



UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET



Marko Đorđević

Sistemi za upravljanje bazama podataka – Seminarski rad

Tema: Interna struktura i organizacija indeksa kod Oracle baze podataka

Profesor:

Doc. dr Aleksandar Stanimirović

Student:

Marko Đorđević, br. ind. 1168

Niš, April 2021.

Sadržaj

1. Uvod	3
2. Logička struktura skladišta kod Oracle baze podataka	4
2.1. Hijerarhija logičke strukture podataka.....	4
2.1. Šema baze podataka	7
3. Indeksi Oracle baze podataka.....	11
3.1. Podela indeksa	12
3.2. Prednosi i mane indeksa	12
3.3. Pamćenje indeksa.....	13
3.4. Tipovi indeksa	15
3.5. Indeksno skeniranje.....	16
3.6. Kompresija indeksa	17
3.7. B-tree indeksi.....	21
3.8. Bitmap indeksi.....	23
3.8.1. Pristupna putanja Bitmap indexa	23
3.8.2. Razlika između Bitmap i B-tree indeksa na nivou pristupa i pamćenja informacija.....	23
3.9. Indeksi bazirani na funkcijama	25
3.10. Tabele organizovane po indeksu.....	27
4. Primeri	29
4.1. Primer B-tree indeksa.....	29
4.2. Primer Bitmap indeksa.....	35
5. Zaključak	41

1. Uvod

Živimo u svetu u kojem se količina podataka uveliko uvećala u raznim oblastima. Danas svaka organizacija poseduje informacioni sistem, počev od malih organizacija do velikih organizacija, koji je zadužen za čuvanje i obradu informacija. Većina kompanija danas koristi baze podataka za automatizaciju svojih informacionih sistema. Baze podataka predstavljaju kolekciju informacija vezanih za određeni subjekat, namenu ili pojavu. Bazu podataka ne čine samo organizovane tabele kao nosioci podataka, već je to kompletan program koji može proračunati, filtrirati podatke, štampati izveštaje itd.

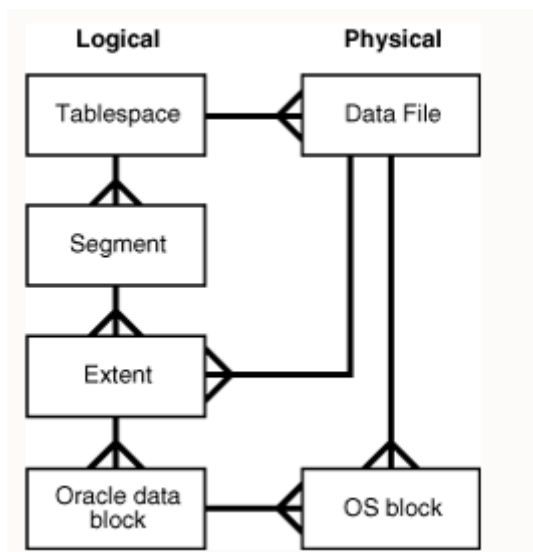
Brigu o podacima, vezama između podataka, ispravnost podataka, sigurnost podataka i sve osobine koje se odnose na podatke preuzima sistem za upravljanje podacima u bazi podataka (DBMS- Database Management System). Svi moderni sistemi za upravljanje bazama podataka koriste relacioni model odnosno koriste sistem za upravljanje relacionim bazama podataka. U svom osnovnom radu iz 1970. godine „Relacioni model podataka za velike zajedničke banke podataka“, E.F.Codd je definisao relacioni model zasnovan na matematičkoj teoriji skupova gde se podaci predstavljani korisniku u vidu tabele i gde svaki red je različit i predstavlja uređenu n-torku relacije R, gde je relacija nista drugo nego tabela koja se sastoji iz kolona i redova. Vrlo je bitno reći da svaka tabela ima isti skup kolona. Relaciona baza podataka je baza podataka koja čuva podatke u relacijama (tabelama).

Sve bržim tehnološkim napredkom i sve većim eksponencijalnim rastom broja prikupljenih podataka javio se problem brze pretrage velike količine podataka. Sve baze podataka teže ka tome da imaju što bolje performanse. Performanse baze podataka se ogledaju u brzini obradi upita i vraćanje rezultata korisniku. Jedan od glavnih faktora koji utiče na performanse jeste i pretraga podataka. Kod skoro svih baza podataka podrazumevano je da se vrši sekvencijalna pretraga podataka. Ovaj način dovodi do pogoršanja performansi za velike baze podataka. Kao rešenje ovog problema predstavljeni su indeksi. Indeksi kod baza podataka predstavljaju posebnu strukturu podataka koja se koristi da bi se ubrzala pretraga podataka. Oni omogućavaju da se smanji broj pristupa disku kako bi se pronašli podaci. Bitno je naglasiti da su indeksi opciona sktutra podataka koju možemo kreirati i da su logički i fizički nezavisni od podataka. Dakle, možemo brisati, kreirati indekse bez uticaja na tabele ili druge indekse.

U ovom radu biće predstavljeni indeksi kod Oracle baze podataka, biće obrađeni primeri na kojima će se videti kako indeksi „pomažu“ pri brzom pretrazi podataka i kako utiču na performanse sistema. U radu korišćena je verzija Oracle 18c baze podataka.

2. Logička struktura skladišta kod Oracle baze podataka

Jedna od glavnih karakteristika RDBMS-a je nezavisnost fizičkog skladištenja podataka od logičkih struktura podataka, kao što su na primer indeksi i tabele. Oracle baza podataka dodeljuje logički prostor za sve podatke u bazi, podaci o tabelama, indeksima. Mi preko logičkih struktura rukovodimo fizičkom strukturom, na primer zadajemo veličinu bloka podataka koji želimo da pamtimo, definišemo kakve podatke čuvamo, i tako dalje. Na slici 1 prikazano je kako je organizovano Oracle skladište, povezanost između fizičke i logičke strukture.

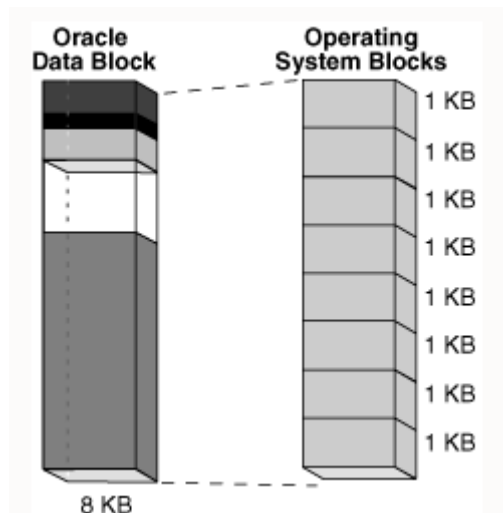


Slika1: Organizacija skladišta kod Oracle baze podataka

U ovom delu upoznaćemo se sa hijerarhijom logičke strukture podataka, upoznaćemo se i sa šemom baze podataka kao i objektima šeme baze podataka.

2.1. Hijerarhija logičke strukture podataka

Na najnižem nivou granularnosti, hijerarhije, Oracle skladišti podatke u blokovima podataka (koji se još nazivaju i logički blokovi, Oracle blokovi ili stranice). Blok podataka predstavlja najmanju jedinicu za skladištenje podataka koje Oracle baza podataka može koristiti ili dodeliti. Jedan blok podataka odgovara određenom broju bajtova prostora fizičke baze podataka na disku. Izgled data bloka prikazana je na slici 2.

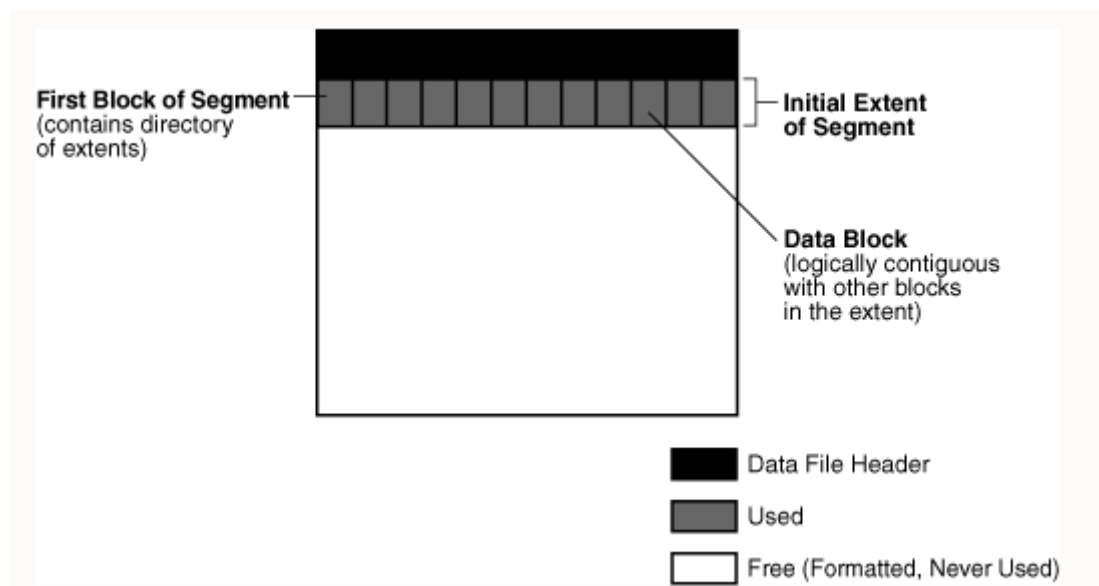


Slika 2: Izgled jednog bloka podataka

Sa slike možemo videti da blok operativnog sistema je minimalna jedinica podatka koju operativni sistem može čitati ili pisati. Suprotno tome, Oracle blok podataka je logička struktura skladišta čija veličina i struktura nisu poznati operativnom sistemu. Baza podataka zahteva podatke u višestrukim blokovima podataka, a ne u blokovima operativnog sistema. Kada baza podataka zatraži blok podataka, operativni sistem prevodi ovu operaciju u zahteve za podacima u trajnom skladištu. Prilikom kreiranja baze podataka moguće je definisati veličinu bloka podataka i ona se ne može promeniti osim ponovnim kreiranjem baze podataka. Standardna veličina bloka podatka je 4KB ili 8KB.

Na sledećem nivou prostora logičke strukture baze podataka je extent. Extent je određeni broj susednih blokova podataka dodeljenih za čuvanje određene vrste informacija. Baza podataka podrazumevano dodeljuje početni opseg za segment podataka kada se segment kreira. Količina je uvek sadržana u jednoj datoteci podataka (eng. Data file). Ukoliko se početni extent popuni i ako je potrebno više prostora, baza podataka automatski dodeljuje inkrementalni extent za ovaj segment. Inkrementalni extent je naknadni extent stvoren za extent. Algoritam dodeljivanja zavisi od toga da li se prostorom tabela upravlja lokalno ili upravlja rečnikom (eng. dictionary-managed). U lokalnom slučaju baza podataka pretražuje bitmapu datoteke podataka za susedne slobodne blokove. Ako datotetka podataka nema dovoljno prostora, tada baza podataka traži drugu datoteku podataka. Proširenja za extent su uvek u istom prostoru tabela, ali mogu biti u različitim datotekama podataka. Lokalni prostor može imati ili iste veličine extenta ili promenljive veličine extenda koju automatski određuje sistem. Svi extenti u prostoru tabela su veličine 1MB.

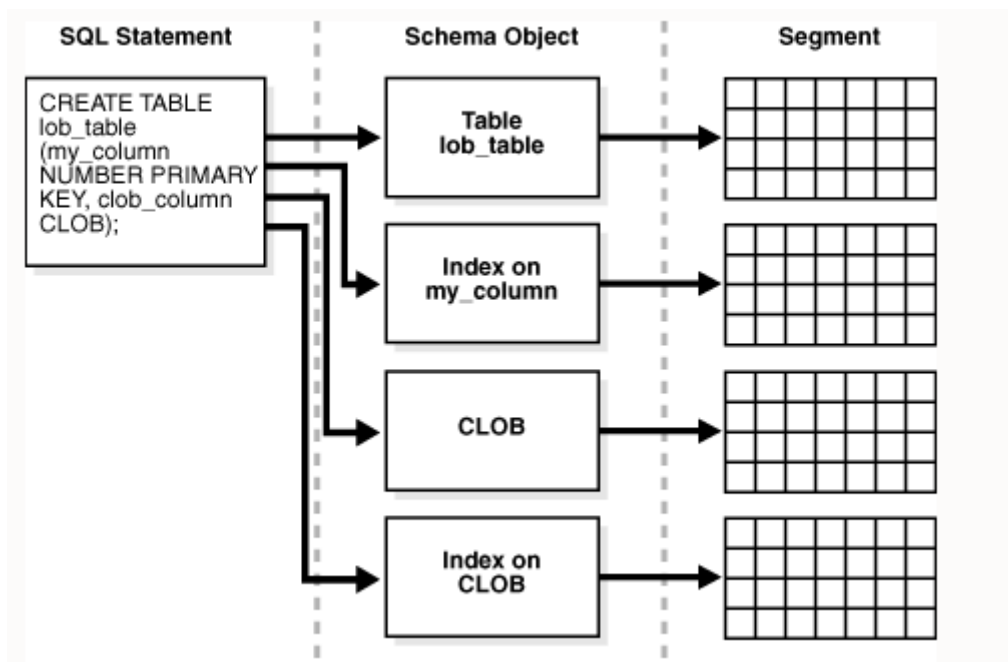
Na slici 3 je prikaz početni obim extent-a



Slika 3: Početni obim extent-a

Nakon extent-a na sledećem nivou granularnosti je segment. Segment je skup extent-a dodeljnih određenom objektu baze podataka kao što je na primer tabela. Na primer, ukoliko pamtimo informacije o zaposlenima, podaci za tabelu zaposlenih se čuvaju u svom segmentu, dok se svaki indeks za zaposlene čuva u svom segmentu indeksa. Jedan segment podataka u bazi čuva podatke za jedan korisnički objekat, odnosno objekat šeme podataka (poglavlje 2.2). Postoje različite vrste segmenata, na primer tabela, particija table ili klaster tabela, LOB ili LOB particija, indeks ili indeksna particija. Svaki neparticionirani objekat i particija objekta čuvaju se u svom segmentu. Podrazumevano baza podataka koristi odloženo kreiranje segmenta za ažuriranje samo metapodataka baze podataka prilikom kreiranja tabela, indeksa i particija. Kada korisnik unese prvi red u tabelu ili particiju, baza podataka kreira segmente za tabelu ili particiju. Paket DBMS_SPACE_ADMIN upravlja segmentima za prazne objekte.

Da bi smo videli kako praktično izgleda kreiranja segmenta, na slici 4 dat je primer preuzet iz dokumentacije koji ilustruje kako se kreiraju segmenti nakon naredbe CREATE, ovde ćemo podrazumevati je opcija odloženog kreiranja segmenta isključeno.



Slika 4: Primer kreiranja segmenata nakon izvršenja naredbe CREATE

Vrlo je bitno naglasiti da segmenti tabele i indeksa su odvojeni.

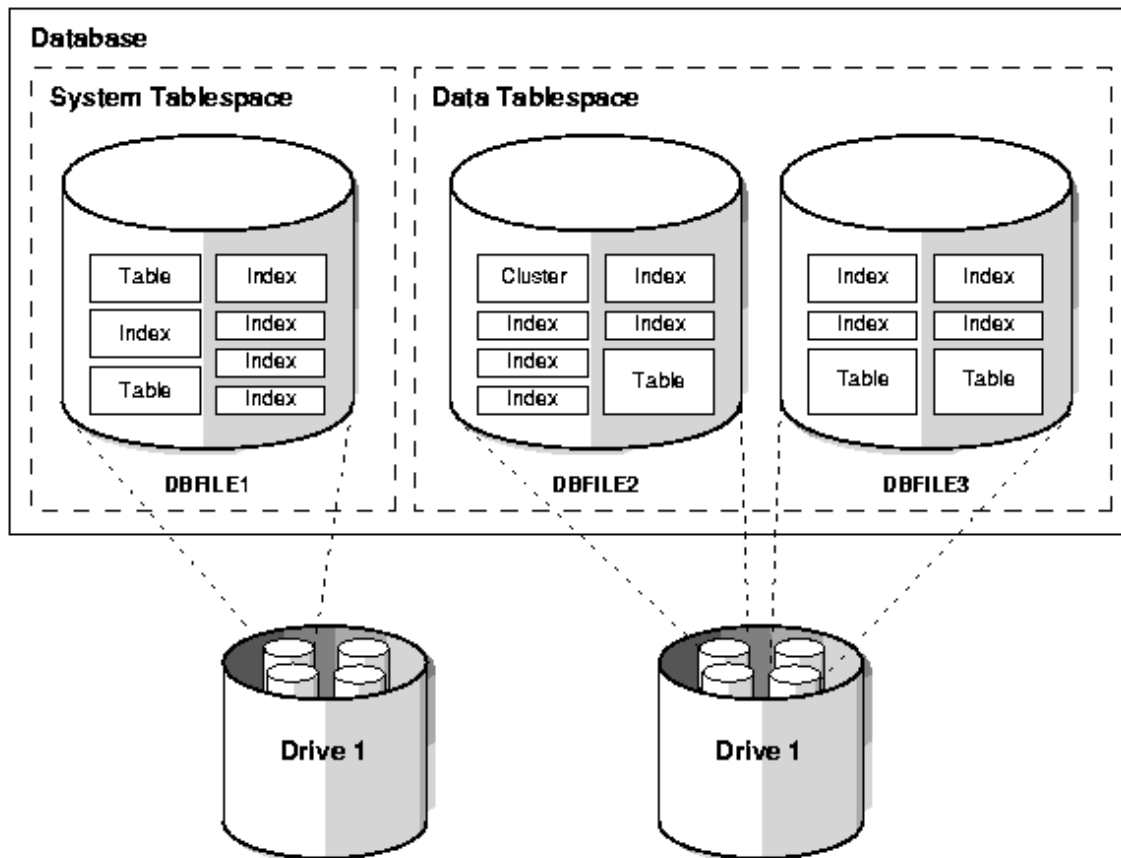
Na krajnjem nivou glanularnosti su prostori tabela. Prostor tabela je jedinica za skladištenje baze podataka koja sadrži jedan ili više segmenata. Svaki segment pripada jednom i samo jednom prostoru tabela. Na fizičkom nivou, prostor tabela skladišti podatke u jednoj ili više datoteka podataka ili privremenih datoteka.

2.1. Šema baze podataka

Proces razvijanja baze podataka podrazumeva formiranje modela realnog sistema, na osnovu izabranih značajnih karakteristika sistema. Modeliranje strukture podataka je zadatak projektanta baze podataka i obuhvata aktivnosti koje se odnose na otkrivanje i izbor podataka koji baza podataka mora pamtit i obrađivati. Ovako dobijene informacije su osnova za kreiranje modela baze podataka izabranim modelom podataka.

Model podataka je integrisana kolekcija koncepata za opis i manipulaciju podacima, vezama između podataka i ograničenjima nad podacima i vezama. Model podataka sadrži skup koncepata i konstrukcija koji se koristi za opis sadržaja baze podataka i veza između podataka. Taj opis koji se zove i šema baze podataka, se odnosi na tip činjenica uključenih u bazu podataka i pravila i ograničenja koja postoje.

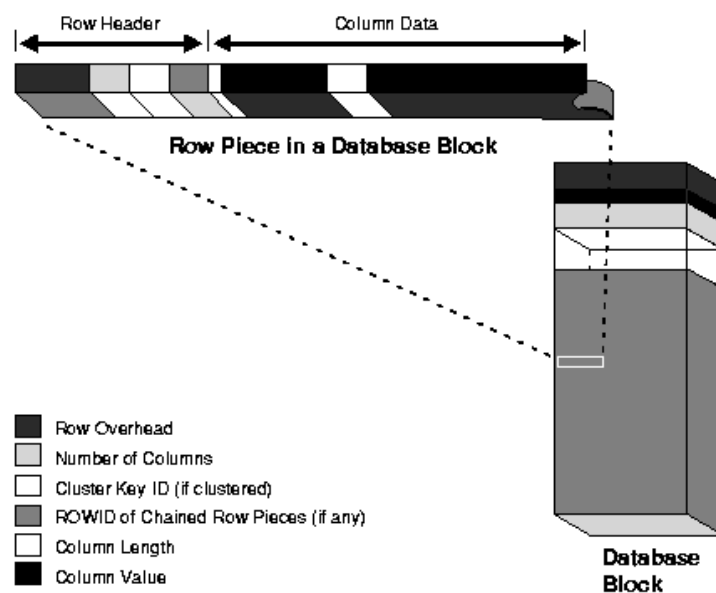
U Oracle bazi podataka šema baze podataka predstavlja kolekciju objekata baze podataka ili logičkih struktura podataka. Sa svakim korisnikom baze podataka povezana je šema baze podataka koja uključuje sledeće objekte: tabele, pogleda, sekvence, sinonime, indekse, klastere, veze do baze podataka, procedure i pakete. Vizuelni prikaz šeme objekata baze prikazan je na slici 5.



Slika 5: Vizuelni prikaz obejkata šeme baze podataka

Objekti šeme Oracle baze podataka:

1. Tabele su osnovna jedinica za skladištenje podataka u Oracle bazi podataka. Podaci se čuvaju u redovima i kolonama. Tabelu definišemo imenom tabele i skupom kolona. Svaka kolona ima svoje ime, tip podatka i veličinu tipa podatka koja pamti određena kolona tabele. U tabeli podaci su smešteni u redovima. Red je skup podataka o koloni koji odgovaraju jednom zapisu u bazi podataka. Za svaki kolonu možemo specificirati pravila koja se nazivaju ograničenja integriteta. Na primer, jedno od ograničenja integriteta je NOT NULL. Kada je ovakvo ograničenje navedeno upitu za kreiranje tabela to znači da vrednost te kolone ne može biti nula, odnosno primorava da sadrži vrednost u svakom redu. Kada kreiramo tabelu, Oracle automatski dodeljuje segment podataka u prostoru tabela za čuvanje budućih podataka tabele. Nakon kreiranja tabele podaci se upisuju u redovima. Oracle čuva svaki red tabele baze podataka koji sadrži podatke za manje od 256 kolona kao jedan ili više redova podataka. Ukoliko je naša tabela ima manje od 256 kolona jedan red će biti sačuvan kao ceo red, neće biti potrebe za čuvanje u više redova, u obrnutom slučaju kolone će biti ulančane i smeštene u istom bloku. Pomoću ovog ulančavanja dobićemo sve podatke u istom bloku. Na slici 6 prikazan je kako izgleda jedan red sačuvan u bloku podataka



Slika6: Izgled jednog reda zapamćenog u bloku podataka

Red sadržan u jednom bloku ima najmanje 3 bajta zaglavlja reda, koji sadrži informacije o redu, ukoliko je ulančan sadrži informacije o drugom delu reda, kolone u redu, ključeve klastera.

Prostor potreban za podatke u koloni zavisi od tipa podataka, ako je tip podataka kolone promenljive dužine, prostor potreban za čuvanje vrednosti može da raste i smanjuje se ažuriranjem podataka. Bitna stvar je reći da Oracle ne čuva podatke o kolonama koji imaju vrednost nula.

Rowid identifikuje svaki deo reda prema njegovoj lokaciji ili adresi. Praktično rowid nam identifikuje gde smo sačuvali podatak, odnosno njegovu lokaciju na fizičkom medijumu.

2. pogledi (eng. Views) – Pogled je prilagođena reprezentacija podataka sadržanih u jednoj ili više tabela ili drugim prikazima. Pogled uzima izlaz upita i tretira ga kao tabelu. Stoga ne možemo na pogled gledati kao skladišteni upit ili kao virtuelnu tabelu. Za razliku od tabele, pogledima se ne dodeljuje prostor za skladištenje, niti pogled zapravo sadrži podatke. Umesto toga, pogled je definisan upitom koji izdvaja ili izvodi podatke iz tabela na koje se referenca odnosi. Pogledi pružaju način da se predstavi drugačiji prikaz podataka koji se nalaze u osnovnim tabelama.

3. indeksi – Indeksi su objekti šeme koji sadrže unos za svaki indeksirani red tabele ili klastera tabela i pružaju direktan, brz pristup redovima (detaljnije o indeksima u poglavlju 3).

4. sinonimi – Sinonim je pseudonim za drugi objekat šeme. Budući da je sinonim jednostavno pseudonim, on ne zahteva nikakvo skladištenje osim njegove definicije u rečniku podataka.

5. Particije – Particije su delovi velikih tabela i indeksa. Svaka particija ima svoje ime i po želji može imati svoje karakteristike memorije.

6. Sekvence - Niz je korisnički kreiran objekat koji može da deli više korisnika sistema za generisanje celih brojeva. Tipično se koriste za generisanje vrednosti primarnog ključa.

7. Dimenzije – Dimenzija definiše odnos roditelja i deteta između parova skupova kolona, pri čemu sve kolone skupa kolona moraju poticati iz iste tabele. Dimenzije se obično koriste za kategorizaciju podataka.

8. PL/SQL potprogrami i paketi – PL/SQL je Oracl proceduralno proširenje SQL-a. PL/SQL potprogram je imenovani PL/SQL blok koji se može pozvati skupom parametara. Paket, PL/SQL grupiše povezane PL/SQL tipove, promenljive i potprograme.

3. Indeksi Oracle baze podataka

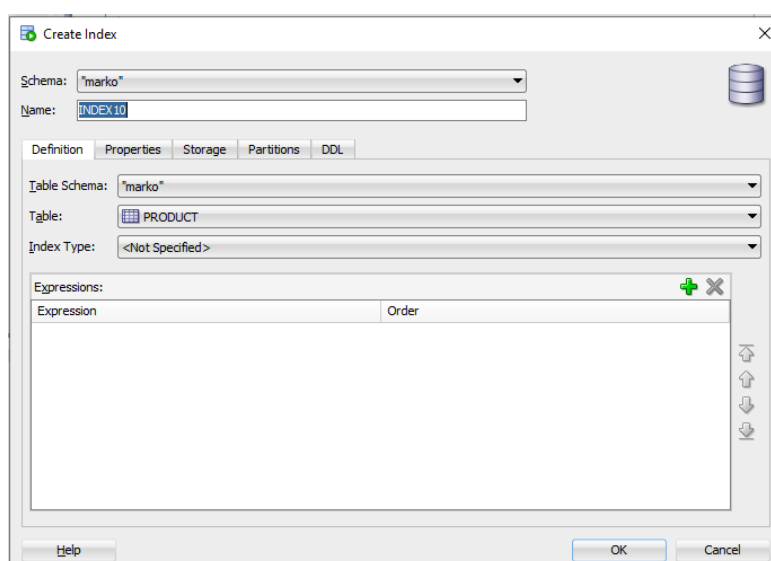
Indeks je opcionalno kreiran objekat baze podataka koji se koristi prvenstveno za povećanje performansi upita. Svrha indeksa baze podataka je povećanje performansi pretraživanja podataka i slična je indeksu na poleđini knjige. Indeks knjiga povezuje temu sa brojem stranice. Kada locirate informacije u knjizi, obično je mnogo brže prvo pregledati indeks, pronaći temu koja vas zanima i identifikovati pridružene brojeve stranica. Pomoću ovih informacija možete direktno da dođete do određenih brojeva stranica u knjizi. Ako se tema pojavljuje samo na nekoliko stranica u knjizi, onda je broj stranica za čitanje minimalan. Na taj način, korisnost indeksa opada sa povećanjem broja pojavljivanja teme u knjizi.

Prilikom obrade upita, baza podataka može koristiti dostupne indekse za efikasno lociranje traženih redova. Indeksi su korisnički kod aplikacije koje često postavljaju upit za određeni red ili opseg redova. Indeksi su logički i fizički nezavisni od podataka. Dakle, možemo brisati, kreirati indekse bez uticaja na tabele ili druge indekse. Sve aplikacije uspešno nastavljaju da funkcionišu i nakon što se indeksi obrišu.

Indekse kreiramo sledećom SQL naredbom:

```
CREATE INDEX <naziv_indeksa> ON <ime_tabele> (<Kolone>);
```

Pored ove naredbe, sql developer, ima napravljen korisnički interfejs za lakše kreiranje indeksa. Izgled interfejsa dat je na slici 7.



Slika 7: Interfejs za kreiranje indeksa u sql developer okruženju

Korisniku se nudi mogućnost podešavanja indeksa i svih ostalih parametara o kojima ćemo pričati u nastavku.

Vrlo je bitno napomenuti da su ključevi indeksa ustvari zadate kolone u naredbi CREATE INDEX, i da na osnovu njih se pristupa vrednosti indeksa za datu kolonu. Broj kolona se kreće u opsegu od jedan pa do koliko kolona ima tabela u bazi podataka. Indeksi koji indeksiraju više kolona zovu se jos i kompozitni indeksi.

Svaka tabela baze podataka može posedovati više indeksa koje se odnose na datu tabelu ali samo pod uslovom da su indeksi različitog tipa (moguće je kreirati nad jednom kolonom više tipova indeksa), možemo kreirati indekse koji se nalaze gde se nalazi i tabela na koju se odnose, ili mogu da se nalaze u drugom prostoru tabela ali pri tome moramo voditi računa da je u jednom trenutku vidljiv samo jedan indeks, indeks možemo napraviti kao jedinstveni i nejedinstveni indeks koji se odnosi na isti skup kolona.

3.1. Podela indeksa

Indeksi mogu biti upotrebljivi (podrazumevani) ili neupotrebljivi, vidljivi (podrazumevani) ili nevidljivi. Ova svojstva definisana su na sledeći način:

1. Upotrebljivost - DML operacije na održavaju neupotrebljiv indeks, koji optimizator ignoriše. Neupotrebljiv indeks može poboljšati performanse rasutih tereta. Umesto da obrišemo indeks i kasnije ga ponovo kreiramo, možemo ga učiniti neupotrebljivim zatim ga ponovo napraviti. Neupotrebljivi indeksi i particije indeksa ne troše prostor. Kada upotrebljivi indeks učinite neupotrebljivim, baza podataka briše svoj segment indeksa.

2. Vidljivost – DML operacije održavaju nevidljiv indeks, ali ga optimizator podrazumevano ne koristi. Učiniti indeks nevidljivim je alternativa tome da ga učinimo neupotrebljivim ili obrisanim. Nevidljivi indeksi su posebno korisni za testiranje uklanjanja indeksa pre njegovog brisanja ili privremene upotrebe indeksa bez uticaja na celokupnu aplikaciju.

Pored ovog svojstva na osnovu broja kolona na osnovu koga se vrši indeksiranje indeksi mogu biti jedinstveni ili nejedinstveni. Jedinstveni indeksi garantuju da nijedna dva reda tabele nemaju duplirane vrednosti u ključnoj koloni ili kolonama (kada se kaže ključna kolona, odnosi se na to za koje kolone je kreiran indeks). U jedinstvenom indeksu postoji jedan ROWID za svaku vrednost podataka. Podaci u blokovima lista sortirani su samo po ključu.

Nejedinstveni indeksi dozvoljavaju dupliranje vrednosti u indeksiranim kolonama ili koloni. Za nejedinstvene indeksi, ROWID je uključen u ključ u sortiranom redosledu, tako da su nejedinstveni indeksi sortirani po indeksnom ključu i rowid-u u rastućem redosledu. Oracle baza podataka ne indeksira redove tabela u koji su svi elementi jednaki nuli, osim indeksa bitmap ili kada je vrednost kolone ključa klastera nula.

3.2. Prednosti i mane indeksa

Prilikom pretraživanja podataka mi pišemo sql upit koji nam izvršava pretraživanje sa bazom. Odsustvo ili prisustvo indeksa ne zahteva promenu bilo kod SQL upita.

Indeks je brza pristupna putanja do jednog reda podataka i utiče samo na brzinu izvršenja. S obzirom na vrednost podataka koja je indeksirana, indeks upućuje direktno na lokaciju redova koji sadrži tu vrednost. Kada indeks postoji na jednoj ili više kolona tabele, baza podataka u nekim slučajevima može iz tabele preuzeti mali skup nasumično raspoređenih redova. Indeksi

su jedno od mnogih sredstava za smanjenje I/O pristupa disku. Ako imamo u bazi podataka tabele koje su organizovane kao heap i ne postoje indeksi, tada baza podataka mora izvršiti potpuno skeniranje tabele da bi pronašla vrednost. Ovakav pristup skeniranja nije dobar jer sa porastom podataka raste i vreme pretraživanje baze podataka. S tim u vezi indeksi rešavaju problem brzog traženja podataka.

Nedostaci indeksa su sledeći:

1. Ručno kreiranje indeksa često zahteva duboko poznavanje modela podataka, primene i distribucije podataka.
2. Kada se podaci menjaju, moramo ponovo pregledati prethodne indekse. Indeks može biti beskoristan jer ukazuje na kolonu koja sada ima izmenjenu vrednost.
3. Vrlo bitno je da posedujemo svest o tome da indeksi zauzimaju prostor na disku. Samim povećanjem indeksa povećava se i prostor za njihovo smeštanje.
4. Baza podataka mora ažurirati indeks kada se DML pojavi na indeksiranim podacima, što stvara režijske troškove. DML naredba su elementi u SQL jeziku koji se koriste za pronalaženje i manipulaciju podacima. Koristeći ove naredbe moguće je izvršiti operacije kao što su: dodavanje novih redova, ažuriranje i brisanje postojećih redova, spajanje tabela i tako dalje.

Sa verzijom 19c Oracle baze podataka, Oracle baza može neprestano nadgledati opterećenje aplikacije, automatski kreirajući i upravljajući indeksima. Automatizovano indeksiranje se implementira kao zadatak baze podataka koji se izvodi u fiksnom intervalu.

3.3. Pamćenje indeksa

Nakon kreiranja indeksa, Oracle automatski dodeljuje segment indeksa za održavanje podataka indeksa u prostoru tabela (eng. Table space). Takođe bitno je napomenuti da nakon kreiranja indeksa, ukoliko se desi promena vrednosti kolone, to će se automatski reflektovati i na same indekse, nije potrebno ažurirati indekse, ovo važi za verzije veće od 19c.

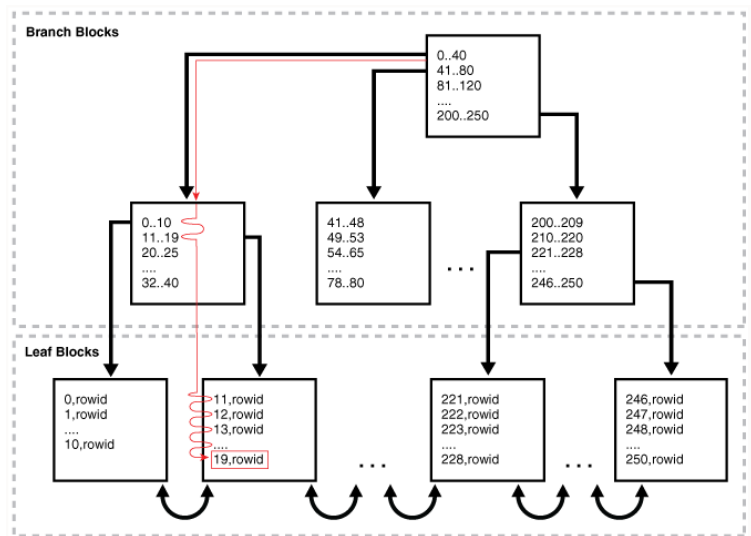
Pre nego što detaljnije objasnimo kako se indeksi pamte u Oracle bazi podataka potrebno je da definišemo sta je to overhead blok. Zaglavlje bloka podataka, direktorijum tabela i direktorijum redova zajednički se nazivaju overhead blok (srp. Nadređeni). Neki overhead blokovi su fiksne veličine dok je ukupna veličina overhead bloka promenljive dužine. U proseku, fiksni i promenljivi delovi overhead-a bloka podatka ukupno čine 88 do 107 bajtova.

Prostor dostupan za podatke indeksa u bloku podataka je veličina bloka podataka umanjena za overhead blok, opsega unosa, ROWID-a i jedan bajt dužine za svaku indeksiranu vrednost. Prostor tabela indeksnog segmenta je ili zadati prostor tabela vlasnika ili prostor tabele posebno imenovan u naredbi CREATE INDEX. Radi lakšeg administriranja, indeks možemo sačuvati u odvojenom prostoru tabele od njegove tabele. Na primer, možemo odlučiti da ne pravimo rezervne kopije prostora tabela koja sadrže samo indekse, koji se mogu obnoviti, i tako smanjimo vreme i skladište potrebne za rezervne kopije.

Kao što smo i rekli, indeksi se smeštaju u posebnom segmentu prostora tabela, i svaka indeksna tabela sastoji se od indeksnih blokova koja predstavlja posebnu vrstu bloka podataka koja upravlja prostorom drugačije od blokova tabela. Oracle baza podataka koristi blokove indeksa za upravljanje logičkim prostorom za skladištenje u indeksu. Postoje 3 tipa indeksnih blokova a to su:

1. Root block – ovaj blok identifikuje ulaznu tačku sledeći indeks blok.
2. Branch block – Prilikom pretraživanja index ključa baze podataka se kreću kroz takozvane branch blokove
3. Leaf block - Ovi blokovi sadrže indeksirane vrednosti (ključeva, ROWID-a) koji upućuju na pridružene redove.

Leaf blokovi skladište vrednost ključeva u sortiranom redosledu tako da baza podataka može efikasno pretražiti sve redove u opsegu vrednosti ključeva. Pod terminom ključ se kod indeks bloka misli na kolonu koju referencira indeks, odnosno kolonu koju smo naveli prilikom kreiranja indeksa. Prikaz strukture dat je na slici 8.



Slika 8: Prikaz strukture pamćenja indeksa

Kada smo definisali kako se pamte indeksi u skladištu, sada ćemo objasniti kako se pamte indeksi u blokovima indeksa.

Unosi indeksa čuvaju se u blokovima indeksa na isti način kao i redovi tabele u bloku podataka. Unosi indeksa u delu bloka ne čuvaju se u binarnom redosledu, već u gomili. Baza podataka upravlja direktorijumom redova (eng. row directory) u bloku indeksa drugačije od direktorijuma u bloku podataka. Unosi u direktorijumu redova (ne odnosi se na unose u sam blok indeksa) poređani su prema vrednosti ključa. Na primer, u direktorijumu redova, unos direktorijuma za indeksni ključ 000000 prethodi unosu direktorijuma za indeksni ključ 111111 i tako dalje. Redosled unosa u direktorijum redova poboljšava efikasnost skeniranja indeksa. Pri skeniranju opsega, baza podataka mora da pročita sve indeksne ključeve navedene u opsegu. Baza podataka obilazi branch blokove da bi identifikovala leaf blok koji sadrži pravi ključ. Budući da su unosi u direktorijum redova sortirani, baza podataka može pomoću

binarnog pretraživanja pronaći prvi indeksni ključ u opsegu, a zatim sekvencijalno pretraživati kroz unose u direktorijumu redova dok ne pronađe poslednji ključ. Na ovaj način baza podataka izbegava čitanje svih ključeva u telu leaf blokova.

Baza podataka može ponovo koristiti prostor unutar bloka indeksa. Blok indeksa obično ima mnogo više redova od bloka tabele organizanih u gomilu. Sposobnost čuvanja mnogih redova u jednom bloku indeksa, olakšava bazi podataka da održi indeks jer izbegava česte podele bloka za čuvanje novih podataka. Vrlo bitna karakteristika je ta da indeks se ne može spojiti, iako ga možemo ručno spojiti koristeći naredbu ALTER INDEX sa opcijama Rebuild ili COALESCE. Na primer, ako popunimo kolonu sa vrednostima od 1 do 500000, a ako zatim obrišemo redove koji sadrže parne brojeve, indeks će sadržati 250000 praznih mesta. Baza podataka ponovo koristi slot samo ako može da upiše podatke koje se uklapaju u indeksni blok koji sadrži prazno mesto.

Da bi baza podataka ponovo koristila slot, neophodno je izvršiti spajanje indeksnog bloka. Spajanje indeksa sabija postojeće podatke indeksa na mestu i ako reorganizacija oslobađa blokove, ostavlja slobodne blokove u strukturi podataka. Dakle spajanje ne oslobađa indeksne blokove za druge namene niti uzrokuje da indeks preraspoređuje blokove. Oracle baza podataka ne sabija automatski indeks već korišćenjem komande ALTER INDEX sa opcijama REBUILD ili COALESCE opcijama.

3.4. Tipovi indeksa

Indeksi B-stabla su standardni tip indeksa. Pretežno se koriste za visoko selektivne indekse (nekoliko redova odgovara svakom unosu indeksa) i indekse primarnog ključa. Koristi se kao spojeni indeksi, indeks B-stabla može vratiti podatke sortiranje po indeksiranim kolonama. Indeksi B-stabla imaju podtipove:

1. Indeksno organizovane tabele (eng. Index-organized tables) – Indeksno organizovana tabela razlikuje se od tabele koja je organizovana kao heap jer su podaci sami po sebi indeks. Tabela koja je organizovana kao heap odnosno gomila smešta podatke, slogove u ne određeni poredak. Ako je na primer jedna kolona u tabeli promenjena, promenio se i njen ROWID.

2. Obrnuti indeksi ključeva (eng. Reverse key indexes) – U ovoj vrsti indeksa bajtovi indeksnog ključa su obrnuti, na primer 103 se čuva kao 301. Preokret bajtova proširuje upis u indeks na više blokova.

3. Opadajući indeksi (eng. Descending indexes) – Ova vrsta indeksa čuva podatke o određenoj koloni ili kolonama u opadajućem redosledu.

4. B-stabla klaster indeksa (eng. B-tree cluster indexes) – Ova vrsta indeksa čuva podatke o određenoj koloni ili kolonama u opadajućem redosledu.

Ukoliko se ne koristi indeksa struktura B-stabla, Oracle baza podatak nudi sledeće tipove indeksa:

1. Bitmap i Bitmap join indekse – Kod bitmap indeksa, unos indeksa koristi bitmapu za usmeravanje na više redova. Suprotno tome, unos indeksa B-stabla pokazuje na jedan red. Indeks bitmap join indeksa je indeks bitmape za svajanje dve ili više tabela.

2. Functional-based indeksi – Ovaj tip indeksa uključuje kolone koje ili transformiše funkcija, kao što je na primer UPPER funkcija ili su uključene u izrra. Indeksi B-stabla ili bitmape mogu biti zasnovani na funkcijama.

3. Indeksi domena aplikacije (eng. Application domain indexes) – Korisnik kreira ovu vrstu indeksa za podatke u domenu specifičnom za aplikaciju. Fizički indeks ne mora koristiti tradicionalnu strukturu indeksa i može se čuvati u Oracle bazi podataka kao tabele ili eksterno kao datoteka.

3.5. Indeksno skeniranje

Indeksno skeniranje je stavljena kao podtema zato što u osnovi indeksi kod Oracle baze podataka su B-tree indeksi, zato je i indeksno skeniranje stavljena kao zasebna tema.

U indeksnom skeniranju baza podataka vraća traženi slog u bazi koristeći indeksirane vrednosti kolona navedene u upitu. Ako baza podataka skenira indeks za neku vrednost, tada će pronaći tu vrednost za $n \cdot I/O$ (gde je I/O vreme potrebno za pristup fizičkoj lokaciji sloga, a n je visina B-tree indeksa). Ovo je osnovni princip indeksiranja kod Oracle baze podataka.

Ako SQL upit pristupa samo indeksiranim kolonama, tada baza podataka čita vrednost direktno iz indeksa a ne iz tabele. Ako upit uz indeksirane kolone pristupa i neindeksiranim kolonama, tada baza podataka koristi rowid za pronalaženje redova u tabeli. Tipično baza prilikom izvršenja upita naizmenično čita podatke iz bloka indeksa a zatim iz bloka tabele.

Postoje sledeći tipovi indeksnog skeniranja:

1. Potpuno indeksno skeniranje - U potpunom skeniranju indeksa, baza podataka čita čitave indekse redom. Potpuno skeniranje indeksa eliminiše operaciju sortiranja, jer podatke prikazuje, orgranizuje po ključu indeksa. Blokove čita pojedinačno. Potpuno skeniranje indeksa može se izvršiti umesto da se izvrši skeniranje kompletne tabele nakon čega sledi sortiranje ako upit ispunjava sledeće zahteve: Sve kolone na koje se upućuje upit, moraju biti u indeksu. Redosled kolona na koje se upućuje upit mora se podudarati sa redosledom vodećih kolona indeksa. Upit može sadržati sve kolone u indeksu ili podskupu kolona u indeksu.

2. Brzo potpuno indeksno skeniranje – ovim skeniranjem baza podataka pristupa podacima u samom indeksu bez pristupa tabeli, a baza podataka čita blokove indeksa bez obređenog redosleda. Brzo potpuno indeksno skeniranje je alternativa potpunom indeksnom skeniranju ako su ispunjeni sledeći uslovi: Indeks mora sadržati sve kolone potrebne za upit i red koji sadrži sve nule ne sme se pojaviti u skupu rezultata upita. Da bi ovaj rezultat bio zagarantovan, najmanje jedna kola u indeksu mora da ima ograničenje NOT NULL, ovim se sprečavanje uzimanje nule u obzir u skupu rezultata upita.

3. Skeniranje opsega indeksa - Skeniranje indeksnog opsega je uobičajena operacija za pristup selektivnim podacima. Može biti ograničen (ograničen sa obe strane) ili neograničen

(sa jedne ili obe strane). Rezultat operacija prikazuje rezultate u rastućem redosledu indeksih kolona (kako su sačuvani indeksi u tabeli indeksa, po ključu indeksne tabele). Više redova sa identičnim vrednostima sortirano je u rastućem redosledu po rowid-u. Ako se podaci moraju sortirati preporučuju se korišćenje ORDER BY klauzule. Ovo skeniranje optimizator koristi kada u where klauzuli imamo ograničenja tipa jednakost, manje, veće, logički operatori, klauzulu LIKE.

4. Jedinstveno indeksno skeniranje - Ovo skeniranje vraća najviše jedan rowid. Oracle izvodi jedinstveno skeniranje ako izraz sadrži UNIQUE ili PRIMARY KEY ograničenje koje garantuje pristup samo jednom redu. Ovakva pristup se koristi kada su sve kolone jedinstvenog indeksa ili indeksa navedenog u where klauzuli koji predstavlja primarni ključ.

5. Preskočno indeksno skeniranje (eng. Index Skip Scan) – ovo skeniranje koristi logičke podindekse složenog indeksa. Baza podataka „preskače“ kroz jedan indeks kroz jedan indeks kao da pretražuje zasebne indekse. Ovakav tip indeksa je koristan ako u vodećoj koloni kompozitnog indeksa ima nekoliko različitih vrednosti, a u navedenom ključu indeksa mnogo različitih vrednosti. Baza podataka može odabrati ovo skeniranje kada vodeća kolona složenog indeksa nije navedena u predikatu upita. Primer ukoliko imamo da je ključ kompozitan, odnosno sastoji se od dve kolone od koje je vodeća kolona prvi element kompozitnog indeksa, dok je druga kolona kompozitnog indeksa neki drugi parametar. Ukoliko izvršavamo SQL upit koji pretražuje indeksnu strukturu po drugoj koloni kompozitnog ključa (ne pretražuje po vodećoj koloni) tada baza podataka koristi ovakav tip indeksiranja. U preskočno indeksnom skeniranju broj logičkih podindeksa se obređuje brojem različitih vrednosti u vodećoj koloni. Ukoliko imamo na primer da vodeća kolona je polje koje je tipa ENUM i sadrži na primer samo 2 vrednosti, tada će baza podataka logično podeliti vrednosti u odnosu na vodeću kolonu, odnosno podeliće indeks tabelu na dva dela i vrši će pretraživanje prvo po prvom delu pa kasnije po drugom delu indeksne tabele. Ovo skeniranje poboljšava skeniranje indeksa po kolonama bez prefiksa. Preskočno indeksno skeniranje omogućava da se složeni indeks logički podeli na manje podindekse. Početna kolona složenog indeksa nije navedena u upitu, drugim rečima preskače se.

3.6. Kompresija indeksa

Često se dešava da u indeks tabeli posedujemo kreirane indekse koje sadrže kolone koje se razlikuju u odnosu na drugi indeks za jednu kolonu. Na primer, ukoliko kreiramo indeks za tabelu korisnike pri čemu indeksiramo kolone aktivni korisnik (da li korisnik koristi sistem ili je ukinuo registraciju), tip korisnika (koje se odnosi na to kakav je tip korisnika sistem, može biti 3 tipa, administrator, partner ili običan korisnik sistema) i kolonu email. U ovakvo definisanom ključu, odnosno pravilnije je reći da u sistemu većina korisnika je po tipu običan korisnik sistema (user) i kolona aktivni korisnik je uobičajeno aktivan (ENUM polje koje ima vrednost ukoliko je korisnik aktivan tada ima vrednost y ukoliko nije ima vrednost n) tada u ovom kompozitnom ključu imamo vrednost kolone email koji je promenljiv. Zapis na primer ovakve indeksne strukture, preciznije je reći leaf blok B-tree načina indeksiranja, je na primer ovakav

0:Y,USER,marko.djordjevic@elfak.ni.ac.rs, AAAPvCAAFAAAAFaAAa
 1:Y,USER,marko.djordjevic@elfak.ni.ac.rs, AAAPvCAAFAAAAFaAAD
 2:Y,USER,korisnik@nova.rs,AAAPvCAAFAAAAFaAAg
 3:Y,USER,korisnik1@nova.rs,AAAPvCAAFAAAAFaAAI
 4:Y,USER,korisnik3@nova.rs,AAAPvCAAFAAAAFaAAm
 5:Y,ADMIN,admin@novaAdmin.rs,AAAPvCAAFAAAAFaABq

Vidimo da u leaf bloku, gde se čuvaju informacije o indeksima, posedujemo za korisnika koji ima istu email adresu poseduje drugačiji rowid. Pored ovog zapažanja vidimo da u ovom primeru imamo skoro sve korisnike koje imaju isti aktivni status i tip korisnika. Oracle baza podataka ovakav problem gomilanja istih podataka rešava korišćenjem kompresiju indeksnog ključa poznatu kao prefiks kompresija. Ovakav tip kompresije omogućava nam kompresovanje delova vrednosti ključeva u indeks segmentu tako što smanjuje neefikasno skladištenje pri čuvanju vrednosti koje se ponavljaju. Kompresija indeksnog ključa kompresuje podatke tako što deli indeksni ključ na dva dela: vodeću grupu kolona, koja se naziva unos prefiksa i potencijalno se dele na više vrednosti ključa i sufiks kolone koja je jedinstvena za svaki indeksni ključ. Kako se prefiksi potencijalno dele na više ključeva u bloku oni se mogu optimalnije čuvati i deliti na više sufiksa što rezultuje kompresiju podataka indeksa. Ovim delimo leaf blok na dva dela u kome ćemo u prvom delu imati prefiks tabelu, a u drugom delu imati sufikse, odnosno vrednosti prefiksa. Sada nakon izvršene ove kompresije grupisace se rezultati na ovakav način :

Prefix tabela:

0: Y,USER,marko.djordjevic@elfak.ni.ac.rs
 1: Y,USER,korisnik@nova.rs
 2: Y,USER,korisnik1@nova.rs
 3: Y,USER,korisnik3@nova.rs
 4: Y,ADMIN,admin@novaAdmin.rs

A sufiks tabela:

0: 0,AAAPvCAAFAAAAFaAAa
 1: 0, AAAPvCAAFAAAAFaAAD
 2: 1, AAAPvCAAFAAAAFaAAg
 3: 2, AAAPvCAAFAAAAFaAAI
 4: 3, AAAPvCAAFAAAAFaAAm
 5: 4, AAAPvCAAFAAAAFaABq

Kompresija indeksnog ključa vrši se u leaf blokovima B-tree indeksa. Ključevi se kompresuju lokalno u leaf bloku, odnosno unosi prefiksa i sufiksa čuvaju se u istom bloku. Unosi sufiksa čine kompresovani prikazi indeksnog ključa. Svaki od ovih kompresovanih redova odnosi se

na odgovarajući prefiks koji se čuva u istom bloku. Lokalnim skladištenjem prefiksa i sufiksa u istom bloku, svaki indeksni blok je samostalan i da bi se konstruisao kompletan ključ nije uključen dodatni IO blok. Ovo je vrlo jeftina operacija samo sa memorijom.

U Oracle bazi podataka moguće je i specificirati koliko kolona želimo da kompresujemo ili ako želimo da kompresujemo podatke po nekoj ključnoj reči, to postizemo korišćenjem naredbe COMPRESS <broj_kolona> ||<ključna reč>.

Ukoliko definišemo jedan tip prefiksa i u drugom trenutku želimo da unesemo drugi tip prefiksa to nije moguće uraditi sve dok se svi sufixi koji se odnose na već definisani prefiks ne obrišu.

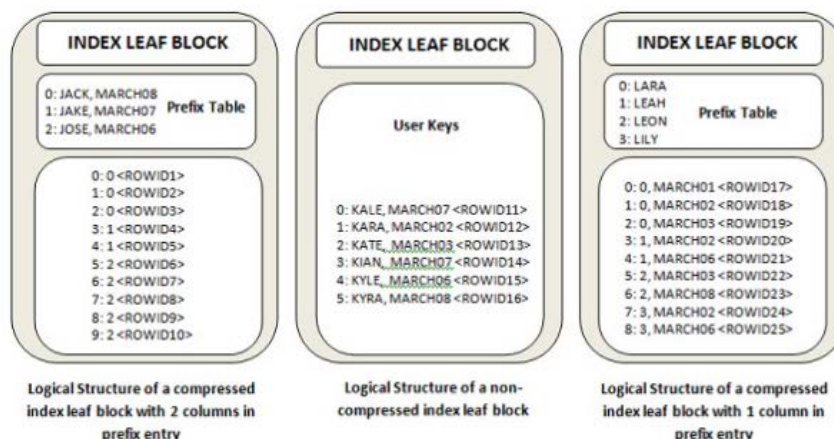
Treba imati na umu, da iako kompresija ključa smanjuje zahteve za skladištenjem indeksa deljenjem delova ključeva u više poddelova, postoji potrošnja procesora pri rekonstrukciji ključeva tokom pretraživanja ili skeniranja indeksa. Prednost jeste optimalniji prikaz indeksa zatim ušteda prostora, povećanje fiksnošći keš memorije, manje leaf blokova i manja dubina stabla što rezultira manje logičkih IO pristupa. Pored ovih prednosti ovakva kompresija je pogodna kod nejedinstvenog indeksa gde je dodat rowid da bi ključ bio jedinstven (u tom slučaju primarna tabela imala bi vrednosti kolona do rowid-a, a sufix tabela imala bi samo identifikator prefiksa i rowid).

Ovakva kompresija je vrlo korisna kada prefiks kolone indeksa se ponavlja više puta u bloku lista. Međutim ako su vodeće kolone vrlo selektivne ovo i nema nekih prednosti, imamo duplo više podataka. Da bi kompresija indeksa bila korisna, trebali bismo osigurati da kolone niske kardinalnosti (kardinalnost se definiše kao broj različitih vrednosti u koloni) budu vodeće kolone u spojenom indeksu. U suprotnom dobijamo negativnu stranu kompresije. Takođe nema smisla kompresovati jedinstveni indeks više kolona. Ključ za dobru kompresiju indeksa je utvrđivanje koji će indeksi imati koristi od toga i kako odrediti <prefix_col_length> tačno.

Kompresija indeksnog ključa (poznata i kao kompresija prefiksa) uklanja duplikate unapred definisanog broja kolona indeksnog prefiksa na nivou leaf bloka i efikasan je način trajnog smanjenja veličine indeksa, kako na disku, tako i u kešu.

Broj prefiks kolona koje treba uzeti u obzir za kompresiju određuje Oracle database administrator u vremenu stvaranja indeksa i konstantan je za sve leaf blokove. Kompresija može biti vrlo korisna kada prefiks kolone indeksa imaju mnogo ponovljenih redova unutar bloka listova. Međutim, kada su vodeće kolone vrlo selektivne ili ako nema mnogo ponovljenih redova unutar leaf blokova. Ovaj pristup za kompresiju indeksnog ključa ima loše strane. Zahtev od DBA je da dobro razume podatke kako bi izabrao najoptimalniji broj kolona prefiksa. Navedeni broj kolona nije možda najoptimalniji za stvaranje najboljeg stepena kompresije za svaki blok u indeksu, i tako dalje.

Oracle u verziji 12.1.0.2 uvodi naprednu kompresiju indeksa koja ima za cilj automatizaciju kompresije indeksa tako da DBA više nije potreban da navede broj kolona prefiksa koje treba uzeti u obzir za kompresiju. Tačan i najoptimalniji broj prefiksa kolona izračunavaće se automatski po blokovima i na taj način se postiže najbolji mogući odnos kompresije. Sada je moguće da različiti leaf blokovi budu različito kompresovani (sa različitim brojem kolona prefiksa) ili da uopšte ne budu kompresovani ako nema prefiksa koji se ponavljaju. Sledeći primer, prikazan na slici 9, preuzet je sa interneta i opisuje primer napredne kompresije indeksa.



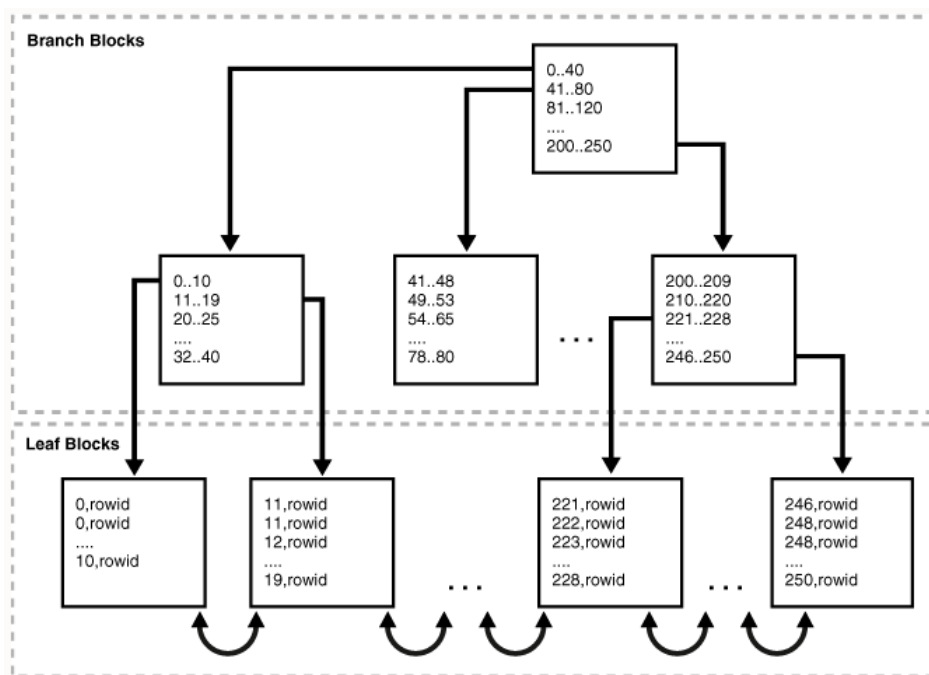
Slika9: Izgled leaf blokova nakon izvršene napredne kompresije

Sa slike 9 se može videti da sada u različitim leaf blokovima imamo različitu kompresiju. Napredna kompresija indeksa se može lako omogućiti tako što ćemo specicificirati opciju COMPRESS za indekse. Novi indeksi se mogu automatski kreirati kao kompresovani ili se postojeći indeksi mogu ponovo stvoriti kompresovani. Komanda koja se ovde koristi jeste CREATE INDEX idxname ON tabname(col1,col2,...coln) COMPRESS ADVANCE LOW/HIGH. Primeti da za razliku od prethodnog primera ovaj način kompresije ne zahteva od korisnika da unese broj kolona za kompresiju, odnosno broj kolona koji će se koristiti za prefiks tabelu, već je samo potrebno navesti ADVANCE i nakon toga se navodi ključna reč HIGH/LOW. U verziji 12c moguće je odraditi samo LOW kompresiju, ali je sada dostupna HIGH kompresija po default-u. Ovim specificiramo jačinu kompresije, ukoliko je HIGH onda metod daje veće koeficijente kompresije u većini slučajeva, istovremeno poboljšavajući performanse za upite koji pristupaju indeksu, takođe pokreću se složeni algoritmi kompresije i smešta podatke u jednici za kompresiju koja je poseban format na disku. Kada specificiramo LOW kompresiju baza podataka možda neće kompresovati sve blokove pa moramo koristiti statistiku koja nam govori koliko blokova je ostalo nekompresovano.

Takođe dobro je napomenuti da napredna tehnika kompresije indeksa dobro funkcioniše na svim podržanim indeksima, uključujući one koji nisu bili dobri kandidati za kompresiju prefiksa.

3.7. B-tree indeksi

B-tree indeks je podrazumevani tip indeksa u Oracle-u. Ovaj tip indeksa je poznat kao B-tree jer se identifikatori reda, ključevi, tabele i pridružene vrednosti kolone čuvaju u indeksnim blokovima u uravnoteženoj strukturi nalik stablu. Primer strukture B-tree indeksa prikazan je na slici 10.



Slika 10: Interna struktura B-tree indeksa

B-tree indeksi poseduju dve vrste blokova: branch blok za pretragu i leaf blok koji se koristi za čuvanje ključnih vrednosti. Branch blokovi b-tree indeksa sadrže podatke indeksa koji upućuju na blokove indeksa nižeg nivoa.

Na slici 10, koja je data kao primer B-tree indeksa, root blok ima unos 0-40 što pokazuje na krajnji levi branch blok u nivou ispod. Ovaj nivo sadrži unose kao što su 0-10, 11-19, ... Svaki ovaj unos ukazuje na leaf blok koji sadrži ključne vrednosti koje spadaju u opsegu definisanom u branch bloku.

Ovaj tip indeksa je uravnotežen jer svi leaf blokovi automatski ostaju na istoj dubini. Dakle, pronalaženje bilo kog zapisa sa bilo kog mesta u indeksu traje približno isto toliko vremena. Visina indeksa je broj blokova potrebnih za prelazak iz root bloka do leaf bloka, dok je visina grane jednaka visini indeksa minus 1.

Branch blokovi čuvaju minimalni prefiks ključa potreba za donošenje odluke o grananju između dva ključa. Ova tehnika omogućava bazi podataka da sačuva što više podataka na svaki branch blok. Branch blok sadrži pokazivač na leaf blok koji poseduje taj ključ. Broj ključeva i pokazivača je ograničen veličinom bloka. Leaf blokovi sadrže svaku indeksiranu vrednost podatka i odgovarajući rowid koji se koristi za lociranje stvarnog podatka u bazi. Svaki unos je sortiran po ključu i rowid-u. Unutar leaf bloka ključ i rowid su povezani sa svojim levim i desnim susednim poljima. Sami leaf blokovi su takođe dvostruko povezani.

Ako je vrednost kolone tabele (ili kombinacija kolona) prilično jedinstvena u svim redovima, onda stvaranje ovakvog tipa indeksa obično rezultira bržim performansama upita. Dodatna poboljšanja performansi se ostvaruje kada sama struktura indeksa sadrži potrebne vrednosti kolone tabele koji zadat upitom. U ovoj situaciji blokovima podataka tabele nije potrebno pristupiti, već baza podataka pristupa indeksnoj tabeli i kao rezultat upita korisniku prikazuje vrednosti kolona.

Indeksi B-tree takođe igraju ključnu ulogu u dizajniranju aplikacije, jer su ovi indeksi usko povezani sa određenim ograničenjima. Oracle koristi ovaj tip indeksa za primenu ograničenja primarnog ključa i jedinstvenog ključa. U većini slučajeva B-tree indeksi se kreiraju automatski prilikom izbora ograničenja primarnog ključa. Takođe indeksi se često ručno kreiraju kako bi se vrednosti podudarale sa vrednostima kolona ograničenja stranog ključa radi poboljšanja performansi upita koji spajaju tabele na kolonama primarnog i stranog ključa (naredbe join).

3.8. Bitmap indeksi

Kod bitmap indeksa, baza podataka čuva bitmapu za svaki indeksni ključ, odnosno niz bitova. U konvencionalnom B-tree indeksu, jedan unos indeksa pokazuje na jedan red. U bitmap indeksu, svaki ključ čuva pokazivače na više redova. Bitmap indeksi su prvenstveno dizajnirani za skladištenje podataka ili okruženja u kojima se upiti upućuju na mnoge kolone na ad hoc način. Situacije u kojima bitmap indeksi pokazuju prednosti su sledeće:

1. Indeksirane kolone imaju malu kardinalnost, odnosno broj različitih vrednosti je mali u poređenju sa brojem redova tabele.

2. Indeksirana tabela je ili samo za čitanje ili ne podleže značajnim modifikacijama pomoću DML upita.

Svaki bit u bitmapi odgovara mogućem rowid-u. Ako je bit postavljen, red sa odgovarajućim rowid-om sadrži vrednost ključa. Funkcija mapiranja pretvara položaj bita u stvarni rowid, tako da indeks bitmape pruža istu funkciju kao indeks B-stabla, iako koristi drugačiju internu strukturu. Ako se indeksirana kolona u jednom redu ažurira, tada baza podataka zaključava unos indeksnog ključa a ne pojedinačni bit preslikan u ažurirani red. Budući da ključ pokazuje na mnoge redove, DML na indeksiranim podacima obično zaključava sve ove redove. Iz tog razloga, indeksi bitmape nisu pogodni za OLTP aplikacije, odnosno aplikacije tipa online bankarstva, online kupovina,...

3.8.1. Pristupna putanja Bitmap indexa

U konvencionalnom B-tree indeksu, jedan unos indeksa pokazuje na jedan red. Kod Bitmap indeksa ključ je kombinacija indeksiranih podataka i rowid opsega. Baza podataka čuva najmanje jednu bitnu vrednost za svaki indeksni ključ. Svaka vrednost u bitmapi, koja je niz vrednosti od 1 i 0 ukazuje na red unutar tabele (gde 1 označava red sadrži taj indeks, a 0 označava red koji ne sadrži taj indeks) sa rowid opsegom. Dakle, u indeksu bitmape, jedan unos indeksa ukazuje na skup redova a ne na jedan red.

3.8.2. Razlika između Bitmap i B-tree indeksa na nivou pristupa i pamćenja informacija

Bitmap indeks koristi drugačiji ključ od indeksa B-tree, ali je sladišten na isti način kao struktura B-tree indeksa. Kod jedinstvenog B-tree indeksa, ključ referencira samo na indeksirane podatka, kod ne jedinstvenog B-tree indeksa, ključ predstavlja kombinaciju indeksirane kolone sa rowid-om. Bitmap indeksima ključ ustvari predstavlja kombinaciju indeksa sa opsegom rowid-a.

Razlog pamćenja bitmap indeksa jeste taj što je pretraživanje jako brzo i na osnovu dela ključa možemo doći do informacije, nije potreban ceo ključ već deo ključa koji će pokazati na vrednost koju referencira indeks.

Bitmap indeksi su obično pogodni za retko modifikovane podatke sa malim ili srednjim brojem različitih vrednosti. Indeksi B-tree, su generalno pogodni za kolone sa visokim stepenom promene, modifikacijom, podataka i sa čestim upita. Na primer, optimizator može odabrati indeks B-tree za upit kolone koje imaju jako malo različitih vrednosti i retko se ažuriraju, kao što je na primer pol, dok bitmap indeksi ovde su pogodniji u odnosu na B-tree indeksi.

Bitmap indeksi su korisni u upitima koji sadrže sledeće karakteristike:

1. Više uslova u WHERE klauzulu upita – Pri pristupanju samoj tabeli, baza podataka filtrira redove koji zadovolja zadati upi ali ne sve redove.

2. AND, OR ili NOT operacije nad kolonama sa malim i srednjim stepenom promene. Kombinaovanje bitmapnih indeksa čini ove operacije efikasnijim.

3. COUNT operacija – baza podataka može pretražiti bitmap indekse bez da prolazi kroz celu tabelu.

4. Predikti koji biraju null vrednost – Za razliku od indeksa B-tree, indeksi bitmapa mogu sadržati nule. Upiti koji broje nule u koloni mogu da koriste indekse bitmape bez skeniranja tabele.

5. Kolone tabele nad kojima se ne izvršava kompleksna DML naredba – Razlog je taj što jedan indeksni ključ pokazuje na više redova. Ako sesija modifikuje indeksne podatke, tada baza podataka ne može zaključati ceo unos indeksa, što u praksi zaključava redove u bitmap indeksima. Pa tako ove vrednosti kolona ne mogu da se izvrše.

Kao što smo gore i napisali, za određenu vrednost u bitmapi vrednost 1 ukazuje na to da se vrednost redova podudara sa uslovima bitmape i 0 ako se ne podudaraju, na osnovu ovih vrednosti, baza podataka koristi interni algoritam za mapiranje bitmape na rowid.

Vrednost 1 ili 0 u bitmapi pomera opseg rowid-a i preslikava u potencijalni red u tabeli, čak i ako red ne postoji. Budući da je broj mogućih redova u bloku unapred određen, baza podataka može da koristi krajnje tačke opsega za određivanje rowid-a proizvoljnog reda u opsegu.

Bitmap indekse moguće je i kreirati kao spajanje (JOIN) dve ili više tabele, primer naredbe:

```
CREATE BITMAP INDEX ime_indeksa ON ime_tabele(kolone)  
  
FROM tabela1, tabela2, ..., tabelaN  
  
WHERE uslov;
```


3.9. Indeksi bazirani na funkcijama

Ovaj tip indeksa je zasnovan na izračunavanju vrednosti funkcije ili izraza koji uključuje jednu ili više kolona i čuva ga u indeksu. Indeks ovakvog tipa može biti B-stablo ili indeks bitmape. Indeksirana funkcija može biti aritmetički izraz ili izraz koji sadrži SQL funkciju, korisnički definisanu PL/SQL funkciju.

Ovakav tip indeksa efikasni su za procenu izraza koji sadrže funkcije u svojim WHERE klauzulama. Baza podataka koristi indeks zasnovan na funkciji samo kada je funkcija ista kao funkcija koja je služila za indeksiranje. Međutim kada baza podataka obrađuje izraze INSERT i UPDATE, ona i dalje mora proceniti funkciju za obradu izraza.

Kada se u WHERE klauzuli upita nalazi izraz koji je jednak funkciji indeksiranja, optimizator koristi skeniranje opsega indeksa na indeksima baziranim na funkcijama. Ovakvo skeniranje je posebno korisno kada je predikat visoko selektivan, odnosno kada bira relativno malo redova. Optimizator da bi prepoznao da li je indeks baziran na funkcijama on vrši podudaranje izraza raščlanjavanjem izraza u SQL upitu, a zatim upoređivanjem stabla izraza i indeksa zasnovanog na funkciji i ako se poklope pročitace ovaj tip indeksa. Ovakvo poređenje ne razlikuje velika i mala slova i zanemaruje prazne prostore.

Kao primer ovakvog tipa indeksa, daćemo primer računanje marže, odnosno dobiti nad tabelom PROMET_ULAZ, koja poseduje realne podatke. Marža se računa kao cena minus nabavna cena sa pdv-om proizvoda. Izgled tabele dat je na slici 11.

MASTER	BRNAL	VD	TD	DATNAL	OJ	SIFRA	PG	GRUPA	ULAZ	IZLAZ	DUGUJE	POTRAZUJE	CENA	RABAT_P	RABAT_I	CENA_NAB_BPDV	VRED_NAB_BPDV	RAZLIKA
3470965	(null)	(null)	1 (null)	11	1575	4	5	2	0	100	0	50	0	0	41.82	83.64	7.27	
3470965	(null)	(null)	1 (null)	11	880	4	5	27	0	675	0	25	0	0	20.91	564.57	49.07	
3470965	(null)	(null)	1 (null)	11	909	4	5	4	0	180	0	45	0	0	37.64	150.56	13.08	
3470965	(null)	(null)	1 (null)	11	3073	4	5	6	0	300	0	50	0	0	41.82	250.92	21.81	
3470965	(null)	(null)	1 (null)	11	2702	4	5	13	0	377	0	29	0	0	24.25	315.25	27.48	
3470965	(null)	(null)	1 (null)	11	1185	4	5	5	0	225	0	45	0	0	37.64	188.2	16.35	
3470965	(null)	(null)	1 (null)	11	460	4	5	11	0	330	0	30	0	0	25.09	275.99	24.01	
3470965	(null)	(null)	1 (null)	11	554	4	5	2	0	100	0	50	0	0	41.82	83.64	7.27	
3470965	(null)	(null)	1 (null)	11	2466	4	5	12	0	360	0	30	0	0	25.09	301.08	26.19	
3470965	(null)	(null)	1 (null)	11	1483	4	5	4	0	160	0	40	0	0	33.45	133.8	11.65	
3470965	(null)	(null)	1 (null)	11	662	4	5	11	0	550	0	50	0	0	39.09	429.99	70.01	
3470965	(null)	(null)	1 (null)	11	2215	4	5	2	0	598	0	299	0	0	244.64	489.28	54.36	
3470965	(null)	(null)	1 (null)	11	1607	4	5	2	0	400	0	200	0	0	156.36	312.72	50.92	
3470965	(null)	(null)	1 (null)	11	1603	4	5	12	0	960	0	80	0	0	62.55	750.6	122.13	
3470966	(null)	(null)	1 (null)	11	460	4	5	-7	0	-210	0	30	0	0	25.09	-175.63	-15.28	
3470939	(null)	(null)	1 (null)	13	460	4	5	16	0	480	0	30	0	0	25.09	401.44	34.92	
3499494	(null)	(null)	1 (null)	11	2013	3	0	5650	0	5650	0	1	0	0	0.804	4542.6	165.73	
3499495	(null)	(null)	1 (null)	13	2013	3	0	9250	0	9250	0	1	0	0	0.804	7437	271.33	
3499544	(null)	(null)	1 (null)	16	2013	3	0	10000	0	10000	0	1	0	0	0.804	8040	293.33	
3499546	(null)	(null)	1 (null)	11	460	4	5	-8	0	-240	0	30	0	0	25.09	-200.72	-17.46	
3499546	(null)	(null)	1 (null)	11	909	4	5	-6	0	-270	0	45	0	0	37.64	-225.84	-19.61	
3499546	(null)	(null)	1 (null)	11	3073	4	5	-1	0	-50	0	50	0	0	41.82	-41.82	-3.63	
3499546	(null)	(null)	1 (null)	11	554	4	5	-3	0	-150	0	50	0	0	41.82	-125.46	-10.9	
3499546	(null)	(null)	1 (null)	11	2702	4	5	-5	0	-145	0	29	0	0	24.25	-121.25	-10.57	
3499546	(null)	(null)	1 (null)	11	1185	4	5	-3	0	-135	0	45	0	0	37.64	-112.92	-9.81	
3499546	(null)	(null)	1 (null)	11	2466	4	5	-5	0	-150	0	30	0	0	25.09	-125.45	-10.91	
3499546	(null)	(null)	1 (null)	11	1483	4	5	-3	0	-120	0	40	0	0	33.45	-100.35	-8.74	
3499546	(null)	(null)	1 (null)	11	766	4	5	-5	0	-245	0	49	0	0	38.31	-191.55	-31.18	
3499546	(null)	(null)	1 (null)	11	836	4	5	-2	0	-240	0	120	0	0	93.82	-187.64	-30.54	
3499546	(null)	(null)	1 (null)	11	880	4	5	-2	0	-50	0	25	0	0	20.91	-41.82	-3.63	
3499546	(null)	(null)	1 (null)	11	908	4	5	-1	0	-150	0	150	0	0	117.27	-117.27	-19.09	
3477477	(null)	(null)	1 (null)	1	4113	4	6	10	0	400	0	40	0	0	33.6	336	27.64	
3470496	(null)	(null)	1 (null)	1	5058	4	6	3	0	180	0	60	11	18	54.55	145.65	17.99	

Slika 11: Izgled podataka u tabeli PROMET_ULAZ

Pored ovih podataka postoji još mnogo podataka koje se pamte u tabeli PROMET_ULAZ i da bi smo videli snagu ovakvih tipova indeksa, odradićemo sql upit za korisnika čija je vrednost kolone MASTER 3470965. SQL upit ćemo izvršiti prvo bez postojanja indeksa, zatim sa definisanim indeksom. SQL naredba koja prikazuje rezultat je sledeća: (*select CENA - cena_NAB_BPDV AS DOBIT FROM "marko".PROMET_ULAZ where MASTER=3470965;*)

Naredba za kreiranje ovakvog tipa indeksa je sledeća (*CREATE INDEX DOBIT_KORISNIKA ON "marko".PROMET_ULAZ (CENA - cena_NAB_BPDV);*).

Rezultat istraživanja dat je u tabeli 1.

Računanje marže	PROMET_ULAZ
Bez indeksiranja	<pre> PLAN_TABLE_OUTPUT Plan hash value: 881843474 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 8 120 137 (0) 00:00:01 * 1 TABLE ACCESS FULL PROMET_ULAZ 8 120 137 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - filter("MASTER"=3470965) </pre>
Sa indeksiranjem	<pre> PLAN_TABLE_OUTPUT Plan hash value: 1012480670 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 8 152 3 (0) 00:00:01 1 TABLE ACCESS BY INDEX ROWID BATCHED PROMET_ULAZ 8 152 3 (0) 00:00:01 * 2 INDEX RANGE SCAN INDEX7 8 1 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 2 - access("MASTER"=3470965) </pre>

Tabela 1:Rezultat indeksa bazirani na funkcijama

Iz priloženog možemo zaključiti da je ovako kreiran indeks jako pogodan za računanje ovakvog tipa zahteva, naravno ovo je osmišljen primer da pokažemo prednost ovakvih tipova indeksa. Marža se računa skoro svakog meseca, kod nekih poslodavaca i svaki dan, tako da ovakav indeks će poslužiti baš tim sistemima da im ubrza proces dobijanja informacija o dobiti.

U ovom primeru pored ovakvog indeksa korišćen je i B-tree indeks, kojim se indeksira MASTER kolona. Izvršićemo izmenu, tako što ćemo umesto B-tree indeksa koristiti bitmap indekse. Rezultat ovakvog korišćenog indeksa dat je na slici 12.

PLAN_TABLE_OUTPUT							
Plan hash value: 1445531846							
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
0	SELECT STATEMENT		8	152	3 (0)	00:00:01	
1	TABLE ACCESS BY INDEX ROWID BATCHED	PROMET_ULAZ	8	152	3 (0)	00:00:01	
2	BITMAP CONVERSION TO ROWIDS						
* 3	BITMAP INDEX SINGLE VALUE	INDEX7					
Predicate Information (identified by operation id):							
3 - access("MASTER"=3470965)							

Slika 12: Rezultat upita izvršen pomoću bitmap indeksa

Kao što se da primetiti bitmap indeksi pristupaju samo 16 redova tabele, i samim tim u poređenju sa b-tree indeksima vidimo da u ovoj situaciji bitmap indeksi pružaju ubrzanje sistema, odnosno postiže se ubrzanje sistema. Razlog toga jeste taj zato što bitmapa čuva informacije o redovima koji poseduju taj indeks, i samim pretvaranjem u stvarne lokacije dolazi se brže do informacija (razlika je 1% korišćenje procesorske moći). Ovaj primer pokazuje pravu razliku između bitmap indeksa i b-tree indeksa.

3.10. Tabele organizovane po indeksu

Na fakultetu i tokom studiranja kao dobru praksu stekli smo rutinu da svaka tabela koju kreiramo poseduje primarni indeks. U ovom delu daćemo prednost ovako kreiranih tabela i zašto je to značajno.

Tabela organizovana po indeksu je tabela koja se čuva kao varijacija strukture indeksa B-tree. Suprotno tome, tabela organizovana kao heap pamti redove tamo gde najviše odgovaraju. Na slici 13 je prikazan primer jedne heap tabele.

ID	ACTIVE	WEB...	ACTIVE_DOC	BARCODE	CODE	INPUTPRICE	INPUTPRE...	MANUF...	MANUFNA...	NAME	ALTERNAME	PRICE	PRICECU...
9	189 y	y	n	(null)	1252529929	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
10	190 y	y	n	(null)	1252529930	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
11	191 y	y	n	(null)	1252529931	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
12	192 y	y	n	(null)	1252529932	0 (null)	(null)	(null)	Makita	Makita ...	(null)	0 (null)	
13	193 y	y	n	(null)	1252529933	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
14	194 y	y	n	(null)	1252529934	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
15	195 y	y	n	(null)	1252529935	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
16	196 y	y	n	(null)	1252529936	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
17	197 y	y	n	(null)	1252529937	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
18	198 y	y	n	(null)	1252529938	0 (null)	(null)	(null)	Akumula...	Akumula...	(null)	0 (null)	
19	199 y	y	n	(null)	1252529939	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
20	200 y	y	n	(null)	1252529940	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
21	201 y	y	n	(null)	1252529941	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
22	202 y	y	n	(null)	1252529942	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
23	203 y	y	n	(null)	1252529943	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
24	204 y	y	n	(null)	1252529944	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
25	205 y	y	n	(null)	1252529945	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
26	206 y	y	n	(null)	1252529946	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
27	207 y	y	n	(null)	1252529947	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
28	208 y	y	n	(null)	1252529948	0 (null)	(null)	(null)	GARDEN...	Elektri...	(null)	0 (null)	
29	209 y	y	n	(null)	1252529950	0 (null)	(null)	(null)	GARDEN...	Pneumat...	(null)	0 (null)	
30	210 y	y	n	(null)	1252529951	0 (null)	(null)	(null)	GARDEN...	GT450 E...	(null)	0 (null)	
31	211 y	y	n	(null)	1252529952	0 (null)	(null)	(null)	GARDEN...	Akumula...	(null)	0 (null)	
32	212 y	y	n	(null)	1252529953	0 (null)	(null)	(null)	GARDEN...	SCD-Fix...	(null)	0 (null)	
33	213 y	y	n	(null)	1252529954	0 (null)	(null)	(null)	GARDEN...	Akumula...	(null)	0 (null)	
34	214 y	y	n	(null)	4324	0 (null)	(null)	4324	Makita	Thodna ...	(null)	0 (null)	
35	215 y	y	n	(null)	1252529955	0 (null)	(null)	(null)	Makita	Akumula...	(null)	0 (null)	
36	1 n	y	n	(null)	MPC610	0 (null)	(null)	(null)	Makita	MPC610 ...	(null)	0 (null)	
37	2 n	y	n	(null)	AC640	0 (null)	(null)	(null)	Makita	AC640 M...	(null)	0 (null)	
38	3 n	y	n	(null)	AC1350	0 (null)	(null)	(null)	Makita	AC1350...	(null)	0 (null)	
39	4 n	y	n	(null)	AC1300	0 (null)	(null)	(null)	Makita	AC1300 ...	(null)	0 (null)	
40	2 n	y	n	(null)	AC640	0 (null)	(null)	(null)	Makita	AC640 M...	(null)	0 (null)	
41	3 n	y	n	(null)	AC1350	0 (null)	(null)	(null)	Makita	AC1350...	(null)	0 (null)	
42	4 n	y	n	(null)	AC1300	0 (null)	(null)	(null)	Makita	AC1300 ...	(null)	0 (null)	
43	5 y	y	n	(null)	435	0 (null)	(null)	(null)	MEGA	Kompree...	(null)	0 (null)	

Slika 13: Primer heap organizovane tabele

U tabeli organizovanoj po indeksu, redovi se čuvaju u indeksu definisanom na primarnom ključu za tabelu. Svaki unos indeksa u B-tree takođe čuva neključne vrednosti kolona (sve kolone osim kolone označene kao primarni indeks). Dakle indeks su podaci, a podaci su indeks. Aplikacija manipuliše tabelama organizovanim indeksom baš kao i tabelama organizovani u heap strukturu koristeći SQL upite.

Tabele organizovane po indeksu pružaju brži pristup redovima tabela pomoću primarnog ključa ili važećeg prefiksa ključa. Prisustvo neključnih kolona redova u bloku lista izbegava dodatni pristup podacima.

Ovakve tabele korisne su kada se povezani delovi podataka moraju čuvati zajedno ili se podaci moraju fizički čuvati u određenom redosledu. Tipična upotreba ove vrste tabela je za pronalaženje informacija, prostorne podatke.

Poređenje heap organizovanih tabela i tabela organizovanih pomoću indeksa dat je u tabeli 2 .

Heap-organizovana tabela	Indeks-organizovana tabela
Rowid jedinstveno definiše red. Ograničenje primarnog ključa može biti definisano opciono.	Primarni ključ jedinstveno identifikuje red. Mora se definisati ograničenje primarnog ključa.
Fizički rowid u ROWID pseudo koloni dozvoljava bildovanje sekundarnih indeksa.	Logički rowid u pseudokoloni ROWID omogućava izgradnju sekundarnih indeksa.
Pojedinačnim redovima se može pristupiti direktno od strane rowid-a.	Pristup pojedinačnim redovima može se postići indirektno pomoću primarnog ključa.
Sekvencijalni full table scan vraća sve redove u nekom redosledu.	Skeniranje punog indeksa ili brzo skeniranje punog indeksa vraća sve redove u nekom redosledu.
Može biti smešten u table cluster-u sa drugim tabelama.	Nije moguće sačuvati u klaster tabeli.
Može sadržati kolonu sa LONG tipom podataka i kolone sa LOB tipom podataka.	Može sadržati LOB kolone, ali ne i LONG kolone.
Može sadržati virtuelne kolone (podržane su samo relacije heap tabele).	Ne može da sadrži virtuelne kolone.

Tabela2: Poređenje heap organizovanih tabela i Indeks organizovanih tabela

4. Primeri

4.1. Primer B-tree indeksa

U ovom seminarskom radu, za test tabelu, napravili smo tabele TEST1,TEST2,TEST3. Struktura test tabela je isti, s tom razlikom što tabela TEST1 ne poseduje nikakve indekse, organizovana je kao heap, tabela TEST2 indeksirane su samo neke kolone, a tabela TEST3 je potpuno indeksirana (za svaku kolonu postoji definisan indeks). Struktura tabela prikazana je na slici 14.

marko.TEST1	
ID	NUMBER
NAME	VARCHAR2 (4000 BYTE)
BROJ	NUMBER
DATUM	DATE
KARAKTER	VARCHAR2 (1 BYTE)
DECIMALNOP	NUMBER
DOUBLEP	NUMBER
DECIMALNOP1	NUMBER
TS	TIMESTAMP WITH TIME ZONE

Slika 14: Struktura TEST tabele

Podaci unutar tabela kreirani su korišćenjem sledeće naredbe

```
create table "marko".test3 as
select rownum as id,
DBMS_RANDOM.string ('A',20) as name,
floor(DBMS_RANDOM.value (10,100)) as BROJ,
TO_DATE( trunc ( DBMS_RANDOM.VALUE (TO_CHAR(DATE '1900-01-01','J'),
TO_CHAR(DATE '2021-12-31','J'))),'J') as DATUM,
decode(round(dbms_random.value), 1, 'F', 'M') as KARAKTER,
DBMS_RANDOM.value (10,5) as DECIMALNOP,
trunc(dbms_random.value(0.01, 9999.99),2) as DOUBLEP,
DBMS_RANDOM.value (10,5) as DECIMALNOP1,
CURRENT_TIMESTAMP as ts
from dual connect by level <=15000
```

Za tabelu TEST2, koristili smo inicijalni, defaultni tip indeksa kod Oracle baze podataka a to je B-tree indeks. Indeksirali smo kolone: id, name, karakter:

```
CREATE INDEX "marko".INDEX2 ON "marko".TEST2 (ID ASC);
```

```
CREATE INDEX "marko".INDEXNAME ON "marko".TEST2 (name ASC);
```

```
CREATE INDEX "marko".INDEXCHAR ON "marko".TEST2 (KARAKTER ASC)
```

Za tabelu TEST3, koristili smo, takođe defaultni tip indeksa B-tree.

```
CREATE INDEX "marko".INDEX3 ON "marko".TEST3 (ID ASC);  
CREATE INDEX "marko".INDEX4 ON "marko".TEST3 (NAME ASC);  
CREATE INDEX "marko".INDEX5 ON "marko".TEST3 (BROJ);  
CREATE INDEX "marko".INDEX6 ON "marko".TEST3 (DATUM ASC);  
CREATE INDEX "marko".INDEX7 ON "marko".TEST3 (KARAKTER ASC);  
CREATE INDEX "marko".INDEX8 ON "marko".TEST3 (DECIMALNOP ASC);  
CREATE INDEX "marko".INDEX9 ON "marko".TEST3 (DOUBLEP ASC);  
CREATE INDEX "marko".INDEX10 ON "marko".TEST3 (DECIMALNOP1);  
CREATE INDEX "marko".INDEX11 ON "marko".TEST3 (TS ASC);
```

Nad ovako kreiranim podacima izvršavali smo naredbe:

(1) *explain plan for select * from "marko".test1 where id=25; SELECT * FROM table(dbms_xplan.display);* - Naredba koja kao rezultat pronalazi redove u tabeli čiji je id=25

(2) *explain plan for select datum from "marko".test1 order by datum ASC; SELECT * FROM table(dbms_xplan.display);* - Naredba koja kao rezultat vraća datume sortiranje u rastućem redosledu

(3) *explain plan for select name from "marko".test1 where karakter like 'M'; SELECT * FROM table(dbms_xplan.display);* - Naredba koja kao rezultat vraća sva imena korisnika koji zadovoljavaju uslov da u koloni karakter sadrže informaciju M.

(4) *explain plan for select decimalnop+broj from "marko".test1; SELECT * FROM table(dbms_xplan.display);* - Naredba koja kao rezultat vraća zbir dve kolone.

(5) *explain plan for select decimalnop+broj from "marko".test1 where id=24; SELECT * FROM table(dbms_xplan.display);* - Naredba koja kao rezultat vraća zbir dve kolone za zadatu vrednost kolone id.

Rezultati po tabelama – upit (1)	Slike
TEST1	<pre> PLAN_TABLE_OUTPUT Plan hash value: 4122059633 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 100 65 (0) 00:00:01 * 1 TABLE ACCESS FULL TEST1 1 100 65 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - filter("ID">=25) </pre>
TEST2	<pre> PLAN_TABLE_OUTPUT Plan hash value: 4188112605 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 100 2 (0) 00:00:01 1 TABLE ACCESS BY INDEX ROWID BATCHED TEST2 1 100 2 (0) 00:00:01 * 2 INDEX RANGE SCAN INDEX2 1 1 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 2 - access("ID">=25) </pre>
TEST3	<pre> PLAN_TABLE_OUTPUT Plan hash value: 1594733951 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 100 2 (0) 00:00:01 1 TABLE ACCESS BY INDEX ROWID BATCHED TEST3 1 100 2 (0) 00:00:01 * 2 INDEX RANGE SCAN INDEX3 1 1 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 2 - access("ID">=25) </pre>

Tabela 3: Prikaz nakon izvršenog jednog testiranja

Rezultat testiranja je prikazan u tabeli 3. Ovako izvršen upit nad ovako kreirane tri tabele došli smo do zaključka da tabele koje poseduju indeks po kome se pretražuje znatno brže i uz potrošnju procesorske moći čitaju podatke u bazi. Kod tabela TEST2 I TEST3 vidimo da tabele koristi tip skeniranja INDEX RANGE SCAN, ovo koriste zato što je upit zadat kao izraz, i samim tim automatski optimizator koristi ovakav tip skeniranja. Kod tabele TEST1, uz odsustvo indeksa, uvek će se pristupati svim informacijama u tabeli čitajući red po red, i samim tim tip skeniranja je TABLE ACCESS FULL. Ovako kreirana tabela nije efikasna jer ima znatnu potrošnju procesorske moći u odnosu na tabele koje imaju indekse jer optimizator mora pristupiti svakom zapisu, redu, u tabeli.

Rezultati po tabelama – upit (2)	Slike
TEST1	<pre> PLAN_TABLE_OUTPUT Plan hash value: 1692556001 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 117K 67 (3) 00:00:01 1 SORT ORDER BY 15000 117K 67 (3) 00:00:01 2 TABLE ACCESS FULL TEST1 15000 117K 65 (0) 00:00:01 ----- </pre>
TEST2	<pre> PLAN_TABLE_OUTPUT Plan hash value: 2452037388 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 117K 67 (3) 00:00:01 1 SORT ORDER BY 15000 117K 67 (3) 00:00:01 2 TABLE ACCESS FULL TEST2 15000 117K 65 (0) 00:00:01 ----- </pre>
TEST3	<pre> PLAN_TABLE_OUTPUT Plan hash value: 3600789041 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 117K 67 (3) 00:00:01 1 SORT ORDER BY 15000 117K 67 (3) 00:00:01 2 TABLE ACCESS FULL TEST3 15000 117K 65 (0) 00:00:01 ----- </pre>

Tabela 4: Rezultat testiranja

Na ovom primeru i sa korišćenjem indeksa i bez njega postiže se isti rezultat, rezultat je prikazan u tabeli 4. Razlog toga je taj što se pristupa celoj tabeli i sam upit vraća sve rezultate, tako da i sa i bez indeksa rezultat i „cena“ upita je ista. Zato indeksi nisu pogodni, oni se uvek koriste kada se izvršava upit koji vraća mali broj rezultata ili čak jedan rezultat. Pored ovog zaključka, kada se radi o poljima tipa DATE, u realnim sistemima, uvek se pretražuje, pored ovog polja, jos jedno polje koje identifikuje neku tabelu. Da bi postigli veću brzinu upita, potrebno je kreirati složeni tip indeksa koji će na primer indeksirati dve kolone, identifikator polje i datum polje. Ovim postizemo da za određenu vrednost polja dobijemo datum. Ovo nije jedini slučaj, ukoliko pored informacije o polju tipa DATE imamo potrebu za prikazivanjem i drugih informacija, poželjno je kreirati indeks koji će obuhvatiti sve kolone za koje se izvršava upit. Naravno, uvek treba obraditi pažnju na samu strukturu sistema, svaka baza podataka je specifična na svoj način i shodno tome treba kreirati odgovarajući tip indeksa specifičnog za potrebe tog sistema

Ovakav rezultat dobio se isti i za test upit 3. Razlog je isti kao i na prethodnom primeru.

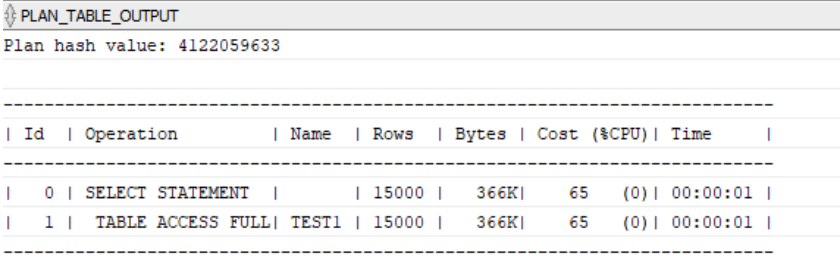
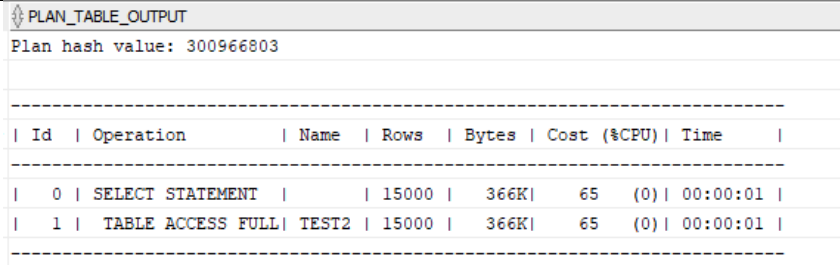
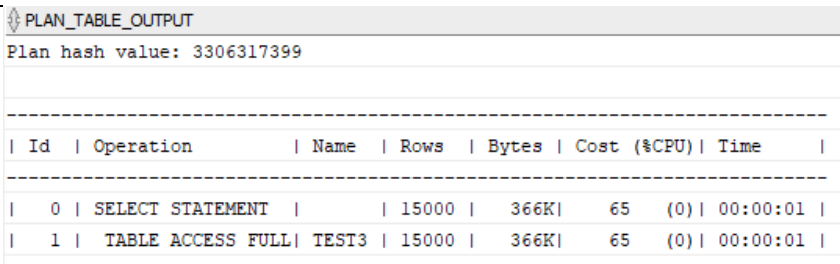
Rezultati po tabelama – upit (4)	Slike
TEST1	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 4122059633 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 366K 65 (0) 00:00:01 1 TABLE ACCESS FULL TEST1 15000 366K 65 (0) 00:00:01 ----- </pre>
TEST2	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 300966803 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 366K 65 (0) 00:00:01 1 TABLE ACCESS FULL TEST2 15000 366K 65 (0) 00:00:01 ----- </pre>
TEST3	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 3306317399 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 366K 65 (0) 00:00:01 1 TABLE ACCESS FULL TEST3 15000 366K 65 (0) 00:00:01 ----- </pre>

Tabela 5: Rezultat testiranja

I u ovom primeru, prikazan u tabeli 5, vidimo istu situaciju kao i na prethodnom. Razlog istih rezultata je taj i ako imamo indeksiranu strukturu tabele, tabela TEST3, ona sadrži za svaku kolonu pojedinačan indeks i prilikom izvršenja upita, optimizator čita vrednosti indeksa i za kolonu *decimalnop* i za kolonu *broj* i pošto rezultat vraća zbir ove dve kolone rezultat će biti isti kao i da smo koristili bez indeksa. U ovakvim slučajevima možemo ubrzati sistem tako što ćemo kreirati kompozitni indeks za kolone *decimalnop* i *broj* i samim tim ubrzamo pretragu ovakvog rezultata (*CREATE INDEX "marko".INDEX12 ON "marko".TEST3 (DECIMALNOP, BROJ);*) ali nećemo postići znatno ubrzanje, isti će rezultat upita biti, ista brzina izvršavanja. Kao i u prethodnom primeru i poželjno je kreirati indeks za one tabele koje vraćaju rezultat.

Za test primer 5, napravićemo izmenu u indeksiranju, preciznije kreiraćemo indeks nad tabelom TEST3 na sledeći način (*CREATE UNIQUE INDEX "marko".INDEX13 ON "marko".TEST3 (BROJ ASC, ID ASC, DECIMALNOP ASC);*) rezultat ovakvog upita je prikazan u tabeli 6.

Rezultati po tabelama – upit (5)	Slike
TEST1	<pre> PLAN_TABLE_OUTPUT Plan hash value: 4122059633 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 30 65 (0) 00:00:01 * 1 TABLE ACCESS FULL TEST1 1 30 65 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - filter("ID">=24) </pre>
TEST2	<pre> PLAN_TABLE_OUTPUT Plan hash value: 4188112605 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 30 2 (0) 00:00:01 1 TABLE ACCESS BY INDEX ROWID BATCHED TEST2 1 30 2 (0) 00:00:01 * 2 INDEX RANGE SCAN INDEX2 1 1 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 2 - access("ID">=24) </pre>
TEST3	<pre> Plan hash value: 2605541091 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 30 2 (0) 00:00:01 * 1 INDEX RANGE SCAN INDEX13 1 30 2 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - access("ID">=24) </pre>

Tabela 6: Rezultat testiranja

Sada vidimo da u ovoj situaciji postizemo ubrzanje ukoliko kreiramo indeks kao što smo naveli, optimizator pristupa samo indeksu, koji sadrži potrebne rezultate, i smanjili smo potrošnju procesorskog vremena.

Zaključak: Bez indeksiranih kolona, baza podataka će raditi sa malom količinom podataka i brzina izvršenja u nekim slučajevima će biti ista kao i kod tabela kod kojih postoje indeksi, sa povećanjem podataka dolazimo do povećanja vremena u pretraživanju. Ukoliko nasumično izaberemo kolone tabele koje indeksiramo dolazimo do nekog ubrzanja, odnosno dolazimo do ubrzanja kod onih upita koji pristupaju indeksiranim kolonama, ukoliko se pristupa ne indeksiranim kolonama nećemo postići ubrzanje (vreme potrebno za izvršenje ovakvog nekog upita biće isto kao i da indeksi ne postoje). Na kraju ako indeksiramo sve kolone, primer kao u

TEST3 tabeli, mi ćemo postići ubrzanje, ali isto tako za one upite koji pristupaju samo indeksiranim kolonama. Zato da bi postigli ubrzanje prilikom pretrage neophodno je dobro proučiti sistem nad kojima se implementiraju indeksi, i kreirati indekse za one kolone čije se vrednosti jako često koriste. U praksi, uvek treba indeksirati kolone koje se nalaze u where klauzuli, indeksiraju se polja koji imaju jaku malu kardinalnost i ona polja koja se nalaze kao uslov JOIN klauzule.

4.2. Primer Bitmap indeksa

Kao i u prethodnom slučaju kreiraćemo 3 tabele, iste kao i u prethodnom primeru samo što ćemo umesto B-tree indeksa koristiti bitmap indekse. Primer naredbe (*CREATE BITMAP INDEX "marko".INDEX2 ON "marko".TEST2 (ID);*)

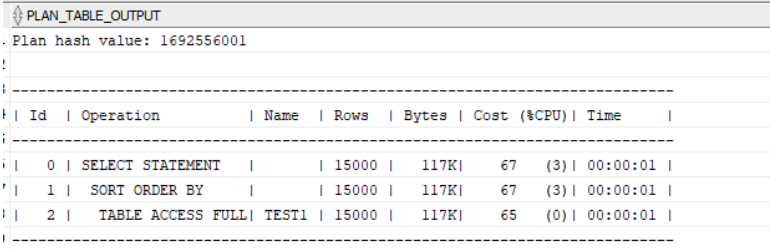
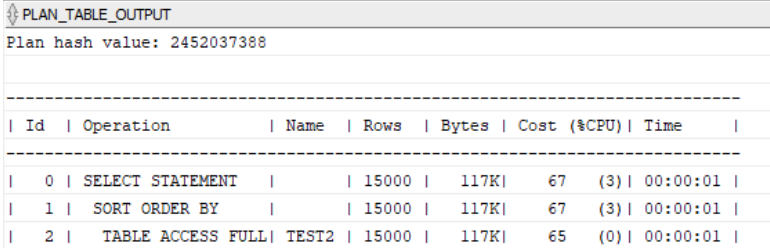
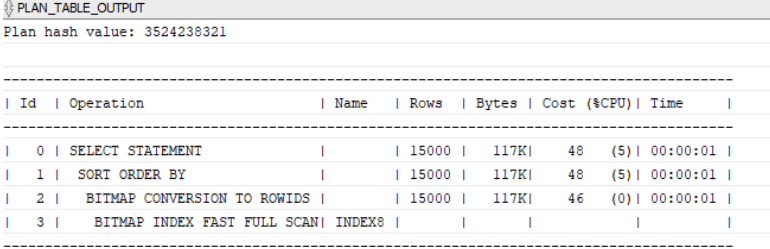
Rezultati po tabelama – upit (2)	Slike
TEST1	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 1692556001 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 117K 67 (3) 00:00:01 1 SORT ORDER BY 15000 117K 67 (3) 00:00:01 2 TABLE ACCESS FULL TEST1 15000 117K 65 (0) 00:00:01 ----- </pre>
TEST2	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 2452037388 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 117K 67 (3) 00:00:01 1 SORT ORDER BY 15000 117K 67 (3) 00:00:01 2 TABLE ACCESS FULL TEST2 15000 117K 65 (0) 00:00:01 ----- </pre>
TEST3	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 3524238321 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 117K 48 (5) 00:00:01 1 SORT ORDER BY 15000 117K 48 (5) 00:00:01 2 BITMAP CONVERSION TO ROWIDS 15000 117K 46 (0) 00:00:01 3 BITMAP INDEX FAST FULL SCAN INDEX8 ----- </pre>

Tabela 7: Rezultat testiranja

U tabeli 7 je prikazan rezultat izvršenja sql naredbe 2. Iz tabele vidimo da smo značajno poboljšali ubrzanje sistema ako smo kao polje datum stavili kao bitmap indeks. U poređenju sa B-tree indeksa došli smo do zaključka da bitmap indeksi su jako pogodni i postiže se

ubrzanje baš iz tog razloga što je mala kardinalnost kolone, sadrži manje sličnih rezultata. S tim u vezi vidimo da polje datum, bitmap indeks brže pretražuje i prikazuje rezultate u poređenju sa B-tree indeksom.

Kao drugo testiranje izabrali smo upit 4 i izvršili smo ga nad bitmap indeksima. Rezultat testiranja dat je u tabeli 8.

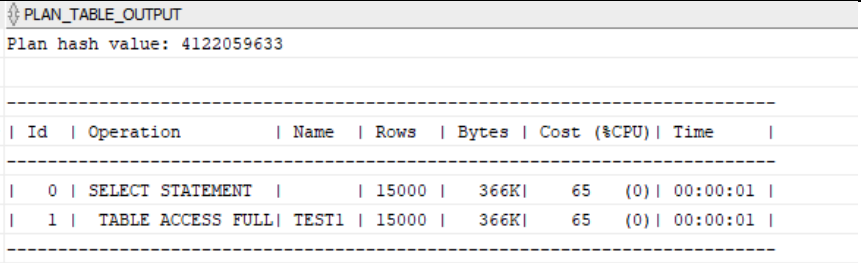
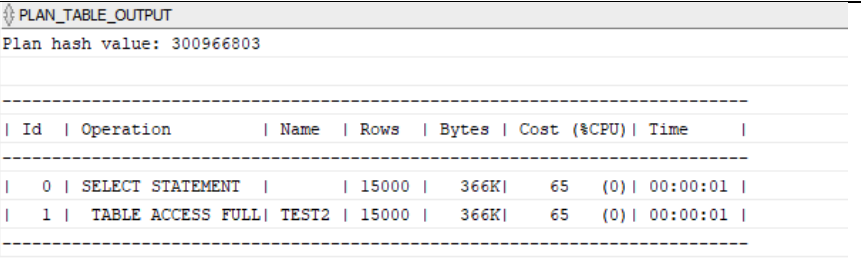
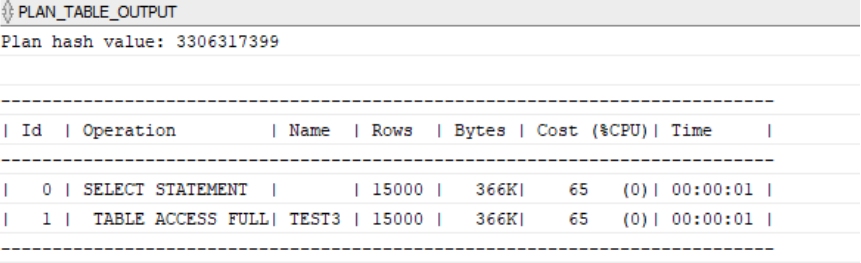
Rezultati po tabelama – upit (4)	Slike
TEST1	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 4122059633 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 366K 65 (0) 00:00:01 1 TABLE ACCESS FULL TEST1 15000 366K 65 (0) 00:00:01 ----- </pre>
TEST2	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 300966803 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 366K 65 (0) 00:00:01 1 TABLE ACCESS FULL TEST2 15000 366K 65 (0) 00:00:01 ----- </pre>
TEST3	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 3306317399 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 15000 366K 65 (0) 00:00:01 1 TABLE ACCESS FULL TEST3 15000 366K 65 (0) 00:00:01 ----- </pre>

Tabela 8: Rezultat testiranja

U poređenju sa indeksima tipa B-tree nismo postigli nikakvo ubrzanje, sada postavlja se pitanja ukoliko kreiramo indeks kao što smo kreirali kod testiranja B-tree indeksa da li ćemo postići bolje rezultate. Jedina razlika biće upit koji izvršavamo, kao i kod testiranja B-tree indeksa testiraćemo nad upitom 5, ali ćemo koristiti bitmap indekse.

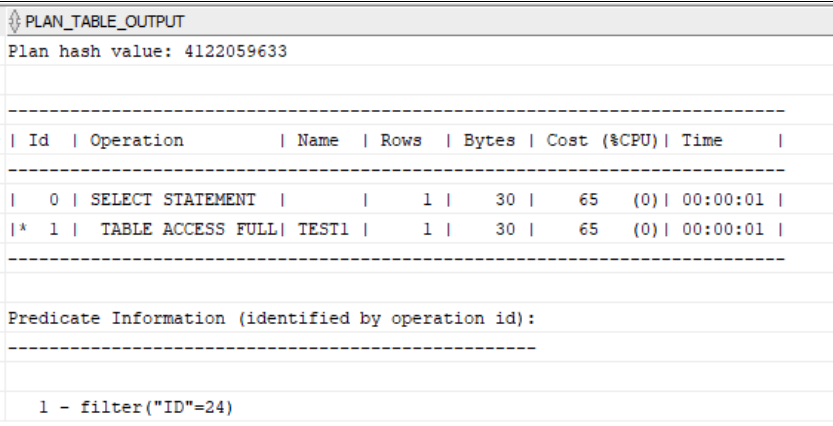
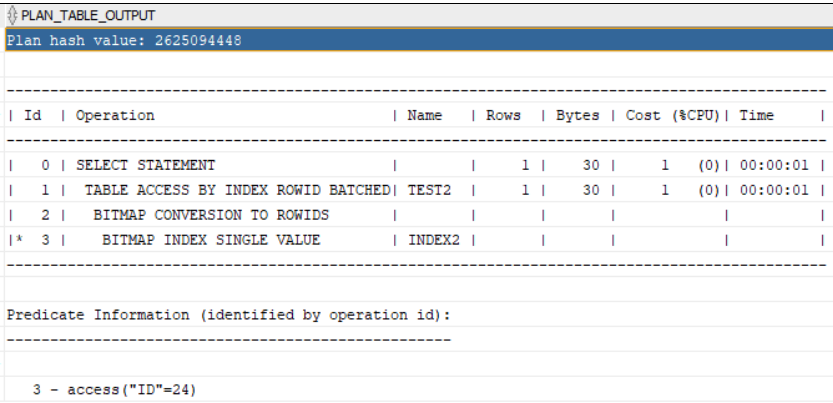
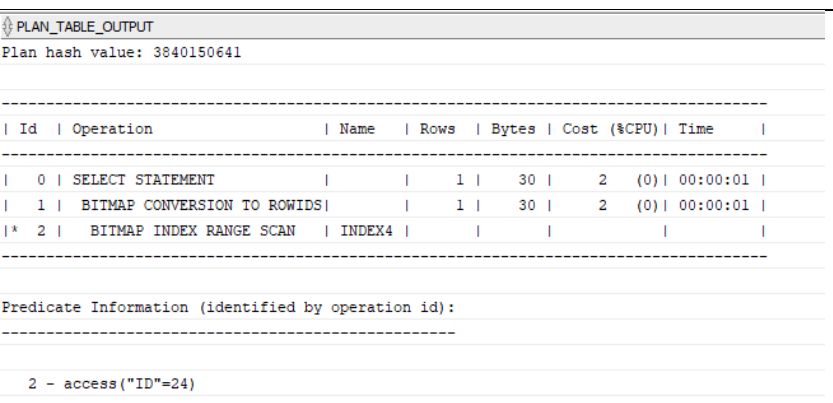
Rezultati po tabelama – upit (5)	Slike
TEST1	 <p>PLAN_TABLE_OUTPUT Plan hash value: 4122059633</p> <pre> ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 30 65 (0) 00:00:01 * 1 TABLE ACCESS FULL TEST1 1 30 65 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - filter("ID">=24) </pre>
TEST2	 <p>PLAN_TABLE_OUTPUT Plan hash value: 2625094448</p> <pre> ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 30 1 (0) 00:00:01 1 TABLE ACCESS BY INDEX ROWID BATCHED TEST2 1 30 1 (0) 00:00:01 2 BITMAP CONVERSION TO ROWIDS * 3 BITMAP INDEX SINGLE VALUE INDEX2 ----- Predicate Information (identified by operation id): ----- 3 - access("ID">=24) </pre>
TEST3	 <p>PLAN_TABLE_OUTPUT Plan hash value: 3840150641</p> <pre> ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 1 30 2 (0) 00:00:01 1 BITMAP CONVERSION TO ROWIDS 1 30 2 (0) 00:00:01 * 2 BITMAP INDEX RANGE SCAN INDEX4 ----- Predicate Information (identified by operation id): ----- 2 - access("ID">=24) filter("ID">=24) </pre>

Tabela 9: Rezultat testiranja

Situacija se nije popravila i samim tim ovde je postignuto maksimalno ubrzanje sistema. Rezultat testiranja je prikazan u tabeli 9. Vidimo da u ovom slučaju brzina izvršavanja upita sa B-tree indeksima i sa bitmap indeksima je identična. U ovoj situaciji B-tree i bitmap indeksi daju isto ubrzanje sistema.

Nakon izvršenog prvog testiranja, kreirane su još dve tabele korišćenjem sledećih naredbi:

```
create table "marko".testwr1 as
select rownum as id,
decode(round(dbms_random.value), 1, 'MP', 'VP') as warehouseID,
DBMS_RANDOM.value (10,5) as price,
250 as testIID
from dual connect by level <=500
```

```
create table "marko".test1NtoN as
select rownum as id,
decode(round(dbms_random.value), 1, 'MP', 'VP') as warehouseID,
DBMS_RANDOM.value (10,5) as price,
DBMS_RANDOM.string ('A',20) as name,
floor(DBMS_RANDOM.value (10,100)) as testIID
from dual connect by level <=500
```

Tabela TESTWR1 napravljena je sa ciljem da pokažemo kakvo bi ubrzanje sistema postigli korišćenjem indeksa kod JOIN naredba, jedan red iz test1 može imati više referenca na redova u testwr1 tabeli. Ovim smo obezbedili da imamo više vrednosti jedne instance tabele test1.

Upit koji je korišćena za testiranje je:

```
explain plan for
select wr1.price
from "marko".test1 t
INNER JOIN "marko".testwr1 wr1 on t.id = wr1.testIID where wr1.warehouseID='MP';
SELECT * FROM table(dbms_xplan.display);
```

Indekse koje smo koristili prilikom testiranja su sledeći:

```
CREATE INDEX "marko".INDEX10 ON "marko".TEST1 (ID ASC);
CREATE INDEX "marko".INDEX14 ON "marko".TESTWR1 (TESTIID, WAREHOUSEID ASC);
```

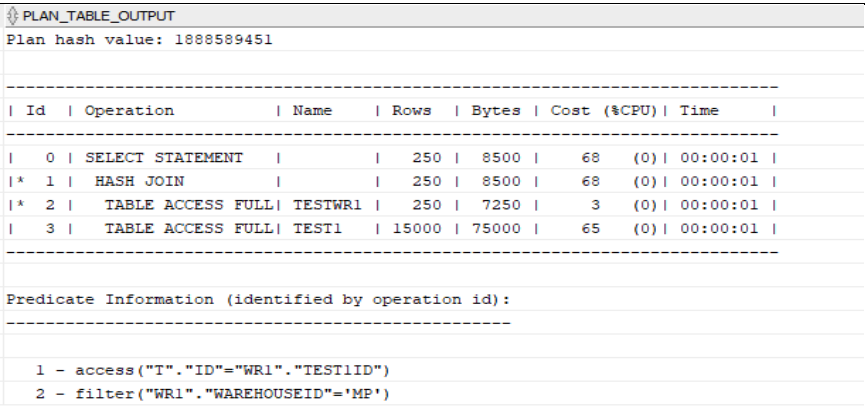
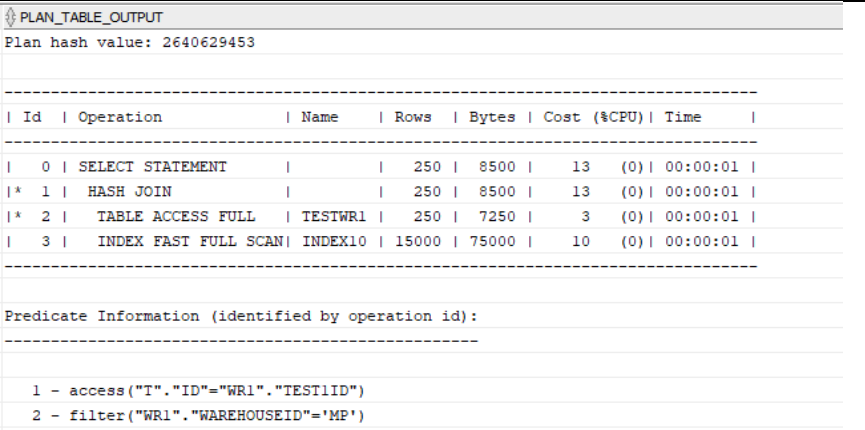
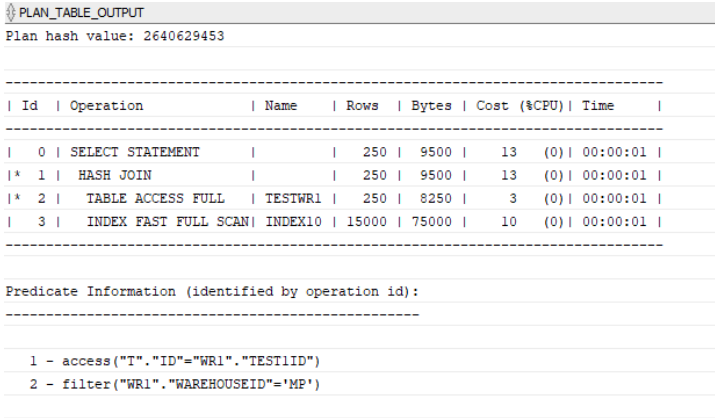
Rezultati testiranja upita	Slike
Bez indeksa	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 1888589451 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 250 8500 68 (0) 00:00:01 * 1 HASH JOIN 250 8500 68 (0) 00:00:01 * 2 TABLE ACCESS FULL TESTWR1 250 7250 3 (0) 00:00:01 3 TABLE ACCESS FULL TEST1 15000 75000 65 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - access("T"."ID"="WR1"."TEST1ID") 2 - filter("WR1"."WAREHOUSEID"='MP') </pre>
Sa indeksom	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 2640629453 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 250 8500 13 (0) 00:00:01 * 1 HASH JOIN 250 8500 13 (0) 00:00:01 * 2 TABLE ACCESS FULL TESTWR1 250 7250 3 (0) 00:00:01 3 INDEX FAST FULL SCAN INDEX10 15000 75000 10 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - access("T"."ID"="WR1"."TEST1ID") 2 - filter("WR1"."WAREHOUSEID"='MP') </pre>

Tabela 10: Rezultat testiranja

Iz tabele možemo zaključiti da korišćenjem ovako kreiranih indeksa postizemo ubrzanje sistema. Sama cena korišćenja procesorske moći prilikom izvršavanja ovakvog upita je veoma niska. Samim tim korišćenje indeksa ubrzalo je proces pretrage. U slučaju da zahtevamo polje koje nije indeksirano, na primer polje "marko".testwr1.id, rezultat operacije je prikazan na slici



```

PLAN_TABLE_OUTPUT
Plan hash value: 2640629453

-----
| Id | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT   |         |    250 |  9500 |    13  (0)| 00:00:01 |
|*  1 |  HASH JOIN         |         |    250 |  9500 |    13  (0)| 00:00:01 |
|*  2 |    TABLE ACCESS FULL| TESTWR1 |    250 |  8250 |     3  (0)| 00:00:01 |
|  3 |      INDEX FAST FULL SCAN| INDEX10 | 15000 | 75000 |    10  (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
 1 - access("T"."ID"="WR1"."TEST1ID")
 2 - filter("WR1"."WAREHOUSEID"='MP')

```

Note

Slika 15: Rezultat testiranja

Sa slike 15 možemo videti da i ako zahtevamo polje koje nije indeksirano opet smo dobili ubrzanje prilikom pretrage, ali smo u poređenju sa prethodnim primerom imali više čitanje podataka. Kada bi smo kreirali indeks dizajniran da vraća vrednosti baš za ovaj upit, dobili smo manji broj čitanja podataka tabele i samim tim obezbedili bismo brzu obradu upita.

Tabela test1NtoN napravljena je sa ciljem da pokažemo kakvo bi ubrzanje sistema postigli korišćenjem indeksa u situaciji kada imamo relaciju N:N.

Upit koji je korišćena za testiranje je:

```
explain plan for
select nwn.price
from "marko".test1NtoN nwn
INNER JOIN "marko".test1 t on t.id = nwn.test1ID
INNER JOIN "marko".testwr1 wr1 on t.id = nwn.test1ID and
nwn.warehouseID=wr1.warehouseID;
SELECT * FROM table(dbms_xplan.display);
```

Prilikom testiranja kreirali smo indekse nad svim poljima nad kojim se vrši upit i kreirali smo kompozitni tip indeksa specifičan za ovaj upit:

```
CREATE INDEX "marko".INDEX18 ON "marko".TESTINTON (ID, WAREHOUSEID,
PRICE);
```

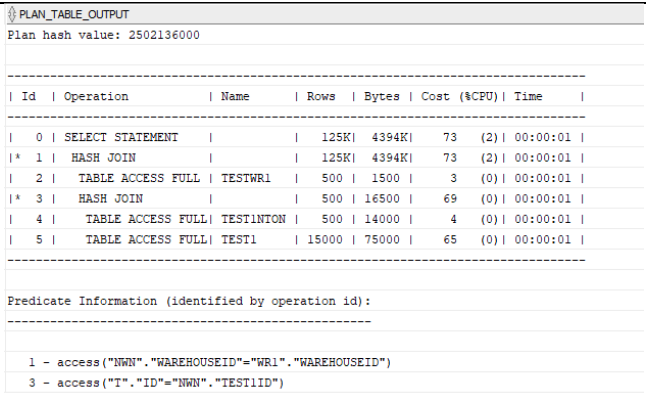
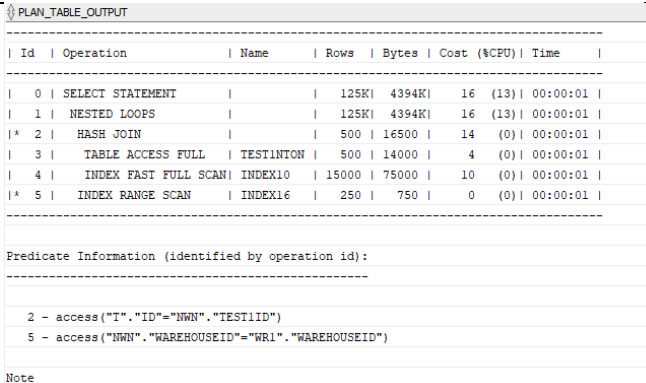
Rezultati testiranja upita	Slike
Bez indeksa	 <pre> PLAN_TABLE_OUTPUT Plan hash value: 2502136000 ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 73 (2) 00:00:01 * 1 HASH JOIN 73 (2) 00:00:01 2 TABLE ACCESS FULL TESTWR1 500 1500 3 (0) 00:00:01 * 3 HASH JOIN 69 (0) 00:00:01 4 TABLE ACCESS FULL TESTINTON 500 14000 4 (0) 00:00:01 5 TABLE ACCESS FULL TEST1 15000 75000 65 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 1 - access("NWN"."WAREHOUSEID"="WR1"."WAREHOUSEID") 3 - access("T"."ID"="NWN"."TEST1ID") </pre>
Sa indeksom	 <pre> PLAN_TABLE_OUTPUT ----- Id Operation Name Rows Bytes Cost (%CPU) Time ----- 0 SELECT STATEMENT 16 (13) 00:00:01 1 NESTED LOOPS 16 (13) 00:00:01 * 2 HASH JOIN 14 (0) 00:00:01 3 TABLE ACCESS FULL TESTINTON 500 14000 4 (0) 00:00:01 4 INDEX FAST FULL SCAN INDEX10 15000 75000 10 (0) 00:00:01 * 5 INDEX RANGE SCAN INDEX16 250 750 0 (0) 00:00:01 ----- Predicate Information (identified by operation id): ----- 2 - access("T"."ID"="NWN"."TEST1ID") 5 - access("NWN"."WAREHOUSEID"="WR1"."WAREHOUSEID") Note </pre>

Tabela 11: Rezultat testiranja

I u ovom testiranju, rezultat testiranja prikazan je u tabeli 11, zadati upit će se brže izvršavati uz posedovanje indeksa. Kao što možemo primetiti broj pročitanih podataka je identičan, ali korišćenje procesorke moći je smanjeno za skoro 5 puta.

5. Zaključak

U ovom seminarskom radu obradili smo temu interna struktura i organizacija indeksa kod Oracle baze podataka. Pored teorijske osnove prikazali smo praktične primere kako indeksiranje utiče na brzinu obrade upita nad podacima u test tabelama. Diskutovali smo o izvršenju operacija prateći njihov plan izvršavanja i na osnovu toga smo izveli zaključke da li je pogodno koristiti B-tree indeks ili bitmap indeks. Pored ovih indeksa, kao primer dali smo i kako se kreiraju indeksi bazirani na funkcijama, gde smo na primeru tabele pokazali realnu situaciju jednog sistema. Takođe prikazali smo kako indeksi utiču na brzinu upita koji poseduju join klauzulu, odnosno „spajanje tabela“.

Oracle svojim korisnicima nudi širok spektar indeksa. Kod Oracle baze podataka indeksi su inicijalno B-tree indeksi ako drugačije ne specificiramo. Pored širokog spektra indeksa, Oracle automatski dodeljuje indekse na osnovu statistike koja se vodi za svaku tabelu ponaosob, tako korisnici Oracle baze podataka ne moraju se opterećivati oko implementacije indeksa. Oracle je napravio mehanizam za automatsko dodavanje indeksa. U nekim situacijama ovakav mehanizam može i usporiti sistem, zato je preporuka da uvek kada se kreira baza podataka za bilo koji sistem, da se detaljno odradi procena i time definišu odgovarajući indeksi na osnovu potrebe aplikacije. Na primer, ukoliko projektujemo bazu podataka koji vrši online kupovinu i pored toga održava skladište sa proizvodima i kategorijama proizvoda, preporuka je koristiti indekse za one tabele čije se informacije najčešće potražuju.