Visoka škola strukovnih studija za informacione i komunikacione tehnologije

Indeksi i korišćenje indeksa u bazama podataka

Predmet: Administriranje baza podataka

Smer: Internet tehnologije

Modul: Web programiranje

Profesor: Miroslava Ignjatović

Student: Nikolina Uzelac 9/17

Beograd, 16.6.2020.

Sadržaj

1. Indeksi i prednosti kreiranja i korišćenja indeksa

Indeks je objekat baze podataka koji može da ubrza dohvatanje redova [1]. Jedna od najboljih tehnika za postizanje prihvatljive performanse upita je kreiranje prikladnih indeksa nad tabelama baze [2]. Indeks daje direktni i brzi pristup redovima tabele, a ukoliko nema indeksa u tabeli pretražuje se cela tabela – svaki red u tabeli će biti pročitan kako bi se utvrdilo da li se podaci poklapaju sa specifikacijama upita. Pretraživanje cele tabele je konzumirajuće ukoliko namera upita nije da procesuira aspolutno svaki red u tabeli. Kako indeks predstavlja alternativni put do podataka u bazi, struktura indeksa čini pretragu podataka u bazi lakšom, sa manje I/O operacija. Stoga, upiti se izvršavaju brže kada se koristi indeks za pretragu podataka baziranim na specifičnim ključnim vrednostima.
Database Management System (DBMS) automatski koristi i održava indekse, naročito u relacionom sistemu. Administrator baza podataka (DBA) odatle mora da kreira indekse bazirane na tipu upita koji će se izvršavati nad bazom, ukoliko ih kreira manuelno, uz CREATE INDEX privilegije. Kod automatskog kreiranja, unique indeks se kreira kada se kreira PRIMARY KEY ili UNIQUE kolona.

```
CREATE INDEX index_name
ON table_name
(column1, column2, ...);
```

```
CREATE UNIQUE INDEX index_name
ON table_name
(column1, column2, ...);
```

Slika 1.1. Kreiranje indeksa gde su dozvoljena ponavljanja vrednosti i indeksa gde nisu

Više indeksa ne znači brži pristup. Svaki COMMIT nad DML operacijama tabele zahteva ažuriranje indeksa, što znači da više indeksa povlači više napora DBMS-a da ažurira indekse.

Kreiranje indeksa treba da se bazira na sledećim uslovima:

- Tabela je velika i indeksi se prave kako bi se podržali upiti koji se najčešće pokreću. Upiti koji pristupaju 25% ili manje redova u tabeli su dobri kandidati za indeksiranje;
- Ukoliko kolona sadrži mnogo različitih vrednosti ili mnogo NULL vrednosti;

- Indeksi bi trebalo da se kreiraju za najčešće referencirane kolone u često pokretanim upitima. Redosled kojim se kolone pojavljuju u indeksu je takođe važan. Birajući ispravan redosled, moguće je napraviti određen indeks kojem mnogi drugi upiti mogu pristupiti;
- Jedna tabela može imati više definisanih indeksa, nismo limitirani na jedan indeks, naročito ako se jedna ili više kolona često koristi u WHERE ili JOIN klauzulama.

Sledeće situacije bi trebalo da podstaknu razmatranje kreiranje indeksa:

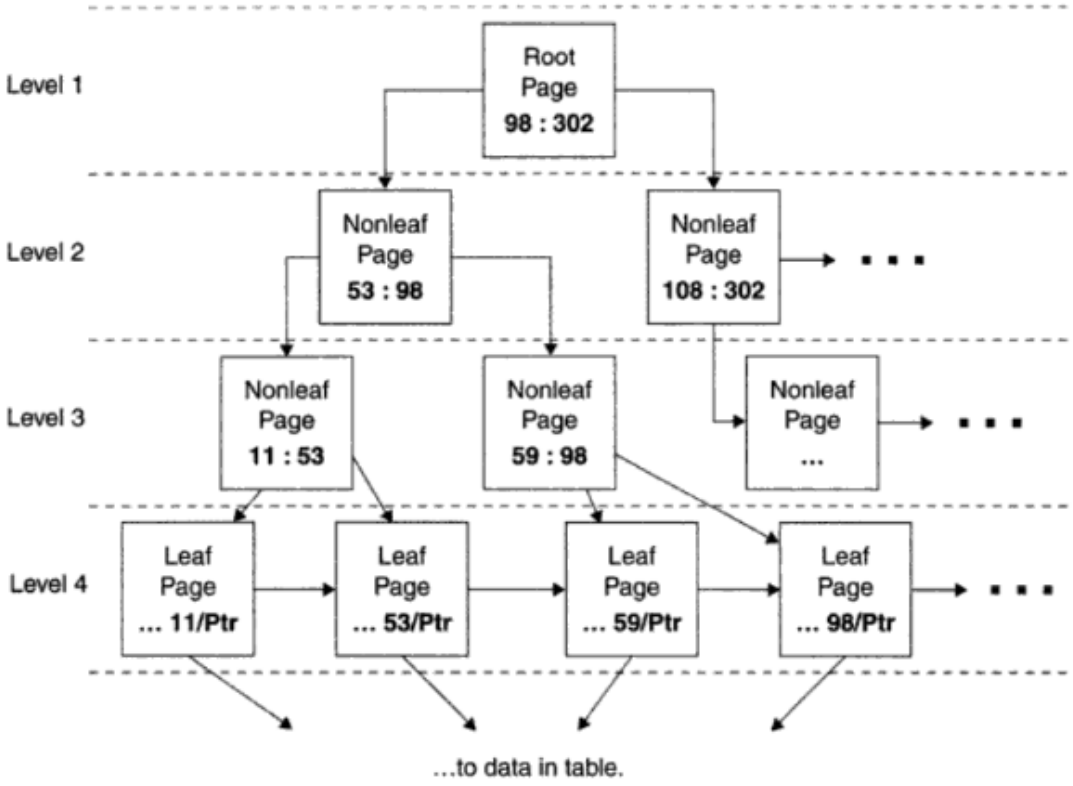
- Strani ključevi – Čak iako DBMS ne zahteva da se kolone stranih ključeva indeksiraju, dobra je praksa uraditi to. Kreiranje indeksa nad kolonama stranih ključeva može poboljšati performanse JOIN-ova koji su bazirani na vezama između tabela, ali mogu isto tako i ubrzati unutrašnju obradu DBMS-a kako bi se osigurao referencijalni integritet;
- Primarni ključevi – Indeks je uobičajeno neophodan nad kolonama primarnih ključeva kako bi se osigurala jedinstvenost;
- Ključevi kandidati – Iako indeksi nisu obavezni nad ključevima kandidatima, dobra je praksa indeksirati ih ukoliko procesi pretražuju podatke na osnovu ključeva kandidata;
- „Index-only access“ – Ako sve kolone iz zahteva za pronalaženje podataka postoje u indeksu, moguće je ispuniti zahtev koristeći samo indeks. Izbegavajući I/O tabele može se poboljšati performansa tako da je nekada dobra ideja prepuniti indeks kolonama da bi se olakšao „index-only access“ za pojedine zahteve. Takav indeks se nekada zove „covering index“ zato što pokriva („covers“) sve podatke koji su zahtevani pojedinim upitima;
- Umanjenje sortiranja – Upiti koji koriste JOIN, ORDER BY, GROUP BY, UNION i DISTINCT mogu da uzrokuju tome da DBMS sortira neposredne ili finalne rezultate tog upita. Ako su indeksi kreirani da podržavaju ovu funkcionalnost, DBMS može da koristi indeks za sortiranje i da izbegne pozivanje konzumirajućeg sortiranja.

2. Vrste indeksa

Većina DBMS-ova podržava veliki broj različitih vrsta indeksa. Indeksi mogu biti jedinstveni ili ne, klasterovani ili neklasterovani, mogu da se sastoje iz jedne ili iz više kolona. Struktura indeksa može biti B-Tree ili Bitmap. Neki DBMS-ovi podržavaju i hashing indekse. U svakom slučaju, osnovni cilj svakog indeksa je da optimizuje obradu upita[3].

2.1. B-Tree indeks

Osnovna tehnika indeksiranja koju podržava većina relacionih sistema baza padataka je b-tree indeks. Njega možemo posmatrati kao balansirano drvo [4].



Slika 2.1.1. Struktura B-Tree indeksa

On počinje od root stranice i širi se do list stranica. Stranice b-tree indeksa referenciramo kao nodove (čvorove). Nodovi postoje u nivoima, sa tim da svaki čvor iznad list nivoa sadrži unose direktorijuma i pokazivače ka nodovima nižeg nivoa. Nodovi na najnižem niovu se zovu list stranice. List stranice sadrže unose sa ključnim vrednostima i pokazivač ka individualnom redu podataka u tabeli. Kako se

podaci dodaju u tabelu, b-tree indeks se ažurira tako što smešta nove ključne vrednosti na odgovarajuću lokaciju u okviru indeks strukture. Za pristupanje podacima koristeći indeks, DBMS kreće od root stranice i prati pokazivače kroz indeks dok se ključna vrednost ne locira na list nivou, gde pokazivač usmerava do konkretnog podatka u tabeli. Svaki roditeljski čvor sadrži najvišu ključnu vrednost koja je smeštena u njegovim direktnim potomcima čvorovima. Listovi stranice b-tree indeksa mogu da se skeniraju za opseg vrednosti nakon što je ključ pretražen. Zahtev za pristup koristeći indeks ima bolje performanse nego pretraživanje cele tabele zbog toga što se traženim podacima može pristupiti direktno koristeći pokazivače iz list čvorova indeksa. Ovo umanjuje I/O i poboljšava performanse za većinu zahteva za pristup podacima.

2.2. Bitmap indeks

Bitmap indeks rešava uži opseg problema, ali pruža odlične performanse. Bitmap indeks je najefikasniji u takozvanim „query-heavy“ tabelama koje se retko menjaju. Najkorisniji su tamo gde kolone koje treba da se indeksiraju imaju vrlo mali broj različitih vrednosti, kao pol (Muški, Ženski, Drugo) ili Boolean (True, False) vrednosti. Skladišta podataka i pregradci za podatke najčešće imaju najviše koristi od bitmap indeksa.

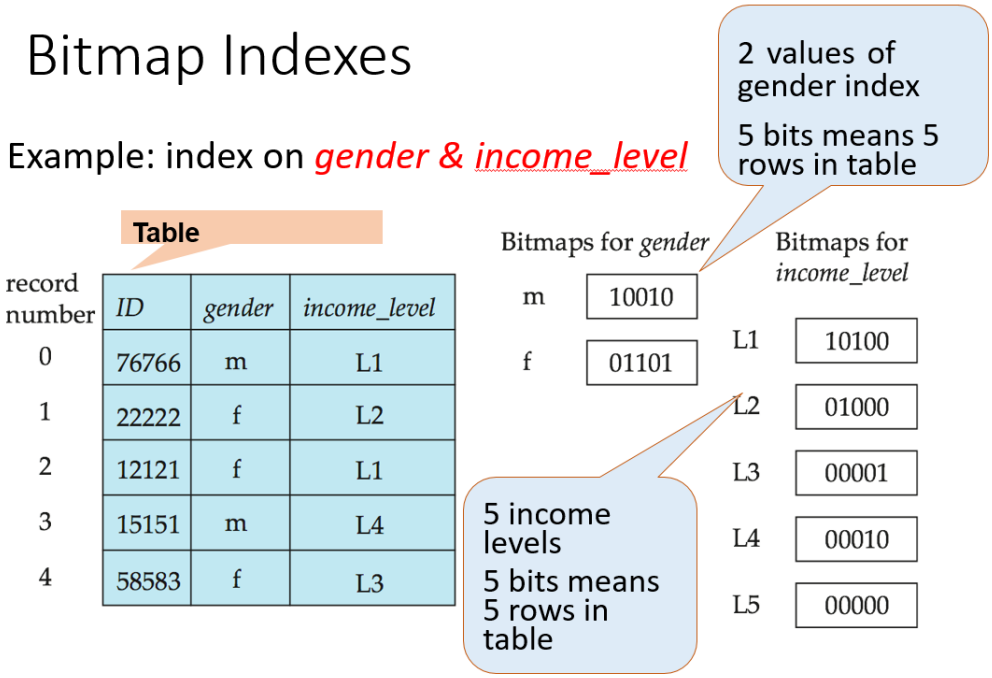
```
CREATE BITMAP INDEX index_name
ON table_name (columns);
```

Slika 2.2.1. Kreiranje bitmap indeksa

Implementacija bitmap indeksa se postiže koristeći string nula i jedinica, ili bita. Za svaku ključnu vrednost bitmap indeksa se čuva poseban string nula i jedinica. Broj različitih ključnih vrednosti određuje broj stringova bita. Na primer, bitmap indeks definisan za kolonu Pol tabele ZAPOSLENI može imati tri stringa, jedan za muški, jedan za ženski, jedan za drugo. Ako posmatramo prethodno pomenuti primer, string je pozicioni, što znači da koja god pozicija ima bit uključen („1“), kolona Pol u tom redu sadrži vrednost za koju je taj posebni string kreiran (muški, ženski, drugo).

Pronalazak seta zapisa sa bilo kojim od nekoliko vrednosti koji su bitmap indeksovani jednostavno zahteva dodavanje stringova za te vrednosti.

Bitmap indeksi mogu biti mnogo brži od pretraživanja tabele, čak i od b-tree indeksa učitavanja pod ispravnim okolnostima. Stringovi bitmap indeksa mogu biti dovoljno mali da se smeste u memoriju, minimizirajući I/O operacije. Čak i za upite koji vraćaju veliki broj redova, kompleksni upit koji koristi bitmap može biti izuzetno efikasan.



Slika 2.2.2. Struktura Bitmap indeksa

Problem sa bitmap indeksima je to što je potreban poseban string za svaku vrednost koja se može pojaviti u polju. Kada kolona može da sadrži veliki broj različitih vrednosti, bitmap indeks nije praktičan iz razloga što to zahteva veliki broj stringova (po jedan za svaku moguću vrednost polja). Dodatno, nule i jedinice bitmap indeksa se ne mogu koristiti za proračune niti mogu biti direktno pročitane da bi se utvrdile konkretne vrednosti koje predstavljaju. Takva ograničenja čine prave bitmap indekse nepraktičnim za većinu aplikacija. Neki DBMS-ovi imaju neku vrstu podrške za proširenje bitmap indeksa kako bi bili praktični za kolone sa većom kardinalnošću.

2.3. Reverse key indeks

Reverse key indeks je u suštini b-tree indeks gde je redosled bitova svake indeksirane kolone obrnut. Redosled kolona u indeksu nije obrnut, samo bitovi unutar svake kolone. Time što je redosled bajtova obrnut postiže se to da susedne

ključne vrednosti nisu fizički smeštene zajedno, tako da ovi indeksi pomažu u raspodeli inače koncentrisanih indeksovanih podataka kroz list čvorove čime poboljšavaju performanse.

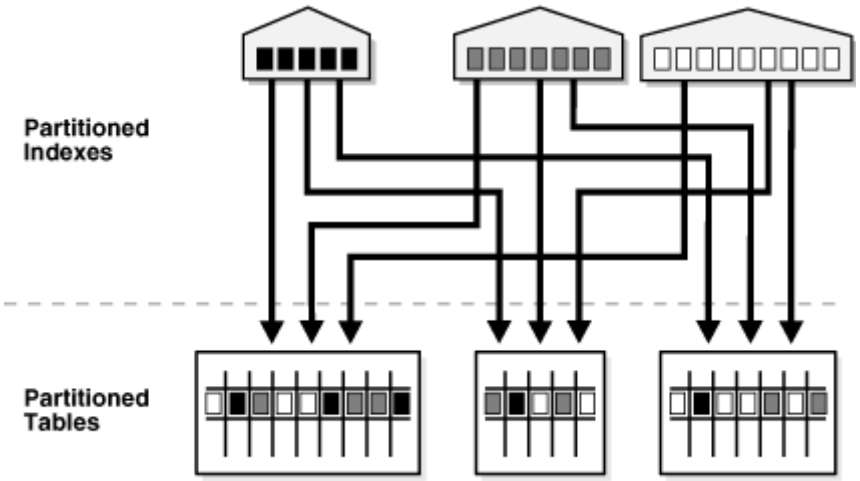
```
CREATE INDEX index_name
ON table_name (column1) REVERSE;
```

Slika 2.3.1. Kreiranje reverse key indeksa

Na primer, ako je reverse key indeks kreiran nad kolonom Ime tabele ZAPOSLENI i vrednost „Craig“ je ubačena, umesto toga se koristi vrednost „giarC“.

2.4. Particionisani indeksi

Particionisani (podeljeni) indeks je takođe u suštini b-tree indeks koji specificira kako da se indeks rasparča (moguće i kod tabela) u podeljene komade, odnosno particije. Particionisanje se uglavnom radi u cilju poboljšanja perfromansi i dostupnosti. Kada su podaci rasprostranjeni preko nekoliko particija, moguće je izvršavati operacije na jednoj particiji tako da to ne utiče na drugu particiju čime se omogućava paralelizam. Paralelni zahtev može biti pozvan da koristi mnoštvo simultanih mehanizama za čitanje jednog SQL iskaza.



Slika 2.4.1. Struktura particionisanih indeksa

```
CREATE TABLE invoices
(invoice_no    NUMBER NOT NULL,
 invoice_date  DATE   NOT NULL,
 comments      VARCHAR2(500))
PARTITION BY RANGE (invoice_date)
```

```
(PARTITION invoices_q1 VALUES LESS THAN (TO_DATE('01/04/2001', 'DD/MM/YYYY'))
TABLESPACE users,
 PARTITION invoices_q2 VALUES LESS THAN (TO_DATE('01/07/2001', 'DD/MM/YYYY'))
TABLESPACE users,
 PARTITION invoices_q3 VALUES LESS THAN (TO_DATE('01/09/2001', 'DD/MM/YYYY'))
TABLESPACE users,
 PARTITION invoices_q4 VALUES LESS THAN (TO_DATE('01/01/2002', 'DD/MM/YYYY'))
TABLESPACE users);

CREATE INDEX invoices_idx ON invoices (invoice_date)
GLOBAL PARTITION BY RANGE (invoice_date)
 (PARTITION invoices_q1 VALUES LESS THAN (TO_DATE('01/04/2001', 'DD/MM/YYYY'))
TABLESPACE users,
  PARTITION invoices_q2 VALUES LESS THAN (TO_DATE('01/07/2001', 'DD/MM/YYYY'))
TABLESPACE users,
  PARTITION invoices_q3 VALUES LESS THAN (TO_DATE('01/09/2001', 'DD/MM/YYYY'))
TABLESPACE users,
  PARTITION invoices_q4 VALUES LESS THAN (MAXVALUE) TABLESPACE users);
```

Slika 2.4.2. Range Partitioning tabele i Global Prefixed particionisani indeksi

2.5. Uređen indeks

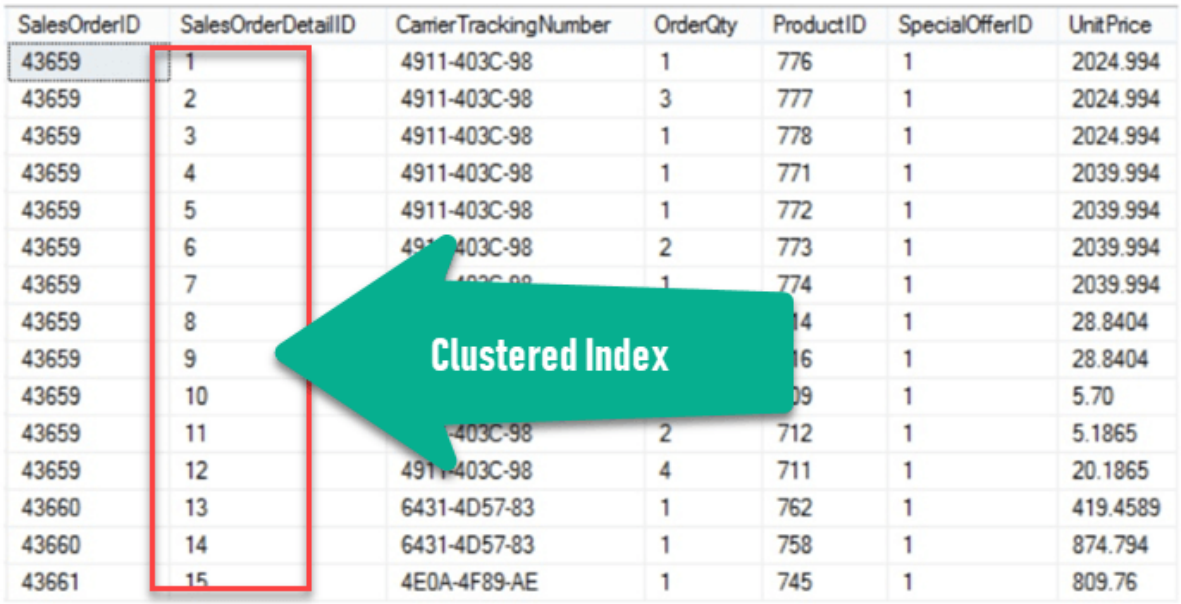
Većina DBMS-ova pružaju opciju da se specificira redosled po kojem su vrednosti b-tree indeksa poređane. Specificirani redosled, bilo da je uzlazni ili silazni, utiče na upotrebljivost indeksa da izbegne operacije sortiranja ili da umanji I/O uslove za vraćanje MIN ili MAX vrednosti. Ovakvi indeksi se kreiraju u ispravnom redosledu kako bi se podržali tipovi upita koji će se izvršavati nad tabelom.

2.6. Klasterovani indeks

Klasterovan indeks sortira i fizički čuva redove podataka na osnovu njihovih ključnih vrednosti[5]. U SQL Serveru ograničenje primarnog ključa automatski kreira klasterovani indeks nad tom kolonom[6]. Samo jedan klasterovan indeks moze da se kreira po tabeli, iz razloga što se redovi podataka u tabeli mogu čuvati u samo jednom redosledu. To je i jedini momenat kada se redovi u tabeli čuvaju u nekom redosledu, baš zato što tabela ima klasterovani indeks.

```
CREATE CLUSTERED INDEX index_name
ON table_name (column1 ASC, column2 DESC);
```

Slika 2.6.1. Primer kreiranja klasterovanog indeksa



Slika 2.6.2. Klasterovani indeks

Kaze se da je tabela u klasterovanom indeksu, odnosno naziva se klasterovanom tabelom, umesto u heap-u kao kod neklasterovanog gde se redovi čuvaju u neuređenoj strukturi. Ne postoji odvojen fajl za indeks, vec je tabela uredjena.

2.7. Neklasterovani indeks

Neklasterovani indeks ima strukturu odvojenu od redova podatka, što će reći da postoje dva fajla: jedan za indekse, a drugi za podatke što zauzima malo više prostora u odnosu na klasterovani indeks.

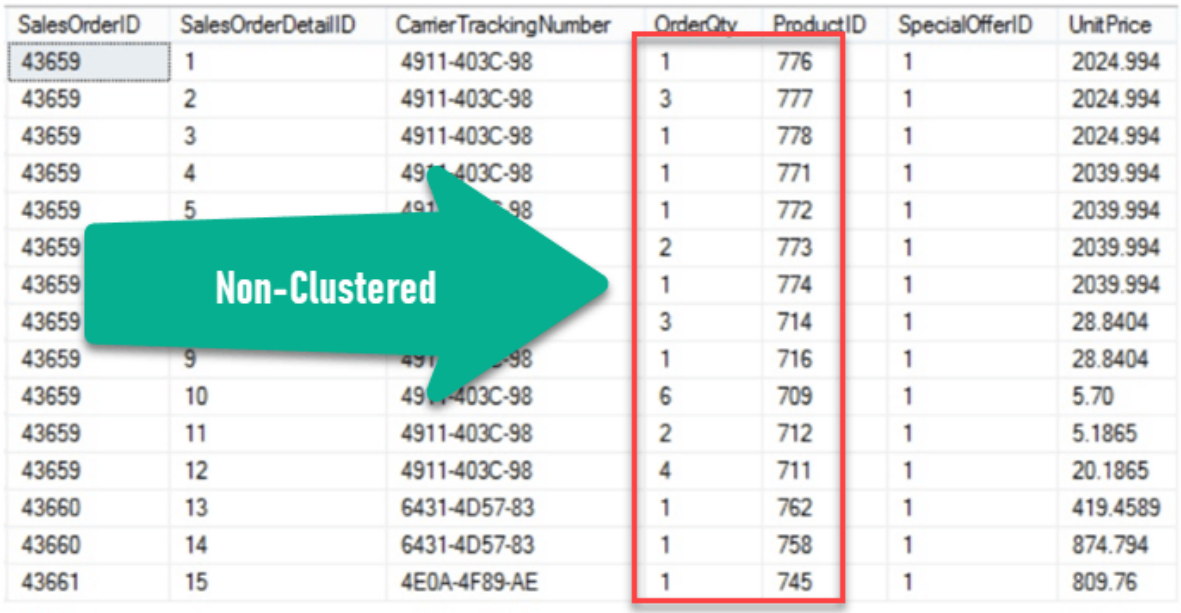
```
CREATE NONCLUSTERED INDEX index_name
ON table_name (column1 ASC);
```

Slika 2.7.1. Primer kreiranja neklasterovanog indeksa

Neklasterovani indeks sadrži ključnu vrednost neklasterovanog indeksa i svaki unos ključne vrednosti sadrži pokazivač ka redu podataka koji čuva ključnu vrednost, ne sortira fizički podatke u tabeli. Taj pokazivač iz reda neklasterovanog indeksa koji vodi ka konkretnom redu podataka se zove lokator reda i njegova struktura zavisi od toga da li su stranice podataka čuvani u heap-u ili u klasterovanoj tabeli.

Za heap, lokator reda je pokazivač direktno na red, dok je za klasterovanu tabelu lokator sam klasterovani indeks. Može postojati više od jednog neklasterovanog

indeksa po tabeli, i ukoliko tabela nema nijedan klasterovani indeks, kaže se da je cela u heap-u.



SalesOrderID	SalesOrderDetailID	CarrierTrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice
43659	1	4911-403C-98	1	776	1	2024.994
43659	2	4911-403C-98	3	777	1	2024.994
43659	3	4911-403C-98	1	778	1	2024.994
43659	4	4911-403C-98	1	771	1	2039.994
43659	5	4911-403C-98	1	772	1	2039.994
43659	6	4911-403C-98	2	773	1	2039.994
43659	7	4911-403C-98	1	774	1	2039.994
43659	8	4911-403C-98	3	714	1	28.8404
43659	9	4911-403C-98	1	716	1	28.8404
43659	10	4911-403C-98	6	709	1	5.70
43659	11	4911-403C-98	2	712	1	5.1865
43659	12	4911-403C-98	4	711	1	20.1865
43660	13	6431-4D57-83	1	762	1	419.4589
43660	14	6431-4D57-83	1	758	1	874.794
43661	15	4E0A-4F89-AE	1	745	1	809.76

Slika 2.7.2. Neklasterovani indeks

3. Primena indeksa u cilju boljih performansi

Kao što je već pomenuto, indeksi se koriste kako bi se ubrzali procesi koje izvršavaju upiti što rezultuje visokim performansama. Možemo ih uporediti sa indeksiranim stranicama knjiga, odnosno sadržajem. Ukoliko želimo da pročitamo određeno poglavlje u knjizi, u sadržaju ćemo proveriti koji je redni broj tog poglavlja i na kojoj se stranici nalazi što nam omogućava da direktno odemo na stranicu koja nam je potrebna. Nije nam neophodno da prođemo kroz celu knjigu stranicu po stranicu kako bismo pronašli neophodne podatke. Bez indeksiranja, proces pronalaženja željenog poglavlja bi bio nepodnošljivo spor.
Ista logika se primenjuje na indeksiranje u bazama podataka. Bez indeksa, DBMS mora prvo da prođe kroz sve podatke u tabeli kako bi vratio potraženi podatak (table-scan) i to je izuzetno spor proces. Ukoliko mudro kreiramo indekse, baza odlazi direktno do konkretnog indeksa i vraća odgovarajuće redove iz tabele.

Administrator baze podataka treba da razume šablone procesa pristupanja tabeli gde se indeks gradi, obzirom da je njihov uticaj velik. Važnu ulogu igraju informacije poput procenta upita koji pristupaju tabeli nasuprot upitima koji modifikuju tabelu,

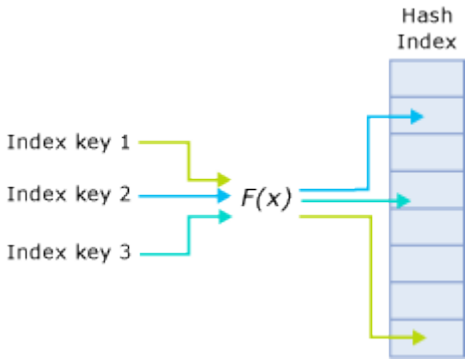
prag performansi koji je postavljen u okviru nivoa servisa kao i uticaj koji će imati u odnosu na reorganizaciju, učitavanje i oporavak.

Objektivni cilj indeksa je uvek bolja performansa i manji I/O, ali koliko nekim upitima indeks pomaže, neke usporava. Strategija efikasnog indeksiranja je najveća moguća redukcija I/O sa prihvatljivim nivoom truda da indeksi ostanu ažurirani. Stoga se testira CPU vreme, proteklo vreme i I/O uslovi kako bi bilo sigurno da indeksi potpomažu performanse.

Često napravljena greška je kreiranje indeksa po objektima, a ne po obimu posla. Indeksiranje po objektima znači da se indeksi kreiraju uporedo sa kreiranjem tabela, što nije optimalno rešenje u toku implementacije fizičkog modela.
Upravo su efikasni indeksi izuzetno važna komponenta fizičkog modela i najvažnija stvar koju administrator baza podataka može da uradi kako bi optimizovao performanse baze podataka.

3.1. Hashing

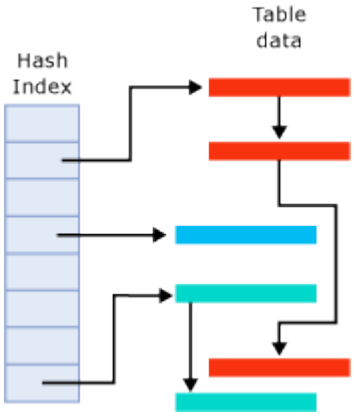
Jedna od tehnika za ubrzavanje pristupa podacima je hashing tehnika, koja koristi indekse kao ulaznu tačku za pristup podacima. Hash indeks se sastoji iz kolekcije „kantica“ organizovanih u niz,a hash funkcija mapira indekse u odgovarajuće „kofice“ u hash indeksu[7]. Hash funkcija je deterministička, isti indeks se uvek mapira u istu „koficu“ u hash indeksu, a moguće je mapirati i više indeksa u istu hash „koficu“.



Slika 3.1.1. Hash indeksi, transformisanje indeksa u „kofice“

Koristi se algoritam koji transformiše ključnu vrednost u pokazivač ka fizičkoj lokaciji reda koji čuva taj podatak. Što je nasumičniji algoritam, to su rezultati bolji, kako bi se izbegla kolizija. Struktura hash indeksa u memoriji se sastoji iz niza pokazivača.

Svaka „kofica“ se mapira u „offset“ u tom nizu, te svaka „kofica“ u nizu ukazuje na prvi red u toj hash „kofici“. Svaki red u „kofici“ ukazuje na sledeći red, rezultujući uvezanom listom redova.



Slika 3.1.2. Hash indeks, tri „kofice“ sa redovima

Broj „kofica“ i količina podataka u okviru indeksa direktno utiču na performanse. Svaka „kofica“ je objekat veličine 8B. Ako se radi sa velikim brojem redova a malim brojem „kofica“, uvezane liste će biti duže i operacije skeniranja će zahtevati brojne korake koji će na kraju degradirati performanse, a naravno, veliki broj „kofica“ a mali broj redova negativno utiču na table-scan i traće memoriju[8]. Ne postoji tačna formula ili zlatna sredina kada se bira broj „kofica“, važno je da se uzme u obzir budući rast u skladu sa prirodom tabele.

```
CREATE TABLE EmpInformation
(
EmpID int NOT NULL IDENTITY(1,1)
PRIMARY KEY NONCLUSTERED,
FirstName nvarchar(25) NOT NULL,
LastName nvarchar(25) NOT NULL,
City nvarchar(25) NULL,
INDEX IX_Hash_FirstName
HASH (FirstName) WITH (BUCKET_COUNT = 100000)
)
WITH (
MEMORY_OPTIMIZED = ON,
DURABILITY = SCHEMA_AND_DATA);
GO
ALTER TABLE EmpInformation
  ADD INDEX IX_Hash_LastName
    HASH (LastName) WITH (BUCKET_COUNT = 80000);
-- Setting BUCKET_COUNT 80000 will result in 131072 buckets
(rounded by the SQL Server to next power of two)
```

Slika 3.1.3. Primer hashing indeksa kroz kreiranje i menjanje tabele

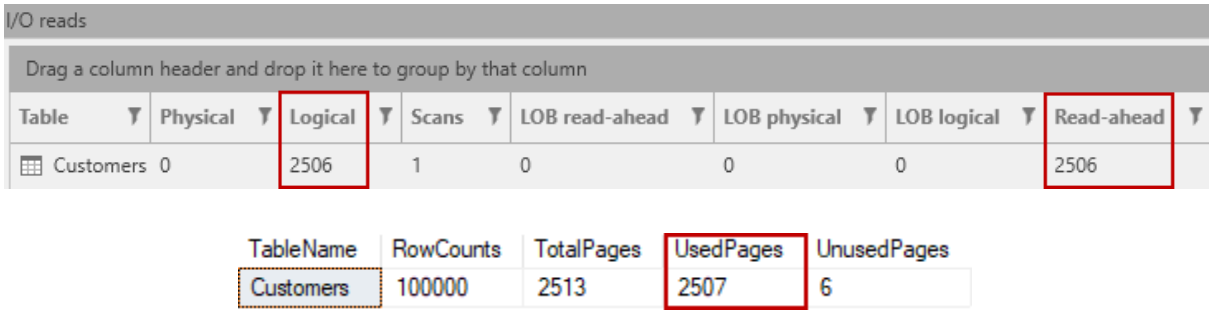
Hash indeksi nisu optimalni u radu sa kolonama sa velikim brojem dupliranih vrednosti, zato što duplirane vrednosti rezultiraju istim hash vrednostima i kreiraju dugačke uvezane liste.

3.2. Klasterovanje

Klasterovanje opisuje način fizičkog smeštanja podataka u tabele, redovi se čuvaju u određenom redosledu koji definiše klasterovani indeks koji je neophodan da bi se klasterovanje podržalo. Podaci kojima se često pristupa su smešteni zajedno na istoj ili na povezanim stranicama baze. Klasterovanje optimizuje performanse zato što je potrebno daleko manje I/O zahteva da bi se podaci vratili.

```
SELECT * FROM Customers where CustomerID = 50000;
```

Slika 3.2.1. Upit za testiranje nad većom tabelom u cilju prikazivanja performansi klasterovanja



Slika 3.2.2. Izvršenje upita nad neklasterovanom tabelom: table-scan je pročitao sve sem jedne stranice korišćene u tabeli

```
CREATE CLUSTERED INDEX CIX_Customers_CustomerID
    ON dbo.Customers (CustomerID);
```

Slika 3.2.3. Kreiranje klasterovanog indeksa

Slika 3.2.4. Izvršenje upita nakon kreiranja klasterovanog indeksa

Iz primera vidimo značajnu razliku i poboljšanje performanse. Obzirom da je tabela ili u heap-u ili je klasterovana, time što smo kreirali klasterovani indeks ona je sada klasterovana. Ali kao bismo pretražili tabelu po nekom drugom, ali čestom kriterijumu, opet bi se radilo skeniranje tabele (samo što bi ovaj put bilo skeniranje uz klasterovani indeks jer je klasterovana tabela) i I/O bi se povećao iz razloga što se koriste i index nodovi za skeniranje. U ovakvim slučajevima moguće je kreirati i neklasterovane indekse kako bi se pospešile performanse, te uz klasterovan indeks (koji može biti samo jedan) postoji i neklasterovan (kojih može biti više).

```
SELECT * FROM Customers WHERE LastName = 'Myers' AND FirstName = 'Kaitlyn';
```

Slika 3.2.5. Upit koji ne pretražuje klasterovani indeks

Drag a column header and drop it here to group by that column

Table	Physical	Logical	Scans	LOB read-ahead	LOB physical	LOB logical	Read-ahead
Customers	0	2550	1	0	0	0	2540

Slika 3.2.6. I/O je porastao nakon izvršenja upita

```
CREATE NONCLUSTERED INDEX IX_Customers_LastName_FirstName
    ON dbo.Customers (LastName, FirstName);
```

Slika 3.2.7. Kreiranje neklasterovanog indeksa

Drag a column header and drop it here to group by that column

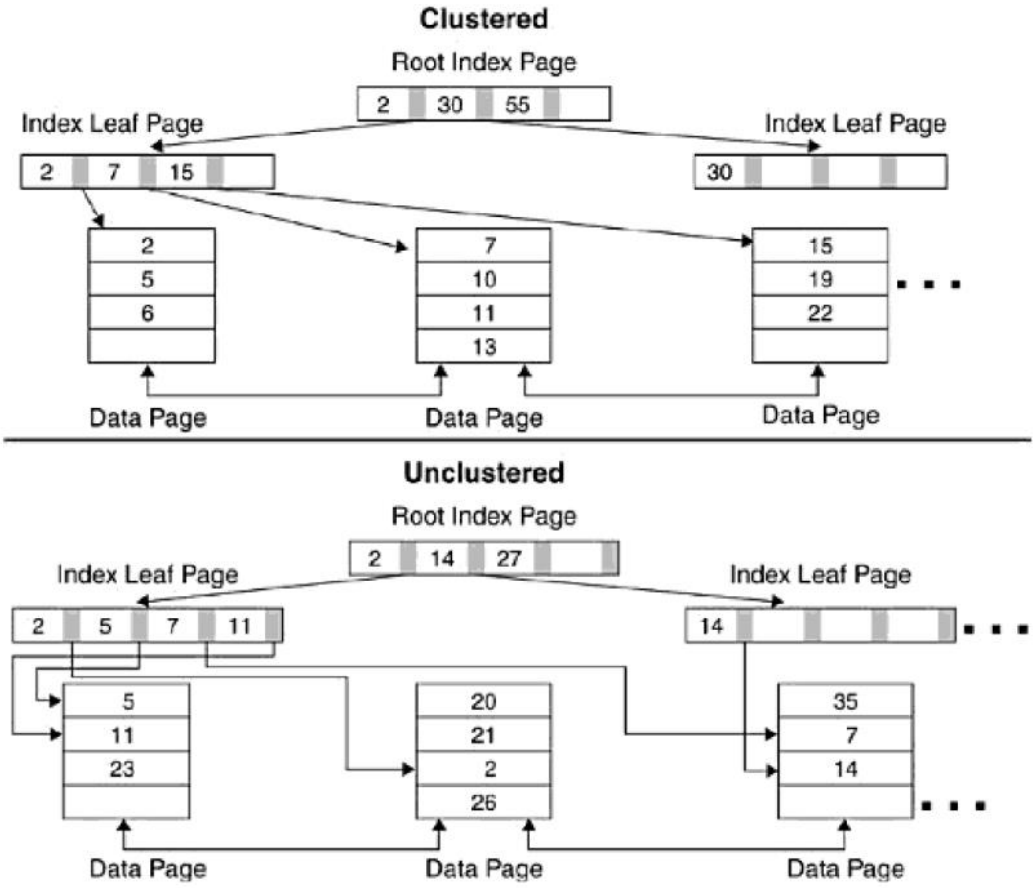
Table	Physical	Logical	Scans	LOB read-ahead	LOB physical	LOB logical	Read-ahead
Customers	51	125	1	0	0	0	2

Slika 3.2.8. Izvršenje upita nakon kreiranja neklasterovanog indeksa (ima više vrednosti upit)

Klasterovanje bi trebalo da se zasniva na sledećim osnovama:

- Veliki broj upita vraća opsege podataka zasnovanim na specifičnim vrednostima kolona;
- Postoji strani ključ u tabeli, odnosno te kolone su često uključenje u JOIN i DBMS pristupa vrednostima stranog ključa kroz referencijalni integritet;
- Kolone se ne menjaju često (smanjuje fizički reklasterovanje);

- Podaci su često sortirani zajedno (ORDER BY, GROUP BY, UNION, SELECT DISTINCT, JOINs).



Slika 3.2.9. Klasterovani i neklasterovani indeks

3.3. Preplitanje podataka

Preplitanje podataka se može posmatrati kao specijalan vid klasterovanja (zapravo, Oracle koristi klaster da definiše ispreplitane podatke). Izvršava se ukoliko su podaci iz dve tabele često JOINovani pa ima smisla fizički ispreplitati podatke u istu fizičku smeštajnu strukturu.

3.4. Particionisanje i paralelizam

Particionisanjem se pospešuje paralelizam. Paralelizam je poželjan jer suštinski smanjuje vreme potrebno za upite nad bazom, naročito kada su u pitanju aplikacije koje pristupaju tabelama i indeksima sa milionima redovima i gigabajtovima

podataka. Koristeći metode particionisanja izbegava se nepotrebno skeniranje tabela i indeksa.

Više tipova paralelizma su zasnovani na resursima koji mogu biti pozvani u paraleli. Na primer, jedan upit može biti rasparčan u mnoštvo zahteva, gde svaki koristi različito CPU jezgro u paraleli. Dodatno, paralelizam se pospešuje i širenjem posla preko nekoliko instanci baze. Svaki DBMS pruža drugačije nivoe podrške za paralelne upite ka bazi.
Na primer, Oracle pruža mogućnosti particionisanja po opsegu, hash particionisanje ili kompozitno particionisanje[9]. Na tabeli koja je na bilo koji od ova tri načina particionisana moguće je kreirati i lokalne i globalne particionisane indekse. Lokalni indeksi nasleđuju particionisane atribiute od njihovih tabela (ako se kreira lokalni indeks na kompozitno particionisanoj tabeli Oracle automatski particioniše lokalni indeks koristeći kompozitnu metodu).

3.5. Index Overloading

Performansi upita se pospešuju u pojedinim situacijama prepunjavanjem indeksa dodatnim kolonama. Indeksi su tipično bazirani na WHERE klauzulama SQL SELECT upita.

```
SELECT zaposleni_id, prezime, plata
FROM Zaposleni
WHERE plata > 15000.00;
```

Slika 3.5.1. Primer SELECT upita

Kreiranje indeksa nad kolonom „plata“ može poboljšati performanse ovog upita. Međutim, administrator baze podataka može dodatno da poboljša performanse upita tako što će prepuniti indeks kolonama „zaposleni_id“ i „prezime“. Sa ovakvim prepunjenim indeksom, DBMS može ispuniti uslove upita koristeći samo indeks. DBMS ne treba da napravi dodatni I/O od pristupanja podacima tabele, obzirom da je svaki zahtevani podatak već u prepunjenom indeksu.

Administratori baza podataka razmatraju prepunjavanje indeksa kako bi podstakli „index-only“ pristup kada mnoštvo upita ima koristi od indeksa ili kada su individualni upiti od izuzetne važnosti.

4. Kada treba izbeći indeksiranje

Postoji nekoliko scenarija kada indeksiranje nije dobra ideja.

Kada su tabele izuzetno male, trebalo bi izbeći indeksiranje. Indeksirani pristup malim tabelama je manje efikasan od jednostavnog skeniranja svih redova zato što čitanje indeksa dodaje I/O zahteve. Na primer, SQL Server će uvek izvršiti table-scan za tabele koje su manje veličine od 64K. Bez obzira, nekada i mala tabela može imati koristi od indeksiranja, na primer, da podstakne jedinstvenost ili ako većina upita vraća jedan red koristeći primarni ključ.

Poželjno bi bilo izbeći indeksiranje kolona sa varljivom dužinom ukoliko DBMS koji se koristi produžava promenljivu kolonu do maksimalne dužine u okviru indeksa. Takve ekspanzije mogu zauzeti neumerenu količinu prostora na disku i može biti neefikasno. Međutim, ako se ove kolone promenljivih dužina koriste u WHERE klauzulama SQL iskaza, treba uporediti cenu memorije diska i cenu skeniranja. Malo više prostora na disku uglavnom ima manju cenu nego traćenje CPU resursa na skeniranje redova. Osim toga, SQL upit može sadržati alternativne predikate koji mogu biti indeksirani umesto kolona sa promenljivom dužinom. Neki DBMS-ovi podržavaju kompresiju indeksa, koja može da se koristi za smanjenje smeštajnih uslova indeksa.

Treba izbeći indeksiranje kolone sa veoma malim brojem vrednosti, odnosno sa malom kardinalnošću. Kada ima tek nekoliko vrednosti, indeks nije od prevelike pomoći i može stvoriti „usko grlo“ u održavanju. Kolona Pol bi bio dobar primer za kolonu sa malim brojem različitih vrednosti.

Dodatno, indeksiranje se izbegava kod tabela kojima se uvek pristupa kroz skeniranje; odnosno, SQL koji se izvršava nad tabelom nikada nema WHERE klauzulu.

Literatura

[1] kursevi.ict.edu.rs : Administriranje baza podataka – DDL naredbe prezentacija

[2] Database Administration: The Complete Guide to DBA Practices and Procedures by Craig S. Mullins

[3] Database Administration: The Complete Guide to Practices and Procedures by Craig Mullins

[4] kursevi.ict.edu.rs : Administriranje baza podataka – Dizajn baze podataka prezentacija

[5] docs.microsoft.com : Clustered and Nonclustered Indexes Described

[6] sqlshack.com : What is the difference between Clustered and Non-Clustered Indexes in SQL Server?

[7] docs.microsoft.com : Hash Indexes

[8] www.sqlshack.com : Deep dive into hash indexes

[9] docs.oracle.com : Index Partitioning