# UTJECAJ DENORMALIZACIJE BAZE PODATAKA NA WEB APLIKACIJE

## 1. Uvod

Svjedoci smo proteklih godina velikog razvoja baza podataka kako u smislu povećanja njihove učinkovitosti tako i u smislu povećanja njihove fizičke veličine (baze reda TB danas više nisu neuobičajene). Danas postoji težnja da se zbog mnogih razloga (sigurnosti, konzistentnosti, itd.) čim više tipova podataka pohrani u baze čime one postaju univerzalni repozitorij. Očito je da takva univerzalna baza nije upotrebljiva ukoliko se iz nje u realnom vremenu ne može dobiti odgovor. Pojam "realnog vremena odgovora" je vrlo rastezljiv, a kako su današnji korisnici sve razmaženiji i žele odgovor i prije nego su postavili pitanje, idealno rješenje bi bio sustav sa čim manjim kašnjenjem (najbolje nultim !). Po drugoj strani razvoj interneta/intraneta nameće kao standardno sučelje korisnik - baza upravo WEB aplikaciju kao najprimjereniju formu za pristup bazama podataka. Tema ovog članka je uz neka osnovna teorijska razmatranja i praktičan primjer iz realnog života kako se približiti navedenim idealima.

## 2. O normalizaciji

Ako prihvatimo činjenicu da je jedina stalna stvar danas promjena onda moramo tako organizirati sebe i druge tako da možemo momentalno reagirati na promjene. Da bi to bilo moguće potrebno je imati što manje redundantnih podataka, jer time smanjujemo vrijeme ažuriranja. Normalna forma trebala bi biti osnovno pravilo pri strukturiranju baza.

Normalizacija je proces spremanja podataka na pravo mjesto. Kad su podaci spremljeni samo na jednom mjestu, dohvaćanje mnoštva različitih, međusobno povezanih podataka obično zahtjeva odlazak na mnogo različitih mjesta. To usporava proces dohvata dok je s druge strane ažuriranje podataka brže jer se obavlja samo na jednom mjestu.

Uobičajeno se podrazumijeva da su sve relacijske baze normalizirane. Sa normaliziranim modelom podataka podatak je spremljen na jedno mjesto a podaci o jednom entitetu spremljeni su zajedno. Bez dubljeg ulaženja u problem normalizacije, kratke definicije prve tri normalne forme bi glasile:

1. normalna forma – svi entiteti moraju imati jedinstveni identifikator (ključ) koji se može sastojati od jednog ili više atributa. Svako polje u tablici mora sadržavati samo jednu vrijednost (atributi moraju biti jednostavni – ne smiju biti sastavljeni od više atributa)
2. normalna forma – svi atributi koji nisu dio ključa moraju u potpunosti ovisiti o njemu
3. normalna forma – svi atributi koji nisu dio ključa ne smiju biti međusobno ovisni

Osnovna pravila normaliziranih tablica:

- svako polje tablice trebalo bi predstavljati jedinstven tip informacije
- svaka tablica mora imati primarni ključ koji se sastoji od jednog ili više polja u tablici
- za svaku jedinstvenu vrijednost primarnog ključa mora postojati jedna i samo jedna vrijednost u bilo kojem stupcu podataka, a ta se vrijednost mora odnositi na subjekt tablice

- mora postojati mogućnost promjene bilo kojeg polja (osim polja u primarnom ključu) bez utjecaja na bilo koje drugo polje

## 3. O denormalizaciji

Često se, međutim, zahtijeva mogućnost brzog dohvaćanja podataka spremljenih u nekoj bazi, smanjenje kompleksnosti ili jednostavniji unos. Rješenje toga ponekad predstavlja denormalizacija baze. Denormalizacija znači spremanje istih podataka na više mjesta čime se povećava zalihost (redundancija). Time povećavamo brzinu odgovora na račun povećanja baze.

Ovime se naravno ne propagira denormalizacija jer normalizirana baza i dalje ostaje optimalno rješenje i treba ga inplementirati kad god je moguće. Ipak u realnim uvjetima denormalizacija je ponekad neophodna i ne predstavlja lošu odluku ako se pametno izvede. Prije denormalizacije treba razmotriti 3 pitanja:

1. da li sistem postiže prihvatljivu brzinu odgovora bez denormalizacije
2. da li će performanse sistema i nakon denormalizacije biti nezadovoljavajuće
3. da li će zbog denormalizacije smanjiti pouzdanost sistema

Ako je odgovor na bilo koje od prethodnih pitanja DA, tada treba izbjeći denormalizaciju jer će dobici biti minimalni u odnosu na moguće gubitke. U slučaju da se ipak odlučite na denormalizaciju, treba se pridržavati nekih preporuka.

Ako postoje mogućnosti, treba imati dvije baze: normaliziranu i denormaliziranu. Tablice u denormaliziranoj bazi nastaju kao sa rezultatima upita u normaliziranu bazu. Aplikacija može raditi sa tablicama u denormaliziranoj bazi (read–only) i tako postizati veće brzine izvođenja. Važno je držati obje baze sinhroniziranim.

Ako ne postoje mogućnosti za dvije baze, tada bi se denormalizirane tablice trebale održavati programski. Treba se voditi računa da se ažuriraju istovremeno sve tablice u kojima se nalaze ponovljeni podaci ili odrediti strogi raspored po kojem će se tablice sinhronizirati. U svakom slučaju svi korisnici koji mogu unositi podatke trebaju biti upoznati sa posljedicama nekonzistentnosti podataka.
Dizajn aplikacije mora biti takav da dozvoljava jednostavan prijelaz rada s denormaliziranih tablica na normalizirane.

## 4. Razlozi za denormalizaciju

Jedini valjan razlog za denormalizaciju je povećanje performansi. Postoji nekoliko indikatora koji ukazuju na tablice pogodne za denormalizaciju. To su:

- postoji puno upita ili izvještaja koji koriste podatke iz više tablica i izvršavaju se on-line
- potrebno je izvršiti puno računanja po jednom ili više stupaca za vrijeme upita
- tablicama pristupa istovremeno više korisnika sa različitim upitom
- postoji puno velikih primarnih ključeva kao polja u upitu
- velik dio vremena se upiti izvršavaju po određenim poljima (60% i više)

Svaka nova verzija RDBMS-a obično donosi povećanje performansi i poboljšan pristup koji tada možda smanjuje potrebu za denormalizacijom Ipak većina RDBMS-a

povremeno zahtijeva denormalizaciju strukture podataka. Postoji više različitih tipova denormaliziranih tablica koje rješavaju problem brzine odgovora koji se javlja kad se pristupa potpuno normaliziranim podacima. U tabeli 1 dane su preporuke kada koji od tipova denormalizacije koristiti.

| | |
|---|---|
| spojene tablice | ako su dobici na brzini veliki |
| tablice za izvještaje | za specijalizirane, važne i česte izvještaje |
| duple tablice | različite aplikacije koriste iste tablice |
| podjeljene tablice | različiti korisnici koriste različite dijelove tablice |
| kombinirane tablice | postoji relacija *jedan-prema-jedan* |
| redundantni podaci | smanjuje se količina tablica koje se spajaju zbog upita |
| ponavljanje grupa | smanjuju se količine izlaznih i ulaznih podataka |
| izračunati podaci | eliminira se računanje i izvođenje algoritama |

**Tablica 1.** Osnovni tipovi denormaliziranih tabela

## 4.1 Spojene tablice

Ako dvije ili više tablica treba spojiti aplikacijom ali je cijena problematična, treba uzeti u obzir mogućnost kreiranja tablice koja nastaje spajanjem više njih. Karakteristike takve tablice:

- ne smije sadržavati redundantne stupce
- sadrži samo ona polja koja su potrebna za aplikaciju (upit)
- kreira se periodički pomoću SQL-a

Upiti u takvu tablicu su vrlo efikasni jer ne uzrokuju ponovna privremena spajanja tablica.

## 4.2. Tablice za izvještaje (report tables)

Često je nemoguće napraviti end-user izvještaj koristeći samo SQL jer oni zahtijevaju posebnu manipulaciju podacima. Za često izvođene izvještaje postavlja se zahtjev njihovog on-line izvođenja, što podrazumijeva i kreiranje tablice iz koje se kreira. U tablicu se postavljaju SQL upiti dok se izvještaj stvara u pozadinskom procesu koristeći određene procedure (aplikacijski programi, SQL, itd). Tada se mogu u dijelovima ukrcavati u tablicu. Tablica za izvještaj treba:

- sadržavati samo jedan stupac za svaki stupac izvještaja
- imati indexe
- ne smije uticati na relacijsku strukturu (veze)

Oakve tablice su idealne za prikaz rezultata vanjskih veza ili kompleksnih SQL upita. Ako izvedemo upit sa *outer join* i rezultat spremimo u tablicu, tada do rezultata stižemo jednostavnom *select* naredbom umjesto kompleksnom *union*. Neki RDBMS podržavaju korištenje explicitne *outer join* funkcije umjesto *union* funkcije. Ipak ovisi o implementaciji koja će od ove dvije funkcije biti jednostavnija.

## 4.3. Mirror tablice

Ako je sistem jako aktivan potrebno je ponekad podijeliti proces na dva ili više odvojenih dijelova. To zahtijeva stvaranje duplih (mirror) tablica. Promotrimo aplikaciju koja ima veliki on-line promet tijekom jutra i ranog poslijepodneva. Promet se sastoji istovremeno od postavljanja upita i ažuriranja podataka. Tijekom poslijepodneva se ista aplikacija

koristi za **decision support proces**. Često rad s podacima ometa **DSP** i uzrokuje zagušenja ili prekidanje izvođenja. Problem se može riješiti kreiranjem duplih tablica. Tablice u prvom planu će se koristiti za produkciju, dok će one u pozadini služiti za izvještaje koji se koriste za donošenje odluka. Treba biti razrađen sistem seljenja tablica iz prvog plana u pozadinu i obratno da bi ostale sinhronizirane i da bi izvještaji na osnovu kojih se donose odluke bili ažurni a time i efikasniji.
Važno je zapaziti da su zahtjevi na tablice za oba procesa različiti i to treba uzeti u obzir kod kreiranja indexa u tablicama koje služe za izvještaje.

## 4.4. Podjela tablica (Split tables)

Ako odvojenim dijelovima normalizirane tablice pristupaju različite i odvojene grupe korisnika aplikacije tada se ona može podijeliti na jednu ili više denormaliziranih tablica, za svaku grupu korisnika posebna. Originalna tablica bi također trebala biti sačuvana i dostupna ako je koriste i drugi dijelovi aplikacije. Ovo bi se moglo smatrati posebnim slučajem duplih tablica. Tablice možemo dijeliti na dva načina: horizontalno i vertikalno. Vertikalna podjela dijeli stupce u različite tablice, dok horizontalna dijeli redove na osnovu raspona vrijednosti ključa.
Kod vertikalne podjele stupac koji sadrži primarni ključ trebao bi se nalaziti u obje nove tablice. Ako originalna tablica više ne postoji, jednu od novih tablica treba proglasiti roditeljskom (parent table), u suprotnom to je original. Mora postojati roditeljska tablica zbog referencijskog integriteta. Kada izvedemo vertikalnu podjelu za svaki primarni ključ mora postojati jedan red. Redove u novim tablicama ne smijemo brisati, jer u tom slučaju bespotrebno kompliciramo proces ažuriranja ili dohvaćanja podataka.
Kod horizantalne podjele redove trebamo podijeliti tako da izbjegnemo njihovo ponavljanje u obje tablice. To se najjednostavnije izvede tako da različite raspone vrijednosti primarnog ključa stavimo u različite tablice. Tada operacijom *union* ne bi smjeli niti izgubiri redove niti dobiti duple.

## 4.5. Kombinirane tablice

Ako su tablice povezane relacijom *jedan-prema-jedan* možemo ih povezati u jednu kombiniranu tablicu. Ponekad se i tablice u *jedan-prema-više* relaciji mogu spojiti u jednu, ali time proces ažuriranja postaje znatno kompliciraniji zbod povećane redundantnosti. Uzmimo na primjer dvije tablice: *zaposlenici* i *odjeli* i spojimo ih u jednu zajedničku veliku tablicu *zaposlenici_i_odjeli*. Nova tablica će sadržavati sva polja iz obje tablice osim polja po kojem je spajano (on neće biti dupli). Dakle, u dodatku svih informacija o svakom zaposleniku imati ćemo i sve informacije o odjelu. To će rezultirati mnogim ponovljenim podacima o odjelima. Kombrniranje ovog tipa trebamo smatrati kao prethodno spajanje i tako ga koristiti. Za tablice u relaciji *jedan-prema-jedan* uvijek treba analizirati i vidjeti da li bi njihova kombinacija bila efikasnija.

## 4.6. Redundantni podaci

Ponekad se jednom ili više stupaca jedne tablice pristupa uvijek kad se pristupa podacima druge tablice. Ako se pojedinim colonama na takav način pristupa češće nego samoj tablici u kojoj se nalaze tada bi trebalo razmisliti o uključivanju tih stupaca u tablicu koja ih poziva kao redundantnih podataka. Dodavanjem tih dodatnih stupaca, možemo eliminirati veze i brzina dohvata podataka će biti znatno veća. Uzmimo opet za

primjer spomenute tablice. Ako većina upita o zaposlenicima zahtijeva i ime odjela onda bi njega trebalo dodati kao redundantni podatak u tablicu zaposlenici. Isti stupac ne bi trebalo brisati iz tablice odjeli u slučaju ažuriranja zbog npr. promjene naziva odjela.
Stupci koji bi mogli biti redundantni trebali bi imati slijedeća svojstva:

- treba ih biti samo nekoliko
- trebaju biti stabilni, rijetko se ažuriraju
- treba ih koristiti ili jako veliki broj korisnika ili nekoliko jako važnih korisnika

## 4.7. Ponovljene grupe (repeating groups)

Kada normaliziramo ponovljene grupe koristimo ih kao pojedinačni red a ne stupac. To rezultira većim zauzimanjem prostora i sporijim dohvaćanjem jer je više redova u tablici i više ih se treba pročitati da bi se odgovorilo na upit. Ponekad, denormalizacija podataka na način da se spremaju u stupce značajno povećava performanse (npr. kvartalne prodaje proizvoda: bolje je za svaki kvartal dodati stupac (ostaje jedan red po proizvodu), nego upisivati novi red (svi se podaci ponavljaju osim novog podatka za novi kvartal). Ipak, ovdje povećanje performansi ide na uštrb fleksibilnosti jer je broj stupaca ograničen (po pretpostavci). Prije odluke da se ponovljene grupe spremaju kao stupci slijedeći uvjeti moraju biti zadovoljeni:

- podaci se gotovo nikad ne spajaju, uspoređuju ili uzimaju srednju vrijednost unutar reda
- podaci se javljaju po točno određenom pravilu određen broj puta
- obično im se pristupa zajedno
- podaci imaju predvidiv obrazac popunjavanja i brisanja

Ako neki od ovih uvjeta nije zadovoljen SQL *select* će se teško definirati i podaci će biti teže dostupni. U tom slučaju bi ovaj način trebalo izbjegavati, jer je pretpostavka da denormalizacijom upiti postaju brži.

## 4.7. Izračunati podaci (derivable data)

Ako je za podatak potrebno izračunavati komplicirane formule, možda je jednostavnije u tablicu upisati rezultat nego ga računati. Pri tome treba voditi računa o tome da kad se promijeni neki dio formule, tada treba preračunati upisane vrijednosti da bi informacije ostale konzistentne. To može imati znatan uticaj na efikasnost i pouzdanost podataka. Ponekad je nemoguće odmah ažurirati rezultat po promjeni podataka o kojima ovisi. To se dešava u slučaju da je tablica koja sadrži rezultate u off-line modu. U tom slučaju promjenu podataka treba izvršiti čim tablica bude spremna za ažuriranje. Ni u kojem slučaju neažurirani podaci ne smiju biti dostupni za upite ili izvještaje.

## 5.Praktični primjeri – TIS 2 relacijska baza podataka

U TKC Rijeka se koristi u radu više službi WEB aplikacija TIS2. Sama koncepcija sustava i aplikacija više puta su od strane autora prezentirani na proteklim savjetovanjima KOM i CASE, pa bi na ovom mjestu samo ponovili najbitnije. TIS (telekomunikacijski informacijski sustav) je tehnička aplikacija koja služi kao osnovna podrška većini odjela TK centra. U TIS-u su pohranjeni svi relevantni podaci za eksploataciju, praćenje i analizu stanja TK mreže na području TKC Rijeka. Kako bi se olakšalo korištenje podataka iz baze (sama aplikacija TIS je terminalskog tipa realizirana u Supri i ima ograničene mogućnosti prezentacije podataka), te omogućilo

povezivanje podataka iz TIS-a sa ostalim bazama podataka TKC koje su relacijskog tipa, odlučeno je realizirati novi sustav TIS2. TIS2 sastoji se od relacijske baze podataka MS SQL Server 6.5 u koji se tijekom noći prebacuju kompletni podaci iz TIS-a, te WEB aplikacije koja služi kao sučelje prema bazi podataka.
Organizacija podataka (tabele, pogledi, indeksi itd.) obavljena je prema svim normativima uspostave relacijskih stuktura (normalne forme itd.) koje smo spomenuli u drugom poglavlju. Međutim u tijeku eksploatacije sustava pojavili su se određeni problemi kod korištenja aplikacije u obliku predugih čekanja na odgovore iz baze (u najgorem slučaju 20 do 40 sekundi što je ocijenjeno neprihvatljivim za ugodan rad). Na ovom mjestu potrebno je napomenuti da je poslužitelj obično računalo Pentium 200 sa 128 MB RAM memorije i standardnim SCSI diskovima. Kako se TIS2 sastoji od preko četrdeset tabela, a svi korisnički upiti postavljani su u dvadesetak pogleda koji povezuju od 3 do 11 tabela, jasno je da se sa navedenim HW nisu mogla postići tražena vremena odziva.

Navedeni problem može se riješiti na dva načina:

a) "grubom silom" (nabaviti višeprocesorsko računalo koje je u stanju "samljeti" tražene upite u realnom vremenu)
b) izvršiti denormalizaciju odnosno pretvoriti postojeće poglede u tabele te time nauštrb prostora olakšati "posao" poslužitelju

Kako Grupa za tehničku dokumentaciju TKC Rijeka (nažalost !) ne raspolaže sa neograničenim financijskim sredstvima odlučili smo se za drugu varijantu. Rezultati su bili iznad očekivanja i u potpunost su opravdali učinjene zahvate.

Kao primjer velikog povećanja odziva sustava dajemo pet karakterističnih denormaliziranih tabela optimiziranih za pristup preko WEB aplikacije TIS2. Odabrano je nekoliko pogleda i tabela sa različitim brojem zapisa, na koje je postavljen upit tipa:

*select * from (tabela ili view), limits = Row Count: 100*

Upit je postavljen na normalizirane i denormalizirane tabele. Iz primjera će biti vidljivo da sa opadanjem broja podataka odnosno spojenih tabela pada i značaj denormalizacije tako da ona za male tabele postaje upitna.

a) PRIMARNE PARICE 263930 zapisa

**upit u denormalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **0 ms**.
Table: QPPARICE scan count 1, logical reads: 5, physical reads: 4, read ahead reads: 0
SQL Server Execution Times: cpu time = 0 ms. elapsed time = **289 ms**.

**upit u normalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **40 ms**.
Table: ZAUZPAR scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: CENTRALE scan count 100, logical reads: 298, physical reads: 3, read ahead reads: 0
Table: PPARICE scan count 1, logical reads: 23, physical reads: 5, read ahead reads: 43
Table: ULICE scan count 0, logical reads: 0, physical reads: 0, read ahead reads: 0
Table: PRETPLAT scan count 1, logical reads: 11350, physical reads: 59, read ahead reads: 11291
Table: Worktable scan count 100, logical reads: 300, physical reads: 19, read ahead reads: 0
SQL Server Execution Times: cpu time = 55 ms. elapsed time = **23024 ms**.

## b) PRETPLATNICI 128836 zapisa

**upit u denormalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **0 ms**.
Table: QPRETPLAT scan count 1, logical reads: 19, physical reads: 5, read ahead reads: 39
SQL Server Execution Times: cpu time = 8 ms. elapsed time = **1640 ms**.

**upit u normalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **507 ms**.
Table: STATUS scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: POBLOZNA scan count 334, logical reads: 667, physical reads: 2, read ahead reads: 0
Table: PRIJEN scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: VRSTPRI scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: KABLOVI scan count 334, logical reads: 6580, physical reads: 10, read ahead reads: 0
Table: RSS scan count 333, logical reads: 999, physical reads: 3, read ahead reads: 0
Table: CENTRALE scan count 334, logical reads: 1000, physical reads: 3, read ahead reads: 0
Table: VRSTURE scan count 334, logical reads: 667, physical reads: 2, read ahead reads: 0
Table: KATEGOR scan count 334, logical reads: 334, physical reads: 1, read ahead reads: 0
Table: ULICE scan count 334, logical reads: 1002, physical reads: 44, read ahead reads: 0
Table: MJESTA scan count 334, logical reads: 667, physical reads: 2, read ahead reads: 0
Table: PRETPLAT scan count 2, logical reads: 11384, physical reads: 704, read ahead reads: 10646
SQL Server Execution Times: cpu time = 16 ms. elapsed time = **10031 ms**.

## c) SEKUNDARNE PARICE 51563 zapisa

**upit u denormalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **0 ms**.
Table: QSPARICE scan count 1, logical reads: 5, physical reads: 0, read ahead reads: 0
SQL Server Execution Times: cpu time = 8 ms. elapsed time = **273 ms**.

**upit u normalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **32 ms**.
Table: CENTRALE scan count 100, logical reads: 298, physical reads: 0, read ahead reads: 0
Table: ZAUZPAR scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: SPARICE scan count 1, logical reads: 8, physical reads: 5, read ahead reads: 11
Table: ULICE scan count 0, logical reads: 0, physical reads: 0, read ahead reads: 0
Table: PRETPLAT scan count 1, logical reads: 11350, physical reads: 0, read ahead reads: 0
Table: Worktable scan count 100, logical reads: 407, physical reads: 95, read ahead reads: 0
SQL Server Execution Times: cpu time = 55 ms. elapsed time = **21601 ms**.

## d) PRIMARNI OBJEKTI 10399 zapisa

**upit u denormalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **0 ms**.
Table: QPOBJEKTI scan count 1, logical reads: 11, physical reads: 5, read ahead reads: 43
SQL Server Execution Times: cpu time = 0 ms. elapsed time = **742 ms**.

**upit u normalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **117 ms**.
Table: ELASTPAR scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: VRSTOBJ scan count 1, logical reads: 1, physical reads: 1, read ahead reads: 0
Table: VLASNOBJ scan count 100, logical reads: 100, physical reads: 1, read ahead reads: 0
Table: POBLOZNA scan count 100, logical reads: 199, physical reads: 2, read ahead reads: 0
Table: MJESTA scan count 100, logical reads: 199, physical reads: 2, read ahead reads: 0
Table: CENTRALE scan count 100, logical reads: 298, physical reads: 0, read ahead reads: 0
Table: POBJEKTI scan count 1, logical reads: 81, physical reads: 5, read ahead reads: 107
Table: VRSTOPR scan count 100, logical reads: 100, physical reads: 1, read ahead reads: 0
Table: ULICE scan count 100, logical reads: 300, physical reads: 54, read ahead reads: 0
SQL Server Execution Times: cpu time = 0 ms. elapsed time = **836 ms**.

## e) SEKUNDARNI OBJEKTI 2698 zapisa

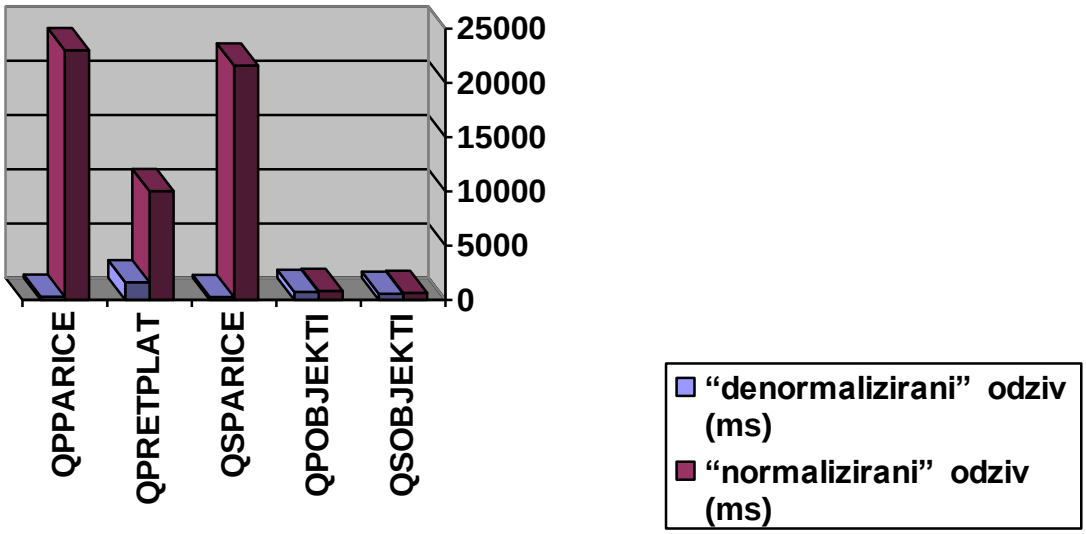
**upit u denormalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **0 ms**.
Table: QSOBJEKTI scan count 1, logical reads: 11, physical reads: 5, read ahead reads: 43
SQL Server Execution Times: cpu time = 16 ms. elapsed time = **571 ms**.

**upit u normalizirane tabele**

SQL Server Parse and Compile Time: cpu time = **101 ms**.
Table: VRSTOBJ scan count 1, logical reads: 1, physical reads: 0, read ahead reads: 0
Table: VLASNOBJ scan count 100, logical reads: 100, physical reads: 0, read ahead reads: 0
Table: POBLOZNA scan count 100, logical reads: 199, physical reads: 0, read ahead reads: 0
Table: MJESTA scan count 100, logical reads: 199, physical reads: 0, read ahead reads: 0
Table: CENTRALE scan count 100, logical reads: 298, physical reads: 0, read ahead reads: 0
Table: SOBJEKTI scan count 2, logical reads: 244, physical reads: 6, read ahead reads: 188
Table: VRSTOPR scan count 100, logical reads: 100, physical reads: 0, read ahead reads: 0
Table: ULICE scan count 100, logical reads: 300, physical reads: 15, read ahead reads: 0
SQL Server Execution Times: cpu time = 0 ms. elapsed time = **641 ms**.

| Naziv tabele | "denormalizirani" odziv (ms) | "normalizirani" odziv (ms) |
|---|---|---|
| QPPARICE | 289 | 23024 |
| QPRETPLAT | 1640 | 10031 |
| QSPARICE | 273 | 21601 |
| QPOBJEKTI | 742 | 836 |
| QSOBJEKTI | 571 | 641 |




## 6. Zaključak

Prethodni primjeri pokazali su vrlo velika povećanja odziva sustava čak do **80 puta** uz istovremeno povećanje baze podataka za samo **35 %** (sa 180 MB na 240 MB) čime su u potpunosti opravdani razlozi za denormalizaciju iz točke 4. Kako se na strani korisničkog sučelja ništa nije promijenilo (izvršena je samo redirekcija upita sa view-ova na novo definirane tabele), korisnici su reorganizaciju baze podataka doživjeli samo kroz drastično ubrzanje same WEB aplikacije. Svaki upit u bazu podataka daje trenutan odgovor što je značajno povećalo efikasnost službi koje koriste TIS2 aplikaciju u svakodnevnom radu. Ponovimo da je poslužitelj klase P200 sa 128 MB RAM-a i

očigledno je da je denormalizacija u ovom slučaju daleko efikasniji način ubrzanja aplikacije nego nadogradnja na novi HW. Kao usporedbu navedimo da sustav TIS opslužuje 4 procesorski Alpha server sa 512 MB RAM i RAID sustavom diskova, a odziv sustava na slične upite je daleko slabiji.

TIS2 je sustav koji služi uglavnom za pozadinske analize i generiranje izvještaja kojih obrada ili dugo traje ili ih nije moguće dobiti iz TIS-a. Time je izvršeno vidljivo rasterećenje i povećana brzina odziva starog sustava što je značajan nusprodukt uspostave novog sustava.

Iako smo vidjeli nesumljive učinke denormalizacije relacijske baze podataka, odluka o denormalizaciji ne smije se donijeti bez prethodne razrade jer ona zahtijeva mnogo administracije. Treba se dokumentirati sve odluke vezane za tip denormalizacije, osigurati točnost podataka, napraviti raspored ažuriranja i dijeljenja podataka (to se mora pažljivo i detaljno isplanirati i dokumentirati zbog povećane redundantnosti) te redovito obavještavati korisnike o stanju baze. Osim ovoga treba vršiti periodičke analize kojima se provjerava konzistentnost i točnost baze.

Kad god imamo denormalizirane podatke aplikacija se mora periodički testirati. Hardver i softver napreduju i mijenjaju se a time i potreba za denormalizacijom. Za odluku da li je denormalizacija i dalje potrebna treba uzeti u obzir ova pitanja:

- da li nova verzija DBMS-a nudi nove načine manipulacije tablicama i time se gubi potreba za npr. spojenim tablicama
- da li novi hardver povećava performanse: npr da li novi procesor tako povećava brzinu procesiranja da se gubi potreba za denormaliziranim podacima, ili povećana memorija omogućuje brže pristupanje podacima

Općenito, treba periodički ispitivati da li dodatni troškovi obrade denormaliziranih podataka opravdavaju prednosti denormalizacije, tj. promatramo:

- brzinu dohvaćanja podataka
- rasterećenje procesora
- kompleksnost ažuriranja
- troškove povratka na normalizirane podatke

Važno je zapamtiti da je najvažniji razlog za denormalizaciju povećanje performansi. Ako se okruženje mijenja tada treba ponovo procijeniti potrebu za denormalizacijom jer je moguće da sa novim hardevrom ili softverom denormalizirani podaci uzrokuju smanjenje performansi.

koraljka.brlas@hpt.hr
damir.medved@hpt.hr

HPT TKC RIJEKA
Erazma Barčića 5
51000 Rijeka