

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ (МАСТЕР) РАД
ОПТИМИЗАЦИЈА ПЕРФОРМАНСИ БАЗЕ
ПОДАТАКА ЗАСНОВАНА НА ПРИСТУПУ
ТРАНСПОНОВАЊА ПОДАТАКА

Ментор

проф. др Ненад Аничих

Студент

Стефан Луковић 2015/3041

Београд, 2017.

Sadržaj

1. Uvod	1
2. Sistemi za upravljanje bazom podataka	3
1.1 Osnovne karakteristike.....	3
2.2 Arhitektura sistema za upravljanje bazom podataka	4
2.3 Logička struktura	6
2.3.1 Logički dizajn	6
2.3.2 Projektovanje logičke strukture	8
3. Fizička struktura baze podataka	12
3.1 Fizičko projektovanje baze podataka	12
3.2 Tabele - različite fizičke organizacije	12
4 Tehnike optimizacije baze podataka	16
4.1 Denormalizacija.....	16
4.2 Materijalizovani pogledi.....	17
4.3 Indeksne strukture i pristupi	19
4.4 Klasterovanje	21
4.5 Distribuirana baza podataka	22
5 ORACLE 12c SUBP	26
5.1 Arhitektura Oracle servera	26
5.2 Baza podataka	27
5.2.1 Instanca	28
5.3 Oracle logička struktura skladišta.....	28
5.3.1 Tablespace	29
5.3.2 Blok.....	29
5.3.3 Ekstent.....	30
5.3.4 Segment	30
5.4 In-memory	31
5.4.1 Format reda i kolona	32
5.4.2 In-memory skladištenje u kolonama.....	33
5.4.3 In-memory kompresija	36
5.4.4 Kontrola upotrebe Oracle IM.....	45
6 Optimizacija fizicke strukture na primeru bankarskog sistema	52
6.1 Opis problema.....	52
6.2 Logička struktura i upit	52

6.3	Primena standardnih pristupa optimizaciji	53
6.4	Izbor pristupa IM u fizičkoj oraganizaciji tabela	55
6.4.1	Optimizacija izvršavanja upita – Primer 1	55
6.4.2	Optimizacija izvršavanja upita – Primer 2	59
6.4.3	Razmatranje optimizacije	61
6.5	Analiza postojećih rešenja optimizacije	62
6.5.1	Istraživanje 1	62
6.5.2	Istraživanje 2	63
7.	Zaključak	65
8.	Literatura	66

1. Uvod

Razvojem poslovanja, preduzeća su primorana da vrše prikupljanje i obradu sve veće količine podataka. Sve su češći zahtevi za alatima koji mogu brzo i efektivno da pomognu pri donošenju odluka koje će pomoći pri unapređenju poslovanja. Često, kako bi reagovali na porast potražnje, kompanije pored skladišta podataka (data warehouse) pokreću analitiku direktno na svojim operativnim sistemima. Ovaj način rada sa podacima dovodi do neizvesnog blansiranja između izvršavanja transakcija, koje mogu biti česti unosi ili izmene, kao i upiti kojima se skenira velika količina podataka radi izveštavanja.

Podaci su najkritičnije sredstvo za svaku organizaciju. Kako su kompanije nastavile da se uvećavaju i njihova potreba za skladištenjem informacija i pristupom se eksponencijalno širi. Baza podataka nudi mesto za čuvanje informacija, ali još važnije je, kada je pravilno organizovana, omogućava efikasano i smisleno pronalaženje informacija. Sistem za upravljanje bazom podataka (SUBP) je softver koji kontroliše skladištenje, organizaciju i prikaz podataka. Sistem za upravljanje relacionim bazama podataka (SURPB), definisan Kodovim pravilima, prilagođava se relacionom modelu sa definisanim objektima skladišta i strukturama. Detaljnije o arhitekturi sistema za upravljanje bazom podataka i logičkoj strukturi pojašnjeno je u prvom delu rada.

U trećem delu rada je opisano je projektovanje fizičke strukture baze podataka, odnosno prevođenje logičke strukture u fizičku, kao i tabele i različite fizičke organizacije.

Baze podataka mogu čuvati ogromne količine informacija, kojima se pristupa pomoću SQL upita. Kako bi se upiti izvršavali brzo, potrebno je izvršiti optimizaciju baze podataka, što podrazumeva maksimizacija brzine i efikasnosti kojom se podaci vraćaju sa diska. Dizajneri baza podataka, administratori i analitičari rade zajedno da optimizuju performance sistema pomoću različitih metoda. Početni dizajn koji pažljivo odgovara funkcionalnim potrebama predstavlja osnovu za poboljšanje performansi. Kreiranje optimizovanih upita može doneti brže rezultate. Različiti načini optimizacije baze podataka su predstavljeni u trećem delu rada.

Oracle je jedan od najviše korišćenih komercijalnih sistema za upravljanje bazom podataka. U četvrtom delu rada je opisana arhitektura Oracle SUBP. Pored arhitekture, opisana je i logička struktura skladišta.

Oracle sa verzijom 12c baze podataka, uvodi dualni format skladištenja podataka, poznatiji kao In-memory. Pored skladištenja podataka u formatu redova, Oracle je omogućio čuvanje podataka u memoriji, u formatu kolona. Koristeći Oracle In-memory opciju, jedna baza podataka može efikasno podržati različite vrste upita koji mogu da optereće sistem, pružajući optimalne performanse za izvršavanje transakcija, istovremeno podržavajući analitiku i izveštavanje u realnom vremenu. Jedinstvena arhitektura dualnog formata Oracle baza podataka, omogućava održavanje podataka u formatu reda za OLTP operacije i u In-memory formatu kolona za analitičku obradu, što predstavlja suštinu ovog istraživanja. Način rada sa In-memory opcijom i njeno efikasno korišćenje je opisano u četvrtom delu.

Rešavanje konkretnog problema, uskog grla bankarskog modela baze podataka je prikazano u petom delu rada. Izvršavanje upita za generisanje izveštaja je upoređeno tradicionalnim načinom optimizacije baze podataka i korišćenjem In-memory opcije na tri različita načina. Pored optimizacije konkretnog primera, prikazana je i analiza postojećih primera optimizacije.

2. Sistemi za upravljanje bazom podataka

1.1 Osnovne karakteristike

Sistem za upravljanje bazom podataka je skup aplikacija i programa koji se koristi za upravljanje objektima baze podataka, kao što su tabele, korisnici, procedure, funkcije i dr. i povezuje bazu podataka sa korisničkim interfejsom (frond-end delom) aplikacije pomoću hardvera. Trenutno postoji nekoliko tipova SUBP, za neke je potrebna kupovina licence, a postoje i sistemi za upravljanje koji su otvorenog koda za koje nije potrebna licenca.

Danas postoji mnogo softverskih sistema za upravljanje bazama podataka. Najveći broj današnjih sistema za upravljanje bazama podataka su višekorisnički što znači da omogućavaju korišćenje i upravljanje bazama koje koristi veliki broj korisnika.

Postoje baze podataka koje su slobodne za korišćenje dok je veći deo softverskih sistema za upravljanje bazama podataka komercijalan i naplaćuje se. Komercijalni sistemi za upravljanje podataka su ogromni softveri koji zahtevaju obuku i stručnost onih koji će te sisteme koristiti. Nije neobično da velike kompanije zaposle jednu ili više osoba koje će biti zadužene za fizičku implementaciju tih sistema. Pored toga tu su i timovi koji će raditi na razvoju logičke strukture baze podataka, kao i timovi koji će razvijati aplikacije koje će komunicirati sa bazom i obezbediti interfejs za one koji ne mogu ili ne bi trebalo da imaju direktan pristup bazama podataka. Pored toga koji sistem za upravljanje bazama podataka je odabran on mora obezbediti neke mogućnosti:

- Sistem za upravljanje bazama podataka mora da obezbedi sredstva za kreiranje strukture baze podataka tako da su programeri u mogućnosti da definišu logičku strukturu podataka koji će biti skladišteni u toj bazi uključujući i veze između tih podataka.
- Sistem za upravljanje bazama podataka mora da obezbedi neki način za unos, menjanje i brisanje uskladištenih podataka. Manji sistemi su zasnovani na formama, dok su oni napredniji zasnovani na „command - line“ interfejsima. Najčešće korišćen jezik za upravljanje podacima kod relacionih baza je SQL (Structured Query Language) koji je prihvaćen kao standardni jezik za upravljanje podacima u relacionim bazama

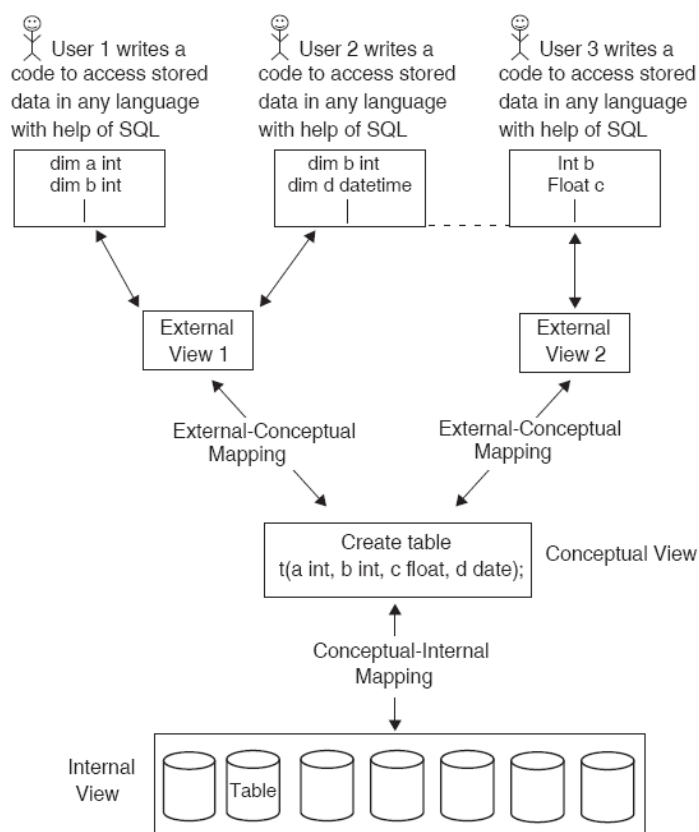
- Takođe ovi sistemi moraju da obezbede i način za pronalaženje podataka. Korisniku mora biti omogućeno da postavlja upite zasnovane na logičkoj povezanosti podataka. Neki od tih upita su složeni pa treba omogućiti i korišćenje logičkih operacija kao što su „AND“, „OR“, „NOT“... kao i da sistem bude u stanju da izvrši neke osnovne proračune na podacima dobijenim kao rezultat upita.
- Iako je moguće korišćenje sistema kroz neke osnovne forme ili SQL komandnu liniju radnicima je neophodna obuka i usavršavanje. U poslovanju postoje ljudi koji treba da upravljaju podacima ali nemaju znanja i za takve korisnike programeri prave program preko koga će lakše upravljati podacima. Obično je to neki softver koji će crtati forme za upite ili izvlačiti neke izveštaje.
- Autentikacija i autorizacija. Sistem za upravljanje bazama podataka treba i da ima mehanizme da različitim korisnicima obezbedi različite nivoe pristupa. Obično se to rešava tako što svaki korisnik ima svoje ime i šifru kojom pristupa bazi i njegovom imenu su dodeljene određene uloge kojima je definisano šta korisnik sme, a šta ne sme da radi. Takođe sistem za upravljanje bazama podataka mora da u nekoj meri garantuje i bezbednost podataka organizacije.

2.2 Arhitektura sistema za upravljanje bazom podataka

Pre više od 40 godine, 1975. godine, ANSI-SPARC (Američki nacionalni institut za standarde – odbor za planiranje standarda) razvio je troslojnu arhitekturu sa sistemskim katalogom. Arhitektura većine komercijalnih SUBP koji su dostupni danas, bazirana je na ANSI-SPARC predlogu.

ANSI/SPARC arhitektura baze podataka ima tri nivoa:

- Interni – predstavlja skladište gde se zapravo čuvaju podaci. Takođe je poznat kao fizički nivo, tj. nivo koji je koristan za računare da razumeju podatke.
- Konceptualni – uz pomoć ovog nivoa, interni i eksterni nivo komuniciraju. Šema baze podataka se definiše na ovom nivou koristeći DDL (Data Definition Language)
- Eksterni – ovaj nivo se odnosi na korisnike. Na ovom nivou korisnici pristupaju podacima sačuvanim u bazi.



Slika 1 – detaljna arhitektura SUBP (1)

Eksterni nivo predstavlja pogled za korisnike, što omogućava pristup više korisnika koji rade na različitim platformama i programskim jezicima, a pristupaju podacima i strukturama podataka skladištenim u bazi podataka. Korisnici koriste različite programske jezike koji imaju različitu sintaksu za pristup podacima. Na slici 1 postoje tri različita korisnika koji koriste istu tabelu "T", ali različita polja tabele. Dva korisnika pristupaju tabeli "T" koristeći JAVA programski jezik, a treći koristeći programski jezik C. Tipovi podataka, za definisanje promenljivih u oba jezika, imaju različite sintakse i mogu imati različite promenljive za skladištenje polja. Korisnici koriste zajednički jezik za baze podataka, tj. SQL (Strukturirani upitni jezik), koji prikazuje podatke iz baze podataka. Tako da je ono što korisnici vide kombinacija programskog jezika i SQL-a.

Konceptualni nivo funkcioniše kao "prevodilac" između internog i eksternog nivoa. Eksterni nivo služi za korisničke poglede, dok je koncetualni nivo pogleda, koji se

naziva i logički pogled, namenjen za korisnike koji pripadaju određenoj grupi. Preko ovog zajedničkog pogleda, svaki korisnik može da pristupi delu baze podataka koji mu je potreban pomoću određenih mapiranja. (1)

2.3 Logička struktura

Jan Harrington u četvrtom izdanju knjige “Relational Database Design and Implementation” (2) navodi da logički dizajn predstavlja konceptualno i apstraktno modelovanje baze podataka u odnosu na fizički dizajn. U logičkom dizajnu se može videti odnos među objektima, dok u fizičkom je akcenat stavljen na najefikasniji način skladištenja i čitanja objekata.

Dizajn treba da bude orijentisan ka potrebama krajnjih korisnika. Krajnji korisnici obično žele da naprave analizu i vide objedinjene podatke, radije od pojedinačnih transakcija. Dobro isplaniran dizajn omogućava rast i promene kako se potrebe korisnika menjaju. Počevši sa logičkim dizajnom, fokus treba da bude na jasnim i konkretnim zahtevima, bez potrebe za definisanjem načina implementacije.

2.3.1 Logički dizajn

Logički dizajn je konceptualni i apstraktni dizajn, gde se definišu vrste informacija koje su potrebne. Proces logičkog dizajna podrazumeva organizovanje podataka u niz logičkih odnosa koji se zovu entiteti i atributi. Entitet predstavlja neku celinu informacija, koji se u relacionim bazama podataka često preslikava u tabelu. Atribut je komponenta nekog entiteta koja pomaže definisanju njegove jedinstvenosti, koja se u relacionim bazama predstavlja kao kolona u tabeli.

Iako se objektno relacioni dijagrami vezuju tradicionalno za normalizovane modele kao što je OLTP, ova tehnika se koristi i kod dimenzionalnog modelovanja, uz primenu različitih pristupa. U dimenzionalnom modelovanju, umesto da se otkriva atomske jedinice informacija i svi odnosi između njih, potrebno je identifikovati koje informacije pripadaju tabeli i informacije koje predstavljaju veze ka dimenzionim tabelama. Izlaz logičkog dizajna je skup entiteta i atributa koji predstavljaju tabele i njihove veze.

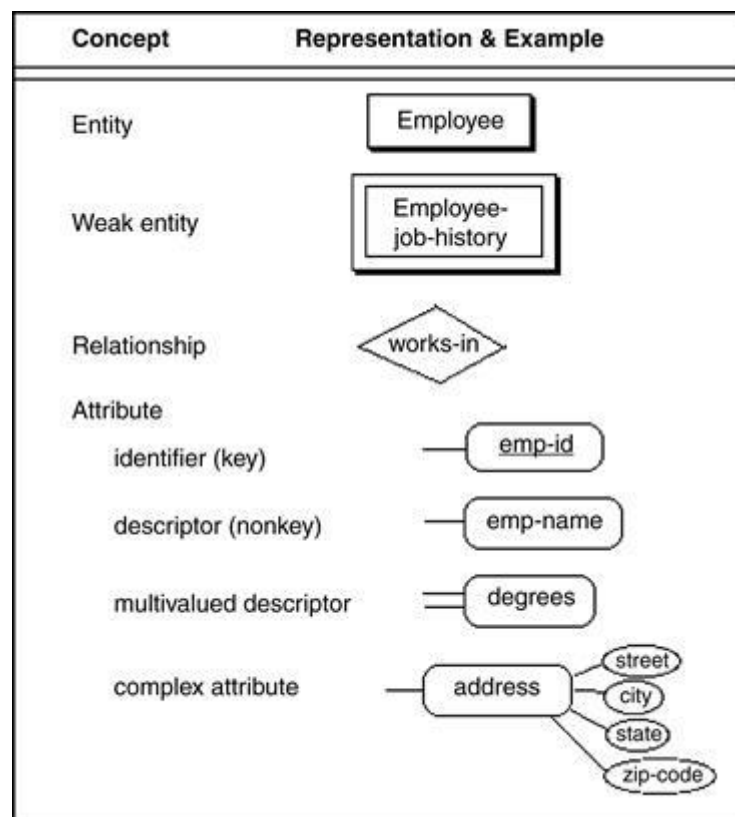
Entiteti

Entiteti su osnovni objekti podataka za koje su informacije prikupljene i oni obično označavaju osobu, mesto, stvar ili događaj informacionih interesa. Poseban događaj entiteta se naziva instanca entiteta ili pojava entiteta.

Veze

Veze predstavljaju realne asocijacije između jednogi ili više entiteta, i kao takva, nema fizičko ili konceptualno postojanje. Relacije ili veze se opisuju u pogledu stepena, poveyanosti i postojanja. Najčešće značenje odnosa asocijacija koje ukazuje na povezanost između entiteta su:

- jedan na jedan (1,1)
- jedan prema više (1,M)
- Više prema više (M,M)



Slika 2 – koncepti za kreiranje logičkog dizajna (2)

Atributi i ključevi

Atributi su karakteristike entiteta koji pružaju detaljne opise o njima. Pojava atributa unutar jednog entiteta ili veze se naziva vrednost atributa.

Postoje dve vrste atributa: identifikatori i deskriptori. Identifikator (primarni ključ) se koristi da jednoznačno odredi instancu entiteta i on predstavlja jedinstvenu vrednost, koji otklanja mogućnost za pojavljivanjem duplikata. Primarni ključ se može sastojati iz jednog ili više atributa koji jedinstveno određuju objekat. Spoljni ključ predstavlja identifikator nekog drugog objekta koji služi za definisanje veze između dva objekta. Deskriptori (sekundarni ključevi) se koriste da opišu nejedinstve karakteristike određenog entiteta.

Slabi objekti

Entiteti imaju unutrašnje identifikatore ili ključeve koji jednoznačno određuju svaku pojavu entiteta, ali slabi objekti su entiteti koji proširuju svoj identitet sa primarnim ključem nadređenog objekta. Slabi objekti se često prikazuju u duplim pravougaonicima, koji označava da su sva pojavljivanja tog entiteta zavisna od postojanja povezanog entiteta u bazi.

2.3.2 Projektovanje logičke strukture

Prolazeći faze projektovanja, svaka faza projektovanja baze podataka ima specifično ime koje najbolje opisuje model koji se kreira. Faze su definisane tako da se olakša postizanje definisanog cilja. U praksi, dobar dizajn i implementacija su se pokazali od suštinskog značaja. Na početku, ciljevi bi trebali da u osnovi opisuju zahteve, a nakon definisanja ciljeva potrebno je implementirati ih koristeći odgovarajuće tehnike i na kraju podesiti implementaciju da funkcioniše u realnom vremenu.

Proces koji nas usmerava kroz stvaranje baze podataka čine 4 faze:

Konceptualna: Tokom ove faze cilje je napraviti skicu baze podataka koja će se dobiti od prikupljenih početnih zahteva i informacija od korisnika. Ovu fazu identifikuje ono šta korisnik želi na visokom nivou. Informacije se prevode u model podataka koji se sastoji od entiteta na visokom nivou i veza između njih. Ime "konceptualno" se zasniva na pronalaženju koncepta, a ne na stvaranju konceptualnog dizajna.

Logička: Logička faza predstavlja implementaciju nedefinisanih detalja u konceptualnoj fazi, pretvarajući koncept u potpuno relacioni dizajn baze podataka koji će biti osnova implementacije. Tokom ove faze nastaje model koji je potreban sistemu i obuhvata sva poslovna pravila koja je potrebno implementirati

Fizička: U ovoj fazi se prilagođava logički model za implementaciju sistemu za upravljanje bazama podataka (Oracle, SQL Server, MySQL...). U najvećoj meri fokus u ovoj fazi je na izgradnji rešenja za dizajn koji odgovara izlazu iz logičke faze.

Podešavanje performansi: U ovoj fazi se radi sa modelom koji je mapiran na skladište. Vršiti se podešavanje, optimizacija i prilagođavanje performansi projekta, zato što je važno da implementacija funkcioniše, bez obzira na hardver i klijentsku aplikaciju. (3)

Za razliku od konceptualnog modela, koji predstavlja opširnu sliku, u delu logičkog projektovanja akcenat je stavljen na detalje. U delu logičkog modelovanja, moguće je dodavanje entiteta modelu, ali su to često entiteti koji će se identifikovati kao složeni atributi (npr. "broj telefona", mogu postojati entiteti koji imaju više brojeva telefona). Tokom ove faze se dodaju atributi, identifikuju pravila, proces itd. kako bi se osigurali svi podaci koji su potrebni.

Identifikovanje atributa i domena

Na početku stvaranja logičkog modela, cilj je pronaći stavke koje identifikuju i opisuju entitete koje je potrebno predstaviti. Na primer, ako je entitet osoba, atributi mogu biti matični broj, pol, boja očiju, datum rođenja visina, e-mail adresa ili e-mail adrese. Svaka od ovih stvari delom predstavlja entitet "Osoba". Identifikovanje koji atributi asociraju entitet zahteva sličan pristup identifikovanju samog entiteta. Često se mogu pronaći atributi tako što će se navesti pridevi koji se koriste da bi se entitet opisao. Neki atributi će se jednostavno pronaći u zavisnosti od toga kojoj vrsti entiteta pripadaju (osoba, mesto itd.).

Informacije o domenu atributa se generalno otkrivaju istovremeno sa atributima, tako da je potrebno identifikovati domen kad god se može otkriti. U nastavku su opisani neki od najčešćih tipova atributa koje treba tražiti tokom procesa identifikacije atributa i njihovih domena:

- **Identifikatori**: sve informacije koje se koriste da identifikuju jednu instancu entiteta. Identifikator je obično i ključ, ali identifikatori neće uvek kreirati odgovarajuće ključeve, već će identifikovati načine na koje korisnik može tražiti određenu instancu.

- Opisne informacije: informacije koje se koriste za opisivanje nečega o entitetu, kao što su boja, status, imena, opisi itd.
- Lokatori: opisuju lokaciju gde je entitet kreiran, fizički u stvarnom svetu, kao što je adresa ili na tehničkom nivou, položaj na ekranu računara.
- Vrednosti: stvari koje kvantifikuju nešto o entitetu, kao što su novčani iznosi, brojevi, datumi itd.

Identifikatori

Svaki entitet treba da ima barem jedan set atributa koji ga identifikuju. Bez atributa, ne postoji način da se različiti objekti mogu identifikovati kasnije u procesu. Ovi identifikatori će verovatno postati kandidati za ključ entiteta, ali ne uvek (ponekad se ne može garantovati jedinstvenost ili da će svaka instanca imati vrednost). Npr. dobri identifikatori mogu biti za ljude matični broj, za knjige ISBN broj, za proizvode bar kod.

Relacioni atributi

Svaka veza koja je identifikovana može podrazumevati neku količinu podataka. Na primer, kada kupac plaća fakturu podrazumeva odnos (vezu) između entiteta Korisnik i entiteta Faktura. Taj odnos implicira da korisnik mora da plati fakturu, što predstavlja atribut iznos.

Identifikovanje poslovnih pravila

Poslovna pravila se mogu definisati kao iskazi koji upravljaju i oblikuju poslovno ponašanje. U zavisnosti od metodologije organizacije, ova pravila mogu biti kao beleške, jednostavni tekstualni dijagrami ili u drugim formatima. Postojanje poslovnog pravila ne podrazumeva mogućnosti implementacije u bazi podataka u trenutku definisanja. Cilj je smanjiti sva pravila orijentisana ka podacima koja će se kasnije koristiti u procesu.

Kada se definišu poslovna pravila, moguće je da se jave dupliranja pravila i atributa domena, ali to neće praviti problem u tom trenutku. Potrebno je definisati sva moguća pravila, jer nedostatak nekog pravila, može biti manje prihvatljivo nego nedostatak atributa, veze ili čak tabele. Često se pri implementaciji sistema mogu pronaći nove tabele i atributi, ali nedostajuća poslovna pravila mogu umanjiti kvalitet podataka za izveštavanje ili čak uništiti ceo dizajn.

Prepoznavanje poslovnih pravila nije težak proces, ali oduzima puno vremena. Za razliku od entiteta, atributa i veza, ne postoji direktno, specifičan gramatički orijentisani pojam za identifikaciju svih poslovnih pravila.

Identifikacija osnovnih procesa

Proces je niz koraka koje preduzima program koji koristi podatke. To može biti računarski proces, na primer “dnevni obračun” ili kreiranje depozita za slanje u banku. Može biti ručni, “kreiranje novog komitenta”, čije detalje prvi put klijent unosi u formu, zatim službenik postavlja različita pitanja i sl. Često proces koji se kreira i koji traje dva koraka i nekoliko minuta, može da se razvuče mesecima i mesecima. Potrebno je otkriti zašto je tako i da li je to dobar i ponekad bezbednosno orijentisan razlog.

Identifikovanje procesa i pravila koja ih određuju su relevantni za modelovanje podataka, jer će mnogi od tih procesa zahtevati manipulaciju podatka. Svaki proces se obično prevodi u jedan ili više upita ili skladištenu proceduru, što može zahtevati više podataka nego što je navedeno, posebno za čuvanje informacija o stanju tokom čitavog procesa.

3. Fizička struktura baze podataka

3.1 Fizičko projektovanje baze podataka

Fizicki dizajn baza podataka

Harington (2) definiše fizički dizajn baza podataka kao proces menjanja strukture baze kako bi se unapredile performanse prilikom izvršavanja programa i pristupa bazi. To znači da se menjaju tabele koje su u trećoj normalnoj nastale kreiranjem logičke strukture kako bi se ubrzala aplikacija koja ih koristi. Mnoštvo modifikacija se može primeniti, od jednostavnog dodavanja indeksa do izmene strukture tabela.

Prevođenje iz logičkog u fizički dizajn

Iako prevođenje zahteva u logički dizajn, primarni cilj faze logičkog projektovanja je da se napravi dizajn koji zadovoljava funkcionalne zahteve aplikacije. Poređenja radi, faza fizičkog dizajna pre svega osigurava da baza podataka ispunava uslove za izvršavanje aplikacije. Najveća greška koja se javlja u ovoj fazi je da se stvori fizički model koji je istovetna kopija logičkog modela. Mapiranje logičkog modela jedan na jedan na fizički model je obično jednostavno postići, ali međutim to rezultira fizički model koji ne podržava primenu visokih performansi. Odvojiti dovoljno vremena kako bi se dobro izvršilo fizičko modelovanje procesa, rezultira model koji može da podrži visoke zahteve performansi.

Prevođenja entiteta u tabele

Entitet u logičkom modelu se često prevodi u tabelu u fizičkom odelu. Ova transformacija je obično jednostavna, osim kada entitet sadrži pod tipove. Pod tipovi ili particije logičkih entiteta pomažu da se klasifikuju informacije koje se nalaze unutar entiteta. Podtip ima obično niz atributa koji se nalaze u nadređenom entitetu i attribute koji su svojstveni samo za njega.

3.2 Tabele - različite fizičke organizacije

Tabela je osnovna jedinica skladištenja u bazi podataka, bez tabele, baza podataka nema vrednost za okruženje. Bez obzira na vrstu tabele, podaci u tabeli se čuvaju u redovima i kolonama. (4)

Relacione tabele

Relaciona tabela je najčešći tip tabele u bazi podataka. Relaciona tabela je organizovana kao heap, drugim rečima, ne postoji red po kom se čuvaju slogovi u tabeli. Svaka tabela sadrži jednu ili više kolona, a svaka kolona ima tip podatka i dužinu. Kolone mogu sadržati korisničke definisane tipove, ugnježdene tabele ili nizove. Tabela može biti definisana kao objektna tabela. Neki od tipova podataka su number, varchar2, date, timestamp, long itd.

Privremene tabele

Ova vrsta tabela je privremena u smislu podataka koji se čuva u tabeli, ne u definiciji same tabele. Naredbom `CREATE GLOBAL TEMPORARY TABLE` kreira se privremena tabela. Kao i kod drugih tabela, korisnici mogu da izvršavaju `SELECT` upit ili DML operacije nad privremenom tabelom. Međutim, svaki korisnik vidi samo svoje podatke u tabeli. Kada korisnik briše privremenu tabelu, briše samo podatke koje je on upisao. Postoje dve vrste privremenih podataka u privremenoj tabeli: privremeni podaci za vreme trajanja transakcije i privremeni podaci za vreme trajanja sesije. Dugovečnost privremenih podataka se kontroliše `ON COMMIT` klauzulom, gde se sa `ON COMMIT DELETE ROWS` uklanjaju svi redovi iz tabele kada se izvrši `COMMIT` ILI `ROLLBACK`, a ukoliko je potrebno čuvati podatke nakon transakcije, onda se poziva klauzula `ON COMMIT PRESERVE ROWS`. U oba slučaja kada se prekine sesija korisnika, svi redovi koje on vidi u privremenoj tabeli se brišu.

Indeksno organizovane tabele

Kreiranje indeksa omogućava efikasnije pronalaženje određenog reda, s tim što baza podataka mora da održava slogove podataka i unose indeksa. Kada tabela nema mnogo kolona, a pristupa se prvenstveno jednoj koloni, indeksno organizovane tabele (IOT) mogu biti pravo rešenje. IOT čuvaju redove u tabeli u indeksu B stablo, gde svaki čvor indeks B stabla sadrži indeksiranu kolonu zajedno sa jednom ili više neindeksiranih kolona. Najuočljivija prednost je u tome što je potrebno održavanje samo jedne strukture skladišta umesto dve, a slično tome, vrednosti primarnog ključa tabele se čuvaju jednom u ION umesto dva puta u regularnoj tabeli. Međutim, postoji nekoliko nedostataka koiršćenja IOT. Neke tabele, poput tabela za logovanje događaja, nemaju primarni ključ, ili bilo koji drugi ključ, dok indeksno organizovana tabela mora da sadrži primarni ključ. Takođe IOT ne može biti deo klastera, a možda

IOT nisu najbolje rešenje za tabele koje imaju veliki broj kolona i kada se često pristupa mnogim kolonama kada se izvlače određeni redovi iz tabele.

Objektne tabele

Korisnički definisani tipovi, zajedno sa definisanim metodama za ove tipove objekata, mogu se implementirati u objektno-orijentisani razvojni projekat. Objektne tabele imaju redove koji su sami po sebi objekti, ili instance definisanih tipova. Redovi u tabeli mogu da se referenciraju preko ID objekta, za razliku od primarnog ključa što je kod relacionih tabela. Takođe, objektne tabele mogu imati i primarne i jedinstvene (unique) ključeve, kao relacione tabele.

Eksterne tabele

Eksterne tabele omogućavaju korisniku da pristupi izvoru podataka, npr. tekstualna datoteka, koji mogu biti kao tabele u bazi podataka. Metapodaci za tabelu se čuvaju u okviru Oracle rečinka podataka, ali se sadržaj čuva spolja.

Definicija eksternih tabela sadrži dva dela. Prvi i najpribližniji deo je definicija tabele sa korisničke tačke gledišta. Ova definicija izgleda kao bilo koja standardna definicija koja se može videti u naredbi CREATE TABLE. Drugi deo je ono što razlikuje eksternu tabelu od regularne. Ovde se vrši se mapiranje između kolona baze podataka i eksternog izvora podataka – od kog elementa počinje kolona, koliko je dužina kolone i da li je format spoljne kolone karakter ili binarni. Sintaksa za podrazumevani tip eksterne table, ORACLE_LOADER, je praktično identična kontrolnoj datoteci u SQL*Loader. Ovo je jedna od prednosti eksternih tabela, a korisniku je potrebno da zna kako da pristupi spoljnoj tabeli da dobije eksternu datoteku.

Nad eksternom tabelom se ne mogu kreirati indeksi i ne mogu se izvršavati DML operacije, što ne predstavlja velike nedostatke, kada se razmatraju prednosti korišćenja spoljnih tabela za punjenje regularnih tabela, na primer u data warehouse okruženju.

Klasteri tabela

Klasteri tabela predstavljaju grupu tabela koje dele zajedničke kolone i čuvaju ih u zajedničkim blokovima. Kada se tabele klasteruju, jedan blok može sadržati redove iz više tabela. Ključ klastera je kolona ili više kolona koje su zajedničke za klasterovane

tabele. Pošto klasteri tabela čuvaju povezane redove različitih tabela u istim blokovima podataka, pravilno korišćenje klastera tabela nudi sledeće pogodnosti u odnosu na neklasterovane tabele:

- Spajanje tabela je smanjeno kod klastera
- Vreme pristupa se poboljšava za spajanje tabela
- Manje skladište je potrebno za čuvanje povezanih tabela i indeksnih podataka, jer se vrednost ne čuva više puta za svaki red.

Nedostaci kod klastera tabela se izražavaju u sledećim situacijama:

- Tabele se često ažuriraju
- Često je potrebno skeniranje cele tabele
- Tabele zahtevaju truncate

Particionisane tabele

Particionisanjem tabela se olakšava upravljanje ogromnim tabelama. Tabela može biti particionisana, ili subparticionisana na manje delove. Sa stanovišta aplikacije, podela je transparentna. Jedini efekat koji korisnik može da primeti je da upiti nad particionisanim tabelama u WHERE uslovu koji se podudara sa particionisanim šemama radi mnogo brže. Ako je particija jedne tabele na oštećenom delu diska, ostale particije su dostupne za izvršavanje upita, dok se oštećeni deo ne oporavi.

4 Tehnike optimizacije baze podataka

4.1 Denormalizacija

Kako bi se sačuvao i održao integritet podataka neophodno je izvršiti normalizaciju, ipak u nekim realnim situacijama potrebno je vratiti redudansu podataka. Proces vraćanja redudanse podacima naziva se denormalizacija. Denormalizacija se radi iz više razloga, kao što su poboljšanje performansi baza ili praćenje istorije. Potpuno normalizovana tabela prikazuje trenutno stanje tako da ukoliko dođe do promene nekih podataka u nekim tabelama korisnik baze neće biti u mogućnosti da sazna šta se dešavalo u prethodnom periodu pa se ovaj problem može rešiti čuvanjem kopija podataka. Npr. ako u nekom projektu vezanom za fakturisanje kupac promeni adresu i traži fakture vezane za pređašnju adresu šema koja je potpuno normalizovana to neće moći da prepozna. Tokom procesa normalizacije vrši se razdvajanje tabela što uzrokuje da je potrebno više spajanja tabela prilikom upita i to negativno utiče na performanse sistema. Ovaj problem se rešava ukoliko se strani ključ jedne tabele ponovi u drugim tabelama koje su vezane. Npr. ukoliko postoji treba izvršiti spajanje tri tabele što je veoma skupa operacija, rešenje može biti da se ključ jedne tabele javi u ostale dve kao strani ključ i na taj način moći će da se izvuku podaci iz tih tabela bez spajanja čime se poboljšavaju performanse sistema.

Ipak denormalizacija donosi opasnost pa je neophodno da se uradi pažljivo i sa određenom namerom. Ceo proces denormalizacije je potrebno dokumentovati. Da bi denormalizovana šema pravilno upravljala informacijama potrebno je koristiti transakcije na pravi način. Transakcija je najmanja jedinica koja će ili potpuno biti izvršena ili se uopšte neće izvršiti. Ukoliko jedan unos u neku tabelu automatski povlači i promene u nekoj drugoj potrebno je da se obe akcije (unos i promena) izvrše ili može doći do ozbiljnih problema u upravljanju podacima. Ukoliko se jedna od akcija ne izvrši uspešno treba izvršiti „roll back“ na početno stanje kako bi zadržali konzistentnost podataka.

Najbolji izbor za održavanje denormalizovanih podataka bio bi da se taj posao prepusti sistemima za upravljanje relacionim bazama podataka. Ovo se može obaviti uvođenjem trigera, jezika za manipulaciju podataka (DML – Data manipulation language). Sistem za upravljanje relacionim bazama podataka ovaj posao obavlja tako što automatski aktivira trigere kao deo svake transakcije. Pored toga, treba imati

procedure koje će u svakom trenutku moći da oporave podatke od mogućih oštećenja.

4.2 Materijalizovani pogledi

Materijalizovani pogledi omogućavaju direktan pristup tabeli smeštanjem rezultata upita u odvojeni objekat šeme. Za razliku od običnog pogleda koji ne zauzima prostor za skladištenje podataka i ne sadrži podatke, materijalizovani pogled sadrži podatke dobijene iz upita nad nekoliko tabela baze podataka. Materijalizovani pogled se može nalaziti i u bazi podataka u kojoj se nalaze tabele nad kojima je upit postavljen kao i u nekoj drugoj bazi podataka.

Materijalizovani pogledi koji su sačuvani u istoj bazi podataka, kao i njegove tabele, može poboljšati performanse sistema koristeći prepisivanje upita (query rewrites). Prepisivanjem upita omogućava se pristup drugim upitima i unapred izračunatim podacima sačuvanih u materijalizovanom pogledu.

Drugi naziv za materijalizovani pogled je i snimak (snapshot). Ovaj pojam se uglavnom koristi za kopiranje podataka u udaljenoj bazi podataka. U SQL jeziku pojmovi „snimak“ i „materijalizovani pogled“ su sinonimi.

Materijalizovani pogledi se mogu koristiti za:

Lakši pristup podacima

Ako je jedan od ciljeva da se smanji opterećenje mreže onda se može koristiti materijalizovani pogled kako bi se distribuirala baza podataka. Umesto da cela organizacija pristupa jednom serveru baze podataka korisnik pristupa na više servera. Korišćenjem višestrukih materijalizovanih upita moguće je kreirati materijalizovan pogled baziran na drugim pogledima što omogućava distribuiranje učitavanja podataka na većem nivou jer korisnik pristupa materijalizovanim pogledima umesto globalnoj bazi podataka. Kako bi se smanjila redundansa podataka materijalizovani pogled može biti deo glavne tabele ili glavnog materijalizovanog pogleda.

Materijalizovani pogledi se ažuriraju periodično. Iz tog razloga one imaju mnogo manje mrežne zahteve i manje su zavisni od glavne baze. Kako glavna baza zahteva konstantnu vezu preko mreže materijalizovani pogledi se koriste samo povremeno i nije neophodna stalna konekcija. Pored toga što ne zahtevaju stalnu mrežnu konekciju kopiranje podataka preko materijalizovanih pogleda omogućuje pristup podacima lokalno

Kreiranje okruženja za masovni razvoj

Šabloni za razvoj omogućavaju da se kreira okruženje materijalizovanih pogleda lokalno. Koristeći ove šablone kreiraju se razna okruženja za masovni razvoj. Parametri omogućuju da se kreira skup podataka prilagođenih korisniku bez da se menja šablon. Ova tehnologija omogućuje rad stotinama i hiljadama korisnika.

Omogućava kreiranje podskupova podataka

Materijalizovani pogledi omogućavaju preslikavanje podataka na nivou reda ili kolone. Podskupovi podataka omogućavaju da nekom korisniku budu dostupne samo oni podaci koji bi mogli da interesuju tog korisnika.

Omogućiti rad bez konekcije

Materijalizovani pogledi ne zahtevaju posebnu mrežnu konekciju. Iako postoji mogućnost automatizacije procesa osveživanja materijalizovanih procesa, oni se mogu osvežiti ručno i na zahtev. Što je idealna situacija za određene vrste aplikacija kod kojih radnik nakon što obavi neku celinu posla može osvežiti informacije u pogledima.

Materijalizovani pogledi mogu biti:

- oni koji su samo za čitanje (read only),
- one koji se mogu menjati (updatable) i
- one u koji se može upisivati (writable)

Pogledi iz kojih se može čitati se kreiraju tako što se izostavi opcija (for update). Materijalizovani pogledi koji su samo za čitanje imaju iste mehanizme kao i druge vrste pogleda osim što ne moraju da pripadaju nekoj grupi materijalizovanih pogleda. Pored toga, materijalizovani pogledi koji se mogu čitati eliminišu mogućnost konflikta podataka.

Primer kreiranja materijalizovanih pogleda koji se može čitati

```
CREATE MATERIALIZED VIEW hr.zaposleni AS  
SELECT * FROM hr.zaposleni@firma.odeljenje
```

Pogledi koji se mogu menjati se kreiraju tako što se doda klauzula „FOR UPDATE“. Ovi pogledi moraju biti smešteni u neku grupu materijalizovanih pogleda

Ova vrsta pogleda smanjuje opterećenje globalne baze jer korisnik može neke podatke da menja lokalne i bez pristupa glavnoj bazi.

Primer kreiranja materijalizovanog pogleda koji se može menjati

```
CREATE MATERIALIZED VIEW hr.odeljenja FOR UPDATE AS  
SELECT * FROM hr.odeljenja@firma.odeljenje;
```

Pored kreiranja materijalizovanih pogleda nekada je potrebno kreirati i grupu materijalizovanih pogleda:

```
BEGIN  
  DBMS_REPCAT.CREATE_MVIEW_REPGROUP (  
    gname => 'hr_repg',  
    master => "firma.odeljenje",  
    propagation_mode => 'ASYNCHRONOUS');  
END;
```

Materijalizovani pogledi u koje se može upisivati se kreiraju tako što se doda klauzula „FOR UPDATE“ ali ovi pogledi ne moraju biti deo neke grupe materijalizovanih pogleda. Korisnik može da obavlja sve operacije nad upitima ali ukoliko korisnik osveži pogled doći će do gubljenja unetih podataka jer se oni neće poslati na glavnu bazu.

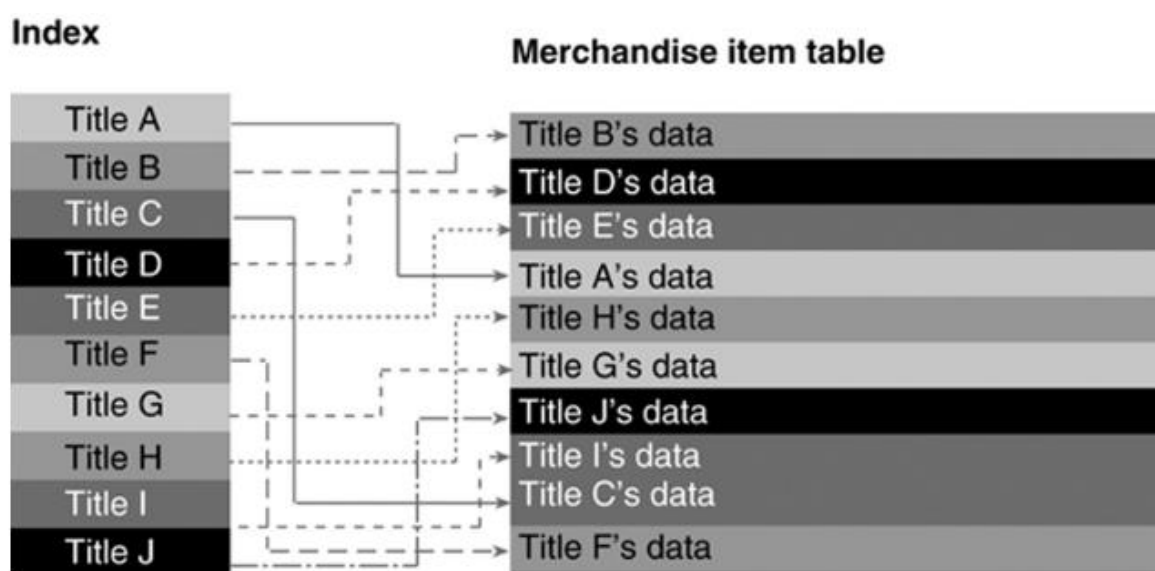
4.3 Indeksne strukture i pristupi

Indeksiranje je obezbeđivanje brzog pristupa vrednostima u kolonama. Pri dodavanju novih redova u tabele oni se obično smeštaju na dnu tabele što rezultuje nasumičnom redosledu vrednosti u koloni. Bez narušavanja redosleda u tabeli sistem za upravljanje bazama podataka mora da izvrši sekvencijalno pretraživanje od vrha do dna kako bi našao neki podatak. Što je tabela veća to će i pretraživanje biti sporije.

Alternativa indeksiranju je sortiranje što i nije tako dobro jer većina sql jezika vrši sortiranje virtuelnih tabela koje su rezultat delovanja upita. Bez obzira na to koliko redova ima tabela i koji sort se koristi ovaj način traženja podataka je veoma spor jer zahteva prolazak kroz sve redove čak i zauzima dodatnu memoriju koja je

potrebna kako bi se neki red pomerio dok mu se ne nađe odgovarajuća pozicija. Takođe pretraživanje sortirane baze je sporije od traženja po indeksu, dok upravljanje indeksima zahteva manje vremena i troši manje performansi nego upravljanje sortiranim tabelama.

Način na koji indeksi rade prikazan je na slici 3. Na slici se vide i stavke relacije kao i ključ koji obezbeđuje brz pristup redovima u tabelama baziranim na nazivima stavki. Indeks sadrži sortiranu listu ključeva (naslova) zajedno sa povezanim redovima u tabelama. Na ovaj način sistem za upravljanje bazama podataka može iskoristiti informacije iz ključa kako bi direktno pristupio redu ili redovima u tabeli izbegavajući sporo sekvencijalno pretraživanje.



Slika 3 – Indeksi i veze sa podacima u tabeli (1)

Kada se jednom napravi indeks, optimizator upita sistema za upravljanje podacima će koristiti taj indeks svaki put kada se utvrdi da će se uz pomoć indeksa brže doći do podataka nego nekim drugim načinom. Korisnik baze više nikada neće morati da pristupa indeksu osim ukoliko želi da ga obriše.

Kada se kreira primarni ključ neke tabele, sistem za upravljanje bazama podataka će automatski napraviti i indeks koristeći kolonu primarnog ključa ili više kolona ukoliko je ključ napravljen od više kolona. Ovo se naziva ideks ključ. Prvi korak u dodavanju reda je potvrda da je primarni ključ tabele jedinstven u indeksu. Na taj način se jedinstvenost celog unosa obezbeđuje pretragom jedinstvenog ključa

u indeksu umesto u celoj tabeli jer je indeks sortiran niz ključeva a tabela je sastavljena od nasumično poređanih redova.

Pored toga što sistem za upravljanje bazama podataka sam pravi indekse nad kolonama koje su primarni ključ za tabelu i sam korisnik može napraviti indeks nad bilo kojom kolonom ili skupom kolona mada pri tome treba obratiti pažnju na nekoliko stvari:

- Indeksi zauzimaju prostor u bazi podataka. Ovo i nije neki nedostatak s obzirom da je danas prostor jeftin
- Kada se ubacuju menjaju ili brišu podaci iz indeksnih kolona sistem za upravljanje bazama podataka treba da ažurira i indeks što dovodi da usporenja sistema
- Indeksi mogu značajno ubrzati pristup podacima

Iz ovoga se vidi da je kompromis između brzine pristupa i brzine ažuriranja podataka.

4.4 Klasterovanje

Najsporiji deo akcija nekog sistema za upravljanje bazama podataka je preuzimanje podataka sa diska ili upisivanje na njega. Ukoliko bi se smanjio broj pristupa disku koji je potreban kako bi se neki podaci pročitali ili uskladištili na disk tada bi se poboljšale performanse baze podataka. Kako bi baza mogla da čita i piše nešto na disk potrebno je pristupiti celoj strani diska na kom su podaci u celini. Veličina strane nije uvek konstanta jer zavisi od raznih kompjuterskih platformi i može se menjati od 512 b do 4 kb pri čemu je 1kb prosečna veličina stranice kod personalnih računara i podaci se uvek kreću u veličini strane. Ukoliko se podaci kojima se češće pristupa drže na jednoj stranici diska ili na stranicama koje su blizu jedna druge može se smanjiti vreme potrebno za upisivanje i čitanje sa diska. Ovaj pristup naziva se klasterovanje ili grupisanje i dostupan je kod mnogih velikih sistema za upravljanje bazama podataka.

U praksi klaster je dizajniran tako da drži zajedno redove koji su povezani poklapanjem primarnih i stranih ključeva. Da bi se definisao klaster treba definisati kolonu ili kolone nad kojima sistem za upravljanje bazama podataka treba da napravi klaster kao i tabele koje treba da budu uključene u klaster. Zatim se svi redovi, kod kojih je vrednost u koloni ili kolonama nad kojima je definisan klaster, čuvaju što je

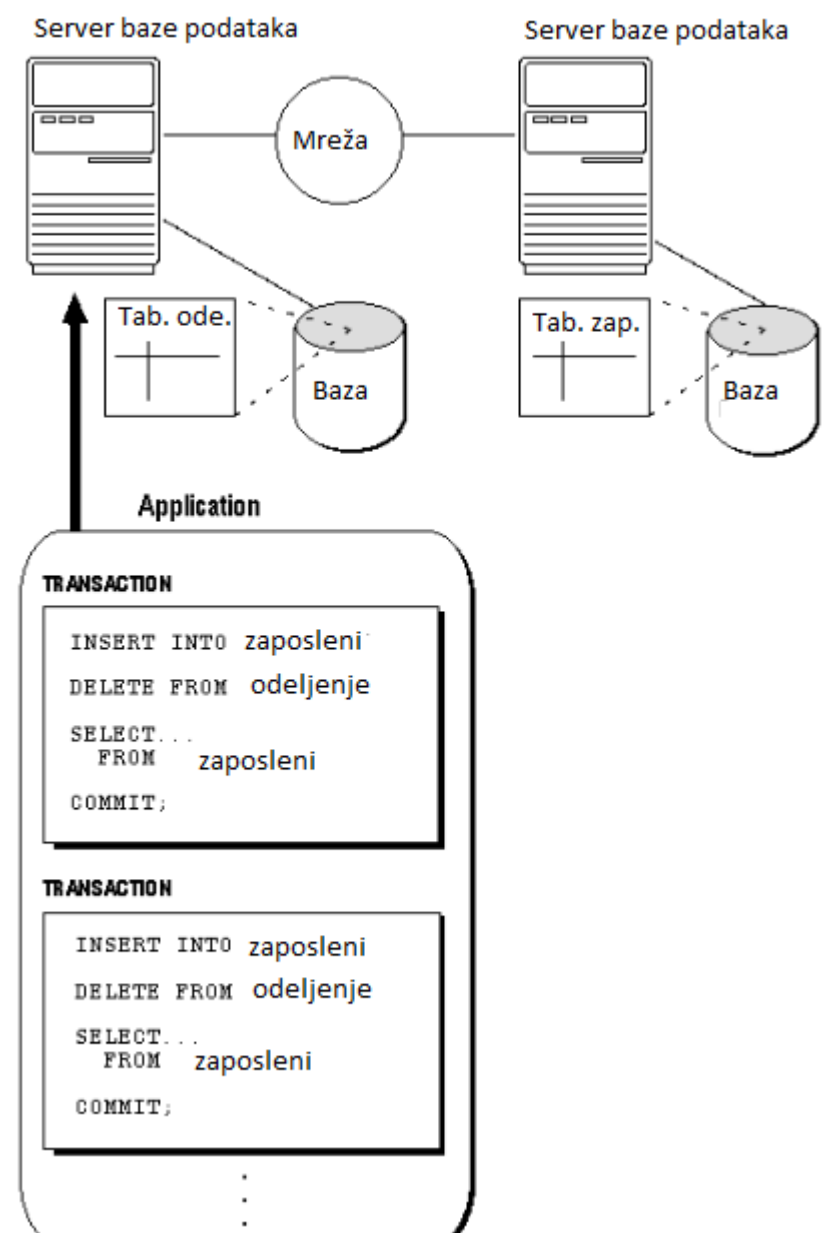
moguće bliže jedan drugom. Kao rezultat toga moguće je da redovi u tabelama budu rasuti na nekoliko različitih strana ali primarni i strani ključevi koji se podudaraju se nalaze na istoj stranici diska.

Na ovaj način mogu se značajno ubrzati rezultati spajanja (join) tabela. Međutim i kod klasterovanja treba uzeti u obzir neke negativne strane:

- Pošto klaster uključuje fizičko postavljanje podataka u fajl, tabela se može klasterizovati samo nad jednom kolonom ili grupom kolona
- Klasterizovanje može da uspori operacije koje zahtevaju skeniranje cele tabele pošto su redovi rasuti na više stranica diska
- Klasterizovanje može da uspori upisivanje podataka
- Klasterizovanje može da uspori promenu podataka u kolonama nad kojima je klaster definisan.

4.5 Distribuirana baza podataka

Distribuirana baza podataka je jedinstvena baza koja se sastoji od više baza podataka uskladištenih na različitim računarima. U takvim bazama se može istovremeno pristupati i menjati ih korišćenjem mreže. Svaki server baze podataka u distribuiranim bazama je pod kontrolom lokalnih sistema za upravljanje bazama podataka i međusobno sarađuju da održe konzistentnost globalne baze podataka.



Slika 4 - je prikazan sistem distribuirane baze podataka (5)

Klijenti, serveri i čvorovi

Server baze podataka je softver koji upravlja bazom, a klijent je aplikacija koja zahteva informacije od servera. Svaki računar u sistemu je čvor. Takođe čvorovi u sistemu distribuiranih baza podataka mogu biti i serveri. Svaki računar se ponaša kao server kada neko zahteva podatke koji se na njemu nalaze, dok se računari ponašaju kao klijenti kada zahtevaju podatke koji se nalaze na udaljenim računarima.

Nezavisnost

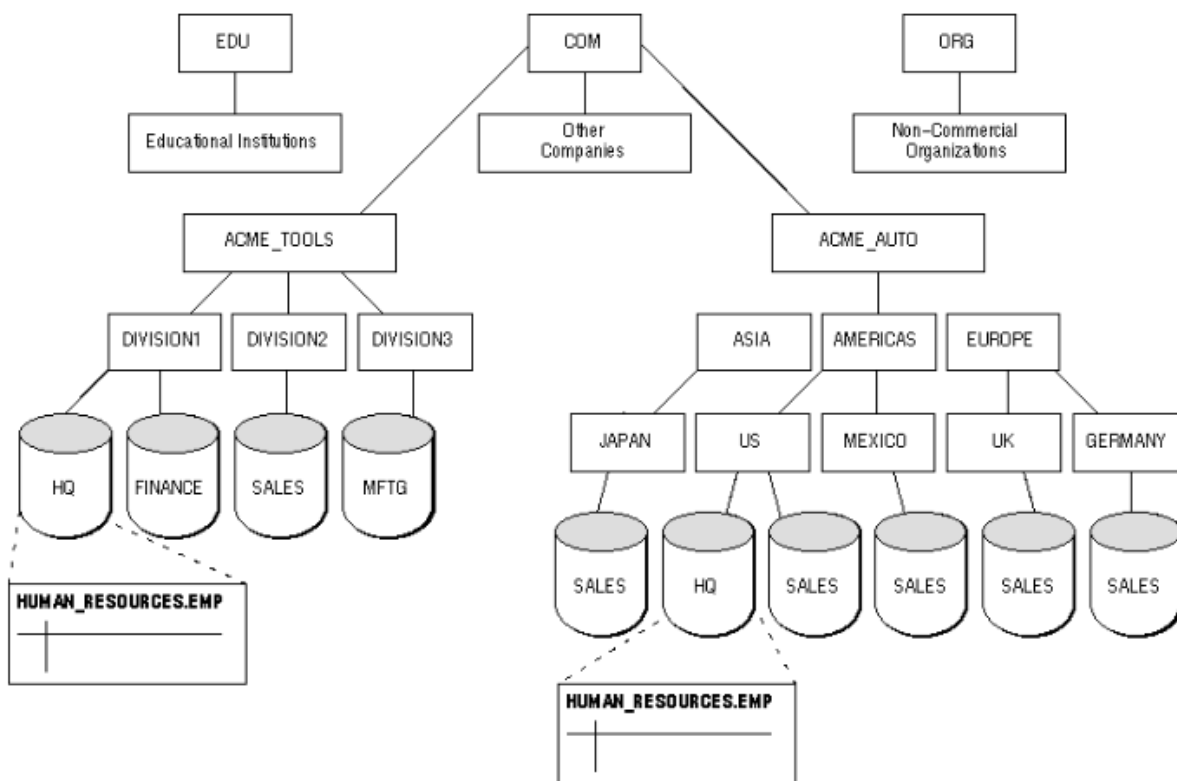
Nezavisnost podrazumeva da svaki server nezavisno učestvuje u distribuiranim bazama podataka (za bezbednost i „backup“ opcije) kao i da svaka

baza pojedinačno nije distribuirana. Iako baze mogu da rade zajedno, one su odvojena skladišta podataka i deluju pojedinačno. Neke od prednosti nezavisnosti su:

- Čvorovi prikazuju logičku strukturu organizacije
- Lokalnim podacima upravlja lokalni administrator. Na taj način domen odgovornosti lokalnih administratora je manji i njime je lakše upravljati
 - o Ukoliko dođe do pada nekog servera neće uticati na druge čvorove. Globalna baza podataka će biti delimično dostupna dok god postoji neki dostupan čvor. Na taj način pad jedne baze neće zaustaviti globalne operacije i u takvom sistemu neće doći do „uskog grla“ u radu.
- Ukoliko dođe do pada oporavak podataka se vrši na lokalnom nivou
- Rečnik podataka postoji u svakoj lokalnoj bazi
- Čvorovi mogu da nadograđuju svoj softver nezavisno

Šema objekata i imenovanje u distribuiranim bazama podataka

Pristup šemi objekata (tabela) je moguć iz svakog čvora koji je deo sistema. Kao što se i kod nedistribuiranih baza podataka mora obezbediti nedvosmisleno imenovanje za referentne objekte tako se i u sistemu za upravljanje distribuiranim podacima mora napraviti šema za jedinstveno imenovanje i referenciranje objekata širom globalne baze. Da bi se rešio problem referenciranja (ovaj proces se još naziva i rezolucija) primenjuje se hijerarhijski pristup. Na primer, sistem za upravljanje bazom podataka garantuje da će u toj bazi svaka šema imati jedinstven naziv kao i da će unutar tih šema svaki objekat imati jedinstven naziv. Kako se ovaj proces sprovodi na svakom hijerarhijskom nivou lokalni nazivi šema i objekata su garancija jedinstvenosti celokupne organizacije.



Slika 5 - Primer rešavanja jedinstvenosti naziva preko hijerarhijske strukture (1)

Povezivanje baza

Kako bi se povezale dve ili više baze podataka u distribuiranim bazama podataka vrši se preko linkova baza podataka, zato što je naziv linka isti kao i globalni naziv. Kreiranjem linkova između baza podataka omogućuje se svakom korisniku ili aplikaciji koja koristi neku lokalnu bazu pristup podacima svim bazama sa kojima je povezan.

Upiti i transakcije u distribuiranim bazama

U distribuiranim bazama razlikuju se udaljeni i distribuirani upiti kao i udaljene i distribuirane transakcije. Udaljeni i distribuirani upiti mogu biti:

- Udaljeni upiti su upiti koji vraćaju informacije koje se nalaze u jednoj ili više tabela, koje se nalaze na jednom čvoru
- Udaljeni update je operacija koja menja podatke na jednoj ili više tabela , koje se nalaze na jednom čvoru
- Distribuirani upit je upit koji vraća informacije sa jednog ili više čvorova
- Distribuirani update je operacija koja vrši promenu informacija na više čvorova. Ovde se koriste okidači (trigeri) i procedure koje koriste udaljeni update kako bi pristupili svakom čvoru i na njima odradili pojedinačni update.

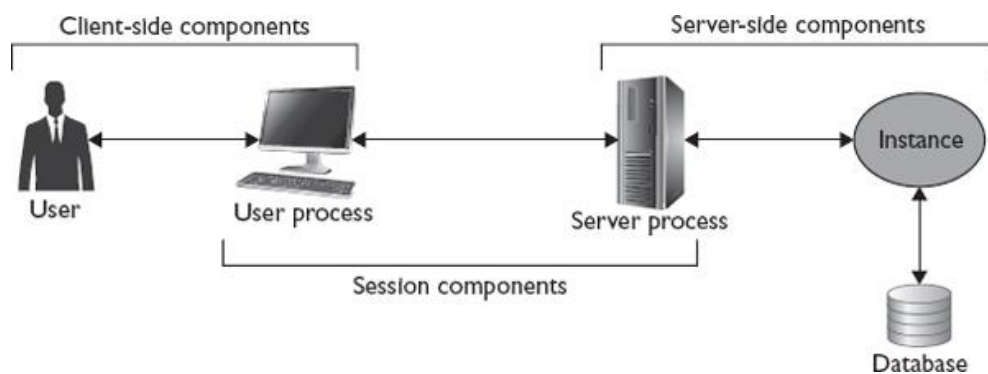
Udaljene transakcije su transakcije koja sadrži jedan ili više upita, koji deluju na jedan udaljen čvor. Distribuirane transakcije su one transakcije koje se sastoje od jednog ili više upita koje se odnose na različite čvorove u mreži.

5 ORACLE 12c SUBP

5.1 Arhitektura Oracle servera

Oracle baza podataka je skup datoteka na disku. Ona postoji dok se ne izbrišu svi fajlovi. Ne postoje ograničenja veličine i broja tih fajlova, i samim tim ne zna se granica veličine baze podataka. Pristup bazi podataka pomoću Oracle instance je skup procesa i memorijskih struktura koje postoje na procesoru i u memoriji servera, i ovo postojanje je privremeno. Instanca se može pokrenuti i zaustaviti. Korisnici baze podataka uspostavljaju sesiju preko instance, a instanca upravlja pristupom bazi podataka. Apsolutno je nemoguće u Oracle okruženju bilo koji korisnik da direktno pristupi bazi. Oracle instanca zajedno sa Oracle bazom podataka čini Oracle server. Model obrade koji implementira Oracle Server je klijent-server obrada. U ovom modelu, korisnički interfejs i aplikaciona logika je odvojena od upravljanja podacima. Za aplikacije razvijene korišćenjem SQL, ovo znači da klijentski sloj generiše SQL komande, a server ih izvršava. Ovo je osnovni prikaz klijent-server modela, gde se za povezivanje koristi lokalna mreža (LAN). Protokol mrežne komunikacije koji se koristi između korisničkog i serverskog procesa je Oracle Net.

Klijentski sloj se sastoji iz dve komponente: korisnika i korisničkih procesa, dok serverski sloj se sastoji iz: serverskog procesa koji izvršava SQL, instance i baze podataka. Svaki korisnik upravlja korisničkim procesom koji deluje na serverski proces, najčešće preko mreže. Serverski proces je u interakciji sa instancom, a instanca sa bazom. Sesija predstavlja komunikaciju korisničkog procesa sa serverskim. Obično je jedan korisnički proces po korisniku i serverski proces po korisničkom procesu. Korisnički i serverski proces pokreću sesiju na zahtev korisnika i prekidaju kada više nije potrebna (4,5).



Slika 6 – Interakcija klijent – baza podataka (4)

Korisnički proces može biti bilo koji softver na strani klijenta koji se u stanju da se konektuje na Oracle server proces. Na primer, jednostavni procesi koje ruža Oracle su SQL*Plus i SQLDeveloper, koji uspostavljaju sesiju preko Oracle servera za izvršavanje SQL upita na zahtev.

5.2 Baza podataka

Baza podataka je kolekcija podataka na disku u jednoj ili više datoteka na serveru baze podataka koji prikuplja i održava relevantne informacije. Baza podataka se sastoji od raznih fizičkih i logičkih struktura, gde tabela predstavlja najvažniju logičku strukturu baze podataka. Tabela se sastoji od redova i kolona koji sadrže odgovarajuće podatke. U najmanju ruku, baza podataka mora imati tabele kako bi skladištila korisne informacije. Pored toga, baza podataka omogućava određeni nivo bezbednosti za sprečavanje neovlašćenog pristupa podacima.

Datoteke koje čine bazu podataka se mogu podeliti u dve kategorije:

- datoteke baze podataka
- ostale datoteke

Razlika između ova dva tipa datoteka je u tome koja vrsta podataka se čuva u svakom od njih. Datoteke baze podataka sadrže podatke i metapodatke, dok druga kategorija sadrži inicijalizacione parametre, informacije logovanja i dr. Fajlovi baze podataka su kritični za tekuće operacije baze koji se konstantno menjaju.

5.2.1 Instanca

Ovo je glavna komponenta tipičnog serverskog okruženja sa jednim ili više procesorom, diskom i memorijom. Dok se baza podataka čuva na disku servera, Oracle instanca egzistira u memoriji. Instanca se sastoji iz velikog bloka memorije alocirane na prostoru koji se naziva System Global Area (SGA), zajedno sa velikim brojem propratnih procesa koji su u interakciji sa SGA i fajlovima na disku.

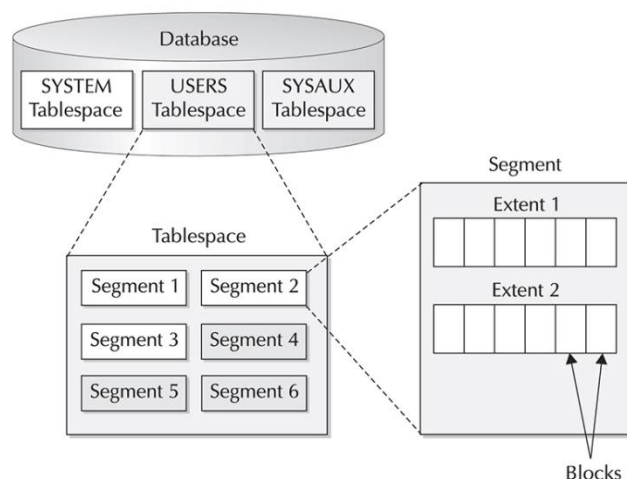
U Oracle RAC (Real Application Clusters), više od jedne instance koristi istu bazu podataka. Iako instance mogu da dele bazu na istom serveru, najverovatnije instance će biti podeljene na različitim serverima koji su međusobno povezani brzim vezama i pristupaju bazi.

Fizička struktura datoteka, redo log i kontrolnih fajlova

Oracle 12c baza podataka predstavlja revolucionaran korak u odnosu na prethodna izdanja Oracle 11g i 10g. Oracle 12c nastavlja tradiciju funkcionalnih poboljšanja tako što je upravljanje planom izvršenja više automatizovano, dodate nove funkcije virtuelizacije, značajno unapređena dostupnost i sposobnost ponovnog pokretanja delova koji prestanu sa radom.

5.3 Oracle logička struktura skladišta

Datoteke u Oracle bazi su grupisane zajedno u jedan ili više tablespaces. U okviru svakog tablespace-a, logičke strukture baza podataka, kao što su tabele i indeksi, su segmenti koji su podeljeni u ekstente i blokove. Ova podela skladišta dozvoljava Oracle-u efikasnije korišćenje prostora na disku.



Slika 7 – Struktura Oracle skladišta (4)

5.3.1 Tablespace

Oracle tablespace se sastoji od jedne ili više datoteke. Nakon instalacije baze, postoje dva tablespace-a, a to su SYSTEM i SYSAUX. Od verzije 10g omogućeno je kreiranje specijalne vrste tablespace-ova zvanih “big file tablespace”, koje mogu veličine i do 128 terabajta, što omogućava administratoru baze upravljanje tablespace-om kao celinom, bez brige o veličini i strukturi datoteka u osnovi.

Koristeći Oracle Managed Files (OMF), upravljanje datotekama je postalo lakše, jer administrator definiše jednu ili više lokacija u sistemu datoteka gde će se smeštati datoteke, kontrolne datoteke ili logovi i Oracle će automatski imenovati i upravljati tim datotekama.

Ako je tablespace privremen, jedino segmenti sačuvani u tom tablespace su privremeni, jer je on sam po sebi permanentan. Oni se koriste za operacije sortiranja i za tabele koje postoje jedino za vreme trajanja sesije korisnika.

5.3.2 Blok

Blok baze podataka je najmanja jedinica skladištenja u bazi podataka. Veličina bloka je definisana brojem bajtova skladištenja u okviru datog tablespace-a baze podataka. Blok je obično veći od bloka operativnog sistema, kako bi se povećala efikasnost izlazno ulaznih operacija diska. Podrazumevana veličina bloka je definisana Oracle inicijalizacionim parametrom DB_BLOCK_SIZE. Mogu se definisati veličine blokava i za druge tablespace-ove u bazi, iako blokovi u SYSTEM, SYSAUX i u privremenim tablespace-ovima moraju biti veličine DB_BLOCK_SIZE. Podrazumevana veličina bloka je 8 kilobajta, a preporuka je iz prakse i raznih testiranja da se koristi ta veličina

bloka za sve tablespace-ove ukoliko ne postoji razlog da se koristi druga veličina. Jedan od razloga je može biti da je prosečna veličina sloga u tabeli 20 kilobajta, pa je u tom slučaju bolje koristiti blok dužine 32 kilobajta, što bi trebalo u potpunosti testirati kako bi se utvrdilo da li utiče na poboljšanje performansi.

5.3.3 Ekstent

Ekstent je sledeći nivo grupisanja u bazi podataka. Ekstent se sastoji od jednog ili više blokova baze podataka. Kada se uvećava objekat baze podataka, prostor dodeljen objektu je alociran u ekstentu.

5.3.4 Segment

Naredni nivo logičke strukture baze podataka je segment koji se sastoji od jednog ili više ekstenta. Oracle izdvaja više od jednog segmenta za tabelu, ako se radi o particionisanoj ili klasterovanoj tabeli.. Segmenti podataka uključuju LOB (veliki objekat) segmente koji čuvaju LOB podatke referencirane LOB lokatorom kolona u segmentu tabele.

Indeksni segment

Svaki indeks se čuva u sopstvenom indeksnom segmentu. Kao i kod particionisanih tabela, svaka particija, particionisanih indeksa je skladištena u svom segmentu.

Privremeni segment

Kada SQL upit zahteva prostor na disku da izvrši operaciju, kao što je operacija sortiranja koja se ne može uklopiti u memoriju, Oracle alocira privremeni segment. Privremeni segment egzistira tokom trajanja izvršavanja SQL upita.

Rollback segment

Od verzije 10g, nasleđenii Rollback segmenti postoje samo u SYSTEM tablespace, a obično administratnor nema potrebu da održava SYSTEM rollback segment. U prethodnim izdanjima Oracle, rollback segment je kreiran da sačuva prethodne vrednosti DML operacija u slučaju da je transakcija poništena (rollback), i da održi prethodnu sliku podataka kako bi se obezbedio konzistentan pregled podataka drugim korisnicima koji pristupaju tabeli. Ovi segmenti su se takođe koristili tokom

oporavka baze podataka za vraćanje nekomitovanih transakcija koje su bili aktivne kada je na primer baza neočekivano stala sa radom.

5.4 In-memory

Memorija transparentno ubrzava analitičke upite različitih razmera, što omogućava donošenje poslovnih odluka u realnom vremenu. Koristeći In-Memory opciju skladištenja podataka, korisnici mogu kad gde žele pokretati analitiku i izveštavanje, za šta čije izvršavanje im je trebalo nekoliko sati, pa čak i dana. Preduzeća imaju koristi od boljih odluka u realnom vremenu, što rezultira niže troškove, veću produktivnost i povećanje konkurentnosti.

Oracle in-memory ubrzava kako skladišta podataka, tako i opterećenje OLTP baze podataka, i lako se primenjuje na svaku aplikaciju koja je kompatibilna sa Oracle 12c bazom podataka, za šta nije potrebna bilo kakva izmena u aplikaciji. In-memory koristi Oracle-ove razvijene tehnologije, kao ušto su scale-in, scale-out i rangiranje skladišta, za efikasnije pokretanje i izvršavanje bilo kakvog zadatka. Oracle je vodeći na tržištu sa karakteristikama poput dostupnosti i bezbednosti svega vezano za In-memory, što čini najjačom ponudom na tržištu.

Sposobnost da se analiza podataka lako obavlja u realnom vremenu zajedno sa obradom svih transakcija na postojećim aplikacijama omogućava organizacijama da se transformišu u okruženja u realnom vremenu, koje donose brzo odluke, odgovaraju odmah na zahteve kupaca i stalno optimizuju ključne procese.

Današnja informaciona arhitektura je mnogo više dinamičnija nego pre nekoliko godina. Poslovni korisnici sada zahtevaju brže dodatne informacije vezane za odlučivanje. Kako bi se održao korak sa porastom potražnje, kompanije su primorane da pokreću analitiku na svojim operativnim sistemima, pored svojih skladišta podataka. Ovo dovodi do nesigurnog balansiranja između transakcionih opterećenja, čestih DML operacija i upita vezanih za izveštavanje koji treba da skeniraju velike količine podataka.

Uvođenjem Oracle In-memory opcije, jedna baza podataka može efikasno podržati različita opterećenja, pružajući optimalne performanse za transakcije dok istovremeno podržava izveštavanje i analitiku u realnom vremenu. To je moguće zahvaljujući jedinstvenom "dvostrukom" arhitekturom koja omogućava podacima da

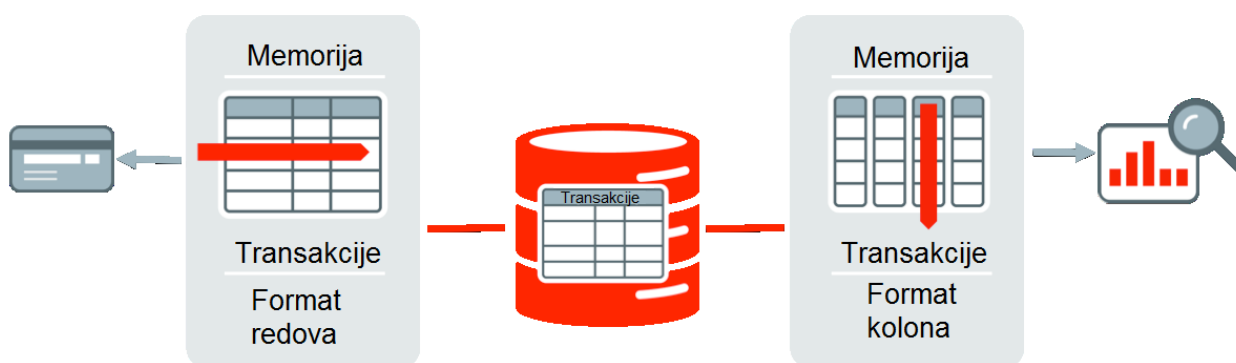
budu podržama u postojećem Oracle formatu skladištenja u redovima, za OLTP operacije, i novim formatu skladištenja u kolonama, optimizovan za analitičku obradu. Interna memorija takođe omogućava skladištima podataka da obezbedi višu analitiku na zahtev, što krajnjim korisnicima omogućava da postave izvršavanje višestrukih upita za isto vreme za koje se izvršava jedan upit.

Ugrađivanje In-memory formata skladištenja u kolonama u postojeći Oracle softver proverava da li je potpuno komatibilan sa svim postojećim funkcijama, a ne zahteva promene na aplikativnom sloju. Kompanije koje teže poslovanju u realnom vremenu mogu lakše ostvariti ciljeve, bez obzira koje aplikacije koriste.

5.4.1 Format reda i kolona

Oracle tradicionalno za skladištenje podataka koristi format reda. U formatu reda, svaka nova transakcija ili rekord sačuvan u bazi podataka, je predstavljen kao novi red u tabeli. Taj red je sastavljen od više kolona, gde svaka kolona predstavlja različiti atribut tog rekorda. Format reda je idealan za online transakcione sisteme, jer omogućava brz pristup svim kolonama u rekordu, jer se svi podaci za dati rekord čuvaju zajedno u memoriji i u skladištu. Format kolona je idealan za analitiku, jer omogućava brže pronalaženje podataka kada je izabrano samo nekoliko kolona koje su selektovane, kada upit pristupa ogromnoj količini setova podataka. Što se tiče DML operacija (insert, update ili delete), koje se javlja u oba formata, format reda je neverovatno efikasan za obradu DML operacija jer manipuliše celim rekordom u jednoj operaciji, tj. kada se dodaje novi red, ažurira ili brše postojeći. Format kolona nije tako efikasan sa obradom DML operacija. Da bi dodali ili obrisali jedan rekord u formatu kolona sve strukture u kolonama bi se morale promeniti. Sve do sada, korisnici su bili primorani da izaberu jedan format i da biraju između optimizacije OLTP ili optimizacije analitičkih performansi. Oracle In-memory pruža najbolje od oba, tako što dozvoljava podacima da budu uskladišteni u In-memory formatu reda (keš baferu) i novom In-memory formatu kolona. Za ovakvu dualnu arhitekturu nisu potrebni dupli memorijski zahtevi. Format kolona u memoriji treba projektovati tako da se priloagode objekti koji moraju biti sačuvani u memoriji, a keš bafer optimizovan za decenije efikasnog rada sa mnogo manjom veličinom nego što je baza podataka. U praksi se očekuje da će dualna arhitektura potrebovati najviše 20% više od ukupne

potrebe za memorijom, što predstavlja malu cenu plaćanja za optimalne performanse zauvek.



Slika 8 – Prikaz formata reda i formata kolona (6)

Uz Oracle jedinstveni pristup, ostaje jedna tabela u skladištu, tako da ne postoje dodatni troškovi skladištenja ili sinhronizacije. Baza podataka održava punu konzistentnost transakcija između formata reda i formata kolona, tako što održava konzistentnost između tabela i indeksa. Oracle Optimizer u potpunosti podržava format kolona, tako što automatski rutira analitičke upite ka formatu kolona i OLTP operacijama u formatu reda, osigurava najbolje performanse i kompletnu doslednost podataka za sve zahteve bez promene aplikacije.

5.4.2 In-memory skladištenje u kolonama

In-memory baza podataka koristi In-memory skladištenje u kolonama, što predstavlja novu komponentu System Global Area (SGA), zvana In-memory oblast. Podaci u IM skladištu u kolonama se ne smeštaju u tradicionalan format redova koji koristi Oracle baza podataka. Umesto toga, koristi se novi format kolona. IM skladište ne zamenjuje keš bafer, ali deluje kao dodatak, tako da podaci mogu sada da se čuvaju u memoriji i u formatu redova i u formatu kolona. In-memory oblast predstavlja statički deo u SGA, čija veličina je kontrolisana od strane inicijalizacionog parametra INMEMORY_SIZE, koji je standardno 0, ukoliko mu se ne dodeli vrednost. Trenutna veličina In-memory oblasti je vidljiva u V\$SGA. Pošto je statički, bilo koja promena parametra INMEMORY_SIZE neće stupiti na snagu dok se baza podataka ne restartuje. Takođe, ne može se uticati ili kontrolisati automatskim upravljanju memorijom (AMM). In-memory oblast mora imati minimalnu veličinu od 100 megabajta. IM oblast je podeljena u dva pool-a: pool od jednog megabajta koji se

koristi za skladištenje kolona formatiranih podataka uskladištenih u memortiju, dok se drugi od 64 kilobajta koristi za čuvanje metapodataka o objektima koji se čuvaju u IM skladištu. Količina slobodne memorije u svakom pool-u je vidljiva u pogledu V\$INMEMORY_AREA. Relativna veličina ova dva poola je određena unutrašnjom heuristikom, većini In-memory oblasti memorije je alocirano pool od 1MB. (6)

```
select pool, alloc_bytes, used_bytes, populate_status, con_id
from V$INMEMORY_AREA;
```

	POOL	ALLOC_BYTES	USED_BYTES	POPULATE_STATUS	CON_ID
1	1MB POOL	1,710,227,456.00	91,226,112.00	DONE	3.00
2	64KB POOL	419,430,400.00	3,342,336.00	DONE	3.00

Tabela 1 – prikaz iskorišćenosti memorije

Punjenje In-memory

Za razliku od čiste In-memory baze, ne trebaju svi objekti iz Oracle baze da se nalaze u In-memory skladištu. IM skladište bi trebalo biti popunjeno sa podacima najkritičnijim po performanse baze, dok manje kritični podaci mogu da se nalaze disku. Naravno, ukoliko je baza podataka dovoljno mala, IM se može popuniti svim tabelama iz baze. In-memory baza podataka dodaje novi INMEMORY atribut za tabele i materijalizovane pogleda (view). Jedino objekti sa atributom INMEMORY se mogu nalaziti u IM skladištu. INMEMORY atribut se može dodeliti na skupove tabela (tablespace), tabele, (pod)particije ili materijalizovane pogleda. Ako je atribut INMEMORY dodeljen na tablespace, onda sve nove tabele i materijalizovani pogledi u njemu će biti definisani u in-memory skladištu, takođe će i sve kolone biti IM.

```
ALTER TABLESPACE TS_DATA DEFAULT INMEMORY;
```

Međutim, moguće je izabrati samo podskup kolona koje će se povući u memoriju. Na primer, sledeća naredba postavlja atribut In-memory na tabelu TRANSAKCIJE u šemi SYSTEM, ali isključuje kolone OPIS, OPIS2 i IZNOS.

```
ALTER TABLE TRANSAKCIJE INMEMORY NO INMEMORY(IZNOS, OPIS,
OPIS1);
```

Slično i za particionisane tabele, gde sve particionisane tabele nasleđuju In-memory atribut, ali je moguće popuniti samo podskup particije. Za označavanje objekta da više nije kandidat i da se odmah isprazni iz memorije koristi se sledeća naredba.

```
ALTER TABLE TRANSAKCIJE_P1 NO INMEMORY;
```

IM skladište je pounjeno nizom pozadinskih procesa koji se nazivaju radni procesi, što znači da će baza podataka biti potpuno aktivna i dostupna kad se svi procesi završe. Sa “čistom” In-memory bazom, datoj bazi podataka se ne može pristupiti, sve dok se podaci nalaze u memoriji, što predstavlja ozbiljne probleme dostupnosti i integriteta podataka. Svaki radni poces dobija podskup blokova baze podataka kako bi se popunilo In-memory skladište, tj. simultano se podaci ubacuju u kolone i kompresuju. Kao što je skup tabela na disku sačinjen na višem nivo, tako je IM skladište sačinjeno od IM kompresionih jedinica (IMCU). Svaki radni process alokira svoju IMC jedinicu i popunjava podskup blokova baze podataka u nju. Podaci se ne sortiraju prilikom punjenja, oni se čitaju istim redosledom kao i u format reda. Objekti su uskladišteni u IM po listi prioriteta, nakon otvaranja baze ili nakon izvršavanja prvog upita. Redosled kojim se objekti skladište, kontorliše se pomoću ključne reči PRIORITY, koja ima pet nivoa.

Prioritet	Opis
CRITICAL	Objekat se povlači u memoriju odmah nakon otvaranja baze podataka
HIGH	Objekat se popunjava u memoriju nakon što se popune svi CRITICAL objekti, ako ostane prostora na raspoloaganju
MEDIUM	Objekat se popunjava nakon svih CRITICAL i HIGH objekata i ukoliko ima prostora u memoriji
LOW	Objekat se popunjava nakon svih CRITICAL, HIGH i MEDIUM objekata i ukoliko ima prostora u memoriji
NONE	Objekti se popunjavaju tek nakon izvršavanja upita nad celom tabelom prvi put i ako ima prostora u memoriji. Ovaj prioritet je podrazumevani.

Tabela 2 – Nivoi prioriteta povlačenja tabela u memoriju

Svi objekti kojima su dodeljeni prioriteti bi morali biti popunjeni pre nego što počne punjenje objekata sa nižim prioritetom. Međutim, redosled punjenja može biti

zamenjen ako se objekat bez prioriteta skenira, aktivira se momentalno skladištenje istog u memoriju.

```
ALTER TABLE TRANSAKCIJE INMEMORY PRIORITY CRITICAL;
```

Ograničenja

Skoro svi objekti u bazi mogu biti uskladišteni u memoriju, ali postoje neki izuzeci.

Objekti koji da se popune u IM format su:

- Bilo koji objekti SYS korisnika i tabele iz SYSTEM ili SYSAUX skupa tabela
- Indeksno organizovane tabele(IOT)
- Grupisane tabele

Sve druge kolone koje sadrže ove tipove podataka se mogu skladištiti u IM. Bilo koji upit koji koristi pristup kolonama koje su u IM ima benefit da pristupa podacima preko skladišta u format kolona. Upiti koji zahtevaju podatke koje nisu podržane u format kolona biće izvršeni preko bafer keša. Objekti koji su manji od 64kb nisu uskladišteni u memoriju, jer mogu da iskoriste značajan prostor unutar IM, gde im je alocirano po 1mb.

5.4.3 In-memory kompresija

Obično se pod pojmom kompresija smatra mehanizam koji štedi prostor. Međutim, podaci uskladišteni u IM se kompresuju pomoću novog seta algoritama za kompresiju koji ne samo da štedi prostor, već i podoljšava performanse upita. Novi Oracle In-memory format kompresije omogućava upitima da se izvršavaju direktno nad kompresovanim kolonama. Ovo znači da će se sve operacije skeniranja i filtriranja izvršiti nad mnogo manjoj količini podataka. Podaci se dekompresuju samo kada je to potrebno za rezultat upita. Za IM kompresiju je definisana ključna reč MEMCOMPRESS, a predstavlja deo atribuda INMEMORY. Postoji šest nivoa, svaki pruža različiti nivo kompresije i performansi.

Nivo komresije	Opis
NO MEMCOMPRESS	Podaci su sačuvani bez korišćenja kompresije

MEMCOMPRESS DML	FOR	Minimalna kompresija optimizovana za DML operacije
MEMCOMPRESS QUERY LOW	FOR	Optimizacija za performanse izvršavanja upita (uobičajno)
MEMCOMPRESS QUERY HIGH	FOR	Optimizacija za performanse upita, kao i uštede memorijskog prostora
MEMCOMPRESS CAPACITY LOW	FOR	Balansirana kompresija, sa akcentom na uštedi memorijskog prostora
MEMCOMPRESS CAPACITY	FOR	Optimizovano za uštedu prostora

Tabela 3 – Načini kompresije podataka

Uobičajno, podaci se kompresuju pomoću opcije FOR QUERY LOW, koja pruža najbolje performanse za upite. Ova opcija koristi postojeće tehnike kao što su Dictionary Encoding, Run Length Encoding i Bit-Packing. Opcija FOR CAPACITY primenjuje dodatne tehnike kompresije, koje akcenat stavljaju na to da svaki novi unos mora biti dekompresovan pre nego što se primeni WHERE uslov. FOR CAPACITY LOW opcija primenjuje tehniku kompresovanja OZIP koja pruža ekstremno brzu dekompresiju prilagođenu za ORACLE bazu podataka. Opcija FOR CAPACITY HIGH koristi jak algoritam za kompresiju sa težom dekompresijom, sve sa ciljem da se pruži veća kompresija. Odnosi kompresije variraju od 2 puta do 20, u zavisnosti od izabrane opcije, tipa podatka i sadržaja tabele. Tehnika kompresije može da varira u zavisnosti od kolone, ili particije u okviru jedne tabele. Na primer, mogu se optimizovati neke kolone za brzo skeniranje, a neke za čuvanje memorije.

```
CREATE TABLE TRANSAKCIJE
(
  ID          NUMBER NOT NULL,
  PARTIJA    VARCHAR2(20),
  KONTO      VARCHAR2(40),
  TIP        VARCHAR2(2),
  VALUTA     VARCHAR2(3),
  IZNOS      NUMBER(20,2),
  EKVIVALENT NUMBER(20,2),
  DATUM      DATE,
  OPIS       VARCHAR2(4000),
```



```

OPIS2          VARCHAR2(4000)
)
INMEMORY MEMCOMPRESS FOR QUERY
NO INMEMORY(OPIS2)
INMEMORY MEMCOMPRESS FOR CAPCITY HIGH(OPIS2);

```

In-memory skeniranje

Analitički upiti tipično referenciraju samo mali podskup kolona u tabeli. Oracle IM pristupa samo kolonama potrebnim za upit, i primenjuje bilo gde WHERE klauzulu u tim kolonama direktno bez potrebe da se prvo dekomprpesuju. Ovo značajno smanjuje količinu podataka kojoj treba da se pristupi i da se obradi.

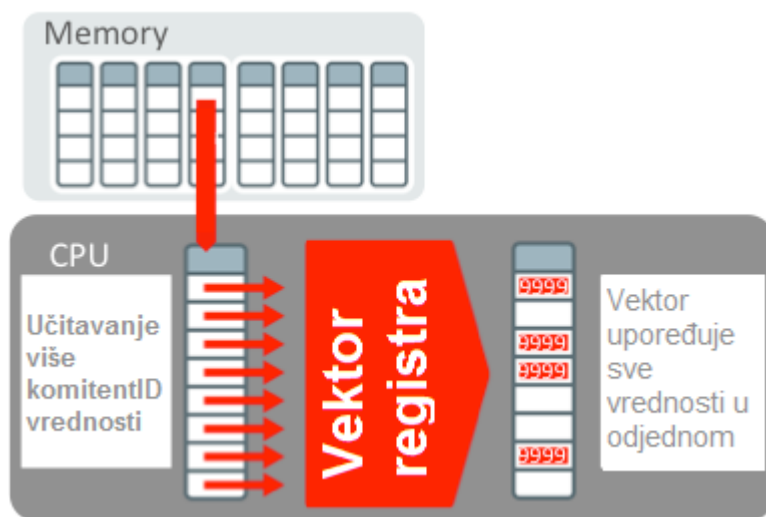
In-memory indeksi

Dalje smanjenje količine podataka kojoj se pristupa moguće je pomoću IM indeksima skladišta koji su automatski kreirani i čuvaju se za svaku kolonu u IM 38ector kolona. Uskladišteni indeksi dozvoljavaju „potkresivanje“ podataka bazirano na filterima u okviru SQL upita. IM indeksi čuvaju minimalne i maksimalne vrednosti za svaku kolonu u IMCU. Kada se definiše WHERE uslov, indeks na referenciranoj koloni je zadužen da ispita da li postoje specifikovane vrednosti u svakom IMCU upoređivanjem definisanih rednosti sa minimalnim i maksimalnim u okviru jednog IMCU. U slučaju da definisana vrednos ne ulazi u okvir između minimalne i maksimalne u okviru IMCU, tada se izbegava pretraga cele IMC jedinice. Za jednakost, u listi, i neki opseg iskaza dodatni nivo isključivanja je moguć pomoću rečinka metapodataka kreiranog za svaki IMCU. Rečnik metapodataka sadrži listu različitih vrednosti za svaku kolonu u IMCU. Tako, pomoću ovih rečnika Oracle baza utvrđuje da li se tražena vrednost u upitu nalazi u okviru datog IMCU, osiguravajući da su samo neophodne IMC jedinice skenirane.

SIMD 38ector obrade

Za podatke koji treba da se skeniraju u IM, baza podataka koristi SIMD (Single Instruction Processing Multiple Data Values) 38ector obrade. Umesto da obrađuje svaki unos u koloni, jedan po jedan, SIMD 38ector omogućava da se skup vrednosti kolone obradi zajedno u jednoj CPU obradi. Format kolona koji se koristi u IM skladište je posebno dizajnirano da poveća broj unosa kolona koji će biti učitane u

39ector registra na CPU i obrađeni u jednom instrukcijskom ciklusu CPU. SIMD 39ector obrade omogućava Oracle IM da skenira milijardu redova u sekundi. Na primeru tabele TRANSAKCIJE, gde je potrebno izračunati stanje za određenu partiju, i pri tom tabela je u potpunosti učitana u memortiju. Upit počinje da skenira samo kolonu PARTIJA, tabele transakcija. Prvih 8 vrednosti iz kolone PARTIJA su učitane u SIMD registar na procesoru i upoređene sa traženom partijom u jendom ciklusu CPU. Broj slogova koji se poklapaju je sačuvan, pa se slogovi uklanjaju i puni se narednih 8 slogova za upoređivanje, sve dok se ne nađu tražene vrednosti iz kolone PARTIJA.



Slika 9 – Vektor obrade (6)

In-memory spajanje tabela

SQL upiti koji sadrže spajanje 39ect tabela mogu biti izvršeni veoma efikasno sa IM jer se mogu iskoristiti prednosti BLOOM filtera. BLOOM filter transformiše JOIN u filter koji se primenjuje na deo pregleda velike tabele. Ovi filteri su uvedeni u Oracle 10g bazi za poboljšanje performansi spajanja tabela i nisu specifični samo za IM. Međutim oni su veoma efikasno primenjeni na format kolona preko SIMD vektorske obrade. Kada se dve tabele spajaju preko HASH JOIN-a, prva tabela (obično manja) se skenira i slogovi koji zadovoljavaju WHERE uslov se koriste za kreiranje IM HASH tabela. Tokom kreiranja ove tabele, bit 39ector ili BLOOM filter se takođe kreira nad kolonom preko koje se vrši spajanje. Bit vektor se šalje kao dodatni uslov za skeniranje druge tabele. Nakon WHERE uslova koji je primenjen na drugu tabelu, rezultujući slogovi će imati heširanu kolonu preko koje se spajaju i biće upoređeni sa

vrednostima u bit vektoru. Ako je pronađena tražena vrednost bit vektora, slog će biti poslat na spajanje, u suprotnom slog će biti ignorisan.

Lako je prepoznati BLOOM filter u planu izvršenja. On će se pojaviti na dva mesta, u vreme stvaranja i tokom primene.

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		32	2912	15053	00:00:01
1	HASH GROUP BY		32	2912	15053	00:00:01
* 2	HASH JOIN		32	2912	15052	00:00:01
3	JOIN FILTER CREATE	:BF0000	4	172	291	00:00:01
* 4	HASH JOIN		4	172	291	00:00:01
* 5	TABLE ACCESS FULL	KOMITENTI	1	19	74	00:00:01
6	TABLE ACCESS FULL	RACUNI	150152	3603648	216	00:00:01
7	JOIN FILTER USE	:BF0000	1203480	57767040	14759	00:00:01
* 8	TABLE ACCESS INMEMORY FULL	TRANSAKCIJE	1203480	57767040	14759	00:00:01

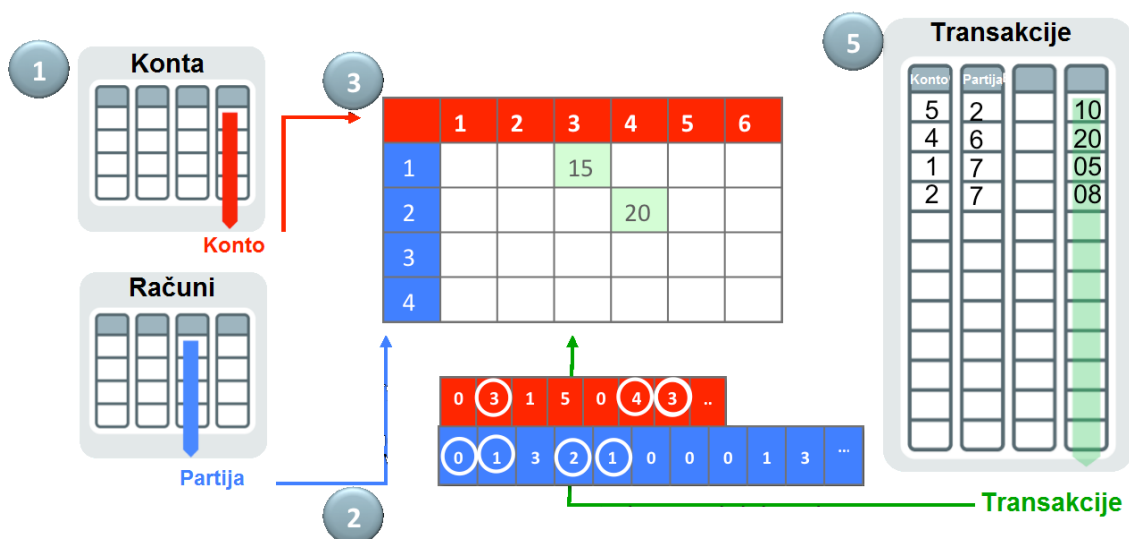
Tabela 4 – Plan izvršenja upita sa BLOOM filterom

In-memory agregacija

Analitički upiti često zahtevaju više od jednostavnih filtera i spajanja, što može biti često i kompleksne agregacije ili razna izvođenja sumarnih vrednosti. Novi alat za optimizaciju transformaciju, zvani "Vektor Group By", je uveden u Oracle 12.1.0.2 bazu podataka, kako bi osigurao procesiranje mnogo kompleksnijih analitičkih upita koristeći CPU algoritme za efikasnost.

VGB transformacija je proces koji se sastoji iz dva dela. Ukoliko je potrebno odrediti stanje po partijama za određenog komitenta, prva faza predstavlja:

1. Upit prvo skenira manje tabele, u ovom primeru to su KONTA i RAČUNI. Nova struktura podataka zvana Key Vector je kreirana na osnovu rezultata ovog skeniranja. Key vektor je sličan BLOOM filteru jer omogućava da se pridruže dodatni uslovi tokom kreniranja tabele TRANSAKCIJE (najveća tabela).
2. Key vektori se takođe koriste za stvaranje dodatnih struktura koje se nazivaju IM akumulaturi. IM akumulator je višedimenzionalni niz izgrađen u PGA koji omogućava ORACLE bazi da izvrši agregaciju ili GROUP BY tokom skeniranja tabele TRANSAKCIJE umesto da to radi kasnije.
3. Na kraju, privremene tabele su kreirane u da čuvaju kolone definisane u SELECT-u iz tabela manjih dimenzija.



Slika 10 – In-memory agregacija

Druga faza obuhvata:

4. Drugi deo plana izvršavanja počinje sa skeniranjem tabele TRANSAKCIJE i primenom Key vektora. Za svaki slog u tabeli TRANSAKCIJE koji se poklaa sa JOIN uslovom (da li je KONTO ili PARTIJA), odgovarajući iznos transakcije će biti dodat u IM akumulator. Ako vrednost već postoji u toj ćeliji, dve vrednosti će biti sabrane i dobijena vrednost će biti smeštena u ćeliju.
5. Na kraju, rezultati skeniranja najveće tabele se spajaju sa pricremenim tabelama kreiranim kao deo skeniranja dimenzija tabela. (6,8)

Kombinacija ove dve faze dramatično poboljšava efikasnost višestrukog spajanja tabela sa kompleksnim agregacijama.

DML i In-memory

Jasno je da se korišćenjem IM sladištenja trastično mogu poboljšati performanse svih tipova upita, ali je vrlo malo baza podataka iz kojih se samo čitaju podaci. Da bi IM skladište bilo zaista efikasno u savremenim uslovima, mora biti u stanju da obradi kako ogromne količine podatka, tako i online transakcije.

Partition Exchange Loads

Preporučuje se da veće tabele u skladištu budu particionisane. Jedan od benefita particionisanja je mogućnost za učitavanje podataka brzo i lako samiminalnim uticajem na korisnike

In-memory skladištenje u kolonama na RAC (Real Application Clusters)

Svaka jedinica u RAC okruženju ima svoje IM skladište. Preporučuje se da na svakom čvoru IM skladište bude jednake veličine, a na RAC jedinica koja ne zahteva IM treba postaviti parametar INMEMORY_SIZE na 0. Podrazumevano je da će svi objekti uskladišteni u IM biti distribuirani preko IM skladišta u klastere, a kontroliše se pomoću podklauzulama DISTRIBUTE I DUPLICATE atributa INMEMORY. U RAC okruženju, ovjekat kojim je definisan jedino INMEMORY atribut će biti distribuiran preko svih IM skladišta u klastere, što čini efikasnijim IM skladište u ovom okruženju. Kako su objekti podrazumevano, koji se distribuiraju preko klastera, kontrolišu sa parametrom DISTRIBUTE, ORACLE je izabrao najbolji način da distribuira objekte preko klastera bez obzira na tip particionisanja. Kao alternativa, može se odrediti DISTRIBUTE BY ROWID RANGE da se distribuira do opsega rowid, DISTRIBUTE BY PARTITION da se distribuiraju particije različitih jedinica ili DISTRIBUTE BY SUBPARTITION za distribuiranje podparticija različitih jedinica.

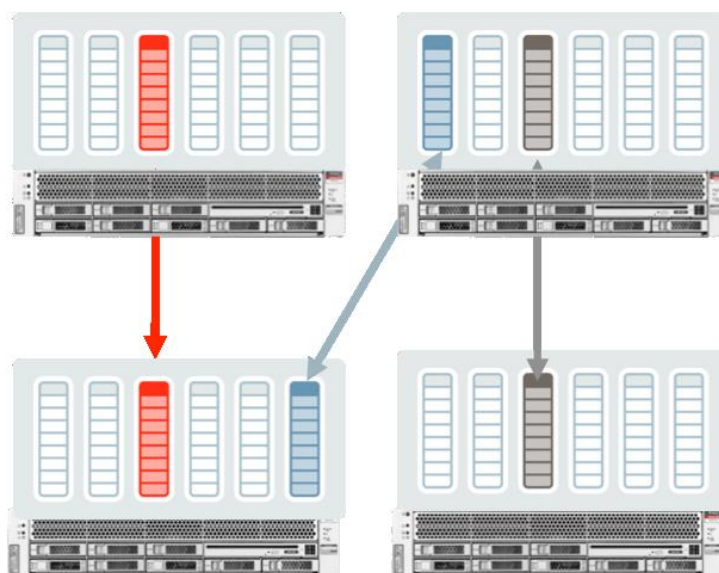
```
ALTER TABLE TRANSAKCIJE INMEMORY DISTRIBUTE BY ROWID RANGE;
```

DISTRIBUTE BY PARTITION ili SUBPARTITION se preporučuje ako su tabele particionisanje HASH i PARTITION-WISE planom. Ovo će omogućiti svakoj particiji da se pridruži raspoređenim u okviru jedne jedinice. DISTRIBUTE BY ROWID RANGE se može koristiti za neparticionisane tabele ili za particionisane gde bi DISTRIBUTE BY PARTITION dovelo do „mimolijaženja“ podataka. Ako je objekat veoma mali (sastoji se od samo jedne IMCU), biće popunjen u IM skladištu na samo jednoj jedinici u klasteru.

Nakon popunjavanja IM skladišta na RAC okruženju pripadaju grupi određenih RAC jedinica, paralelni server obrade mora izvršavati upite na svakoj RAC jedinici nad objektima koji se u njoj nalaze. Koordinator upita objedinjuje rezultate svi paralelnih servera obrade zajedno pre vraćanja istih krajnjem korisniku.

IM greška tolerancije

S obzirom na arhitekturu IM skladišta na RAC okruženju, neke osetljive performanse aplikacije mogu zahtevati rešenje za grešku tolerancije. Korišćenjem ključne reči **DUPLICATE** pored atributa **INMEMORY** vrši se kreiranja duplih podataka u IM skladištu. Ovo znači da svaki uskladišteni će svaki IMCU u IM imati identičnu kopiju lociranu u nekom drugom RSC klasteru. Dupliranje IMC jedinica pruža IM grešku tolerancije takvu da sa se podacima može pristupiti iako neka od jedinica prestane sa radom. Takođe, unapređuju se performanse, tako da upiti mogu uvek koristiti kako primarnu tako i kopiju IMCU bilo kada.



Slika 11 - IMCU dupliranje na drugu jedinicu RAC klastera (7)

Ukoliko RAC jedinica prestane sa radom neko vreme, jedina način da se pristupi podacima je kreiranje kopije primarnog IMCU lociranog na jedinici. Samo ako druga jedinica prestane sa radom na neko vreme, podaci se moraju preraspodeliti. Ukoliko je potrebna dodatna tolerancija greške, moguće je popuniti u objekat u IM skladištu na svakoj jedinici u klasteru definisanjem **DUPLICATE ALL** uz **INMEMORY** atribut. Ovo će obezbediti najviši nivo redundanse i pružiće linearnu skalabilnost, jer će upiti biti u mogućnosti da u se u potpunosti izvrše u okviru jedne jedinice.

```
ALTERTABLE TRANSAKCIJE INMEMORY DUPLICATE ALL;
```

DUPLICATE ALL opcija se može koristiti takođe da se rasporedi spajanje između velikih tabela i tabela manjih dimenzija. Definisanjem DUPLICATE ALL opcije na manjim tabelama potpuna kopija će biti uskladištena u IM skladištu na svakoj jedinici.

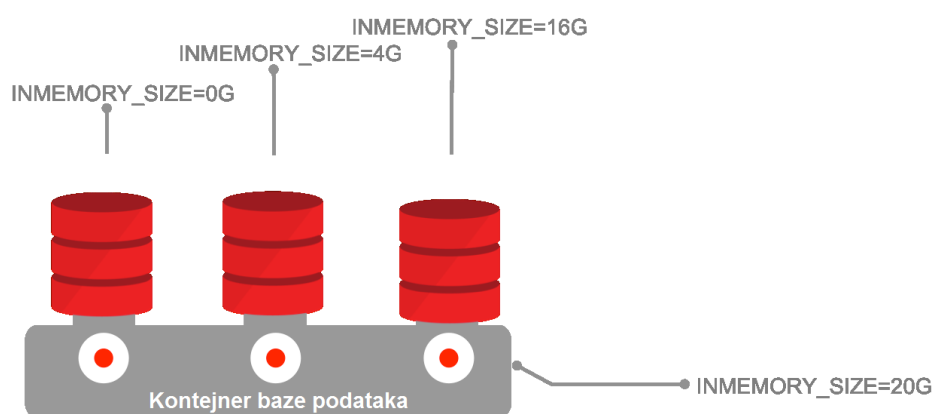
DUPLICATE klauzula se primenjuje samo na ORACLE Engineered sistemu i biće ignorisana bilo gde drugo. Ako RAC jedinica stane sa radom ne Engineered sistemu, podaci popunjeni u IM skladište u toj jedinici neće biti više dostupni IM na tom klasteru. Upiti nad delovima nedostajućih objekata neće propasti. Umesto toga, upit će pristupiti podacima iz bafer keša ili skladišta, što će uticati na performanse izvršenja ovih upita. Ukoliko se RAC ne pokrene neko vreme, objekte ili delove objekata koji se nalaze u IM na toj jedinici, biće uskladišteni na ostalim jedinicama u klasteru (pod pretpostavkom da je ostalo prostora). Da bi se smanjio uticaj na performanse zbog oborenih RAC jedinica, preporučuje se da se ostavi neki prostor u IM na svakoj jedinici u klasteru. Važno je znati da podaci nisu odmah raspoređeni drugim jedinicama klastera nakon pada neke jedinice zato što je velika verovatnoća da će se jedinica brzo pokrenuti. Ako su podaci odmah redistribuirani, proces preraspodele bi dodatno opteretio sistem, koji će se brzo poništiti nakon vraćanja jedinice u službu, pa će samim tim sistem čekati nekoliko desetina minuta pre pokretanja preraspodele podataka. Čekanje omogućava jedinici da se ponovo pridruži klasteru. Distribucija se vrši na osnovi IMCU i objekti su u potpunosti dostupni tokom procesa

IM skladištenje u višeklijentskom okruženju

Oracle Multitenant je novi model konsolidacije baze podataka u kojoj se više PLUGGABLE baza podataka (PDB) konsoliduje u kontejneru baze (CDB). Zadržavajući mnoge aspekte izolacije jedne baze podataka, to dozvoljava PDB da deli sistemski globalni prostor (SGA) i pozadinske procese zajedničkog CDB, tako da sve PDB dele jedno IM skladište u kolonama.

Ukupna površina IM skladišta se definiše postavljanjem vrednosti parametra INMEMORY_SIZE u CDB. Svaka PDB precizira koliko zajedničkog IM prostora može koristiti za postavljanje INMEMORY_SIZE parametra. Ne moraju sve PDB u kontejneru baze podataka koristiti IM skladište, što znači da se na nekima

INMEMORY_SIZE može postaviti na 0 kako ne bi koristile IM. To nije neophodno jer zbir veličine INMEMORY_SIZE svih PDB mora biti manji ili jednog datom parametru na nivou kontejnera. Moguće je da su sve PDB “pretplaćene” na IM skladište, jer se time osigurava da se prostor ne rasipa ukoliko se jedna baya isključi. Pošto je parametar INMEMORY_SIZE statički (zahteva da se baza restartuje kako bi se promene primenile), bolje je dozvoliti PDB da pređu granicu IM skladišta, kako bi sav prostor u IM bio iskorišćen. Međutim moguće je da je jednoj bazi nedostaje memorija koju koristi druga baza zbog veće definisane veličine. Preporučuje se da ako se ne očekuje gašenje baze duži period da se ne prelazi ukupna definisana veličina IM na celom kontejneru.



Slika 12 – Kontejner baze podatka (8)

Svaka PDB je u potpunosti Oracle baza podataka, pa svaka ima listu prioriteta. Kada PDB pokrene objekte na svojoj listi prioriteta, oni će biti popunjeni u IM skladište, pod pretpostavkom da ima dovoljno prostora.

5.4.4 Kontrola upotrebe Oracle IM

Postoji nekoliko inicijalizacionih parametara koji omogućavaju kontrolu kada i kako će se koristiti IM skladište, a postoje oni parametri koji predstavljaju jezgro i koji su opciono. Parametri inicijalizacije jezgra sa INMEMORY prefiksom su uvedeni da direktno kontrolišu različite aspekte funkcionalnost u memoriji, a tu je i novi parametar optimizacije koji može da utiče na to da li upiti koriste IM ili ne.

NAME	TYPE	VALUE

inmemory_clause_default	string	
inmemory_force	string	DEFAULT
inmemory_max_populate_servers	integer	2
inmemory_query	string	ENABLE
inmemory_size	big integer	2G
inmemory_trickle_repopulate_servers_percent	integer	1
optimizer_inmemory_aware	boolean	TRUE

Tabela 5 – Upotreba In-memory

INMEMORY_SIZE

Ovaj parametar kontroliše memoriju dodeljenu IM skladištu. Podrazumevana veličina je 0 bajtova. Ovaj parametar se menja jedino na nivou sistema i zahteva ponovno pokretanje baze podataka, kako bi se izmene primenile. Minimalna veličina zahtevana za INMEMORY_SIZE je 100MB

INMEMORY_QUERY

Postavljanje ovog parametra na DISABLE kako na nivou sesije ili sistema se onemogućuje korišćenje IM skladišta u potpunosti. To će onemogućiti izvršavanje upita pomoću IM. Podrazumevana vrednost parametra je ENABLE.

INMEMORY_MAX_POPULATE_SERVERS

Maksimalan broj procesa koji može da se pokrene je pod kontrolom ovog parametra, koji je postavljen podrazumevano na 0,5 X CPU_COUNT. Smanjenje broja radnih procesa će smanjiti trošenje CPU resursa tokom uskladištavanja, ali će verovatno proširiti potrebno vreme da se uradi čuvanje podataka u IM skladište.

Dodatni inicijalizacioni parametri

INMEMORY_CLAUSE_DEFAULT

Ovaj parametar omogućava određivanje podrazumevanog režima za In-memory tabele novodeći vežeće setove vrednosti za sve INMEMORY pod parametre koji nisu

eksplicitno navedeni u sintaksi. Podrazumevana vrednost je prazan string, što znači da se samo eksplicitno navedene tabele skladište u IM.

```
ALTER SYSTEM SET inmemory_clause_default= 'INMEMORY PRIORITY LOW'
```

Vrednost parametra je prevedena na isti način kao INMEMORY klauzula, sa istim podrazumevanim ako jedna od pod klauzula nije definisane. Bilo da je eksplicitno definisana tabela za In-memory, naslediće neodređene vrednosti ovog parametra.

INMEMORY_TRICKLE_REPOPULATE_SERVERS_PERCENT

Parametar koji kontroliše procenat vremena koje radni proces mogu obavljati repopulacijom. Vrednost ovog parametra je procenat parametra INMEMORY_MAX_POPULATE_SERVERS. Postavljanje ovog parametra na '0' onemogućava repopulaciju, a podrazumevano 1 znači da će radni procesi provesti jedan procenat vremena na repopulaciju.

INMEMORY_FORCE

Podrazumevano svaki objekat sa navedenim INMEMORY atributom je kandidat da bude deo IM skladišta. Međutim ako je INMEMORY_FORCE postavljen na OFF, onda čak i ako je podešen IM prostor, nijenda tabela neće biti u memoriji. Podrazumevana vrednost je DEFAULT.

OPTIMIZER_INMEMORY_AWARE

Moguće je isključiti sva IM poboljšanja kreirana za poboljšanje modela postavljanjem OPTIMIZER_INMEMORY_AWARE parametra na FALSE. Čak iako je optimizacija IM poboljšanja isključena, i dalje se može dobiti InMemory plan.

Praćenje i upravljanje Oracle IM bazom podataka

Monitoring objekata koji se nalaze IM skladištu se obavlja pomoću dva nova pogleda, v\$IM_SEGMENTS i v\$IM_USER_SEGMENTS, gde se mogu videti objekti koji su uskladišteni u IM.

SQL>desc v\$IM_SEGMENTS

Name	Null?	Type
OWNER		VARCHAR2 (128)
SEGMENT_NAME		VARCHAR2 (128)
PARTITION_NAME		VARCHAR2 (128)
SEGMENT_TYPE		VARCHAR2 (18)
TABLESPACE_NAME		VARCHAR2 (30)
INMEMORY_SIZE		NUMBER
BYTES		NUMBER
BYTES_NOT_POPULATED		NUMBER
POPULATE_STATUS		VARCHAR2 (9)
INMEMORY_PRIORITY		VARCHAR2 (8)
INMEMORY_DISTRIBUTE		VARCHAR2 (15)
INMEMORY_DUPLICATE		VARCHAR2 (13)
INMEMORY_COMPRESSION		VARCHAR2 (17)
CON_ID		NUMBER

Tabela 6 – Sadržaj pogleda v\$IM_SEGMENTS

Ovi pogledi ne prikazuju samo objekte koji se nalaze u IM skladištu, oni tagode prikazuju kako se objekti distribuiraju u RAC klasteru i da li se ceo objekat nalazi u memortiji (Objekat je u potpunosti u memoriji ako je atribut BYTES_NOT_POPULATED = 0). Takođe je moguće iz ovih pogleda utvrditi nivo kompresije ostvaren za svaki objekat koji se nalazi u IM, pod pretpostavkom da nisu kompresovani na disku.

```
SELECT v.owner, v.segment_name,
v.bytes orig_size,
v.inmemory_size in_mem_size,
v.bytes / v.inmemory_size comp_ratio
FROM v$im_segments v;
```

	OWNER	SEGMENT_NAME	ORIG_SIZE	IN_MEM_SIZE	COMP_RATIO
1	HR	TRANSAKCIJE	12,050,235,392.00	88,014,848.00	136.91
2	HR	RACUNI	7,340,032.00	4,325,376.00	1.70
3	HR	KOMITENTI	3,145,728.00	2,228,224.00	1.41

Tabela 7 – Prikaz veličine tabela na disku i u memoriji

Takođe novi pogled V\$IM_COLUMN_LEVEL, sadrži detalje o kolonama koje se nalaze u IM skladišti, ukoliko nije potrebno popuniti IM sa svim kolonama iz jedne tabele.

```
select      table_name,column_name,      inmemory_compression      from
v$im_column_level
```

	TABLE_NAME	COLUMN_NAME	INMEMORY_COMPRESSION
1	TRANSAKCIJE	ID	DEFAULT
2	TRANSAKCIJE	PARTIJA	DEFAULT
3	TRANSAKCIJE	KONTO	DEFAULT
4	TRANSAKCIJE	TIP	DEFAULT
5	TRANSAKCIJE	VALUTA	DEFAULT
6	TRANSAKCIJE	IZNOS	NO INMEMORY
7	TRANSAKCIJE	EKVIVALENT	DEFAULT
8	TRANSAKCIJE	DATUM	DEFAULT
9	TRANSAKCIJE	OPIS	NO INMEMORY
10	TRANSAKCIJE	OPIS2	NO INMEMORY

Tabela 9 – In-memory na nivou kolono

USER_TABLES

Najnovija verzija Oracle baze podataka 12c proširuje rečnik *_TABLES kolonom tipa boolean pod nazivom INMEMORY, koja ukazuje koja tabela ima definisan taj atribut.

```
select table_name, inmemory from user_tables
```

	TABLE_NAME	INMEMORY
1	REGIONS	DISABLED
2	COUNTRIES	DISABLED
3	LOCATIONS	DISABLED
4	DEPARTMENTS	DISABLED
5	JOBS	DISABLED
6	EMPLOYEES	DISABLED
7	JOB_HISTORY	DISABLED
8	KOMITENTI	ENABLED
9	RACUNI	ENABLED
10	TRANSAKCIJE	ENABLED
11	TRANSAKCIJE_IM	ENABLED
12	TRANSAKCIJERDB	DISABLED

Tabela 10 – Prikaz tabela koje se nalaze u memoriji i na disku

Tabele koje su particionisane neći imati nikakvu vrednost, jer se atribut INMEMORY definiše na nivou segmenta. INMEMORY atribut za ove tabele će biti definisan preko particija ili podparticija u *_TAB_(SUB)PARTITIONS. Tri dodatne kolone INMEMORY_PRIORITY, INMEMORY_DISTRIBUTE, i INMEMORY_COMPRESSION su takođe dodate u *_TABLES pogled da ukažu na IM attribute za svaku tabelu.

Upravljanje korišćenjem CPU prilikom skladištenja IM formatu kolona

Inicijalno popunjavanje IM skladišta predstavlja veoma intenzivnu operaciju za CPU, koja može da utiče na performanse izvršavana drugih procesa. Za kontrolu korišćenja centralnog procesora tokom uskladištavanja podataka u IM i ukoliko je potrebna promena prioriteta, može se koristiti Resource Manager. Moguće je koristiti standardni i kreirati sopstveni plan za korišćenje RM centralnog procesora. Podrazumevano, IM skladištenje se pokreće u *ora\$autotask* korisničkoj grupi, osim na zahtevano skladištenje, koje se pokreće u korisničkoj grupi koje je prouzrokovalo populaciju. Ako *ora\$autotask* ne postoji u planu, populacija se pokreće u OTHER_GROUPS. Druge operacije u *ora\$autotask* uključuju operacije automatskog održavanja kao što je zbirna statistika i segmentska analiza. Procedura SET_CONSUMER_GROUP_MAPPING se može koristiti za promenu korisničke grupe pri populaciji In-memory skladišta.

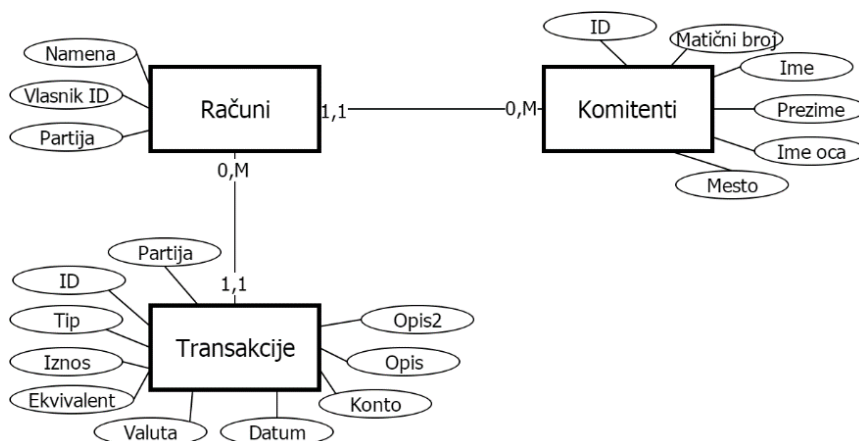
```
BEGIN
  dbms_resource_manager.Set_consumer_group_mapping(
    attribute => 'ORACLE_FUNCTION',
    value      => 'INMEMORY',
    consumer_group => 'BATCH_GROUP');
END;
```

6 Optimizacija fizicke strukture na primeru bankarskog sistema

6.1 Opis problema

Prikazani model bankarskog Sistema (slika 13), predstavlja “usko grlo” rada bankarskog softvera. Veliki broj klijenata koristi usluge banke. Svi klijenti su evidentirani u tabeli “Komitenti”, dok se računi koje klijenti poseduju nalaze u tabeli “Računi”. Količina podataka u ovima tabelama zavisi od veličine banke, što u posmatranom okruženju ne predstavlja veliki problem. Ključni problem predstavlja evidencija transakcija. Evidencija transakcija vezanih za račune komitenata banke se vrši u tabeli “Transakcije”. Transakcija može biti uplata mesečne zarade, plaćanje karticom, podizanje novca sa bankomata, on-line plaćanje, nalozi platnog prometa i dr.

Prema regulativi NBS, banke su dužne da šalju izveštaje o stanjima računa po kontima za svakog komitenta. Kako bi se utvrdilo tačno stanje na određeni datum, potrebno je grupisati sve transakcije po partij i i kontu na kojem su proknjižene. Pored izveštaja za NBS, stanje na određeni dan se izračunava na zahtev klijenta (na šalteru, putem mobilne aplikacije...) za šta je potreban brz odgovor sistema. Izvršavanjem svakodnevnih transakcije, broj slogova u tabeli se konsatno uvećava, dok pronalaženje transakcija po određenom računu postaje sporije.



Slika 13– izdvojen model bankarskog Sistema

Upit pomoću kojeg se prikazuje stanje na računu komitenta na 01. Januar 2014.

```
SELECT R.PARTIJA, T.KONTO, T.VALUTA, SUM(DECODE(T.TIP, 'P', 1, -1) * T.EKVIVALENT) STANJE
FROM TRANSAKCIJE T, RACUNI R, KOMITENTI K
WHERE K.ID = R.KOMITENTID
AND R.PARTIJA=T.PARTIJA
AND T.DATUM < '01jan14'
AND K.MATICNIBROJ='0110965766300'
GROUP BY T.VALUTA, T.KONTO, R.PARTIJA
```

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		34	3094	398979	00:00:16
1	HASH GROUP BY		34	3094	398979	00:00:16
* 2	HASH JOIN		34	3094	398979	00:00:16
3	HASH JOIN		4	172	291	00:00:01
* 4	TABLE ACCESS FULL	KOMITENTI	1	19	74	00:00:01
* 5	TABLE ACCESS FULL	RACUNI	150152	3603648	216	00:00:01
6	TABLE ACCESS FULL	TRANSAKCIJE	1014529	48697392	398685	00:00:16

Tabela 11 – Plan izvršenja upita na tradicionalan način

6.3 Primena standardnih pristupa optimizaciji

Kako se izvršavanjem upita pristupa celim tabelama (može se videti na slici EXPLAIN PLAN), moguće je izvršiti optimizaciju dodavanjem indeksa.

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		19	1729	43	00:00:01
1	. HASH GROUP BY		19	1729	43	00:00:01
2	.. NESTED LOOPS		19	1729	42	00:00:01
3	... NESTED LOOPS		39	1729	42	00:00:01
4 NESTED LOOPS		3	129	5	00:00:01
5 TABLE ACCESS BY INDEX ROWID	KOMITENTI	1	19	2	00:00:01
* 6 INDEX UNIQUE SCAN	I_MATBR	1		1	00:00:01
7 TABLE ACCESS BY INDEX ROWID BATCHED	RACUNI	3	72	3	00:00:01
* 8 INDEX RANGE SCAN	I_KOM	3		1	00:00:01
* 9 INDEX RANGE SCAN	I_PARTRA	13		2	00:00:01
* 10	... TABLE ACCESS BY INDEX ROWID	TRANSAKCIJE	6	288	16	00:00:01

Tabela 12 – Plan izvršenja optimizovanog upita tradicionalan način

Na prethodnom planu izvršavanja upita se može videti da je dodavanje indeksa doprinelo smanjenju koštanja pristupa tabelama. Dodavanjem indeksa se postiže približno vreme izvršavanja kao i kada se radi sa InMemory tabelom. Optimizacija pomoću indeksa u navedenom primer pomaže u velikoj meri kada je potrebno prikazati rezultate za specifičan uslov. U ovom slučaju za određenog komitenta i njegove partije, gde je kreiran indeks nad kolonom partije u tabeli transakcija.

6.4 Izbor pristupa IM u fizičkoj oraganizaciji tabela

U ovom delu će biti prikazana upotreba In-memory skladištenja podataka u formatu kolona na različite načine.

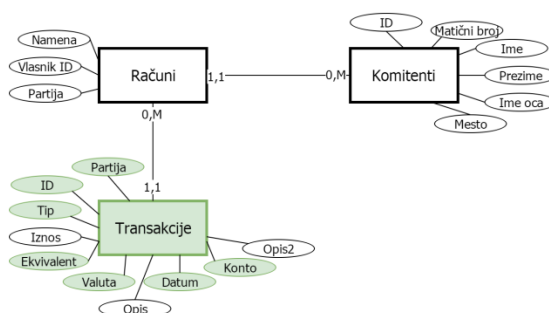
6.4.1 Optimizacija izvršavanja upita – Primer 1

Jedan od čestih akcija koje klijent obavlja je provera stanja na računu. Potrebno je omogućiti klijentu da u bilo kom trenutku u što kraćem roku dobije stvarno stanje sredstava na njegovom računu. Optimizacija izvršavanja problematičnog upita koji prikazuje stanje sredstava na određeni dan za traženog komitenta, moguća je na tri načina:

- Samo kolone kritične tabele koje se koriste u upitu se definišu kao In-memory
- Cela kritična tabela
- Sve tabele koje se koriste u problematičnom upitu

Samo kolone koje se koriste u upitu se prebacuju u format kolona

Jedna od značajnih mogućnosti In-memory opcije je izbor određenih kolona tabele koja je kritična za izvršavanje upita. Definisanje određenih kolona koje će se popuniti u memoriji, smanjuju vreme i trošak memorije.



Slika 14 – Samo određene kolone In-memory

Naredba pomoću koje se kolone id, partija, konto, tip, valuta, ekvivalent i datum smeštaju u memoriju:

```
ALTER TABLE transakcije INMEMORY no
inmemory(iznos,opis,opis2);
```

Kako bi se tabela učitala u memoriju, potrebno izvršiti upit kojim će se skenirati cela tabela.

```
select count(1),id from transakcije_im group by id;
```

U rečniku USER_TABLES se mogu videti tabele i vrednosti određenih atributa koji su vezani za In-memory opciju.

```
SELECT table_name,
       inmemory,
       inmemory_priority,
       inmemory_distribute,
       inmemory_compression,
       inmemory_duplicate
FROM   user_tables f
```

	TABLE_NAME	INMEMORY	INMEMORY_PRIORITY	INMEMORY_DISTRIBUTE	INMEMORY_COMPRESSION	INMEMORY_DUPLICATE
1	RACUNI	DISABLED				
2	TRANSAKCIJE_IM	ENABLED	CRITICAL	AUTO	FOR QUERY LOW	NO DUPLICATE
3	KOMITENTI	DISABLED				

Slika 15 – Prikaz rečnika USER_TABLES

Detalji vezani za In-memory na nivou kolona se mogu videti u pogledu V\$IM_COLUMN_LEVEL.

```
select table_name,column_name, inmemory_compression from v$im_column_level;
```

	TABLE_NAME	COLUMN_NAME	INMEMORY_COMPRESSION
1	TRANSAKCIJE_IM	ID	DEFAULT
2	TRANSAKCIJE_IM	PARTIJA	DEFAULT
3	TRANSAKCIJE_IM	KONTO	DEFAULT
4	TRANSAKCIJE_IM	TIP	DEFAULT
5	TRANSAKCIJE_IM	VALUTA	DEFAULT
6	TRANSAKCIJE_IM	IZNOS	NO INMEMORY
7	TRANSAKCIJE_IM	EKVIVALENT	DEFAULT
8	TRANSAKCIJE_IM	DATUM	DEFAULT
9	TRANSAKCIJE_IM	OPIS	NO INMEMORY
10	TRANSAKCIJE_IM	OPIS2	NO INMEMORY

Slika 16 – Pregled view-a V\$IM_COLUMN_LEVEL

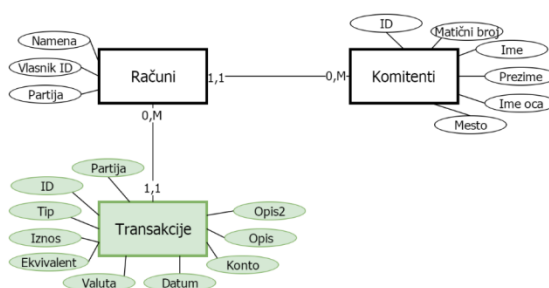
Veličinu tabele tj. prostor koji tabela zauzima na disku i u memoriji, kao i procenat kompresije podataka, može se videti upitom nad pogledom V\$IM_SEGMENTS.

	OWNER	SEGMENT_NAME	ORIG_SIZE_MB	IN_MEM_SIZE_MB	COMP_RATIO
▶ 1	stefan	TRANSAKCIJE_IM	11492	183.9375	62.4777437988447

Slika 17 – Pregled view-a V\$IM_SEGMENTS

Cela tabela se povlači u memoriju

Kada se tabela koja sadrži ogroman broj redova koristi za više različitih analitičkih upita, dobro bi bilo da se cela tabela povuče u memoriju. Jedan od razloga je i mogućnost da se često koriste različite kolone kritične tabele za različite upite.



Slika 18- Cela tabela TRANSAKCIJE In-memory

Naredba kojom se definiše da se cela tabela TRANSAKCIJE prebacuje u memoriju:

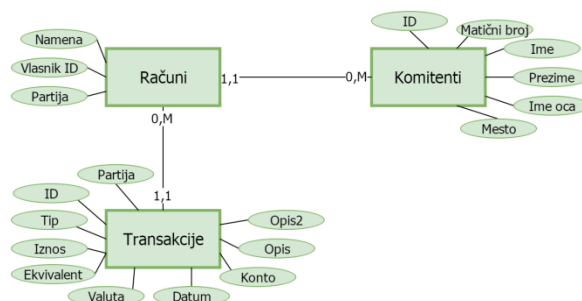
```
ALTER TABLE transakcije INMEMORY;
```

Isto kao i u prethodnom primeru, potrebno je skenirati celu tabelu.

```
select count(1),id from transakcije_im group by id;
```

Sve tri tabele, koje su deo modela, koje se koriste u upitu

Ukoliko postoji dovoljno prostora u memoriji, za postizanje boljih performansi sistema, moguće je učitati u memoriju sve tabele koje se koriste u kritičnom upitu.



Slika 19– Sve tri tabele problematičnog dela modela InMemory

```

ALTER TABLE komitenti INMEMORY;
ALTER TABLE racni INMEMORY;
ALTER TABLE transakcije INMEMORY;
  
```

Kako bi se tabele učitale u memoriju, potrebno je izvršiti upite koji će u potpunosti skenirati sve tri tabele.

```

select count(1),id from transakcije_im group by id;
select count(1),id from komitenti group by id;
select count(1),partija from racuni group by id;
  
```

Razmatranje optimizacije

Izvršavanje upita koji prolazi kroz celu tabelu TRANSAKCIJE traje približno 380 sekundi. Isto to važi i za ostale tabele, s tim što proces prebacivanja traje dosta kraće jer su tabele KOMITENTI i RAČUNI mnogo manje od tabele transakcije.

	Komitent br. 1 [s]	Komitent br. 2 [s]
Tradicionalno bez korišćenja InMemory	195.04	170.8
InMemory određene kolone	0.076	0.060
InMemory – najveća tabela	0.081	0.062
InMemory – sve tabele	0.043	0.039

Tabela 13 – Poređenje brzine izvršavanja upita

Nakon izvršavanja još oko 250000 transakcija, tabela TRANSAKCIJE se značajno povećala. U tabeli je prikazano vreme izvršavanja upita nakon što je ažurirana tabela TRANSAKCIJE koja se već nalazi u IM skladištu.

	Komitent br. 1 [s]	Komitent br. 2 [s]
Tradicionalno bez korišćenja InMemory	210.44	214.97
InMemory određene kolone	0.179	0.172
InMemory – najveća tabela	0.169	0.166
InMemory – sve tabele	0.148	0.144

Tabela 14 – Poređenje brzine izvršavanja upita

6.4.2 Optimizacija izvršavanja upita – Primer 2

Za potrebe kontrole NBS neophodno je kreirati dnevni izveštaj koji će prikazivati stanja po partijama i kontima svih na određeni dan. Za razliku od prethodnog primera gde je bilo potrebno pronaći transakcije za jednog klijenta (transakcije po svim njegovim partijama), sada je potrebno proći kroz sve transakcije i sumirati ih po partiji i kontu.

```
SELECT K.IME || ' ' || K.PREZIME, R.PARTIJA, T.KONTO,
T.VALUTA, SUM(DECODE(T.TIP, 'P', 1, -1) * T.EKVIVALENT) STANJE
FROM TRANSAKCIJE T, RACUNI R, KOMITENTI K
WHERE K.ID = R.KOMITENTID
AND R.PARTIJA=T.PARTIJA
AND T.DATUM < '01jan14'
GROUP BY T.VALUTA, T.KONTO, R.PARTIJA
```

U nastavku će biti prikazani planovi izvršenja prethodnog upita bez korišćenja InMemory (sa indeksima) i sa InMemory tabelom.

Planu izvršenja koji je prikazan u tabeli 15, pokazuje da se tabele spajaju pomoću HASH JOIN-a. Idući plan izvršenja je i u primeru koji koristi tabelu

TRANSAKCIJE koja se nalazi u memoriji. Razlika je u ceni izvršenja upita za svaki korak.

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		956199	87970308	422232	00:00:17
1	HASH GROUP BY		956199	87970308	422232	00:00:17
* 2	HASH JOIN		956199	87970308	401984	00:00:16
3	TABLE ACCESS FULL	KOMITENTI	49999	999980	74	00:00:01
* 4	HASH JOIN		966469	69585768	401907	00:00:16
5	TABLE ACCESS FULL	RACUNI	150152	3603648	216	00:00:01
* 6	TABLE ACCESS FULL	TRANSAKCIJE	966469	46390512	398689	00:00:16

Tabela 15 – Plan izvršenja upita na tradicionalan način

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		956199	87970308	80912	00:00:04
1	HASH GROUP BY		956199	87970308	80912	00:00:04
* 2	HASH JOIN		956199	87970308	60664	00:00:03
3	TABLE ACCESS FULL	KOMITENTI	49999	999980	74	00:00:01
* 4	HASH JOIN		966469	69585768	60587	00:00:03
5	TABLE ACCESS FULL	RACUNI	150152	3603648	216	00:00:01
* 6	TABLE ACCESS INMEMORY FULL	TRANSAKCIJE	966469	46390512	57369	00:00:03

Tabela 16 - Plan izvršenja upita koristeći In-memory opciju

Iz priloženog se može videti kako se cena izvršavanja upita umanjila više od 5 puta, ali se rezultat optimizacije vidi najviše kod izvršavanja upita.

	Stanje na 01.01.2014 [s]
Tradicionalno bez korišćenja InMemory	282.56
InMemory određene kolone	3.718
InMemory – najveća tabela	3.522
InMemory – sve tabele	2.78

Tabela 17 – Poređenje brzine izvršavanja upita

6.4.3 Razmatranje optimizacije

Iz rezultata dobijenih u oba primera, može se videti da korišćenje In-memory opcije značajno ubrzava izvršavanje upita nad tabelama koje imaju ogroman broj kolona. U prvom primeru se pomoću skladištenja podataka u formatu kolona postižu identični rezultati kao što je kod optimizacije korišćenjem indeksa. Rezultati za povlačenje podataka se dobijaju veoma brzo, s tim što je indeksiranja utiče negativno na performanse DML upita.

Drugi primer optimizacije u nedostižnoj prednosti u odnosu na tradicionalne načine optimizacije, zbog obaveznog potpunog skeniranja tabela. Suštinski značaj formata kolona je skeniranja tabela sa velikim brojem redova. U tabeli 15 se može videti uporedni prikaz izvršenja različitih sa i bez In-memory opcije.

	Primer 1 Komitent br. 1 [s]	Primer 1 Komitent br. 2 [s]	Primer 2 Stanje na 01.01.2014 [s]
Bez korišćenja InMemory	210.44	214.97	282.56
InMemory određene kolone	0.179	0.172	3.718
InMemory – najveća tabela	0.169	0.166	3.522
InMemory – sve tabele	0.148	0.144	2.78

Tabela 15 – Vreme izvršavanja upita

Ovo istraživanje je pokazalo da, nema uticaja spajanje blokova za redove koji zauzimaju memorijski prostor na disku veći od veličine bloka, što predstavlja problem kod tradicionalnog načina izvršavanja upita. Prednost prvog načina In-memory optimizacije je što se ne zauzima prostor u memoriji, nepotrebnim podacima, tj. podacima koji se ne koriste u problematičnom upitu. Ukoliko se izostave neke kolone iz upita ili kolone koje se koriste u drugim upitima koji su problematični, može negativno da utiče efiksanost povlačenja podataka. Drugim načinom optimizacije se otklanja mogućnost ne postojanja određene kolone u memoriji, što će sigurno ubrzati bilo koji upit koji radi sa ogromnom tabelom. Treći način zasigurno ubrzava izvršavanje upita, ali i utiče na kapacitet memorije, što može biti problem ukoliko je veličina memorije ograničavajući faktor.

6.5 Analiza postojećih rešenja optimizacije

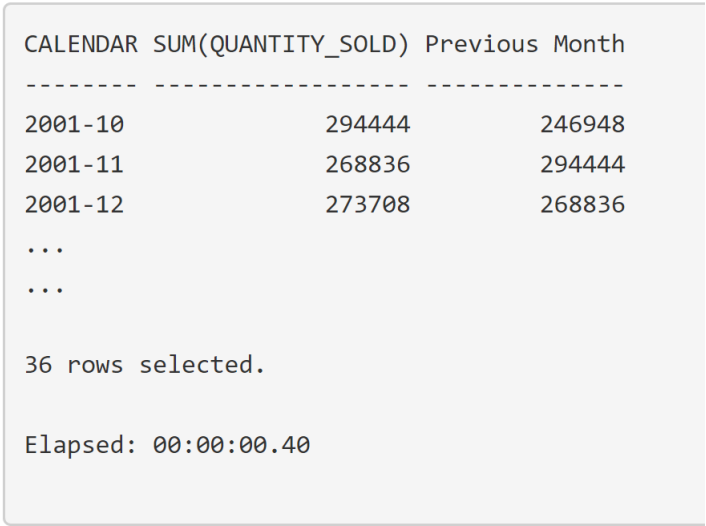
6.5.1 Istraživanje 1

Istraživanje koje je sproveo Soom Garvin (9) bavi se optimizacijom upita koji predstavlja osnovu za kreiranje izveštaja prodaje po mesecima. Tabela gde se beleže podaci o prodaji se naziva SALES i napunjena je sa 11 miliona redova, što može mnogo otežati upitima koji koriste podatke iz ove tabele.

Upit za generisanje izveštaja o prodaji:

```
SELECT CALENDAR_MONTH_DESC,  
       sum(quantity_sold),  
       lag(sum(quantity_sold),1) OVER (  
           ORDER BY CALENDAR_MONTH_DESC) "Previous Month"  
FROM SH.sales2,  
     SH.TIMES  
WHERE SH.times.TIME_ID = SH.sales2.TIME_ID  
GROUP BY CALENDAR_MONTH_DESC  
ORDER BY CALENDAR_MONTH_DESC ASC;
```

Problem kod datog upita se ogleda u tome što je potrebno sumiranje količine prodatih proizvoda za svaki mesec. Korišćenjem In-memory opcije, brzina dobijanja rezultata daje mogućnost korisniku da povlači izveštaj kad god želi za manje od 1 sekunde.



CALENDAR	SUM(QUANTITY_SOLD)	Previous Month
2001-10	294444	246948
2001-11	268836	294444
2001-12	273708	268836
...		
...		

36 rows selected.

Elapsed: 00:00:00.40

Slika 20 – Rezultat izvršavanja upita u istraživanju (9)

U planu izvršavanja upita datog istraživanja se može videti da se tabele skeniraju u potpunosti. Pomoću VECTOR_GROUP_BY operacije se grupišu rezultati iz tabele TIMES, zatim spajaju sa tabelom SALES2 koristeći HASH JOIN.

PLAN_TABLE_OUTPUT							
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
0	SELECT STATEMENT				770 (100)		
1	TEMP TABLE TRANSFORMATION						
2	LOAD AS SELECT						
3	VECTOR GROUP BY		60	1200	5 (20)	00:00:01	
4	KEY VECTOR CREATE BUFFERED	:KV0000					
* 5	TABLE ACCESS INMEMORY FULL	TIMES	1461	29220	4 (0)	00:00:01	
6	WINDOW BUFFER		60	2520	765 (26)	00:00:01	
7	SORT GROUP BY		60	2520	765 (26)	00:00:01	
* 8	HASH JOIN		60	2520	764 (26)	00:00:01	
PLAN_TABLE_OUTPUT							
9	VIEW	VW_VT_0834CBC3	60	1320	762 (26)	00:00:01	
10	VECTOR GROUP BY		60	660	762 (26)	00:00:01	
11	HASH GROUP BY		60	660	762 (26)	00:00:01	
12	KEY VECTOR USE	:KV0000					
* 13	TABLE ACCESS INMEMORY FULL	SALES2	11M	115M	758 (25)	00:00:01	
14	TABLE ACCESS FULL	SYS_TEMP_0FD9D6600_5B3351	60	1200	2 (0)	00:00:01	

Slika 21 – plan izvršavanja upita u istraživanju (9)

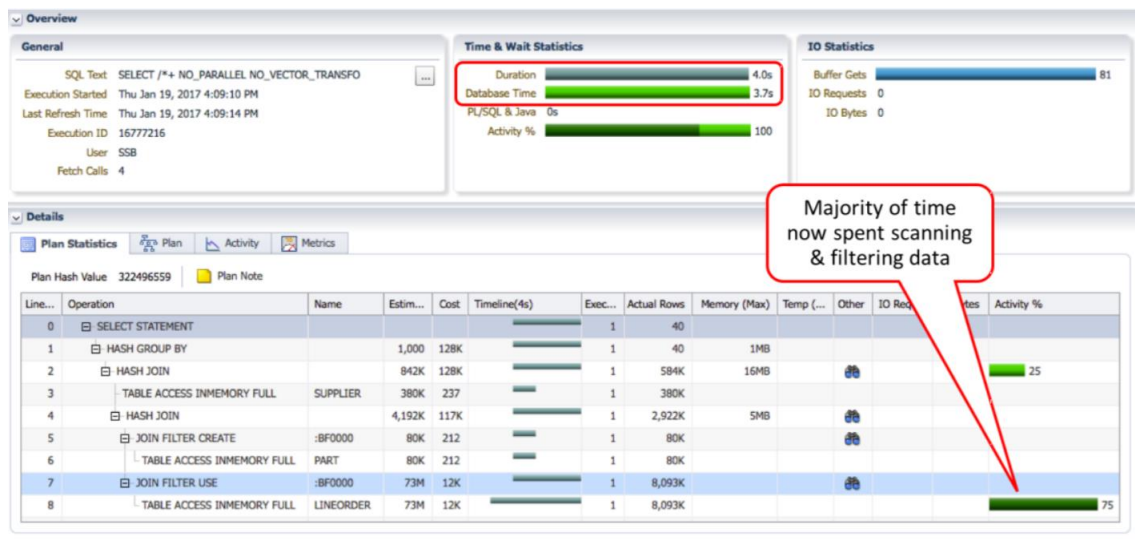
6.5.2 Istraživanje 2

U primeru koje je prikazao Andy Rivenes (10) u istraživanju pod nazivom “Oracle Database In-Memory Implementation and Usage”, može se videti optimizacija upita za povlačenje izveštaja o ukupnim prihodima od prodaje po brendu. Podaci od prodaje se beleže u tabeli LINEORDER koja sadrži 75 miliona slogova, koje je potrebno skenirati kako bi se utvrdili ukupni prihodi.

Upit za prikaz prihoda od prodaje:

```
SELECT p.p_brand1,
       SUM(lo_revenue) rev
FROM lineorder l,
     part p,
     supplier s
WHERE l.lo_partkey = p.p_partkey
AND l.lo_suppkey = s.s_suppkey
AND p.p_category = 'MFGR#12'
AND s.s_region = 'AMERICA'
GROUP BY p.p_brand1
```

U planu izvršenja za dati upit se može videti korišćenje BLOOM filtera kako bi se efikasno izvršilo HASH spajanje prilikom skeniranja nakon.



Slika 23 – Plan izvršenja upita nakon optimizacije

Iz datog plana se može videti da je najviše vremena potrošeno za pristup tabeli LINEORDER u memoriji koristeći BLOOM filter. BLOOM filter se kreira odmah nakon sneiranja tabele PART (linija 5) i zatim se primenjuje kao deo skeniranja cele tabele LINEORDER. Izvršenje upita traje samo 4 sekunde.

7. Zaključak

Aplikacije koje zahtevaju brzinu i pristup velikoj količini podataka, kako bi se odgovorilo na pitanja koja se odnose na poslovanje, mogu iskoristiti benefite koje pruža Oracle In-memory opcija. Izvršavanje transakcija koje sadrže ogromne analitičke upite korišćenjem tradicionalnog načina čuvanja podataka u formatu reda mogu da traju neko vreme. Kašnjenje sa donošenjem odluka predstavlja rizik od konkurentske prednosti ili mogućnosti za neostvarivanjem dodatnih prihoda.

Oracle In-memory pruža jedinstvenu arhitekturu koja omogućava tabelama da se istovremeno nalaze na disku i u memoriji, korišćenjem skladištenja u formatu redova na disku i u formatu kolona u memoriji. Oracle optimizator upita automatski usmerava analitičke upite ka formatu kolona, a OLTP upite na format reda, što transparentno koristi najbolje performanse oba formata. Oracle podržava konzistentnost transakcija između oba formata i nije potrebno dodatno proširivanje kapaciteta diska jer je format kolona u potpunosti In-memory i ne perzistira na disku. Jedan od razloga zašto je sad pravo vreme za uvođenje načina čuvanja podataka u memoriji je zato što je memorija mnogo pristupačnija danas i ekonomski je izvodljivo za implementaciju. Kompanije zahtevaju mogućnost da analiziraju svoje podatke u realnom vremenu bez negativnog uticaja na performanse OLTP operacija. Analitički upiti često imaju tendenciju da pristupaju određenom broju kolona nad tabelama koje sadrže nekoliko miliona ili milijardi redova, dok se OLTP transakcije izvršavaju nad svim kolonama u nekoliko redova istovremeno. Kompanije zahtevaju podatke koji su struktuirani automatski u formatu kolona za analitičke upite i u formatu reda za OLTP.

Oracle In-memory može da obezbedi i do 100 puta efikasnije izvršavanje analitičkih upita. Implementacija In-memory opcije je jednostavna jer nije potrebna promena koda aplikacija, osim dodeljivanja dodatne memorije i identifikacije objekata koji se će se popunjavati u IM skladišta, nije potrebna nikakva izmena u bazi podataka.

8. Literatura

1. **Naik, Shefali.** *Concepts of Database Management System.* s.l. : Pearson India.
2. **Harrington, Jan.** *Relational Database Design and Implementation, 4th Edition.* Kembridž, SAD : Morgan Kaufmann, 2016.
3. **Kalen Delaney, Jessica Moss, Louis Davidson.** *Pro SQL Server Relational Database Design and Implementation, Fifth Edition.* s.l. : APRESS, 2016.
4. **Bryla, Bob.** *Oracle Database 12c DBA Handbook.* Kolumbus, SAD : McGraw-Hill, 2015.
5. **Lance Ashdown, Tom Kyte.** Oracle Database Concepts, 12c Release 1 (12.1). Oracle. [Online] <http://docs.oracle.com/database/121/CNCPT/toc.htm>.
6. **Colgan, Maria.** White Paper Oracle Database In-Memory. Oracle. [Online] Jul 2015. <http://www.oracle.com/>.
7. **Banerjee, Joyjeet.** *Oracle Database 12c Release 2 In-Memory: Tips and Techniques for Maximum Performance.* s.l. : Oracle Press.
8. **Lahiri, Tirthankar.** White Paper When To Use Oracle Database In-Memory . [Online] <http://www.oracle.com/technetwork/database/in-memory/overview/twp-dbim-usage-2441076.html>.
9. **Soorm, Gavin.** ORACLE ACE - ORACLE 12C IN-MEMORY DATABASE. [Online] Septembar 2014. <https://gavinsoorma.com/2014/10/oracle-12c-in-memory-database/>.
10. **Rivenes, Andy.** Oracle Database In-Memory Implementation and Usage. [Online] <http://www.oracle.com/technetwork/database/in-memory/learnmore/twp-oracle-dbim-implementation-3863029.pdf>.
11. **Oracle.** Oracle Database In-Memory - Powering the Real-Time Enterprise. [Online] <http://www.oracle.com/technetwork/database/options/database-in-memory-ds-2210927.pdf>.
12. **Lahiri, Tirthankar.** When to Use Oracle Database In-Memory. [Online] 2015. <http://www.oracle.com/technetwork/database/in-memory/overview/twp-dbim-usage-2441076.html>.
13. **Tirthankar Lahiri, Markus Kissling.** Oracle's In-Memory Database Strategy for OLTP and Analytics. Oracle. [Online] https://www.doag.org/formes/pubfiles/7378967/2015-K-DB-Tirthankar_Lahiri-Oracle_s_In-Memory_Database_Strategy_for_Analytics_and_OLTP-Manuskript.pdf.
14. **Ashdown, Lance.** Oracle. [Online] <http://docs.oracle.com/database/122/INMEM/toc.htm>.

15. **Henschen, Doug.** Oracle In-Memory Option: 6 Key Points. [Online]
<https://www.informationweek.com/software/information-management/oracle-in-memory-option--6-key-points/a/d-id/1269584?>.
16. **Rittman, Mark.** Taking a Look at the Oracle Database 12c In-Memory Option. [Online] <https://www.rittmanmead.com/blog/2014/08/taking-a-look-at-the-oracle-database-12c-in-memory-option/>.
17. **Brian T. Berkowitz, Sreenivas Simhadri, Peter A. Christofferson, Gunnar Mein.** In-memory database system. [Online]
<https://www.google.com/patents/US6457021>.
18. **Oracle.** Oracle Database In-Memory and CustomerXPs Delivers CrossChannel Fraud Management in Real Time. [Online]
<http://www.oracle.com/us/products/database/dbim-customerxps-case-study-3208833.pdf>.
19. **GmbH, Bosch.** Major Improvement in SAP CRM use with Oracle Database In-Memory at Bosch GmbH. [Online] <http://www.oracle.com/us/solutions/sap/nl25-oradb4sap-inmemory-bosch-3013580.pdf>.
20. **Boch, Villeroy &.** Oracle Database In-Memory at Villeroy & Boch. [Online]
<http://www.oracle.com/us/solutions/sap/nl25-oradb4sap-inmemory-vb-2995008.pdf>.
21. **Gill, Philip J.** Memorable Performance. [Online]
<http://www.oracle.com/technetwork/issue-archive/2015/15-may/o35diemobiliar-2541568.html>.
22. **Oracle.** Why Exadata Is the Best Platform for Oracle Database In-Memory. [Online] <http://www.oracle.com/technetwork/database/in-memory/learnmore/twp-dbim-exadata-2556211.pdf>.
23. **Lahiri, Tirthankar.** When to Use Oracle Database In-Memory. [Online]
<http://www.oracle.com/technetwork/database/in-memory/overview/twp-dbim-usage-2441076.html>.
24. **Olofson, Carl W.** Memory-Optimized Transactions and Analytics in One Platform: Achieving Business Agility with Oracle Database. [Online]
<http://www.oracle.com/us/products/database/idc-business-agility-with-oracle-db-2641912.pdf>.
25. **Colgan, Maria, Exley, Richard and Rivenes, Andy.** In-memory - High Availability Best Practices. [Online]
<http://www.oracle.com/technetwork/database/availability/dbim-maa-2658757.pdf>.
26. **Kyte, Tom.** On Oracle Database In-Memory. [Online]
<http://www.oracle.com/technetwork/issue-archive/2015/15-jan/o15asktom-2398997.html>.

27. **Oracle.** A Survey of In-Memory Databases from SAP, Microsoft, IBM, MemSQL and Oracle. [Online] <http://www.oracle.com/technetwork/database/in-memory/learnmore/dbim-competitors-2412331.pdf>.
28. **Agarwal, Brijesh.** Database system with methods for improving query performance with cache optimization strategies. [Online] <https://www.google.com/patents/US5822749>.
29. **Graefe, Goetz.** Query evaluation techniques for large databases. [Online] <https://dl.acm.org/citation.cfm?id=152611>.
30. **Kristin Bennett, Michael C. Ferris, Yannis E. Ioannidis.** A Genetic Algorithm for Database Query Optimization. [Online] [https://s3.amazonaws.com/academia.edu/documents/](https://s3.amazonaws.com/academia.edu.documents/).
31. **Matthias Jarke, Jurgen Koch.** Query Optimization in Database Systems. [Online]
32. **Guy Maring Lohman, Eugene Jon Shekita, David E. Simmen, Monica Sachiye Urata.** Relational database query optimization to perform query evaluation plan, pruning based on the partition properties . [Online] <https://www.google.com/patents/US6092062>.
33. **Taylor, Robert.** Query Optimization for Distributed Database Systems. [Online] <http://www.cs.ox.ac.uk/people/dan.olteanu/theses/Robert.Taylor.pdf>.
34. **Ioannidis, Yannis E.** Query Optimization. [Online] <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=38C868EB0F2AC1A907014961A733D83F?doi=10.1.1.24.4154&rep=rep1&type=pdf>.