POBOLJŠANJE PERFORMANSI APLIKATIVNIH SISTEMA DENORMALIZACIJOM RELACIONIH BAZA PODATAKA

APPLICATION SYSTEMS PERFORMANCE IMPROVEMENT BY DENORMALIZATION OF RELATIONAL DATABASES

Slavko Marić, Velibor Đalić, *Elektrotehnički fakultet, Banja Luka*
Siniša Jakovljević, Fond PIO Republike Srpske, Aleksandar Damljanović, *Lanaco d.o.o., Banja Luka*

Sadržaj - *Normalizovana organizacija relacione baze podataka minimizuje/eliminiše probleme vezane za redundanciju podataka, nemogućnost predstave i rekonstrukcije podataka, i u funkciji je podrške održavanja i očuvanja integriteta podataka. Denormalizacijom se svjesno i namjerno uvodi redundancija podataka, sa (najčešćim) ciljem postizanja boljih performansi/odziva aplikativnog sistema. U ovom radu su analizirani i opisani različiti scenariji kada se denormalizacijom organizacije relacionih baza podataka može postići poboljšanje aplikativnih performansi. Prezentovani su eksperimenti, bazirani na realnim aplikativnim situacijama, koji ilustruju pozitivne i negativne efekte primjene postupka denormalizacije na strukture koje reprezentuju generalizacije/specijalizacije. Eksperimenti su realizovani korištenjem Oracle 10g SUBP i Toad for Oracle 10.5.0.41 alata.*

Abstract - *Normalization of relational databases minimizes/eliminates problems related to repetition of information, inability to represent certain information, and supports data maintenance and integrity. Denormalization is the process of intentionally and consciously backing away from normalization to improve performance. In this paper some denormalization scenarios of the relational database organisation, that provide higher application system performance, are presented. Presented experiments are based on real application cases, and illustrate positive and negative effects of denormalization process. Experiments were realized using Oracle 10g DBMS and Toad for Oracle 10.5.0.41 tool.*

1. UVOD

Dug period kontinualnog razvoja sistema za upravljanje bazama podataka (SUBP) i pratećih razvojnih alata, omogućio je relativno lak i jednostavan razvoj robusnih softverskih aplikacija i sistema sa bazama podataka, otpornih na sistemske otkaze, koje rade korektno i u radnom okruženju sa velikim brojem konkurentnih korisnika. Ovakve sisteme standardno prate visoki zahtjevi u pogledu performansi (brzine odziva i propusne moći), koje često realizovani sistemi ne zadovoljavaju i pored primjene savremenih tehnologija i alata, te visokih performansi korištenih hardverskih platformi. Razlog tome je velika interna kompleksnost sistema, čiji su detalji skriveni od razvojnih inžinjera i korisnika, i u kojem mnogi elementi i njihove interakcije mogu da utiču na performanse u toku izvršavanja kompleksnih internih operacija.

Faktori koji mogu da utiču na performanse aplikativnih sistema sa bazama podataka, između ostalih su: način specifikacije i izvršavanja upita, indeksne strukture i način korištenja indeksa, primijenjeni nivoi izolacije transakcija i mehanizmi zaključavanja, trajanje transakcija, karakteristike određenih funkcionalnosti i parametara operativnog sistema, mehanizmi trajnog registrovanja potvrđenih transakcija i oporavka u slučaju sistemskih otkaza, način logičke organizacije strukture baze podataka, karakteristike hardverskih elemenata sistema itd. [1]. Navedeni faktori mogu imati kompleksne interakcije koje nije jednostavno identifikovati, kao ni pojedinačne faktore koji učestvuju u degradaciji performansi niti stepen njihovog pojedinačnog doprinosa toj degradaciji.

Poboljšanje performansi aplikativnih sistema sa bazama podataka može se postići podešavanjem relevantnih parametara i odgovarajućim korištenjem prethodno navedenih faktora i mehanizama (eng. *database tuning*). I pored određenih napora za formalizaciju i matematičku specifikaciju problema i optimizaciju parametara sistema, u praksi dominiraju pristupi zasnovani na logičkom i racionalnom prosuđivanju i empirijskim/iskustveno zasnovanim pravilima i principima. Podešavanje nije jednostavan proces, jer zahtijeva poznavanje i razumijevanje internih principa rada više komponenti sistema: aplikacije, SUBP, operativnog sistema i hardverskih komponenti.

U ovom radu je analizirana jedna opcija poboljšanja performansi sistema sa relacionim bazama podataka - denormalizacijom organizacije relacione baze. Denormalizacijom se svjesno i namjerno narušava normalizovana organizacija uz propratne negativne efekte, i sa samo jednim motivom - postizanje boljih performansi/odziva aplikativnog sistema.

Rad je organizovan na slijedeći način. U sekciji 2 su navedene osnovne karakteristike normalizovane i denormalizovane organizacije relacione baze podataka, i opisane situacije kada projektant baze podataka može razmišljati o kontrolisanoj denormalizaciji, kao i najčešći postupci denormalizacije. U sekciji 3 je opisan postupak denormalizacije tabelarne reprezentacije specijalizacija, a zatim su prezentovani eksperimentalni rezultati za simulirane aplikativne situacije realnog sistema. U sekciji 4 su opisani principi, pristup i preporuke za denormalizaciju relacionih baza podataka. Na

kraju, u sekciji 5, izloženi su zaključci vezani za poboljšanje aplikativnih performansi denormalizacijom.

2. POSTUPCI DENORMALIZACIJE

'Dobra' organizacija baze podataka uobičajeno podrazumijeva organizaciju baze u trećoj normalnoj formi (3NF). 3NF je pogodna jer se od neke polazne organizacije baze podataka uvijek dekompozicijom može doći do organizacije baze u 3NF, pri čemu dekompozicije čuvaju i održavaju funkcionalne zavisnosti koje vrijede na polaznoj relaciji (eng. *dependency preserving decomposition*) i obezbjeđuju spajanje dekomponovanih relacija bez gubitka informacija (eng. *lossless join decomposition*). Ovakve karakteristike nema dekompozicija u poželjniju Boyce-Codd'ovu normalnu formu (BCNF), koja ne može da garantuje očuvanje funkcionalne zavisnosti na dekompozicijama u svim slučajevima. Osim toga, mnogi CASE alati omogućuju automatsko generisanje relacionog modela baze podataka na implementacionom nivou koji je u 3NF iz konceptualnog (najčešće E-R) modela. Zbog navedenog, prihvatljivom se smatra dekompozicija u 3NF (ako BCNF nije moguće ostvariti), a toleriše se nedostatak 3NF zbog određene redundantnosti podataka, koji je posljedica relaksacije BCNF. Strogo gledano, organizacija baze podataka u 3NF (i u BCNF) takođe može sadržavati određene redundantnosti i anomalije, koje se mogu eliminisati višim stepenima normalizacije (4NF i 5NF).

Suštinski, normalizovana organizacija baze podataka omogućava da se podaci koji reprezentuju određene činjenice, objekte i pojave realnog svijeta registruju na jednom mjestu. Time se poboljšava i olakšava proces ažuriranja, podržava integritet i konzistentnost baze, smanjuje/eliminiše redundantnost podataka, ali to može da naruši performanse pretraživanja i dobijanja podataka. Kontrolisana redundantnost može da bude poželjan projektni cilj, dok je nekontrolisana redundancija podataka nepoželjna u svakom slučaju [2].

U određenim situacijama, kada aplikativni zahtjevi diktiraju potrebu za brzim pretraživanjem i pristupom podacima, projektanti baza podataka se mogu odlučiti za kontrolisanu denormalizaciju organizacije baze, sa ciljem poboljšanja aplikativnih performansi. Poboljšanje performansi na specifične upite se dobija procesom koji je inverzan dekompoziciji u procesu normalizacije – objedinjavanjem podataka iz različitih tabela na jedno mjesto (tabelu). Ovim se izbjegava vremenski zahtjevan proces spajanja (dekomponovanih) relacija za vrijeme izvršavanja upita, jer se željeni podaci već nalaze u jednoj tabeli. Kao propratni negativni efekat imamo postojanje istih podataka na više mjesta u bazi, što komplikuje ažuriranje, povećava memorijski prostor i potencijalno predstavlja prijetnju konzistentnosti i integritetu podataka. Postoji više situacija i varijanti primjene kontrolisane denormalizacije [3], [4], [5], [6].

Dodavanje redundantnih podataka u neku tabelu. Ako se nekim čestim upitom zahtijeva dobijanje podataka iz neke tabele i sa njima povezanih podataka iz neke druge tabele, onda je pri realizaciji svakog upita neophodno izvršiti spajanje tabela. Ako brzina odziva ne zadovoljava, onda se može razmisliti o redundantnom dodavanju potrebnih podataka u odgovarajuću tabelu. Na primjer, posmatrajmo u bazi podataka nekog preduzeća (pojednostavljenu) strukturu tebele *stanje_skladišta* (*sklad_id*, *artikl_id*, *količina*), gdje su *sklad_id*, *artikl_id* strani ključevi (identifikatori skladišta i artikla), koji čine primarni ključ tabele *stanje_skladišta*. Česti *on-line* uvidi u stanje određenog skladišta (u procesu naručivanja robe od strane kupca, npr.) zahtijeva *on-line* prikaz stanja skladišta. Prikaz treba da pored *artikl_id* i atributa *količina* prikaže i (minimalno) *naziv_artikla,* što zahtijeva spajanje tabela *stanje_skladišta* i *artikl.* Kako bi se izbjeglo spajanje tabela, ukoliko performanse prikaza stanja skladišta nisu zadovoljavajuće, moguće rješenje je dodavanje atributa *naziv_artikla* (iz tabele *artikl*) i u tabelu *stanje_skladišta*. Ovim se narušava 3NF (a i 2NF), jer *artikl_id* $\rightarrow$ *naziv_artikla,* a *artikl_id* je podskup primarnog ključa tabele. Pri tome, mora se programski osigurati konzistentnost podataka koji se ponavljaju u različitim tabelama. Slične situacije imamo i za denormalizaciju iz 3NF u 2NF.

Koncept denormalizacije se može primijeniti i na više normalne forme (4NF i 5NF), ali su ovi slučajevi u praksi rijetki. Denormalizacija 4NF na primjer, gotovo uvijek vodi izraženim problemima održavanja (konzistentnosti) podataka prilikom ažuriranja, jer se normalizacija u 4NF radi sa ciljem izbjegavanja višestrukog dodavanja zapisa (eng . *tuple generating problem*) prilikom generisanja nekog zapisa.

Višeznačni atributi nekog entitetskog skupa se u normalizovanoj reprezentaciji smještaju u posebne tabele, sa jednim redom u tabeli za svaku vrijednost atributa. Denormalizovanom reprezentacijom se sve vrijednosti višeznačnog atributa smještaju u jedan red tabele (kojom se reprezentuje odnosni entitetski skup), u fiksan broj polja. Denormalizacijom se izbjegava spajanje podataka koju imamo kod normalizovane reprezentacije u najvećem broju upita, a po cijenu neiskorištenog prostora na disku (kada je broj vrijednosti višeznačnog atributa entiteta manji od rezervisanog broja polja) i nemogućnosti predstave više vrijednosti od fiksno predviđenog broja. Neki SUBP (Oracle npr.) podržavaju atribute koji mogu imati više vrijednosti (elementi *varray* tipa i ugniježdene relacije), što je dovelo do uvođenja pojma 0NF, kako bi se relacije sa ovim tipom atributa uklopile u relacionu paradigmu [4].

Ako se dvije ili više tabela često (korištenim upitom) spajaju, sa ciljem poboljšanja brzine odziva može se voditi posebno tabela koja će (redundantno) sadržavati već spojene podatke u jednoj tabeli (eng. ***pre-joined table***). Pri tome, skup podataka iz drugih tabela koji se vode u spojenoj tabeli, mora da bude minimalan potreban skup koji zadovoljava zahtjeve za brzo dobijanje informacija za kritični upit. Najpovoljnija implementaciona varijanta je preko materijalizovanog pogleda, jer je teret održavanja konzistentnosti i integriteta podataka u materijalizovanom pogledu na SUBP. Cijena za dobijenu brzinu odziva na upit iz jedne, prethodno spojene tabele, jeste dodatno vrijeme potrebno za ažuriranje redundantne tabele, kao i dodatni prostor na disku za smještanje redundantnih podataka.

U određenim situacijama sračunavanje i dobijanje **izvedenih podataka** može biti komplikovano i vremenski zahtjevno, pa je u takvim situacijama moguće rješenje vođenje izvedenih podataka u bazi. Na primjer, tekuće stanje artikla na nekom skladištu je izvedeni podatak, i može se izračunati na osnovu početnog stanja i ukupnog prometa (ulaza/izlaza) do tekućeg trenutka. Broj transakcija prometa

nekog artikla može biti vrlo veliki, pa pretraživanje tih transakcija i računanje prometa može biti dugotrajno. U procesu *on-line* prodaje na primjer, potrebno je imati 'trenutan' uvid o stanju artikla na skladištu, što se može realizovati vođenjem trenutnog stanja u odgovarajućoj **izvedenoj tabeli** (kao u prethodno pomenutoj tabeli *stanje_skladišta*). Cijena za dobijanje brzog odziva o stanju na skladištu je da se sa svakom transakcijom prometa artikla mora ažurirati odgovarajuće stanje na skladištu, kako bi se osigurala konzistentnost podataka.

Dijeljenje tabele (eng. *table split*) se može primijeniti za poboljšanje performansi ako se najčešće pristupa distinktnim dijelovima tabele iz različitih aplikacija i od različitih grupa korisnika. U takvim situacijama, dijeljenje tabele neće zahtijevati spajanje prilikom pristupa, jer se svi podaci nalaze u odgovarajućoj particiji, a rad sa manjim tabelama je efikasniji. Dijeljenje može biti **po kolonama (**vertikalno) i **po redovima** (horizontalno). Ukoliko postoje aplikacije koje zahtijevaju brz odziv, i koje rade sa kompletnom originalnom tabelom, onda se može voditi i originalna (normalizovana) tabela, i (nenormalizovane) komponentne particije istovremeno, pri čemu se mora obezbijediti konzistentnost podataka.

Obrnut proces, **objedinjavanje** tabela, može se primijeniti ukoliko je kardinalnost mapiranja veza tipa 1:1. U ovim slučajevima ne postoji problem ponavljanja i održavanja konzistentnosti podataka, niti problem gubitka performansi zbog spajanja tabela. Ipak, iz razloga navedenih u prethodnom pasusu, objedinjavanje povećava dužinu (broj bajta) reda u tabeli čime se smanjuje brzina pristupa po zapisu, pa objedinjavanje može biti opravdano ako aplikacije dominantno pristupaju međusobno povezanim podacima iz obje (pojedinačne) tabele istovremeno.

Hijerarhijske strukture podataka **(**organizacione strukture, sastavnice itd.**)** se jednostavno implementiraju u relacionim bazama, ali procesiranje kompleksnijih hijerarhijskih struktura implementiranih normalizovanim tabelama često nije efikasno. Za brzo i efikasno dobijanje podataka mogu se koristiti alternativne nenormalizovane tabele (eng. *speed table*) [5].

Pored navedenih, moguće su i druge specifične varijante denormalizacije, kao na primjer kreiranje redundantnih tabela (eng. *mirror tables*), kreiranje tabela iz generisanih izvještaja, i posebno kreiranje (redundantnih) skladišta podataka (eng. *data warehouse*) iz produkcionih baza, sa ciljem efikasnog dobijanja informacija potrebnih za odlučivanje [6].

3. PRIMJER DENORMALIZACIJE TABELARNE REPREZENTACIJE SPECIJALIZACIJA

U aplikativnim sistemima, za predstavljanje sličnih entiteta uobičajeno koristimo koncepte specijalizacije/generalizacije[1]. Ove koncepte u relacionim sistemima prevodimo u tabele po odgovarajućim pravilima [7]. Tabela koja odgovara entitetskom skupu superklase sadrži zajedničke atribute specijalizovanih klasa, dok tabele koje odgovaraju podklasama (specijalizacijama) sadrže primarni ključ superklase i specifične atribute podklase (Slika 1a). Ukoliko specijalizaciju karakteriše totalnost i disjunktnost, onda je alternativna (normalizovana) predstava preko skupa tabela koje predstavljaju samo specijalizovane klase sa svim atributima tih specijalizovanih klasa (Slika 1b).

Tabela	Atributi
vozilo	*reg_broj, proizvođač, god_proizvodnje*
putničko	*reg_broj,broj_osoba*
teretno	*reg_broj,nosivost*

a)

Tabela	Atributi
Putničko	*reg_broj, proizvođač, god_proizv,broj_osoba*
teretno	*reg_broj, proizvođač, god_proizv,nosivost*

b)

Slika 1. Varijante normalizovane predstave specijalizacije

U realnim poslovnim informacionim sistemima, standardne poslovne transakcije su ulazi/izlazi robe, materijala i usluga, na primjer u poslovnim sistemima koji se bave veletrgovinom. Podaci o ovim poslovnim transakcijama se tipično registruju u tabelu zaglavlja transakcija i tabelu transakcionih detalja – stavki. Za predstavljanje ovih transakcija pogodan je koncept specijalizacije. Najčešće, za *on-line* i *off-line* preglede su potrebne stavke, jer su u njima zapisani transakcioni detalji. Iako se stavke pojedinih transakcija razlikuju u nekim detaljima, one imaju i zajedničke atribute, te se za predstavljanje stavki može koristiti koncept specijalizacije. U ovom radu je analiziran efekat različite organizacije podataka (uključujući i denormalizovanu organizaciju) na performanse upita, na primjeru organizacije podataka koja reprezentuje stavke transakcija. Pored dvije normalizovane organizacije (šeme Normal1 i Normal2 koje korespondiraju varijantama predstavljenim na Slici 1a i 1b, respektivno) razmatrana je i denormalizovana organizacija transakcionih stavki (šema Denormal), koju čini jedna tabela sa atributima koji su unija atributa svih tipova transakcionih stavki. Broj i karakteristike atributa predstavljenih navedenim šemama, kao i broj zapisa i njihova raspodjela po vrstama transakcionih stavki predstavljaju simulaciju podataka realnog sistema. Sumarne karakteristike pojedinih organizacija su predstavljene slijedećom tabelom:

Tabela 1. Karakteristike organizacija Normal1, Normal, Denormal

Šema	Broj tabela	Parent Tabela	Zapisa ukupno	Velič [Mb] ukupno
Normal1	6	Da	783 279	131
Normal2	5	Ne	783 279	119
Denormal	1	Ne	783 279	144

Kao što se vidi iz Tabele 1, Normal1 ima jednu (parent) tabelu sa zajedničkim atributima svih transakcionih stavki koja reprezentuje nadklasu i 5 tabela koje odgovaraju specijalizacijama, dok Normal2 ima 5 tabela koje sadrže kompletne zapise različitih tipova transakcionih stavki. Najmanje prostora na disku zauzima Normal2 (predstavljaju se samo potrebni podaci), zatim Normal1 (zahtijeva dodatni prostor za vezu zapisa iz parent tabele i specijalizovanih tabela preko surogat ključa *id*), dok denormalizovana tabela u zapisu stav-

[1] Ovi termini se mogu kao sinonimi, u daljnjem tekstu koristiće se termin specijalizacija.

ke pored potrebnih podataka sadrži i *null* vrijednosti na atributima koji potiču iz drugih tipova stavki.

Efekat različitih organizacija transakcionih stavki na performanse upita testiran je kroz simulaciju realnih aplikativnih upita za karticu prometa nekog artikla u periodu (**kartica**) i sumarnog prometa za period, sortiranog po organizacionim jedinicama (skladištima) i vrstama prometa (**promet**). Upit **kartica** selektuje sve transakcije nekog artikla na skladištu, sortirane po datumu nastanka, i prikazuje potrebne podatke u odgovarajućim kolonama. Rezultati izvršavanja upita su predstavljeni Tabelama 2 i 3.

Tabela 2. Kartica

Šema	Artikl	Zapisa Ukupno	Korišteni Indeksi	Odziv [s]
Normal1	A	2375	Ne	3
Normal2	A	2375	Ne	4
Denormal	A	2375	Ne	3
Normal1	A	2375	Da	8.5
Normal2	A	2375	Da	2.3
Denormal	A	2375	Da	2
Normal1	B	388	Da	9
Normal2	B	388	Da	0.67
Denormal	B	388	Da	0.6

Tabela 3. Promet

Šema	Period prometa	Korišteni Indeksi	Odziv [s]
Normal1	6 mjeseci	Ne	5
Normal2	6 mjeseci	Ne	5.3
Denormal	6 mjeseci	Ne	6
Normal1	6 mjeseci	Da	4.5
Normal2	6 mjeseci	Da	4.7
Denormal	6 mjeseci	Da	45
Normal1	1 dan	Da	3
Normal2	1 dan	Da	0.7
Denormal	1 dan	Da	0.68

Memorijski bafer za podatke, indekse, plan izvršenja itd. je bio veličine 260 Mb (~ 180 Mb za baferovanje podataka i ~80Mb za dijeljeni bafer). Sve tabele mogu kompletno da se učitaju u memorijski bafer za podatke. Baferi su pražnjeni prije startovanja svakog testa pokretanjem odgovarajućih skriptova. Razlog tome je što su izvršavanja brza (reda par stotina milisekundi) ako se podaci i plan izvršenja nalaze u memoriji, a realne aplikativne situacije za navedene upite su da se (najčešće) sukcesivni upiti neće ponavljati. Za analizu izvršenja i mjerenje odziva korišten je alat Toad for Oracle 10.5.0.41 a korišteni SUBP je Oracle 10g Express Edition. Aplikacija je testirana na personalnom racunaru sa Windows XP OS, Intel Core 2 Duo procesorom na 2.5 Mhz i 2Gb memorije.

Eksperimenti pokazuju interesantne rezultate. Može se zaključiti da nijedan tip organizacije nema apsolutan i isključiv uticaj na performase upita. Na performanse upita (pored organizacije) utiče i broj zapisa kojima se pristupa, (ne)postojanje odgovarajućih indeksa i zaključak optimizatora o planu izvršenja koji će biti primijenjen.

Za upit **kartica**, ako se ne koriste indeksi, svi upiti su koristili puni pristup tabelama (eng. *full table access*). Izvršavanje upita za Normal1 organizaciju je koristilo efikasno *hash join* spajanje (nakon učitavanja podataka u memorijski bafer) i nakon toga operaciju unije selektovanih spojenih zapisa i sortiranje, operacije koje na relativno malom broju podataka ne traju dugo. Normal2 je imao učitavanje cijelih (specijalizovanih) tabela, selekciju potrebnih podataka, uniju i sortiranje. Upit za denormalizovanu organizaciju je imao puni pristup samo jednoj tabeli, selekciju i sortiranje podataka. Iako su operacije u memoriji za Denormal organizaciju jednostavnije u odnosu na prethodne dvije, veći je broj pristupa disku jer je ukupna veličina prostora na disku za ovu organizaciju veća, pa su na kraju vremena izvršenja približno ista (3, 4, 3 sekunde).

Ako se koriste indeksi, onda Normal2 i Denormal koriste preko indeksa pristup samo malom broju (2375 u odnosu na ukupno 783279 zapisa), pa su ti upiti efikasniji u odnosu na upite bez indeksa (Denormal je nešto efikasniji, 2s u odnosu na 2.3s za Normal2, zbog toga što nema izvršavanje *union* operacije). Normal1 se sa indeksima izvršava neočekivano dugo. Plan izvršenja je pokazao da je optimizator nakon pristupa parent tabeli po indeksu vršio *nested join* spajanje sa tabelama specijalizacije, što je prouzrokovalo značajno vrijeme izvršavanja ovog upita za Normal1 organizaciju. Korištenje indeksa za rezultate koje produkuju mali broj izlaznih redova (artikl B) je veoma efikasan za Normal2 i Denormal, jer se preko indeksa pristupa malom broju zapisa (a time i disk blokova). Iz prethodno navedenog razloga (*nested join* spajanje za Normal1), izvršavanje upita za Normal1 organizaciju je neefikasno kao i u prethodnom slučaju.

Slična razmatranja su i za upit **promet**. Bez indeksa svi upiti imaju puni pristup i čitanje svih tabela, i približna vremena izvršavanja (najefikasniji Normal1, 5 sec). Pristup preko indeksa pokazuje sljedeće: ako se selektuje mnogo podataka (promet za 6 mjeseci) pristup preko indeksa (kreiranog po datumu stavki) se izvršava za veliki broj podataka (a time se pristupa i velikom broju disk blokova - u najgorem slučaju broj blokova je jednak broju indeksnih blokova + broj selektovanih podataka), pa je iz tog razloga pristup za Denormal vrlo skup i niskih performansi (korištenje indeksa je neadekvatno). Optimizator nije prihvatio pristup preko indeksa za Normal1 i Normal2, već je forsirao pristup cijeloj tabeli (eng. *full table scan*) sekvencijalno, i procesiranje kao i za karticu. Zbog većeg broja podataka koji su selektovani za obradu, vrijeme procesiranja upita je nešto duže za ove šeme i ovaj upit nego za korespondentne upite za karticu. Međutim, za upit sa indeksima za mali broj podataka koji se selektuje (promet za jedan dan), selekcija na bazi indeksa za šeme Normal2 i Denormal daje visoke performase odziva. Za Normal1, optimizator je odbio korištenje indeksa i radio slično kao za karticu bez indeksa, pa je i odziv sličan.

Rezime prethodne analize je kao što slijedi. Nijedna organizacija podataka ne garantuje automatski najbrži odziv. Ako bi za navedene tipove upita izabrali najpovoljnije odzive za pojedine načine organizacije, onda bi Denormal imao najbolje performanse u najvećem broju upita a Normal1

najlošije. Rezultati takođe pokazuju da su performanse bilo koje organizacije ekstremno osjetljive na druge faktore (korištenje indeksa i plan izvršenja na primjer). Iako bi brzina odziva na upite mogla biti prihvatljiva za normalizovane verzije za jedan broj korisnika, u stvarnoj implementaciji je izabrana denormalizovana implementacija. Pored (sveukupno) najboljih performansi/brzine odziva, ova vrsta denormalizacije ne generiše višestruke kopije nekog podatka, tako da ne komplikuje ažuriranje, iako sadrži redundantne *null* vrijednosti, čime se povećava potreban prostor na disku. Dodatno, upite je jednostavnije specifikovati za Denormal, nego za Normal1 i Normal2.

4. PRINCIPI, PRISTUP I PREPORUKE ZA DENORMALIZACIJU RELACIONIH BAZA PODATAKA

Generalno, motivi i ciljevi denormalizacije su poboljšanje performansi pretraživanja i dobijanja podataka iz baze, po cijenu ponavljanja i redundancije podataka i komplikacije i otežavanja procesa ažuriranja. Preciznije rečeno, denormalizacijom se može postići poboljšanje performansi pretraživanja i dobijanja podataka **određenih aplikativnih funkcija**, uz, ne rijetko, negativne efekte na performanse pretraživanja drugih aplikativnih funkcija. Dalje, denormalizacija je proces koji se zasniva na iskustvenim, pragmatičnim i subjektivnim procjenama i postupcima, pri čemu nema jasnih kriterijuma kada i gdje se zaustaviti u ovom procesu, za razliku od dobro razrađene i utemeljene teorije normalizacije koja definiše sam postupak i njegovo nastavljanje sve dok se ne dostigne najviša moguća normalna forma [2].

Iz navedenog, može se zaključiti da denormalizaciju treba primjenjivati sa pažnjom, u posebnim situacijama, kada je to zaista potrebno, a nikako kao *per se* poželjnu aktivnost. Zbog toga je najprije potrebno utvrditi da li je denormalizacija uopšte potrebna. Polazna tačka razmatranja treba da bude realizovani sistem sa potpuno normalizovanom organizacijom baze podataka, za koji je potrebno utvrditi:

- da li su performanse sistema sa normalizovanom organizacijom zadovoljavajuće,
- ukoliko nisu, da li su loše performanse posljedica drugih uzroka, i da li se željene performanse mogu postići na drugi način (optimizacija SQL upita, podešavanje indeksa, mogućnost poboljšanja performansi zamjenom kritičnih hardverskih komponenti, novih verzija SUBP itd.).

Ukoliko su sve druge opcije iscrpljene, potrebno je pažljivo razmotriti mogućnost poboljšanja perfomansi denormalizacijom strukture relacione baze. Potrebno je:

- identifikovati aplikativne funkcije koje ne daju zadovoljavajući odziv i pored drugih poduzetih mjera i akcija, i selektovati samo one čije se nezadovoljavajuće performanse ne mogu tolerisati kao kandidate za poboljšanje perfomansi denormalizacijom,
- analizirati postupke i opcije denormalizacije koje bi mogle unaprijediti performanse i selektovati odgovarajući postupak denormalizacije,
- obezbijediti očuvanje konzistentnosti i integriteta podataka u radu sa denormalizovanom bazom. Denormalizacija mora da zadovolji zahtjev za mogućnost (obrnute) konverzije u normalizovanu formu bez gubitaka informacija. Denormalizaciju primijeniti samo ako rješenje na bazi denormalizacije ne umanjuje pouzdanost sistema,
- analizirati potencijalno negativne efekte na druge aplikativne funkcije (ako je denormalizacija primijenjena i ako su poboljšane performanse ciljnih aplikativnih funkcija),
- dokumentovati i prihvatiti postupak denormalizacije kao dio aplikativnog rješenja (ako su eventualni negativni efekti denormalizacije na performanse drugih aplikativnih funkcija prihvatljivi).

Kada se prihvati i implementira rješenje sa denormalizacijom baze u nekom segmentu, potrebna je periodična analiza da li su razlozi zbog kojih je urađena denormalizacija još uvijek aktuelni. Vremenom se mijenjaju procesni zahtjevi i unapređuju hardversko-softverske komponente sistema, što može dovesti do prestanka razloga za denormalizaciju.

5. ZAKLJUČAK

Normalizovana organizacija baze podataka omogućava da se podaci koji reprezentuju određene činjenice, objekte i pojave realnog svijeta registruju na jednom mjestu. Time se poboljšava i olakšava proces ažuriranja, podržava integritet i konzistentnost baze, smanjuje/eliminiše redundantnost podataka, ali to može da naruši performanse pretraživanja i dobijanja podataka. Kontrolisana denormalizacija može da bude moguća projektna opcija, sa ciljem postizanja boljih performansi/odziva aplikativnog sistema, kao što je to pokazano na primjeru denormalizacije tabelarne reprezentacije specijalizacija u ovom radu. Mnogo različitih faktora sa kompleksnim interakcijama može da utiče na aplikativne performanse. Denormalizaciju treba primijenjivati sa pažnjom, u posebnim situacijama i kada je to zaista potrebno. Polazna tačka razmatranja treba da bude realizovani sistem sa potpuno normalizovanom organizacijom baze podataka. Ukoliko se prihvati i implementira rješenje sa denormalizacijom relacione baze u nekom segmentu, potrebno je periodično analizirati aktuelnost razloga zbog kojih je denormalizacija urađena i mogućnost ponovne normalizacije.

LITERATURA

[1] D. Sasha and P. Bonnet, *Database Tuning*, Morgan Kaufmann Publishers 2003.

[2] C.J. Date, "Some Relational Myths Exploded (An examination of some popular misconceptions concerning relational database management systems)", InfoMIS 4 No. 2, 1984.

[3] D.B. Bock and J.F., "Benefits of Denormalized Relational DatabaseTtables", Proceedings of the Decision Sciences Institute, Orlando, Florida, November, 1996.

[4] D. K. Burleson, *Oracle High-Performance SQL Tuning*, Oracle Press 2001.

[5] C. Mullins, "Denormalization Guidelines", The Data Administration Newsletter, June 1997.

[6] J. Singh, B. Singh, Y. Sriveni, "A Convenient Way From Normalized Database To Denormalized Database", Proceedings of the IJJCET 2010, pp 84-87.

[7] A.Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts, McGraw-Hill, 2002.