ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA I ANALIZA OGRANIČENJA U RELACIONIM BAZAMA PODATAKA VEB SAJTOVA

–Master rad –

Kandidat:

Nenad Živanović 2012/3258

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2015.

SADRŽAJ

1. UVOD

U raznim metodologijama razvoja softvera, testiranje predstavlja nezaobilazni korak. U toj fazi se mogu identifkovati svi problemi, nepravilnosti i nedostaci aplikacije. Testiranjem različitih implementacija istih funkcionalnosti i merenjem određenih parametara može se znatno povećati kvalitet softvera.

Prilikom kreiranja baze podataka postoji nekoliko opcija za implementaciju ograničenja nad bazom podataka. Svaki od načina ima svoje prednosti i mane. U zavisnosti od konkretnog slučaja, osoba koja kreira bazu podataka će se odlučiti za odgovarajući način. Da bi se odlučio za pravi način, mora da zna koje su razlike u implementacijama i u kojim slučajevima je dobro koristiti koji način. Ukoliko se izabere pravi način, performanse baze podataka se mogu znatno povećati.

Predmet ovog rada je različita implementacija ograničenja nad bazom podataka. Opisan je životni ciklus softera i način projektovanja baza podataka. Akcenat je stavljen na fazu analize i na detaljan opis jednog od najpoznatijih modela podataka – MOV (model objekti-veze) (en. *entity–relationship model (ER model*). Takođe, opisana su i ograničenja nad bazom podataka koja su podeljena u nekoliko kategorija.

U trećem poglavlju će biti dat kratak opis tehnologija u kojoj je implementiran studijski primer. Opisan je SQL Server, način na koji on izvršava i optimizuje upite i Profiler, alat sa kojim su vršenja merenja.

U četvrtom poglavlju će se demonstracijom kroz primer proći kroz svih 5 osnovnih faza životnog ciklusa softvera. U ovom poglavlju biće opisana i imeplementacija sa veb sajtom koji je povezan na bazu podataka koja će biti korišćena tokom testiranja. Biće formalno opisana ograničenja koja će biti implementirana. Ograničenja će se implementirati putem *Trigger*-a i putem *Stored Procedure*-a. Biće data detaljna analiza performansi po vrstama ograničenja. Nakon detaljnih analiza, biće dat uporedni prikaz performansi, prednosti i mana različitih načina implementacija.

2. SPECIFIKACIJA SOFTVERA

U ovom poglavlju biće opisan jedan od načina za specifikaciju softvera. Akcenat će biti stavljen na MOV.

2.1. Životni ciklus softverskog sistema

Razvoj (životni ciklus) softverskog sistema se sastoji iz sledećih faza:

1) Prikupljanja zahteva od korisnika
2) Analize
3) Projektvanja
4) Implementacije
5) Testiranja

- U fazi prikupljanja zahteva se definišu svojstva i uslovi koje softverski sistem ili šire gledajući projekat treba da zadovolji.
- U fazi analize se dobija struktura i sistemske operacije sistema
- U fazi projektovanja se pravi arhitektura softverskog sistema
- Implementacione komponente, iz faze implementacije, treba da realizuju komponente koje su dobijene u fazi projektovanja
- Svaka od implementacionih komponenti se testira u fazi testiranja.

2.2. Model objekti-veze

Model podataka je intelektualno sredstvo za opis statičkih karakteristika sistema, opis karakteristika sistema u nekom stacionarnom stanju. Stacionarno stanje nekog sistema karakteriše se skupom zavisnosti koje postoje između objekata sistema. Ove zavisnosti se, u modelu podataka, mogu predstaviti bilo strukturom podataka, bilo skupom ograničenja na vrednosti podataka. Pored toga, neophodno je definisati i skup operacija modela podataka, da bi se preko njih, u modelima procesa, mogla opisati i dinamika realnog sistema. Zbog toga svaki model podataka poseduje tri osnovne komponente:

1) Strukturu modela, odnosno skup koncepata za opis objekata sistema njihovih atributa i njihovih međusobnih veza.
2) Ograničenja - semantička ograničenja na vrednosti podataka koja u svakom stacionarnom stanju moraju biti zadovoljena. Ova ograničenja se obično nazivaju pravilima integriteta modela podataka.
3) Operacije nad konceptima strukture, pod definisanim ograničenjima, preko kojih je moguće opisati dinamiku sistema u modelima procesa.

Model objekti-veze je najpopularniji i u praksi najviše korišćeni semantički model podatka. Postoji više različitih verzija ovog modela. Ovde će biti opisana jedna specifična verzija ovog

modela. Model objekti-veze (MOV) u kome se definišu i jezik za specifikaciju ograničenja i operacije modela, čime se dobija alat za formalnu specifikaciju. Postoje i drugi modeli podataka, jedan od popularnijih je UML dijagram klasa.

2.3. Struktura Modela objekti-veze

Struktura MOV predstavlja se dijagramima objekti-veze (DOV).

2.3.1. Objekti i veze

U modelu MOV sistem se opisuje kao skup objekata i njihovih veza.

Koristeći apstrakciju klasifikacije, pojedinačni objekti u sistemu se klasifikuju u tipove objekata. Tip objekta je opšti predstavnik neke klase, a svaki pojedinačni objekat predstavlja jedno pojavljivanje (primerak) datog tipa. Klasa objekata je skup pojavljivanja objekata datog tipa. Na dijagramima objekti veze (DOV) klase objekata se prikazuju pravougaonicima. Svaka klasa definiše istovremeno i tip objekta.

Veze u modelu opisuju način povezivanja (uzajamna dejstva) objekata. Apstrakcija klasifikacije može se primeniti i na veze i definisati pojmovi tipa, klase i pojavljivanja veze. Klase veza se na DOV predstavljaju sa rombovima. U MOV se pretpostavlja korišćenje samo binarnih veza, veza između dva tipa objekata. Svaki tip veze između dva tipa objekata e1 i e2 definiše dva tipa preslikavanja, preslikavanje sa skupa pojavljivanja (klase) E1 u skup pojavljivanje E2 (E1 je domen, a E2 kodomen preslikavanja) i (inverzno) preslikavanje sa skupa E2 u skup E1.

Veze se mogu uspostavljati i između pojavljivanja objekata istog tipa (nad istom klasom objekata). Isto tako moguće je uspostaviti više različitih tipova veza između istih tipova objekata.

Očigledno je da dva preslikavanja definišu jednu binarnu vezu, pa je koncept veze izvedeni pojam, što ne znači i da je potpuno redudantan, odnosno da bi ga trebalo izostaviti. Nazivom veze se uspostavlja odnos između dva međusobno inverzna preslikavanja. Veza predstavlja agregaciju dva objekta, i može se tretirati kao poseban agregirani objekat.

Jedna od bitnih karakteristika veza između objekata je kardinalnost preslikavanja koja je čine. Kardinalnost preslikavanja E1 --> E2 definiše se parom (DG, GG), gde DG (donja granica) daje najmanji mogući, a GG (gornja granica) najveći mogući broj pojavljivanja tipa objekata E2, za jedno pojavljivanje tipa objekata E1. DG može imati vrednost 0, 1 ili neki poznat ceo broj > 1. GG može imati vrednost 1, neki poznat ceo broj > 1, ili nepoznat ceo broj > 1 koji se označava sa M. Očigledno je da u jednom preslivanju mora biti zadovoljeno DG GG. Bez obzira da li se naziv preslikavanja izostavlja ili ne, kardinalnost preslikavanja se uvek mora definisati.

2.3.2. Atribut i domen

Objekti u sistemu se opisuju preko svojih svojstava, odnosno atributa. Svaki atribut u jednom trenutku vremena ima neku vrednost. Atributi uzimaju vrednost iz skupa mogućih vrednosti. Ovi skupovi se nazivaju domenima.

Domeni mogu biti:

- predefinisani, odnosno standardni programsko-jezički domeni, kao što su *integer, character, real, logical i date*.
- semantički, kada se definišu posebno, preko svoga imena, predefinisanog domena i, eventualno, ograničnenja na mogući skup vrednosti predefinisanog domena.

Pored ograničenja na vrednosti atributa, odnosno vrednosti domena koja su data u primerima definišu se i druga. Ograničenja mogu biti prosta i složena. Lista dozvoljenih prostih ograničenja je:

1) konstanta, gde je bilo koji operator poređenja koji se na datom domenu može definisati (na primer, <, , >, , = za brojne domene), a konstanta je neka definisana vrednost iz datog domena.
2) *between* konstanta , konstanta, gde su konstante vrednosti iz datog domena.
3) *in* (lista vrednosti), gde se lista formira od konstanti iz odgovarajućeg domena
4) *not null*, kada dato polje ne može da dobije "nula vrednost", odnosno mora uvek da ima vrednost.

Pretpostavlja se da svaki domen uključuje u sebe i "nula vrednost", specijalnu vrednost koja se dodeljuje nekom atributu objekta kada se njegova vrednost ne poznaje (ili, preciznije, još uvek ne poznaje).

Složena ograničenja se formiraju od prostih ili drugih složenih ograničenja vezujući ih logičkim operatorima *and*, *or* i *not*.

Bilo prosto, bilo složeno ograničenje se može imenovati, odnosno posebno definisati kao Bullova (logička funkcija) i samo ime navesti kao ograničenje.

2.4. Ograničenja u MOV

U prethodnom delu, pri uvođenju i definisanju pojedinih koncepata strukture MOV, definisana su i neka ograničenja - kardinalnost preslikavanja i ograničenja na vrednosti domena. Međutim , postoje i mnogo složenija semantička ograničenja (uzajamni odnosi objekata i njihovih atributa) koje je nemoguće ili nepraktično prikazati strukturom MOV, pa je neophodno definisati jezik za njihovu specifikaciju. Pre nego što uvedemo jezik za specifikaciju ograničenja, pogodno je klasifikovati ograničenja u sledeće klase:

- Strukturna ograničenja (ograničenja na preslikavanja)
- Vrednosna ograničenja (ograničenja na vrednosti atributa)

2.4.1. Strukturna ograničenja

Sama struktura MOV, odnosno dijagram objekti veze iskazuje ova ograničenja. Na svakom DOV zadate su kardinalnosti preslikavanja za sve veze. Isto tako, s obzirom na to da se koriste samo jednoznačni atributi, da svaki atribut mora biti primenljivo svojstvo za sva pojavljivanja posmatranog objekta i da su identifikatori objekata označeni, implicitno su definisane kardinalnosti direktnog (atributa) i inverznog preslikavanja *objekat*----*>domen*.

Pored ovih ograničenja, u ređim slučajevima je potrebno definisati i ograničenje tipa redosled za preslikavanja čija je gornja granica kardinalnosti GG > 1, koje definiše zahtevani redosled povezivanja pojavljivanja objekta kodomena preslikavanja sa datim pojavljivanjem objekta domena preslikavanja.

2.4.2. Vrednosna ograničenja

Mada se u osnovi zadaju na isti način, iz praktičnih razloga je pogodno ovu klasu ograničenja podeliti na dve podklase: ograničenja na domene i medjuzavisnosti atributa. Ograničenja na domene se jednostavno iskazuju i daju se uz definiciju domena kako je to ranije pokazano. Za iskazivanje medjuzavisnosti atributa koriste se formule *objektnog računa*, predikatskog računa prvog reda u kome su promenljive objekti u sistemu.

Osnovni pojmovi predikatskog računa su:

1) Afirmativna rečenica, koja ima smisla i koja je istinita ili neistinita naziva se sud.
2) Afirmativna rečenica koja ima smisla i koja sadrži jedan ili više promenljivih parametara i koja postaje sud uvek kada parametri iz rečenice dobiju konkretnu vrednost naziva se predikat. Broj parametara u predikatu se naziva dužina predikata.
3) Promenljive u objektnom računu su objekti odnosno, promenljiva može da uzima vrednost iz neke klase objekta
4) Predikatski ili kvantifikatorski račun je matematička teorija čiji su objekti formule koje predstavljaju predikate. Simboli koji se koriste da označe neki sud nazivaju se atomskim formulama ili atomima.

2.5. Operacije u MOV

Pogodno je operacije u MOV podeliti na operacije održavanja (ažuriranja) baze podataka i operacije pretraživanja (izveštavanja).

2.5.1. Operacije održavanja

Iz same strukture MOV očigledno je da se operacije održavanja baze podataka svode na operacije dodavanja novog objekta u klasu, izbacivanja objekta iz klase, izmenu vrednosti nekog atributa, zatim povezivanja, razvezivanja i prevezivanja dva objekta preko zadatog preslikavanja, odnosno veze. Sintaksa i detaljnije objašnjenje ovih operacija slede:

1) Ubacivanje - *Insert*

U listi parova atribut-vrednost neophodno je zadati sve atribute koji ne mogu imati nula vrednost. Podrazumeva se da je među njima i atribut identifikator objekta, jer on, očigledno ne može da ima nula vrednost.

Podrazumeva se da operacija ubaci prouzrokuje operacije poveži prema svim okolnim objektima prema kojima posmatrani objekat ima obavezno preslikavanje.

2) Izbacivanje - *Delete*

Podrazumeva se da operacija izbaci prouzrokuje operacije razveži za sva preslikavanja u kojima učestvuje posmatrani objekat.

3) Ažuriranje - *Update*

Atributi pojavljivanja objekta zadatog identifikatorom dobijaju zadate vrednosti. Identifikator objekta se ne može ažurirati.

4) Povezivanje

Operacija povezivanja se ne može direktno primeniti na preslikavanja čije su kardinalnosti DG = 1 i GG = 1, jer se to povezivanje ostvaruje automatski sa ubacivanjem objekta domena preslikavanja.

Podrazumeva se da operacija poveži za jedno preslikavanje prouzrokuje i operaciju poveži za njeno inverzno preslikavanje.

5) Razvezivanje

Operacija razvezivanja se ne može direktno primeniti na preslikavanja čije su kardinalnosti DG = 1 i GG = 1 inače bi uvek narušila uslov integriteta.

Podrazumeva se da operacija razveži za jedno preslikavanje prouzrokuje i operaciju razveži za njeno inverzno preslikavanje.

6) Prevezivanje

Operacija prevezivanja se sastoji od uzastopne primene operacije preveži sa jednog kodomena i poveži na drugi, u jednoj atomskoj transakciji. Ona se primenjuje na ona preslikavanja na koja se operacije razvezivanja i povezivanja ne mogu direktno primeniti.

2.5.2. Operacije pretraživanja

U MOV je moguće definisati dve vrste operacija za pretraživanje. Prvi skup operacija je navigacione prirode, slično operacijama u mrežnom i hijerarhijskom modelu, i one omogućuju kretanje kroz model i pristup pojedinačnim pojavljivanjima objekata tekuće klase, klase kojoj se u tom trenutku pristupilo. Drugu grupu operacija čine tzv. specifikacione operacije, odnosno odgovarajući upitni jezik, kojima se iskazuje struktura i uslovi koje rezultat operacije treba da zadovolji.

2.6. Dinamička pravila integriteta

U prethodnim poglavljima definisane su različite vrste ograničenja, odnosno pravila integriteta.

Ova pravila su statička, odnosno daju uslove koje podaci u bazi podataka treba da zadovolje u stacionarnom stanju, odnosno po okončanju bilo koje transakcije. Međutim, za potpunu specifikaciju baze podataka nekog informacionog sistema, pored ograničenja od interesa je i akcija koju treba preduzeti kada neka operacija ažuriranja baze podataka naruši definisano ograničenje. Ovakva specifikacija daje se preko tzv. dinamičkih pravila integriteta. Jedno dinamičko pravilo integriteta čini trojka

<OPERACIJA, OGRANIČENJE, AKCIJA>

preko koje se, za svaku *operaciju* održvanja baze podataka i svako *ograničenje* koje ona može da naruši, definiše *akcija* koju treba preduzeti ako je ograničenjne narušeno.

Kako smo izvršili podelu ograničenja na ograničenja na domene atributa, složena vrednosna ograničenjnja i strukturna ograničenja, na isti način možemo podeliti i dinamička pravila integriteta.

2.6.1. Dinamička pravila integriteta za atribute

Ograničenja na vrednosti atributa mogu narušiti Operacije održavanja *insert* i *update*. Očigledno je da će se, bez obzira na to koja je od ovih operacija narušila ograničenje, preduzeti ista akcija. Zbog toga u definisanju dinamičkog pravila integriteta ove vrste nije neophodno navoditi operaciju. Ranije je pokazano da je najpraktičnije ograničenje na atribute prikazati preko tabele sa kolonama *atribut, domen, ograničenje*.

Za iskazivanje dinamičkog pravila integriteta za atribute dovoljno je ovu tabelu proširiti sa kolonom *akcija*.

2.6.2. Složena vrednosna dinamička pravila integriteta

Kao što je ranije rečeno složena vrednosna ograničenja definišu međuzavisnosti vrednosti različitih atributa i specifikuju se pomoću iskaza objektnog računa. Za svako vrednosno ograničenje i listu operacija koje ga mogu narušiti definiše se akcija koju treba preduzeti kada je ograničenje narušeno.

2.6.3. Strukurna dinamička pravila integriteta.

Strukturna ograničenja u MOV definisana su samom strukturom modela, odnosno načinom međusobnog povezivanja različitih klasa objekata. Operacije održavanja baze podataka mogu da naruše strukturna ograničenja narušavajući kardinalnost preslikavanja preko kojih su ostvarene veze između klasa objekata. Pri definiciji semantike pojedinih operacija održavanja baze podataka pretpostavljeno je sledeće:

- Operacija insert poziva automatsko izvršavanje operacije *connect* za sva obavezna preslikavanja (preslikavanja sa DG > 0) koja poseduje posmatrana klasa;
- Operacija delete poziva automatsko izvršavanje operacije *disconnect* za sva preslikavanja date klase;
- Operacija *reconnect* se svodi na uzastopnu primenu operacija disconnect i connect, u jednoj atomskoj transakciji;
- Operacije *connect* i *disconnect* na jednom preslikavanju podrazumevaju da se odgovarajuća operacija obavi i na njemu inverznom preslikavanju;

Pretpostavlja se da se operacija update neće primenjivati na identifikator objekta.

Imajući ovo u vidu, očigledno je da strukturna ograničenja mogu narušiti samo operacije connect i disconnect, bilo da se neposredno primenjuju, bilo da su automatski indukovane od drugih operacija, pa je samo za ove operacije potrebno definisati dinamička strukturna pravila integriteta.

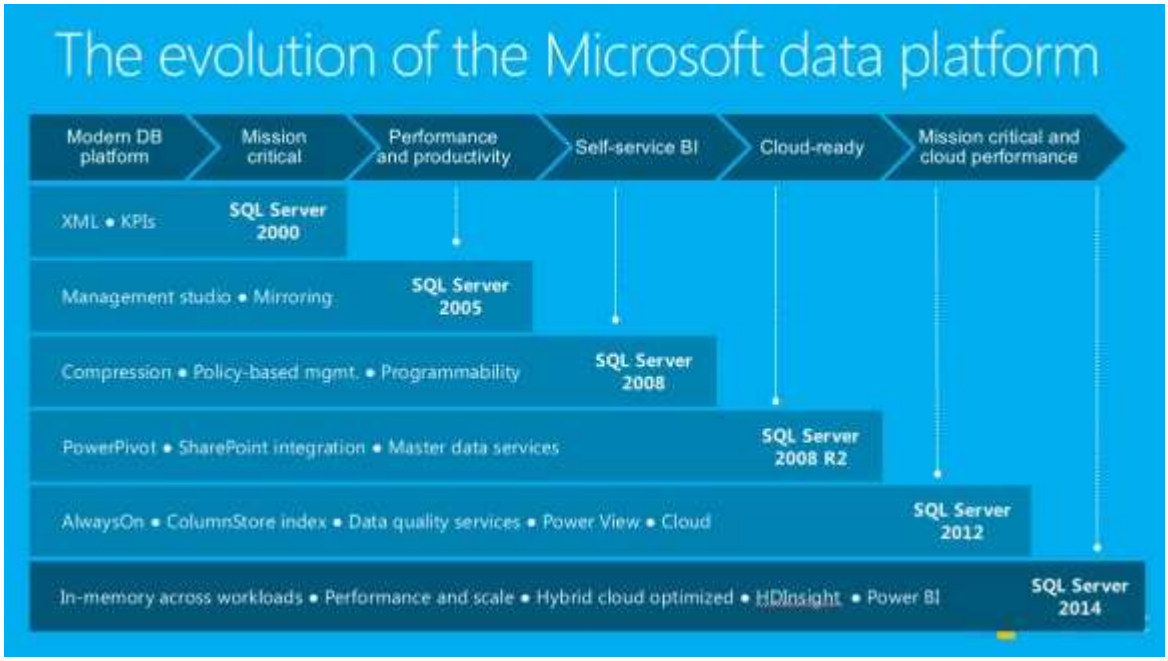
3. IMPLEMENTACIONE TEHOLGIJE

Za implementaciju navedenih ograničenja korišćen je SQL Server, kao sistem za upravljanje bazom podataka. Verzija SQL Servera koja je korišćena je 2008 R2. Za testiranje i merenje rezulata upita korišćen je Express Profiler v2.0.

Prilikom implementacije veb sajta korišćene su sledeće tehnologije: PHP, HTML, CSS, JavaScript i MSSQL server.

3.1. SQL Server

SQL Server je Microsoftov sistem za upravljanje relacionim bazama podataka. Kao i svakom drugom sistemu za upravljanje bazama podataka, glavni zadatak mu je kreiranje, čuvanje i manupulisanje podacima , kao i kontrola prava pristupa podacima. Najpopularniji alat za pristup SQL Serveru je SQL Server Management Studio. Na sledećoj slici, vidi se kratak pregled istorije SQL Servera.



Slika 3.1.1. Istorija SQL Servera[7]

3.1.1. Arhitektura SQL servera

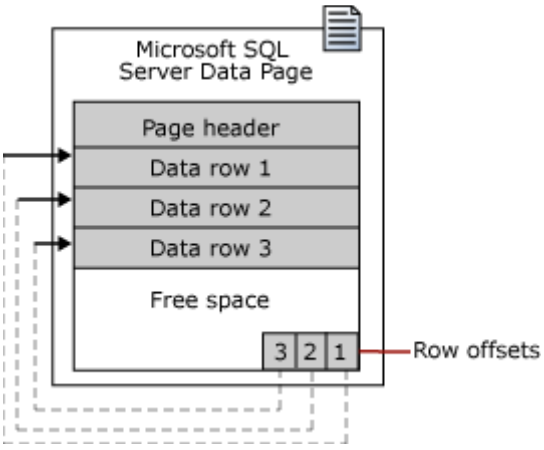
U narednom delu će biti opisane strukture podataka koje se koriste za upravljanje stranicama i dodacima. Razumevanje arhitekture stranica i dodataka je bitno za dizajniranje i razvoj baza podataka koje treba da budu efikasne.

Stranica je osnovna jedinica skladištenja podataka u SQL serveru. Dodatak (eng. extent) je skup od 8 fizički susednih stranica. Dodaci pomažu da se efikasno upravlja stranicama.

Osnovna jedinica baze podataka u SQL Serveru je stranica. Prostor na disku koji je lociran u fajlu (.mdf ili .ndf) u bazi podataka je logički podeljen na stranice koje su numerisane od 0 do n. Operacije na disku I/O se izvršavaju na nivou stranice. To jest, SQL Server čita ili piše čitave stranice podataka. Sve stranice su uskladištene na dodacima.

Na SQL Serveru veličina stranice je 8 KB. Ovo znači da baza podataka na SQL Serveru ima 128 stranica po megabajtu. Svaka stranica započinje sa 96-bajtnim zaglavljem koji se koristi da uskladišti sistemske informacije o stranici. Ove informacije uključuju broj stranice, tip stranice, količinu slobodnog prostora na stranici i dodelu indentifikatora jedinice objekta koji poseduje stranicu.

Redovi podataka su postavljeni na stranicu serijski, počevši odmah posle zaglavlja. *Row offset* počinje na kraju stranice, a svaka row offset tabela sadrži jedan ulaz za svaki red na stranici. Svaki ulaz zapisuje koliko je daleko prvi bajt reda od početka stranice. Ulazi na row offset tebele su u obrnutom redosledu u odnosu na redosled redova na stranici.[5]

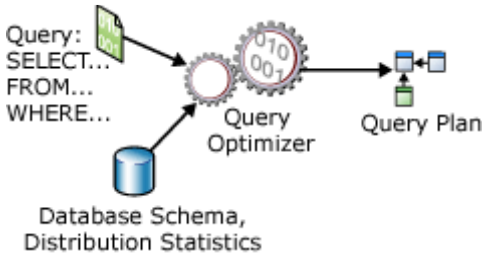


Slika 3.1.2. SQL Server stranica[5]

Obrada jednog SQL upita je osnovni način na koji SQL Server obavlja svoj posao. Koraci koji se koriste za obradu jednog SELECT upita koji se oslanja samo na lokalne tabele.

3.1.2. Optimizovanje SELECT upita

SELECT upit je neproceduralni, on ne iznosi tačne korake koje bi server baze podataka trebao da koristi da izvadi zahtevane podatke. Ovo znači da server baze podataka mora da analizira upit da bi utvrdio najefikasniji način da izvuče zahtevane podatke. Ovo je poznato kao optimizovanje SELECT upita. Komponenta koja ovo radi se naziva optimizator upita. Ulaz za optimizator se sastoji od upita, šeme baze podataka (tabela i definicije indeksa) i statistike baze podataka. Izlaz optimizatora je plan izvršavanja upita. Ulazi i izlazi optimizatora upita u toku optimizacije pojedinačnog SELECT upita su ilustrovane na sledećoj slici:



Slika 3.1.3. Optimizator upita[5]

SELECT upit definiše sledeće:

- Format rezultata. Ovo je navedeno najviše u selektovanoj listi. Ipak, druge klauzule kao što su ORDER BY i GROUP BY takođe utiču na finalnu formu rezultata.
- Tabele koje sadrže izvor podataka. Ovo je navedeno u FROM klauzuli.
- Kako su tabele logički povezane u svrhe SELECT upita. Ovo je definisano u zajedničkim specifikacijama, koje se mogu pojaviti u WHERE klauzuli ili ON klauzuli nakon FROM klauzule.
- Uslovi koje redovi u izvornim tabelama moraju da zadovolje da se kvalifikuju za SELECT upit. Oni su navedeni u WHERE i HAVING klauzulama.

Izvršni plan upita je definicija sledećeg:

- Redosled kojim je pristupano izvornim tabelama

Postoji mnogo načina kojim server baze podataka može da pristupi tabelama baze da bi napravio skup rezultata. Na primer, ako se SELECT upit odnosi na tri tabele, server baze podataka može prvo da pristupi tabeli A, da iskoristi podatke iz Tabele A kako bi izdvojio poklapajuće redove iz tabele B , i da iskoristi podatke iz Tabele B da bi izdvojio podatke iz Tabele C. Drugi načini kojima bi server baze podataka mogao da pristupi su:

- TableC, TableB, TableA, ili
- TableB, TableA, TableC, ili
- TableB, TableC, TableA, ili
- TableC, TableA, TableB
- Metode koje su korišćene za vađenje podataka iz svake tabele.

Generalno, postoje različite metode za pristup podacima u svakoj tabeli. Ako je potrebno dohvatiti samo nekoliko redova sa specifičnim ključem, server baze podataka može da koristi indeks. Ako su svi redovi u tabeli potrebni, server baze podataka može da ignoriše indekse i da izvrši skeniranje tabele. Ako su svi redovi u tabeli potrebni, ali postoji indeks čije su ključne kolone u ORDER BY klauzuli, izvršavajući skeniranje indeksa umesto skeniranja tabela, može se sačuvati neki deo rezultujućeg skupa. Ako je tabela veoma mala, skeniranje tabele može biti najefikasnija metoda za skoro svaki pristup tabeli.

Proces selektovanja jednog plana izvršenja od potencijalno mnogo mogućih planova se naziva optimizacija. Optimizator upita je jedna od najvažnijih komponenti SQL baze podataka. Optimizator upita SQL Servera je optimizator koji se bazira na troškovima. Svaki mogući plan izvršenja ima povezane troškove po pitanju količine računarskih resursa koji se koriste. Optimizator

upita mora da analizira moguće planove i da izabere onaj sa najnižim predviđenim troškovima. Neki kompleksni SELECT upiti imaju na hiljade mogućih planova izvršenja. U ovim slučajevima Optimizator upita ne analizira sve moguće kombinacije. Umesto toga, on koristi kompleksne algoritme kako bi našao plan izvršenja koji ima troškove koji su blizu minimalnim mogućim troškovima.

Optimizator upita SQL Servera ne bira samo plan izvršenja sa najnižim troškovima resursa. On bira plan koji vraća rezultate korisniku sa razumnim troškovima u resursima i koji vraća najbrže rezultate. Na primer, paralelno procesuiranje upita najčešće koristi više resursa u odnosu na serijsko, ali završava upit brže. Optimizator SQL Servera će koristiti paralelni plan izvršenja da vrati rezultate ako neće negativno uticati na opterećenje na serveru.

Optimizator upita je važan jer on omogućava serveru baze podataka da se automatski prilagodi promenljivim uslovima u bazi podataka bez zahtevanja ulaza od programera ili administratora baze podataka. Ovo omogućava programerima da se fokusiraju na opisivanje konačnog rezultata upita. Oni mogu da veruju da će optimizator upita napraviti efikasan plan izvršenja za stanje baze podataka svaki put kada se upit vrši

3.1.3. Izvršavanje SELECT upita.

Osnovni koraci koje SQL Server koristi da procesuira pojedinačni SELECT upit uključuje sledeće:

1) Analizator skenira SELECT upit i deli ga na logičke jedinice kao što su ključne reči, izrazi, operatori i identifikatori.
2) Drvo upita je izgrađeno opisujući logičke korake koji su potrebni da se transformišu izvorni podaci u onaj format koji je zahtevan rezultatom.
3) Optimizator upita analizira različite načine na koje se izvornim tabelama može pristupiti. Onda selektuje seriju koraka koji vraćaju rezultate najbrže koristeći najmanje resursa. Drvo upita se ažurira kako bi zapisalo tačne serija koraka. Konačna, optimizovana verzija drveta upita se naziva plan izvršenja.
4) Relacijski engine započinje izvršenje plana izvršenja. Dok se ostvaruju koraci koji su potrebni da se podaci iz baze podataka procesuiraju, Relacijski engine zahteva da Engine skladištenja dostavi podatke iz setova redova koji su zaahtevani od relacijskog engina.
5) Relacijski engine(mehanizam) procesuira podatke povraćene iz engina skladištenja u format koji je definisan za set rezultata i vraća rezultate klijentu.

3.1.4. Izvršavanje drugih upita

Osnovni koraci koji su opisani za procesuiranje SELECT upita odnose se na druge SQL upite kao što su INSERT,UPDATE, i DELETE. UPDATE i DELETE upiti imaju za cilj set redova koji treba da se modifikuju ili izbrišu. Proces identifikacije ovih redova je isti proces koji je korišćen za identifikovanje izvornih redova koji doprinose setu rezultata SELECT upita, UPDATE i INSERT upiti mogu posedovati ugrađene SELECT upite koji pružaju vrednosti koje treba da se ažuriraju ili ubace.

3.1.5. Stored Procedure i trigger-i

Stored Procedure na SQL serveru predstavljaju kolekciju T-SQL (*Transact*-SQL) naredbi, varijabli i naredbi za kontrolu toka, koje su grupisane kako bi izvršile specifičan zadatak. Generalno, *Stored Procedure* rade kao i procedure u programskim jezicima. *Stored Procedure* je imenovani objekat baze podataka i čuva se na strani servera gde se i izvršava, a klijentu se prosleđuju samo rezultati. Veoma su moćne i preko njih mogu da se izvršavaju sve operacije u DML-u (*data manipulation language*) kao, na primer, kreiranje tabele, izvršavanje UPDATE iskaza nad više tabela, umetanje, brisanje podataka ali i postavljanje vrednosti (SET) kao i prihvatanje transakcije (COMMIT) ili vraćanje baze u prethodno stanje (ROLLBACK).

Trigger se može definisati kao proceduralni kod koji se automatski izvršava svaki put kada se desi definisani događaj nad određenom tabelom ili *view*-om. *Trigger* predstavlja operaciju ažuriranja baze podataka, uslov je proizvoljni SQL predikat, a akcija je sekvenca SQL naredbi. Pod ažuriranjem se podrazumevaju operacije INSERT, UPDATE ili DELETE. *Trigger-i* se pre svega koriste za očuvanje integriteta baze podataka. Na taj način se mogu implementirati pravila integriteta. *Trigger*-i se, takođe, mogu koristiti za validaciju podataka, koji se unose i to definisanjem seta pravila napisanih korišćenjem T-SQL jezika. *Trigger*-i omogućavaju administratorima baze podataka da uspostave dodatne veze između odvojenih baza podataka.

3.1.6. Izvršavanje Stored Procedure i trigger

SQL Server skladišti samo izvor za *stored procedure* i *trigger*. Kada je *stored procedure* ili *trigger* izvršen, izvor je sastavljen u izvršni plan. Ako je *stored procedure* ili *trigger* ponovo izvršen pre nego što plan izvršenja izađe iz memorije, relacioni engine detektuje postojeći plan i ponovo ga koristi. Ako je plan izašao iz memorije, izrađuje se novi plan. Ovaj proces je sličan procesu koji SQL Server prati za sve SQL upite. Glavna prednost po pitanju performansi koje *stored procedure* i *trigger* imaju u SQL Serveru u poređenju sa serijom dinamičnog SQL-a je u tome što su njihovi SQL upiti uvek isti. Zbog toga, relacioni engine ih lako spaja sa postojećim planovima izvršenja. Planovi st*ored procedure*-a i *trigger*-a se lako koriste ponovo. Plan izvršenja za *stored procedure* i *trigger* izvršava odvojeno od plana izvršenja dela koji poziva *stored procedure* ili okida *trigger*. Ovo omogućava veću mogućnost ponovnog korišćenja planova izvršenja *stored procedure* i *trigger*.

3.2. Profiler

Profiler je grafički korisnički interfejs SQL Trace dela sql servera za praćenje dešavanja na bazi ili servisa za analize. Ima mogućnost snimanja i čuvanja podataka o svakom događaju u fajl ili tabelu za kasniju analizu. Na primer, može se pratiti koja *stored procedure* utiče na performanse jer se izvršava previše dugo.

Osnovne komponente koje bi trebalo poznavati o Profileru su:[5]

1) Događaji

Događaj je akcija koja se desila u okviru instance sql server engine-a. Primeri

- Ostvarivanje konekcije, greške i prekidi konekcije
- T-SQL SELECT, INSERT, UPDATE, i DELETE upiti
- Poziv *stored procedure*
- Početak i kraj *stored procedure*
- Početak i kraj upita u okviru *stored procedure*

- Početak i kraj grupe upita
- Greške koje su upisane u log grešaka
- Zaključavanja koja su se dogodila nad nekim objektom
- Otvaranje kursora
- Bezbednosne provere

Svi podaci koji su generisani kao događaj su prikazani kao jedan red. Taj red sadrži kolone koje opisuju detalje događaja.

2) Klasa događaja

Klasa događaja predstavlja tip događaja koji može biti snimljen. Klasa događaja sadrži sve podatke koji mogu biti predstavljeni kao događaj. Primeri klasa događaja su:

- *SQL:BatchCompleted*
- *Audit Login*
- *Audit Logout*
- *Lock:Acquired*
- *Lock:Released*

3) Kolona

Kolona je atribut klase događaja koji je sačuvan na snimku. Zbog prirode klasa događaja i samih atributa, nema svaki red vrednost za svaki atribut.

4) Obrazac

Obrazac definiše podešavanja za snimak. Na primer mogu se podesiti događaji i kolone koji se žele snimati.

5) Snimanje

Snimak beleži sve događaje koji odgovaraju datim događajima i filterima koji s odabrani. Nakon toga, snimak se može sačuvati, može se sačuvati obrazac snimka i slično.

6) Filter

Kada se pokreće snimanje ili kada se kreira obrazac, postoji mogućnost filtera samo određenih događaja, kolona, dodatnih atributa. Ovo je veoama korisno jer se može snimak suziti samo na kritične tačke koje se žele analizirati.

4. PRAKTIČNI PRIMER

Implementacija svih gore navedenih ograničenja će biti detaljnije objašnjena kroz studijski primer. Biće naveden verbalni opis primera, pobrojani slučajevi korišćenja. Biće dat MOV za dati prikaz strukture podataka, relacioni model. Biće formalno opisana definisana ograničenja nad bazom podataka. Nakon toga, testiraće se različite implementacije navedenih ograničenja, i biće upoređeni rezultati.

4.1. Prikupljanje zahteva

U ovoj fazi, koja je prva, a koja često se naziva i nulta faza, vrši se prikupljanje korisničkih zahteva, njihova selekcija itd. Nakon toga dobija se takozvani "verbalni opis sistema" i tada se može pristupiti detaljnijem projekovanju arhitekure samog sistema, a takođe se mogu definisati slučajevi korišćenja.

4.1.1. Verbalni opis

Potrebno je napraviti bazu podatka za unos, izmenu i pregled organizacionih jedinica nekog preduzeća, radnika, i njihovih zarada. Ograničenja koja su definisana su da radnik koji je rukovodilac nekoj organizacionoj jedinici, mora biti zaposlen u toj organizacionoj jedinici. Da bi radnik bio rukovodilac, mora imati odgovarajuću stručnu spremu i da ima status "Aktivan". Dozvoljeni statusi radnika su "Aktivan", "Neaktivan". Nije moguće dodeliti radniku status „Neaktivan" ukoliko je rukovodilac nekom OJ. Prilikom unosa isplata potrebno je da ukupan iznos isplate bude jednak zbiru stavki. Dozvoljeni osnov je "Zarada", "Prevoz", "Prekovremeni rad". Ukoliko je osnov "Prevoz", zarada ne može biti veća od 5000 dinara.

4.1.2. Slučajevi korišćenja

Slučajevi korišćenja za datu bazu podataka su:

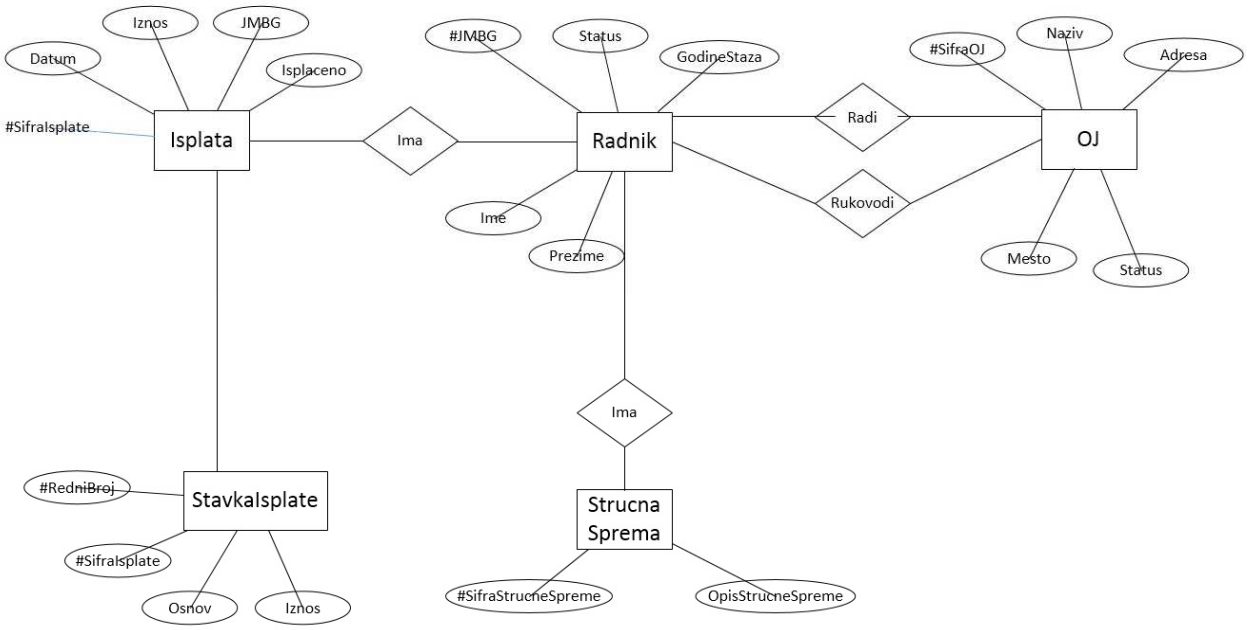
1) Unos OJ
2) Izmena OJ
3) Pregled OJ
4) Unos radnika
5) Pregled radnika
6) Izmena radnika
7) Unos isplate

4.2. Analiza

U sledećoj fazi razvijanja softvera biće definisani model podataka i relacioni model, biće dat formalni prikaz zadatih ograničenja

4.2.1. Model podataka

Struktura softverskog sistema opisana je preko modela objekti veze.



Slika 4.2.1. Model objekti veze

4.2.2. Relacioni model

Na osnovu modela objekti-veze, napravljen je relacioni model. Kreirane su sledeće tabele u bazi sa svojim kolonama:

Radnik (**JMBG**, Ime, Prezime, SifraOJ, Status, SifraStrucneSpreme,GodineStaza)

OJ (**SifraOJ**, Naziv, Adresa,Mesto,Status, JMBGRukovodioca)

StrucnaSprema (**SifraStrucneSpreme**, OpisStrucneSpreme)

Isplata(**IDIsplate**, JMBG, Iznos, Datum,Isplaceno)

StavkeIsplate(**RedniBroj,IDIsplate**, IznosStavke, Osnov)

Kolone koje predstavljaju primerne ključeve u svojim tabela su prikazane bolodovano,

Tabelarni prikaz vrednosnih i strukturnih ograničenja dat je u nastavku

Tabela Radnik		**Prosto vrednosno ograničenje**		**Složeno vrednosno ograničenje**		**Strukturno ograničenje**
	Ime	**Tip atributa**	**Vrednost atributa**	**Međuz. atributa jedne tabele**	**Međuzav. atributa vise tabela**	INSERT RESTRICTED StrucnaSprema, OJ
Atributi	JMBG	String	Not null, LEN=13 SUBSTRING (1,2) < 32, SUBSTRING (3,4) < 13, SUBSTRING (5,7) BETWEEN 000, 999 AND 000.			
	Ime	String	Not null, Start with UPPER CASE			UPDATE RESTRICTED StrucnaSprema, OJ
	Prezime	String	Not null, Start with UPPER CASE			
	SifraOJ	int	Not null			DELETE RESTRICTED OJ
	Status	String	In (“Aktivan”, “Nekaktivan”)			
	SifraStrucneSpreme	int	Not null			
	GodineStaza	int				

Tabela 4.2.1. Ograničenja tabele Radnik

Tabela OJ		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Medjuz. atributa jedne tabele**	**Medjuz. atributa vise tabela**	INSERT RESTRICTED Radnik UPDATE RESTRICTED Radnik DELETE RESTRICTED Radnik
	SifraOJ	int	Not null			
	Naziv	String	Not null, Start with UPPER CASE			
	Adresa	String	Not null, Start with UPPER CASE			
	Mesto	String	Not Null Start with UPPER CASE			
	Status	String	In (“Aktivan”, “Nekaktivan”)			
	JMBGRukovodioca	String	LEN=13 SUBSTRING (1,2) < 32, SUBSTRING (3,4) < 13, SUBSTRING (5,7) BETWEEN 000, 999 AND 000.			

Tabela 4.2.2. Ograničenja tabele OJ

Tabela StrucnaSprema		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Medjuz. atributa jedne tabele**	**Medjuzav. atributa vise tabela**	INSERT / UPDATE CASCADE Radnik DELETE RESTRICTED Radnik
	SifraStrucneSpreme	int	Not null			
	OpisStrucneSpreme	String	Not null			

Tabela 4.2.3. Ograničenja tabele Stručna Spreme

Tabela Isplata		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Medjuz. atributa jedne tabele**	**Medjuz. atributa vise tabela**	INSERT RESTRICTED Radnik UPDATE RESTRICTED Radnik CASCADE StavkeIsplate DELETE CASCADE StavkeIsplate
	SifraIsplate	int	Not null			
	JMBG	String	Not null, LEN=13 SUBSTRING (1,2) < 32, SUBSTRING (3,4) < 13, SUBSTRING (5,7) BETWEEN 000, 999 AND 000.			
	Iznos	Decimal	>0		∑ iznosa stavki	
	Isplaceno	int				
	Datum	Datetime	Not null			

Tabela 4.2.4. Ograničenja tabele Isplata

Tabela StavkeIsplate		Prosto vrednosno ograničenje		Složeno vrednosno ograićenje		Strukturno ograničenje
Atributi	**Ime**	**Tip atributa**	**Vrednost atributa**	**Medjuz. atributa jedne tabele**	**Medjuza v. atributa vise tabela**	INSERT RESTRICTED Isplate UPDATE RESTRICTED Isplate DELETE /
	RedniBroj	int	>0			
	SifraIsplate	int	Not null			
	Iznos	decimal	>0	IF Osnov= ”Prevoz ” THEN <5000		
	Osnov	String	In („Zarada“, “Prevoz“, “Prekovremeni rad“)			

Tabela 4.2.5. Ograničenja tabele StavkeIsplate

4.2.3. Formalni iskazi ograničenja

U narednom delu će biti prikazani formalni iskazi ograničenja nad bazom podataka za svaku tabelu pojedinačno.

Tabela Radnik:

- Prosta vrednosna ograničenja

Ime i prezime moraju počinjati velikim slovima. Vrednost atributa Status može biti “Aktivan, Neaktivan”.

Ime `varchar(100) SUBSTRING (1,1) IS UPPER CASE`

Prezime `varchar(100) SUBSTRING (1,1) IS UPPER CASE`

Status `varchar(100) IN (“Aktivan”,”Neaktivan”)`

JMBG mora imati 13 karaktera, prve dve cifre moraju biti manje od 32, druge 2 manje od 13 i 5.,6. i 7. cifra moraju biti u opsegu [000,014] U [900,999]

JMBG
```
varchar(13) LEN()=13 AND SUBSTRING (1,2) < 32, SUBSTRING (3,4)
< 13, SUBSTRING (5,7) BETWEEN 000, 014 OR BETWEEN 900, 999
```

- Međuzavisnost atributa više tabela

Nije moguće dodeliti status „Neaktivan“ radniku koji je rukovodilac nekom OJ.

```
OGR1 (OJ,Radnik):=foreach Radnik (if Radnik.Status=’Neaktivan’ then
NOT EXIST OJ.Rukovodilac=Radnik.JMBG)
```

- Strukturna ograničenja:

Nije moguće uneti/izmeniti stručnu spremu koja nije definisana u tabeli StrucnaSprema.

Nije moguće uneti/izmeniti OJ radnika koji ne postoji u tabeli OJ.

Nije moguće obrisati radnika ukoliko je rukovodilac nekom OJ.

```
INSERT RESTRICTED StrucnaSprema, OJ,
UPDATE RESTRICTED StrucnaSprema, OJ,
DELETE RESTRICTED OJ
```

Tabela OJ

- Prosta vrednosna ograničenja:

Adresa, Mesto i naziv OJ moraju počinjati velikim slovima, vrednost atributa Status može biti “Aktivan, Neaktivan”.

Naziv `varchar(100) SUBSTRING (1,1) IS UPPER CASE`

Adresa `varchar(100) SUBSTRING (1,1) IS UPPER CASE`

Status `varchar(100) IN (“Aktivan”,”Neaktivan”)`

JMBGRukovodioca mora imati 13 karaktera, prve dve cifre moraju biti manje od 32, druge 2 manje od 13 i 5.,6. i 7. cifra moraju biti u opsegu [000,014] U [900,999]

JMBGRukovodioca `varchar(13) LEN()=13 AND SUBSTRING (1,2) < 32, SUBSTRING (3,4) < 13, SUBSTRING (5,7) BETWEEN 000, 014 OR BETWEEN 900, 999`

- Međuzavisnost atributa više tabela

Nije moguće da radnik bude rukovodilac OJ ukoliko mu stručna sprema nije veća od 3 (minimum diploma fakulteta) i ako mu status nije „Aktivan“.

```
OGR2 (OJ,Radnik):=foreach OJ (if OJ.Rukovodilac.SifraOJ = OJ.SifraOJ)
THEN OJ.Rukovodilac.Status = ”Aktivan” or
OJ.Rukovodilac.StrucnaSprema >3
```

- Strukturna ograničenja:

Nije moguće uneti/izmeniti JMBGRukovodioca koji nije definisan u tabeli radnik.

Nije moguće obrisati OJ ukoliko bar jedan zaposleni radi u tom OJ.

```
INSERT RESTRICTED Radnik
UPDATE RESTRICTED Radnik
DELETE RESTRICTED Radnik
```

Tabela Isplata

- Prosta vrednosna ograničenja:

JMBG mora imati 13 karaktera, prve dve cifre moraju biti manje od 32, druge 2 manje od 13 i 5.,6. i 7. cifra moraju biti u opsegu [000,014] U [900,999]

JMBG `varchar(13) LEN()=13 AND SUBSTRING (1,2) < 32, SUBSTRING (3,4) < 13, SUBSTRING (5,7) BETWEEN 000, 014 OR BETWEEN 900, 999`

- Međuzavisnost atributa više tabela:

Iznos mora biti jednak sumi svih stavki koje odgovaraju toj isplati.

```
OGR3 (Isplata, StavkeIsplate) := foreach Isplata.Iznos = SUM(
Isplata.StavkeIsplate.Iznos)
```

- Strukturna ograničenja:

Nije moguće uneti Isplatu sa JMBG koji ne postoji u tabeli Radnik.

Nije moguće izmeniti JMBG u tabeli Isplata koji ne postoji u tabeli Radnik.

Izmene nad tabelom isplata odraziće se i na tabelu StavkeIsplate

Ukoliko se obriše slog u tabeli Isplata, obrisaće se i povezani slogovi u tabeli StavkeIsplate.

```
INSERT RESTRICTED Radnik
UPDATE RESTRICTED RadnikCASCADE StavkeIsplate
DELETE CASCADE StavkeIsplate
```

Tabela StavkeIsplate:

- Prosta vrednosna ograničenja:

  Vrednost atributa Osnov može biti “Zarada”, “Prevoz”,”Prekovremeni rad”.

  Osnov `varchar(100) IN (“Zarada”, “Prevoz”,”Prekovremeni rad”)`

- Međuzavisnost atributa jedne tabele:

  Ako je osnov isplate „Prevoz“, iznos isplate ne može biti veći od 5000.

```
OGR4 (StavkaIsplate):=foreach StavkaIsplate
IF(StavkaIsplate.Osnov=“Prevoz“) THEN (Iznos<5000)
```

- Strukturna ograničenja

  Potrebno je obezbediti da je moguće uneti samo stavke koje su povezane sa postojećom isplatom.

```
INSERT RESTRICTED Isplate
UPDATE RESTRICTED Isplate
```

4.3. Projektovanje

Faza projektovanja opisuje fizičku strukturu i ponašanje softverskog sistema (arhitekturu softverskog sistema). U našem primeru arhitektura sistema se sastoji od jednog nivoa – skladišta podataka.

Prilikom projektovanja skladišta podataka, potrebno je projektovati tabele baze podataka i ograničenja nad bazom podataka.

4.3.1. Projektovanje baze podatka

Na osnovu strukture sistema projektovane su tabele sledeće definicije:

Tabela Radnik		
Naziv kolone	Tip podatka	Veličina
JMBG	Varchar	13
Ime	Varchar	100
Prezime	Varchar	100
SifraOJ	int	/
Status	Varchar	100
SifraStrucneSrpeme	int	/
GodineStaze	int	/

Tabela 4.3.1. Projektovanje tabele Radnik

Tabela OJ		
Naziv kolone	Tip podatka	Veličina
SifraOJ	int	/
Naziv	Varchar	100
Adresa	Varchar	100
JMBGRukovodioca	Varchar	13
Mesto	Varchar	200
Status	Varchar	20

Tabela 4.3.2. Projektovanje tabele OJ

Tabela StrucnaSprema		
Naziv kolone	Tip podatka	Veličina
SifraStrucneSpreme	Int	/
OpisStrucneSpreme	Varchar	255

Tabela 4.3.3. Projektovanje tabele StrucnaSprema

Tabela Isplata		
Naziv kolone	Tip podatka	Veličina
SifraIsplate	int	/
JMBG	Varchar	13
Iznos	Decimal	38,2
Datum	DateTime	/
Isplaceno	int	

Tabela 4.3.4. Projektovanje tabele Isplata

Tabela StavkeIsplate		
Naziv kolone	Tip podatka	Veličina
RedniBroj	int	/
SifraIsplate	int	/
Iznos	Decimal	38,2
Osnov	Varchar	50

Tabela 4.3.5. Projektovanje tabele StavkeIsplate

Nakon projektovanja, dijagram baze izgledan kao na slici 4.3.1.

Slika 4.3.1. Dijagram baze podataka

4.3.2. Projektovanje ograničenja nad bazom podataka

Ograničenja koja su definisana nad bazom podataka će biti implementirana na dva načina:

- implementacija putem *stored procedure*. Za svaku tabelu će biti potrebno napraviti jednu *stored procedure* koja će da vrši proveru definisanih ograničenja i operaciju ubacivanja.
- implementacija putem *trigger*-a. Za svaku tabelu će biti potrebno napraviti *trigger*-a. koji će vršiti proveru datih ograničenja.

4.4. Implementacija na SQL serveru

Kao što je već spomenuto, sistem za upravljanje bazom podataka koji je korišćen je SQL Server, verzija 2008 R2. Kao razvojno okruženje korišćen je SQL Server Management Studio.

4.4.1. Kreiranje baze

U prvom koraku implementacije, kreirana je baza podataka

```
CREATEDATABASE [RadniciMaster]
```

Potom su kreirane su tabele:

```
CREATETABLE [dbo].[Radnik](
      [JMBG] [nvarchar](13)NOTNULL,
      [Ime] [nvarchar](100)NOTNULL,
      [Prezime] [nvarchar](100)NOTNULL,
      [SifraOJ] [int] NOTNULL,
      [Status] [nvarchar](100)NOTNULL,
      [SifraStrucneSpreme] [int] NOTNULL,
      [GodineStaza] [int] NULL,
CONSTRAINT [PK_Radnik] PRIMARYKEYCLUSTERED
(
```

```
[JMBG] ASC
)WITH
(PAD_INDEX=OFF,STATISTICS_NORECOMPUTE=OFF,IGNORE_DUP_KEY=OFF,ALLOW_R
OW_LOCKS=ON,ALLOW_PAGE_LOCKS=ON)ON [PRIMARY]
)ON [PRIMARY]
GO

ALTERTABLE [dbo].[Radnik] WITHCHECKADDCONSTRAINT [fk_radnik]
FOREIGNKEY([SifraOJ])
REFERENCES [dbo].[OJ]([SifraOJ])
GO

ALTERTABLE [dbo].[Radnik] WITHCHECKADDCONSTRAINT
[fk_Sifrastrucnespreme] FOREIGNKEY([SifraStrucneSpreme])
REFERENCES [dbo].[StrucnaSprema]([SifraStrucneSpreme])
GO


CREATETABLE [dbo].[OJ](
      [SifraOJ] [int] NOTNULL,
      [Naziv] [nvarchar](100)NOTNULL,
      [Adresa] [nvarchar](100)NOTNULL,
      [JMBGRukovodioca] [nvarchar](13)NULL,
      [Mesto] [nvarchar](200)NOTNULL,
      [Status] [nvarchar](20)NOTNULL,
CONSTRAINT [PK_OJ] PRIMARYKEYCLUSTERED
(
      [SifraOJ] ASC
)WITH
(PAD_INDEX=OFF,STATISTICS_NORECOMPUTE=OFF,IGNORE_DUP_KEY=OFF,ALLOW_R
OW_LOCKS=ON,ALLOW_PAGE_LOCKS=ON)ON [PRIMARY]
)ON [PRIMARY]
GO

ALTERTABLE [dbo].[OJ] WITHCHECKADDCONSTRAINT [fk_JMBGRukovodica]
FOREIGNKEY([JMBGRukovodioca])
REFERENCES [dbo].[Radnik]([JMBG])
GO


CREATETABLE [dbo].[StrucnaSprema](
      [SifraStrucneSpreme] [int] NOTNULL,
      [OpisStrucneSpreme] [nvarchar](100)NULL,
CONSTRAINT [pk_strucnasprema] PRIMARYKEYCLUSTERED
(
      [SifraStrucneSpreme] ASC
)WITH
(PAD_INDEX=OFF,STATISTICS_NORECOMPUTE=OFF,IGNORE_DUP_KEY=OFF,ALLOW_R
OW_LOCKS=ON,ALLOW_PAGE_LOCKS=ON)ON [PRIMARY]
)ON [PRIMARY]
GO


CREATETABLE [dbo].[Isplata](
      [SifraIsplate] [int] IDENTITY(1,1)NOTNULL,
      [JMBG] [nvarchar](13)NOTNULL,
      [Iznos] [decimal](18, 0)NOTNULL,
      [Datum] [datetime] NOTNULL,
CONSTRAINT [PK__Isplata__3E4FE3CB0BC6C43E] PRIMARYKEYCLUSTERED
(
```

```
      [SifraIsplate] ASC
)WITH
(PAD_INDEX=OFF,STATISTICS_NORECOMPUTE=OFF,IGNORE_DUP_KEY=OFF,ALLOW_R
OW_LOCKS=ON,ALLOW_PAGE_LOCKS=ON)ON [PRIMARY]
)ON [PRIMARY]
GO

ALTERTABLE [dbo].[Isplata] WITHCHECKADDCONSTRAINT [fk_isplata]
FOREIGNKEY([JMBG])
REFERENCES [dbo].[Radnik]([JMBG])
GO


CREATETABLE [dbo].[StavkeIsplate](
      [RedniBroj] [int] NOTNULL,
      [SifraIsplate] [int] NOTNULL,
      [Iznos] [decimal](18, 0)NOTNULL,
      [Osnov] [nvarchar](100)NULL,
CONSTRAINT [PK__StavkeIs__863A8DE60F975522] PRIMARYKEYCLUSTERED
(
      [RedniBroj] ASC,
      [SifraIsplate] ASC
)WITH
(PAD_INDEX=OFF,STATISTICS_NORECOMPUTE=OFF,IGNORE_DUP_KEY=OFF,ALLOW_R
OW_LOCKS=ON,ALLOW_PAGE_LOCKS=ON)ON [PRIMARY]
)ON [PRIMARY]
GO

ALTERTABLE [dbo].[StavkeIsplate] WITHCHECKADDCONSTRAINT
[fk_StavkaIsplate] FOREIGNKEY([SifraIsplate])
REFERENCES [dbo].[Isplata]([SifraIsplate])
GO
```

4.4.2. Implementacija ograničenja

Napisani su *trigger*-i koji proveravaju da li su zadovoljeni kriterijumi integriteta baze podataka i da li su zadovoljena sva ograničenja. Napisane su i *stored procedure* koje proveravaju ulazne parametre u istu svrhu.

trigger-i koji u napisani:

tg_Radnik

tg_OJ

tg_StavkeIsplate

stored procedure koje su napisane:

sp_sacuvajRadnika

sp_sacuvajOJ

sp_sacuvajIsplatu

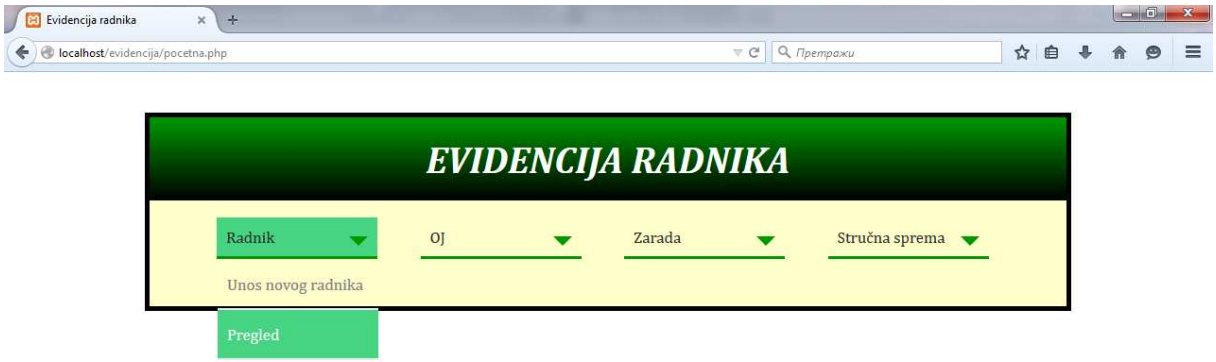
sp_sacuvajStavku

4.5. Implementacija veb sajta

U ovom delu opisana je implementacija veb sajta sa kreiranom bazom podataka. Kao što je već ranije napomenuto, tehnologije koje su korićene su PHP, HTML, CSS, JavaScript. Na ovom sajtu omogućene su sve funkcionalnosti koje će biti testirane u daljem radu a to su:

1. Unos, izmena, pregled i brisanje OJ
2. Unos, izmena, pregled i brisanje radnika
3. Unos, izmena, pregled i bisanje stručne spreme
4. Unos, pregled, i brisanje isplate

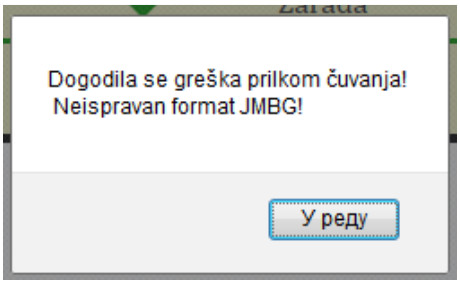
Početna strana je prikazana na slici 4.5.1. Kao što se vidi sa slike, postoje četiri padajuća menija Radnik, OJ, Zarada i Stručna sprema. Svaki padajući meni se sastoji od dve opcije, a to su Unos novog podatka i pregled podatka.



Slika 4.5.1. Početna strana veb sajta

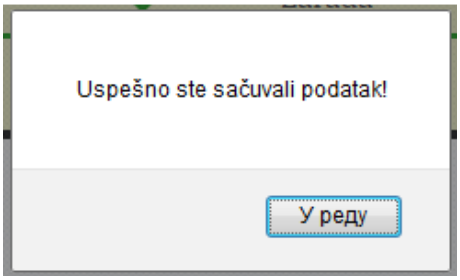
Nakon odabira opcije Unos novog radnika dobija se forma za unos podataka . U ovom slučaju namerno ćemo napisati neispravan JMBG da bismo dobili kao poruku imeplementirano ograničenje o ispravnosti JMBG-a u tabeli radnik.

Slika 4.5.2. Forma za unos novog radnika



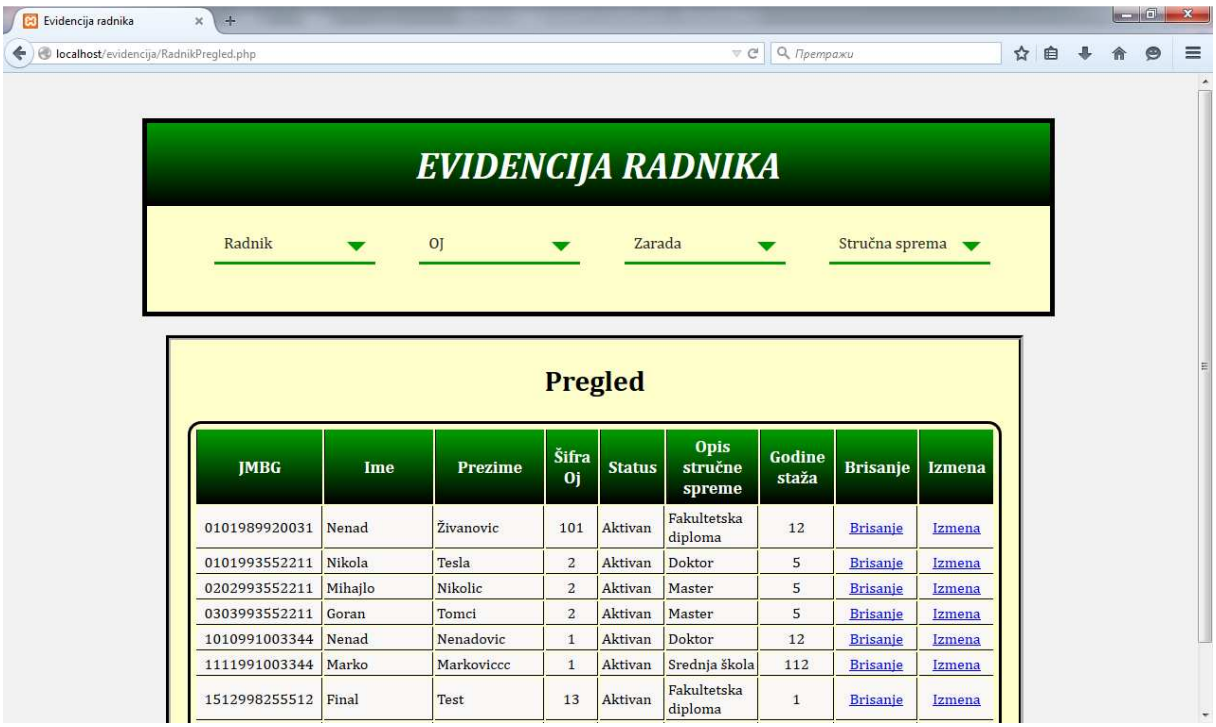
Slika 4.5.3. Poruka prilikom pokušaja ubacivanja neispravnog podatka

I situacija kada su svi podaci ispravni i ubacivanje novog radnika je uspešno.



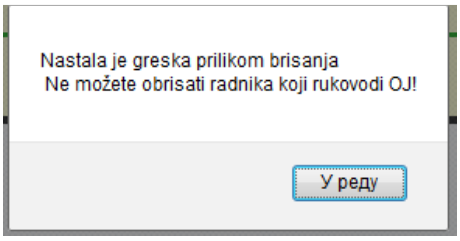
Slika 4.5.4. Poruka prilikom uspešng ubacivanja novog podatka

Kada se odabere Pregled iz padajućeg menija dobija se forma koja je prikazana na slici 4.5.5. Ova forma izlistava spisak svih radnika koji se nalaze u bazi. Za svaki zapis omogućena je izmena i brisanje.



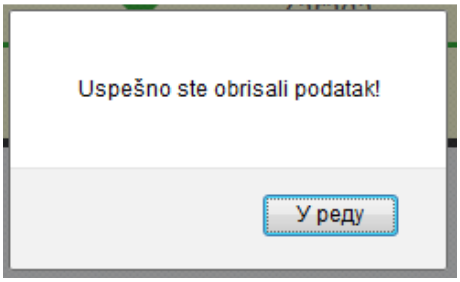
Slika 4.5.5. Pregled radnika

Proverićemo ograničenje da ne možemo da obrišemo radnika ako je on rukovodilac nekom OJ.



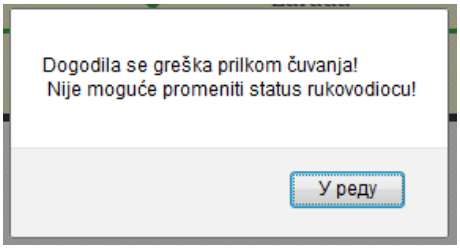
Slika 4.5.6. Poruka prilikom pokušaja brisanja radnika

I situacija kada je brisanje podatka uspešno.



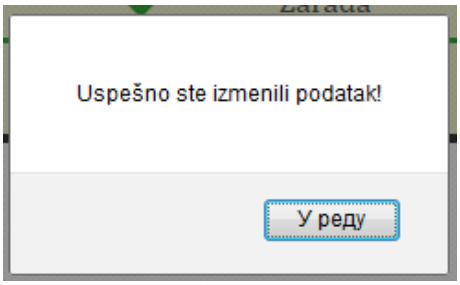
Slika 4.5.7. Poruka prilikom uspešnog brisanja radnika

Sada ćemo proveriti ograničenje da ne možemo da promenimo status radniku iz aktivan u neaktivan ako je on rukovodilac nekom OJ.



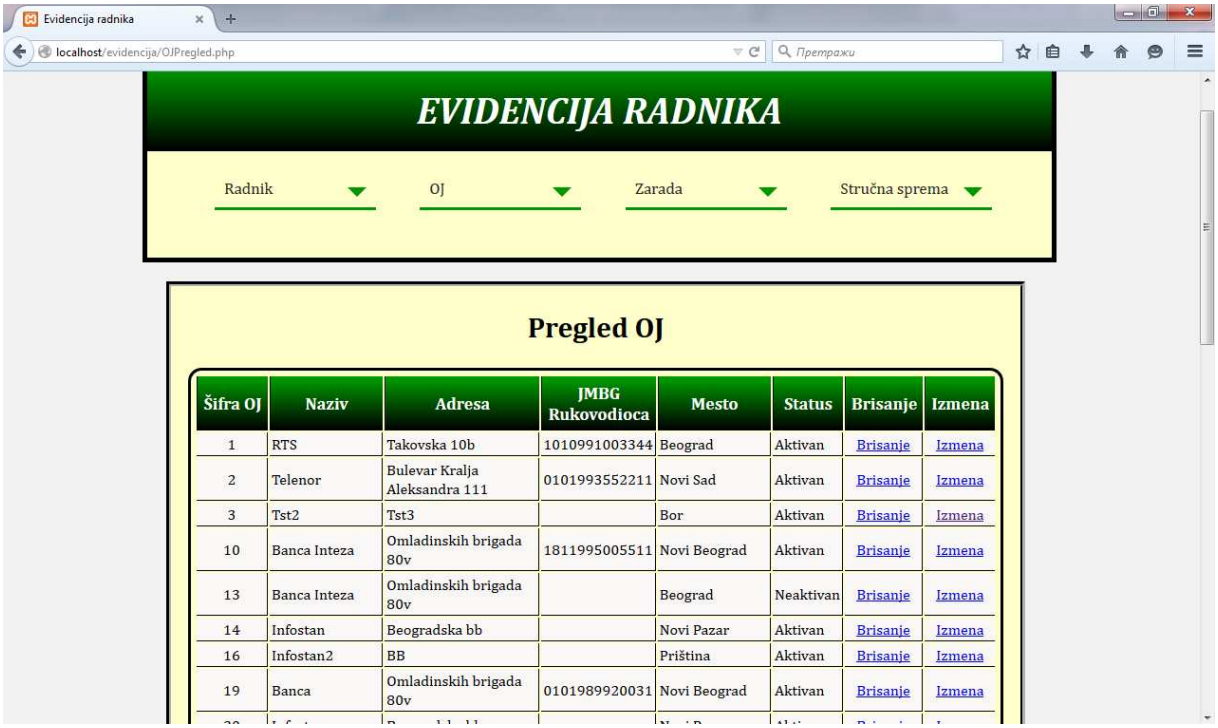
Slika 4.5.8. Poruka prilikom pokušaja izmene radnika

I situacija kada je izmena podatka uspešno izvršena

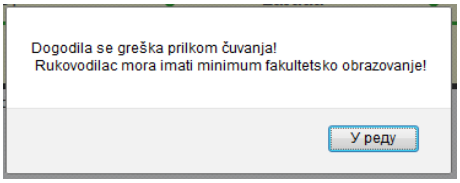


Slika 4.5.9. Poruka prilikom uspešne izmene radnika

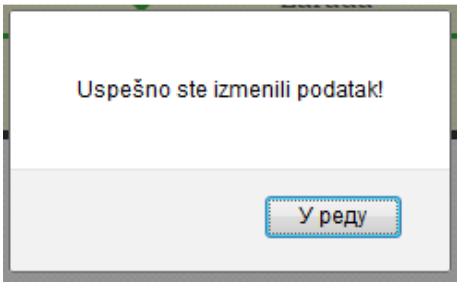
Ograničenje nad tabelom OJ proverićemo tako što ćemo OJ staviti da je rukovodilac neko ko nema fakultetsku diplomu. A nakon toga ćemo izabrati “regularnog” radnika za rukovodioca da bi izmena bila uspešna.



Slika 4.5.10. Pregled OJ



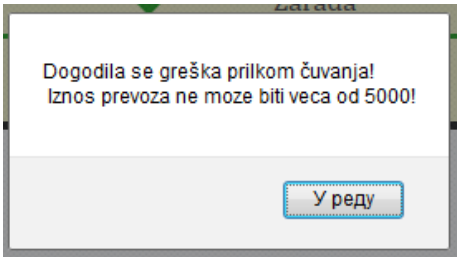
Slika 4.5.11. Poruka prilikom pokušaja izmene OJ



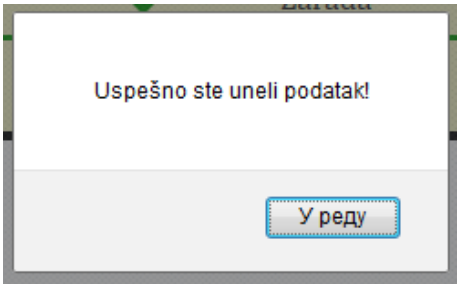
Slika 4.5.12. Poruka prilikom uspešne izmene OJ

I na kraju proverićemo ograničenje nad tabelom StavkeIsplate tako što ćemo za stavku prevoz staviti vrednost veću od 5000, a to nije dozvoljeno. A nakon toga ćemo staviti vrednost manju od 5000 i uspešno izvršiti čuvanje u bazi.

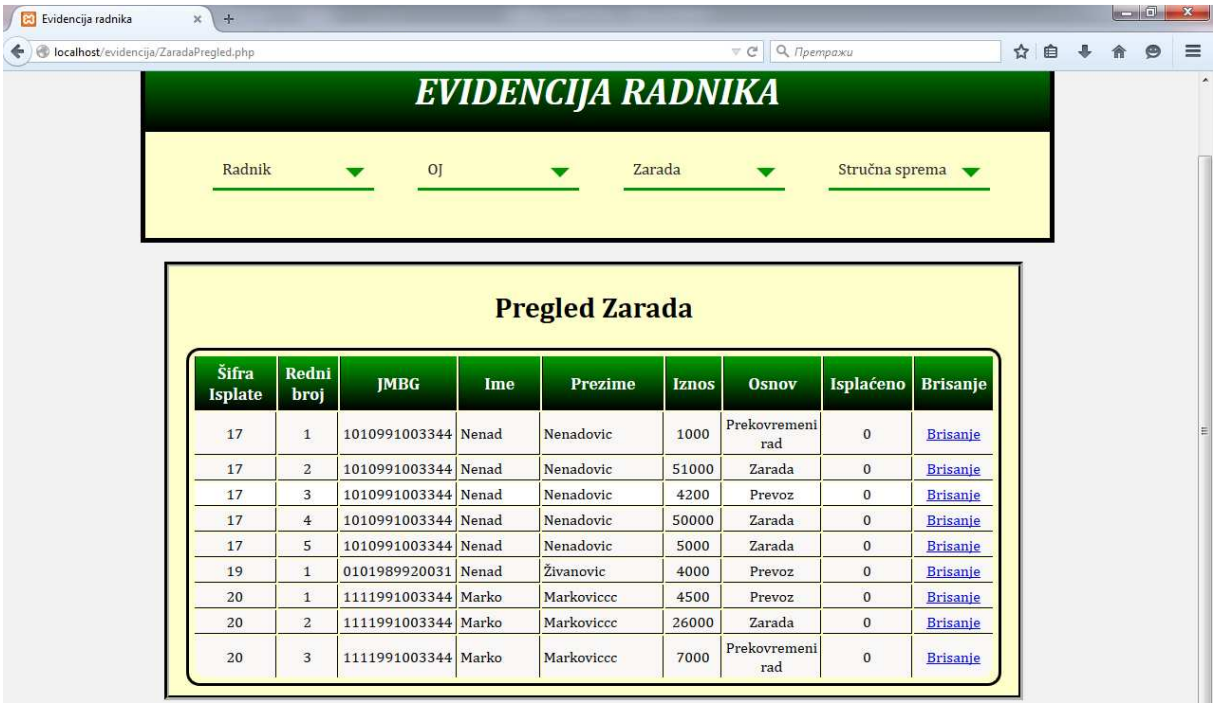
Slika 4.5.13. Unos nove stavke i zarade.



Slika 4.5.14. Poruka prilikom pokušaja čuvanja stavke isplate



Slika 4.5.15. Poruka prilikom uspešnog čuvanja stavke isplate



Slika 4.5.16. Pregled zarada

4.6. Testiranje i merenje performansi

U fazi testiranja vršena je provera da li sve funkcionalnosti rade po korisničkim zahtevima. Nakon detaljnog testiranja utvrđeno je da je svaka funkcionalnost u skladu sa zahtevom.

Merenje performansi je vršeno na računaru sa procesorom Intel(R) Pentium(R) Dial CPU T3200 @2GHz, RAM memorijom od 3GB, 32-bitnim operativnim sistemom Windows 7.

Prilikom testiranja vršena su merenja po različitim kriterijumima:

1) Vreme trajanja
2) Procesorsko vreme
3) Broj čitanja sa diska
4) Broj pisanja po disku

Za testiranje je korišćena statistika sql servera i “Profiler”, koji su detaljnije opisani u trećem poglavlju ovog rada.

4.6.1. Merenje performansi prostih vrednosnih ograničenja

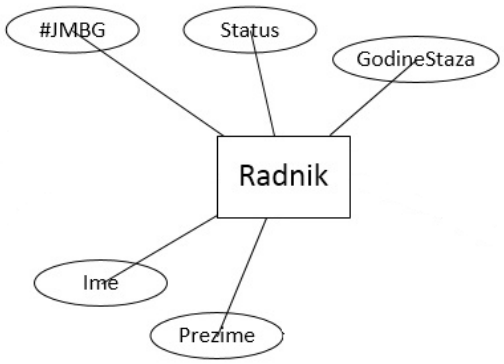
Prosta vrednosna ograničenja se odnose na ograničenja na domene atributa. Tu postoje dve podkategorije prostih vrednosnih ograničenja:

- ograničenja koja se sastoje od jednog pravila koje atribut treba da ispuni
- ograničenja koja se sastoje od više pravila

Po toj kategorizaciji je vršeno testiranje.

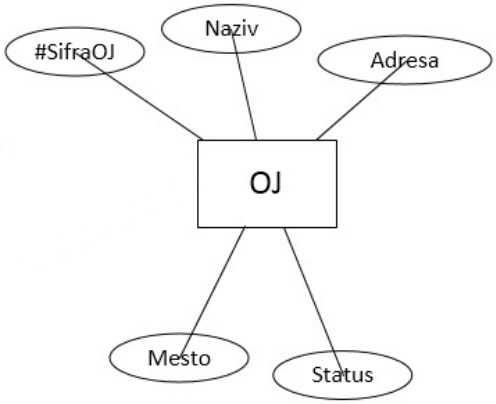
Ograničenja koja se sastoje od jednog pravila

U tabeli Radnik ime i prezime moraju počinjati velikim slovima. Vrednost atributa Status može biti “Aktivan, Neaktivan”.



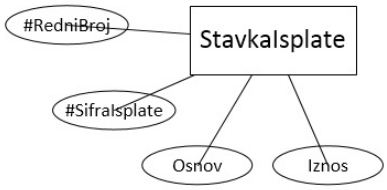
Slika 4.6.1. MOV tabele Radnik

U tabeli OJ adresa i naziv OJ moraju počinjati velikim slovima:



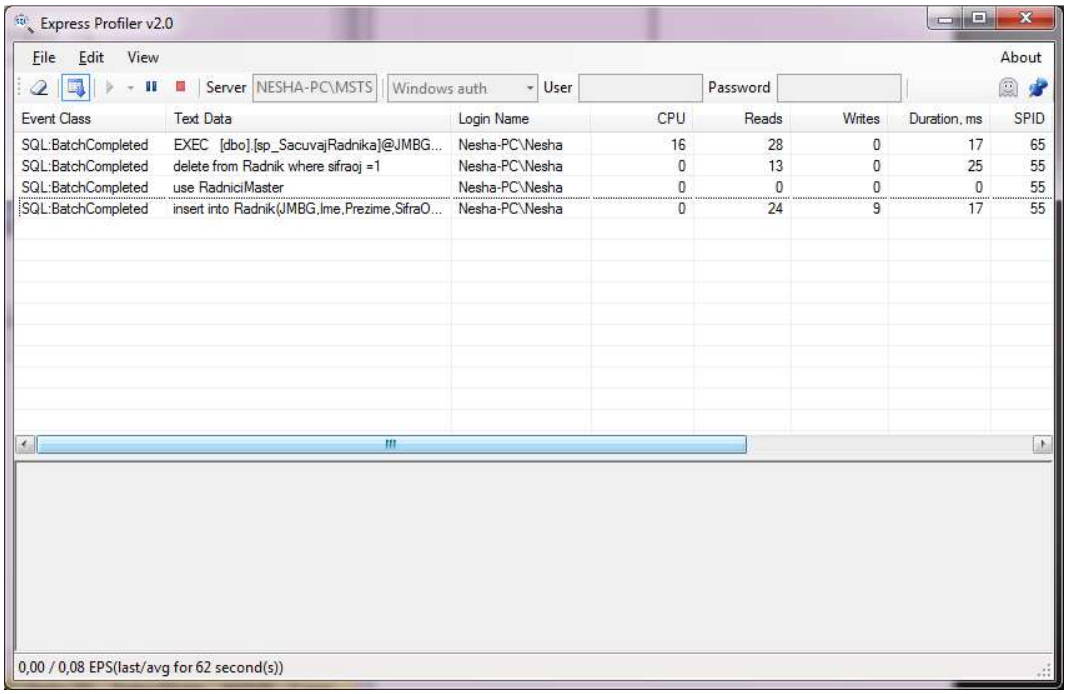
Slika 4.6.2. MOV tabele OJ

U tabeli Isplata vrednost atributa Osnov može biti “Zarada”, “Prevoz”,”Prekovremeni rad”.



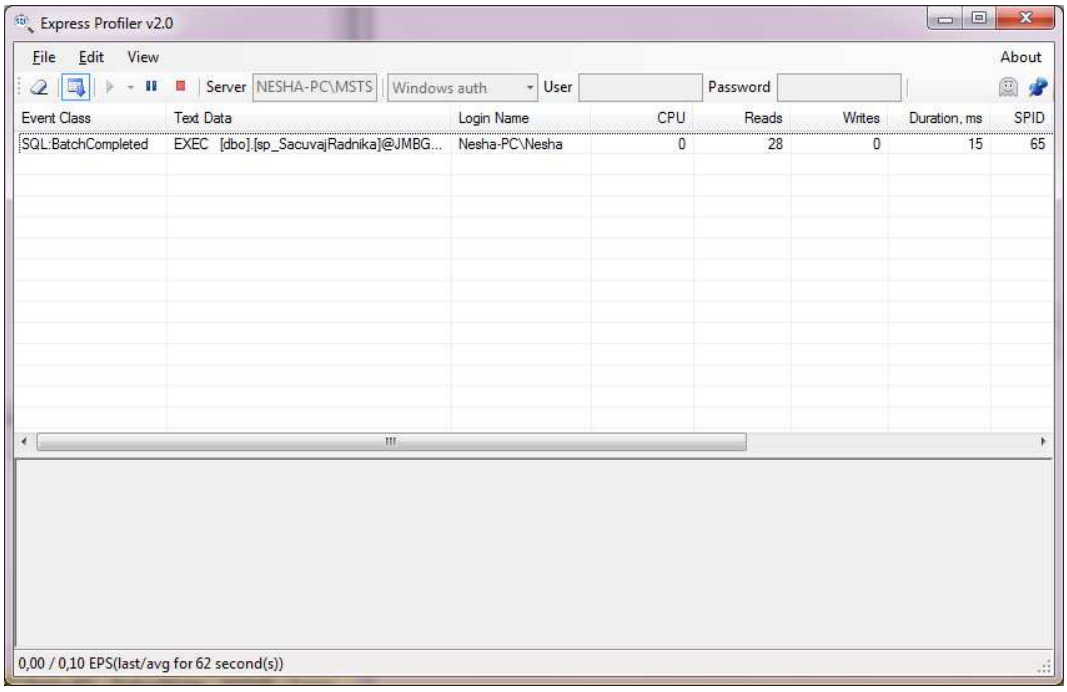
Slika 4.6.3. MOV tabeleStavkaIsplate

U ovom primeru, ovaj tip ograničenja se odnosi na prvo slovo u nazivu i na dozvoljene vrednosti atributa. Stoga će biti dovoljno test izvršiti nad tabelom Radnik. Ukoliko ograničenja implementiramo putem *trigger*-a, prilikom izvršavanja upita dobijamo sledeće rezultate:



Slika 4.6.4. Prikaz izvršavanja implementacije prostih ograničenja putem *trigger*-a

Ukoliko ograničenja implementiramo putem *stored procedure*, prilikom izvršavanja upita dobijamo sledeće rezultate:



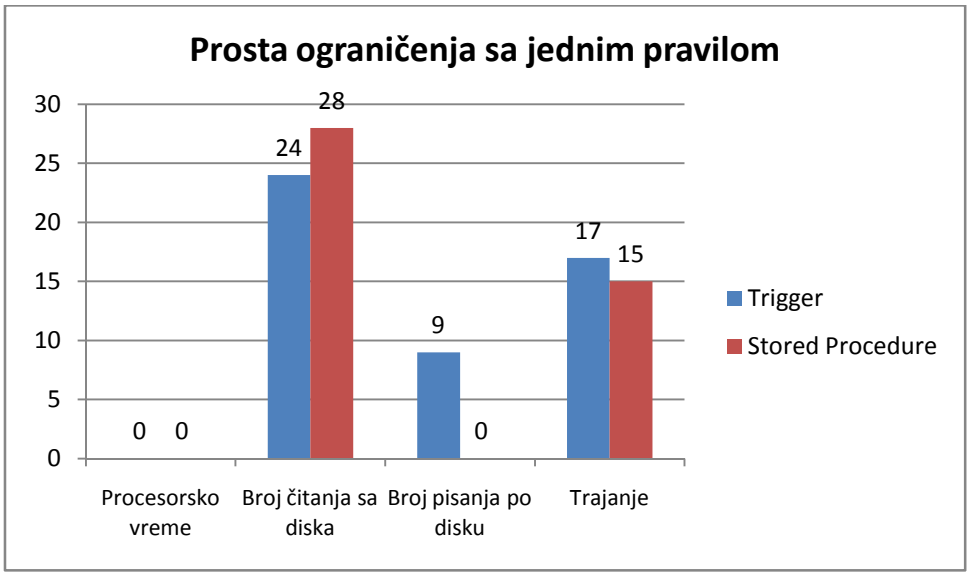
Slika 4.6.5. Prikaz izvršavanja implementacije prostih ograničenja putem *stored procedure*

Tabelarni prikaz datih rezultata:

	Procesorsko vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
Trigger	0	24	9	17
Stored Procedure	0	28	0	15

Tabela 4.6.1. Rezultati merenja prostih vrednosnih ograničenja

Grafički prikaz rezultata:



Slika 4.6.6. Grafički prikaz rezultata merenja prostih vrednosnih ograničenja

Na osnovu dobijenih rezultata dolazimo do zaključka da je po dva kriterijuma implementacija putem *Stored Procedure*-a bolja, dok je putem *trigger*-a jedan kriterijum bolji. Za kriterijum procesorsko vreme su dobijeni isti rezultati, stoga je ovaj podatak irelevantan. Broj čitanja sa diska je za 4 veći kod *Stored Procedure*. U ovom slučaju, to je procentutalno oko 14%. Kao što se vidi sa grafikona broj pisanja po disku je 9 puta veći kod implementacije putem *trigger*-a. Dok je ukupno trajanje za 2ms veće kod *trigger*-a.

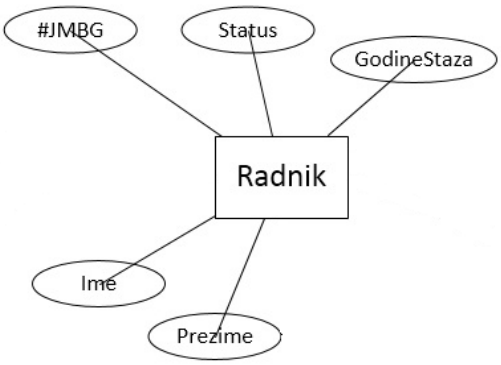
Zaključak je da je za ovaj tip ograničenja efikasnije koristiti *Stored Procedure*-u.

ograničenja koja se sastoje od više pravila

Jedino prosto ograničenje sa više pravila koje se nalazi u primeru je da JMBG mora imati 13 karaktera, prve dve cifre moraju biti manje od 32, druge 2 manje od 13 i 5.,6. i 7. cifra moraju biti u opsegu [000,014] U [900,999] i da su dozvoljene vrednosti atributa definisani u prethodnom testiranju. Stoga će biti dovoljno test izvršiti nad tabelom Radnik.

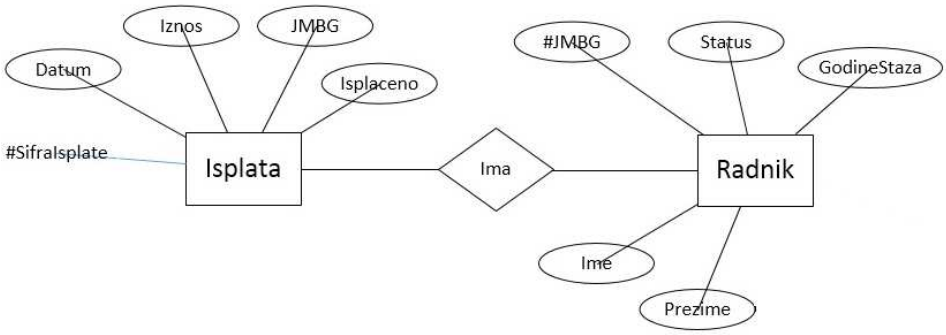
Tabele u kojima se nalazi kolona JMBG su: Radnik, OJ i Isplata.

U tabeli Radnik JMBG predstavlja primarni ključ te tabele.



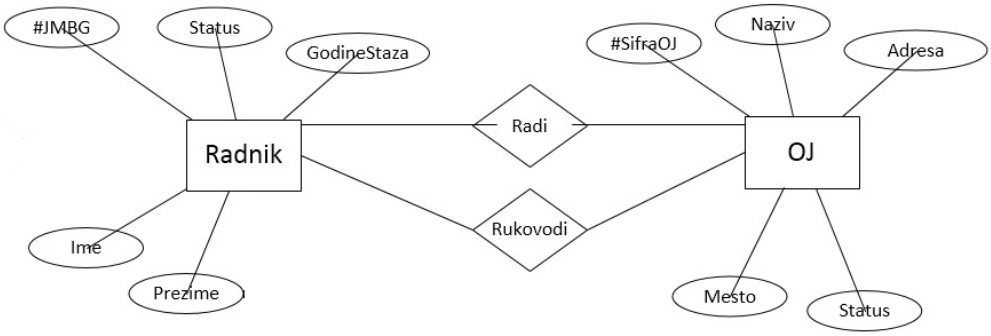
Slika 4.6.7. MOV tabele Radnik

U tabeli Isplata JMBG je spoljni ključ.



Slika 4.6.8. MOV tabelaRadnik i Isplata

U tabeli OJ JMBG je, takođe, spoljni ključ.



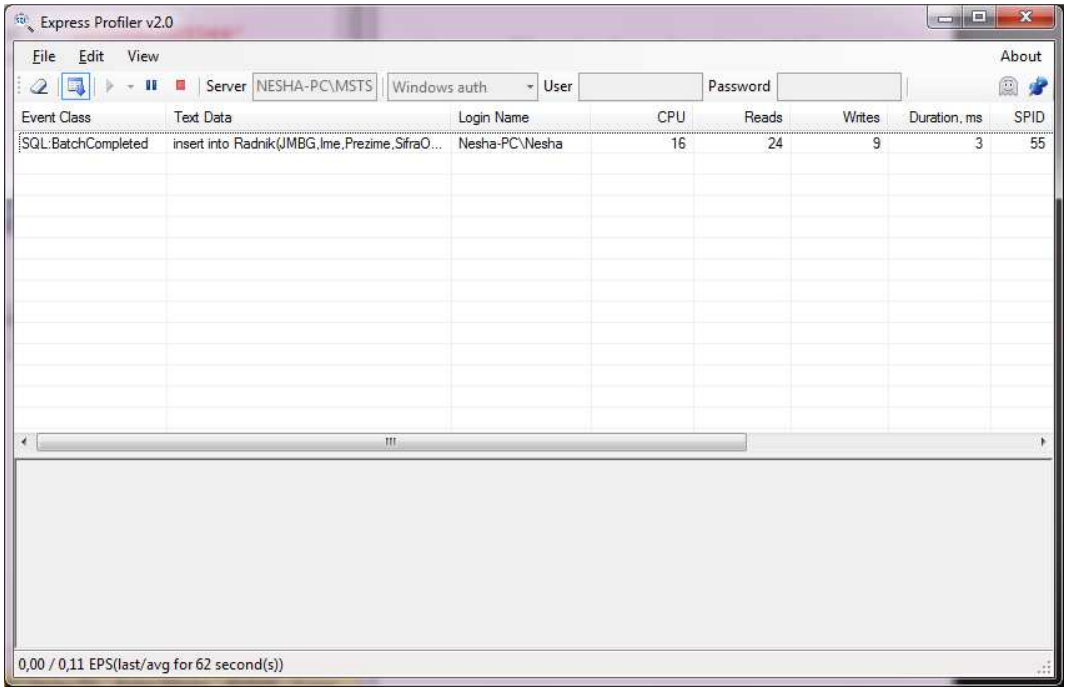
Slika 4.6.9. MOV Tabela Radnik i OJ

U svim tabelama u kojima se nalazi kolona JMBG implementirano je isto ograničenje nad tom kolonom.

Za primer testiranja i merenja performansi implementacije ovog tipa ograničenja, biće dovoljno testirati samo Tabelu Radnik. Pravila koja se moraju zadovoljiti su: JMBG mora biti u

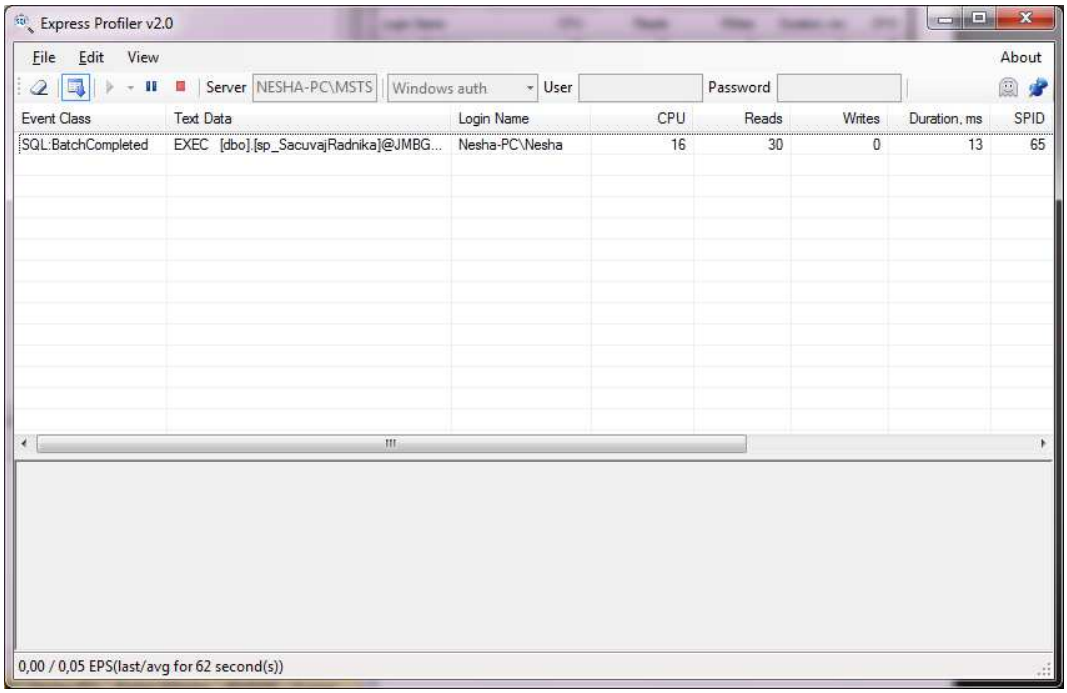
prethodnom definisanom formatu, ime i prezime moraju počinjati velikim slovima, vrednost atributa Status može biti “Aktivan, Neaktivan”.

Ukoliko ograničenja implementiramo putem *trigger*, prilikom izvršavanja upita se dobija sledeći rezultat:



Slika 4.6.10. Prosto vrednosno ograničenje sa više pravila putem *trigger*-a

Ukoliko ograničenja implementiramo putem *stored procedure*, prilikom izvršavanja upita dobijamo sledeće rezultate:



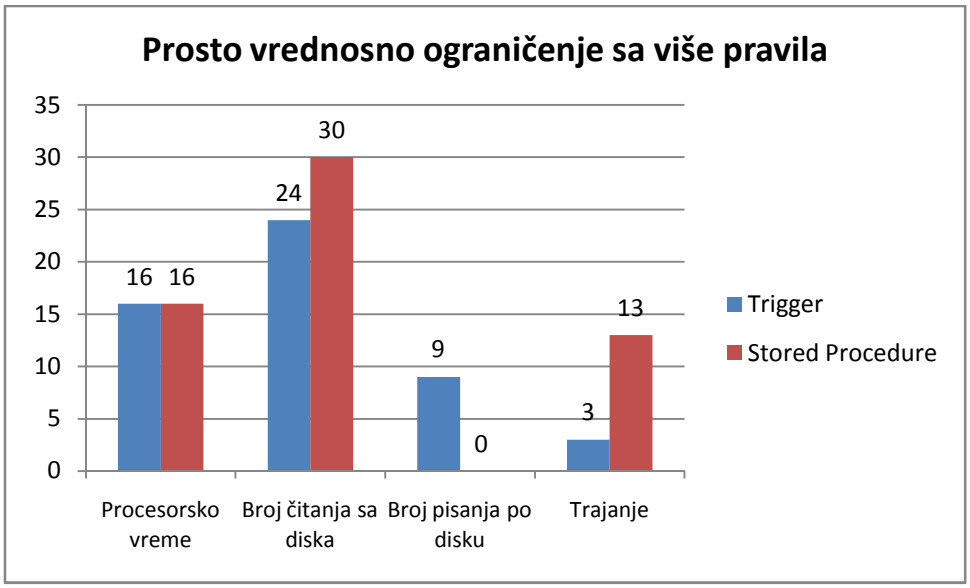
Slika 4.6.11. Prosto vrednosno ograničenje sa više pravila putem *stored procedure*

Tabelarni prikaz dobijenih rezultata:

	Procesorsko vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
Trigger	16	24	9	3
Stored Procedure	16	30	0	13

Tabela 4.6.2. Prosto vrednosno ograničenje sa više pravila

Grafički prikaz rezultata:



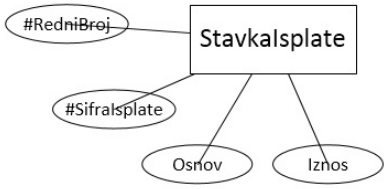
Slika 4.6.12. Grafički prikaz rezultata merenja prostih vrednosnih ograničenja sa više pravila

U oba slučaja, procesorsko vreme je potpuno isto. Implementacijom istog ograničenja putem *Stored Procedure* broj čitanja sa diska je veći za oko 18%. Broj pisanja po disku je 9 puta veći kod ograničenja putem *trigger*-a, dok je znatno veće trajanje izvršenja putem *Stored Procedure*, tako da je ona ima slabije performanse u ovom slučaju.

Po svemu viđenom, po dva kriterijum bolje rezultate pokazuje implementacija putem *Trigger*-a. Iako je prvi zaključak da je ona bolja, ipak je bolje koristiti *Stored Procedure* za implementaciju prostih vrednosnih ograničenja sa više pravila, zato što je broj pisanja po disku mnogo manji.

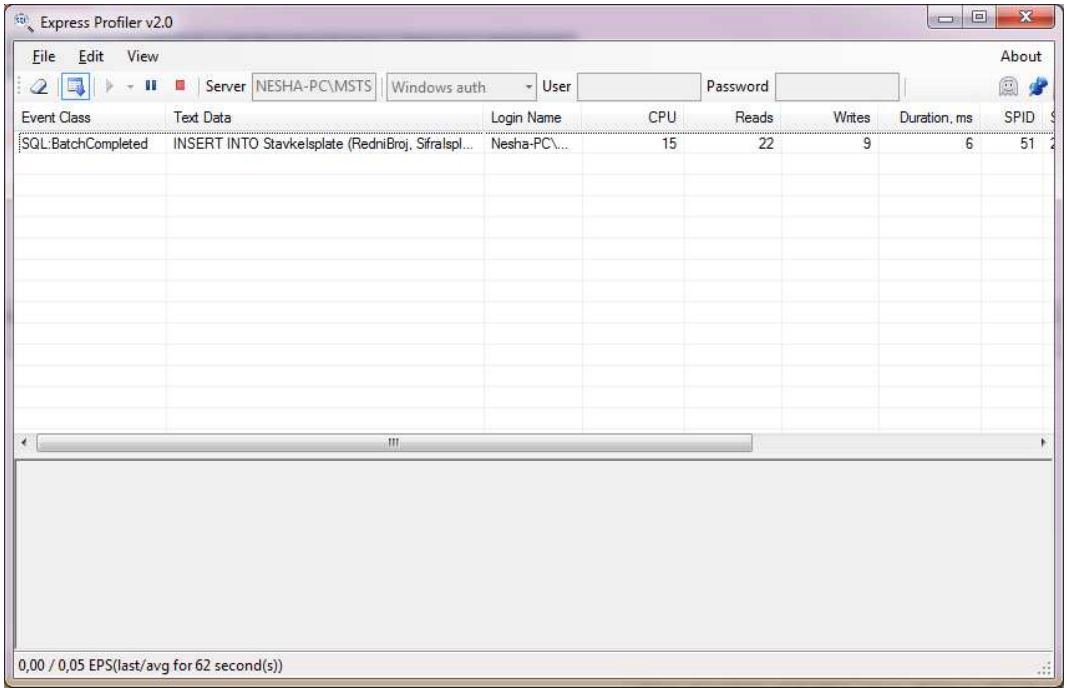
4.6.2. Merenje performansi za međuzavisnost atributa jedne tabele

U tabeli StavkaIsplate imamo ograničenje da u jednoj tabeli vrednost jednog atributa zavisi od vrednosti drugog atributa. Ako je osnov isplate „Prevoz“, iznos isplate ne može biti veći od 5000.



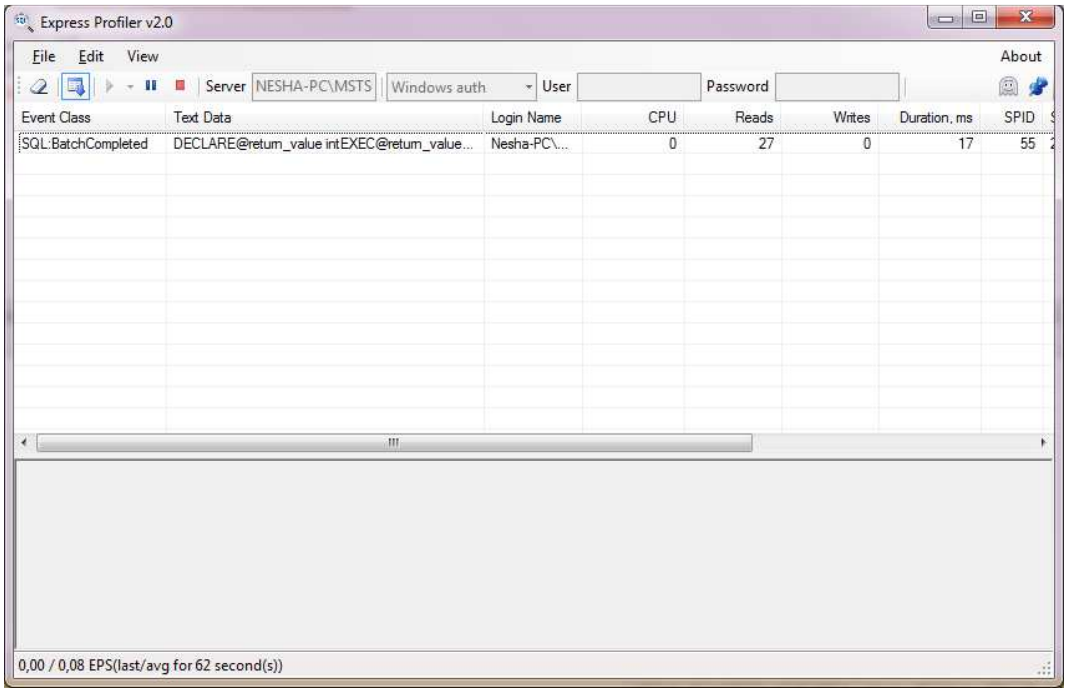
Slika 4.6.13. MOV tabele StavkeIsplate

Rezultati implementacije ograničenja putem *trigger*-a:



Slika 4.6.14. Ograničenje nad kolonom jedne tabele putem *trigger*-a

Ukoliko ograničenja implementiramo putem *stored procedure*, prilikom izvršavanja upita dobijamo sledeće rezultate:



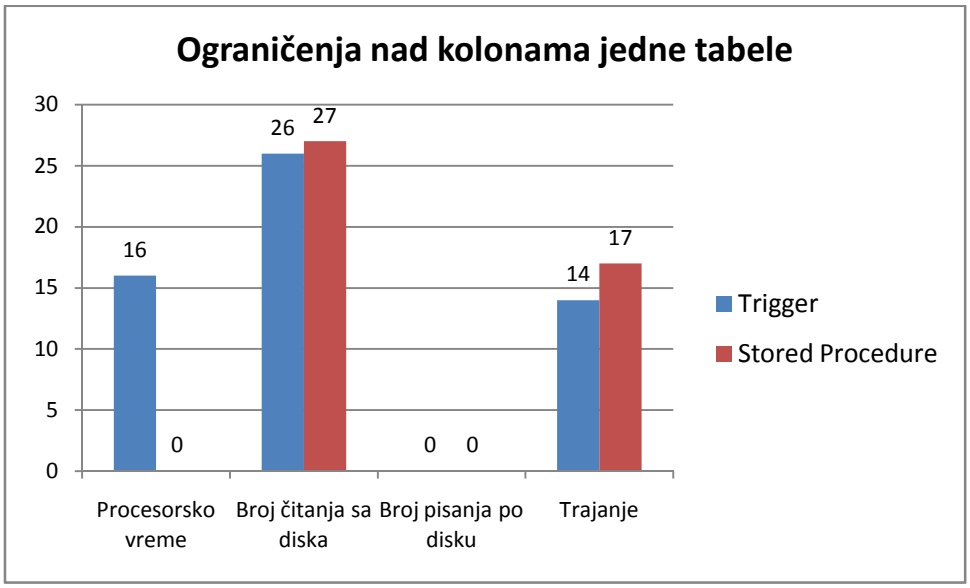
Slika 4.6.15. Ograničenje nad kolonom jedne tabele putem *stored procedure*

Tabelarni prikaz rezultata:

	Procesorsko vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
Trigger	16	26	0	14
Stored Procedure	0	27	0	17

Tabela 4.6.3. Rezultati merenja složenih vrednosnih ograničenja

Grafički prikaz rezultata:



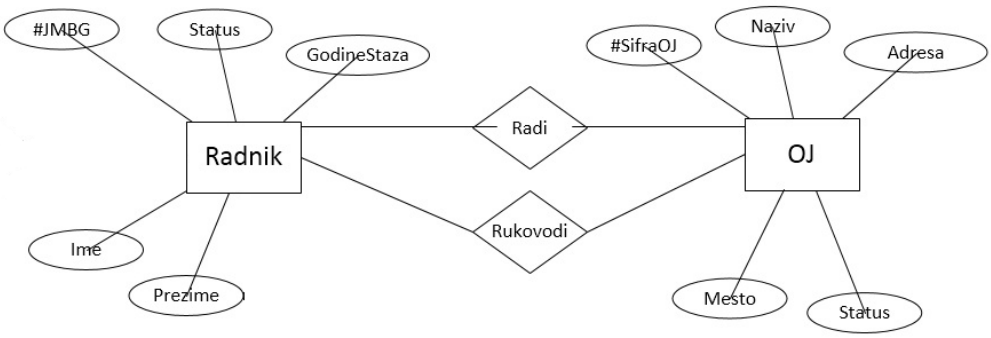
Slika 4.6.16. Grafički prikaz rezultata merenja implementacije ograničenja nad kolonama jedne tabele

Kod ove vrste ograničenja procesorsko vreme igra veoma značajnu ulogu. Kao što se vidi sa slike 4.6.16. rezulati merenja idu u prilog *stored procedure*-ama. Iako imaju lošije performanse kada je u pitanju ukupno vreme izvšavanja i broj čitanja sa diska, procesorsko vreme nam govori da ova vrsta ograničenja ostvaruje bolje performanse. Broj pisanja po disku je zanemarljiva kategorija u ovom slučaju.

4.6.3. Merenje performansi za medjuzavisnost atributa više tabela

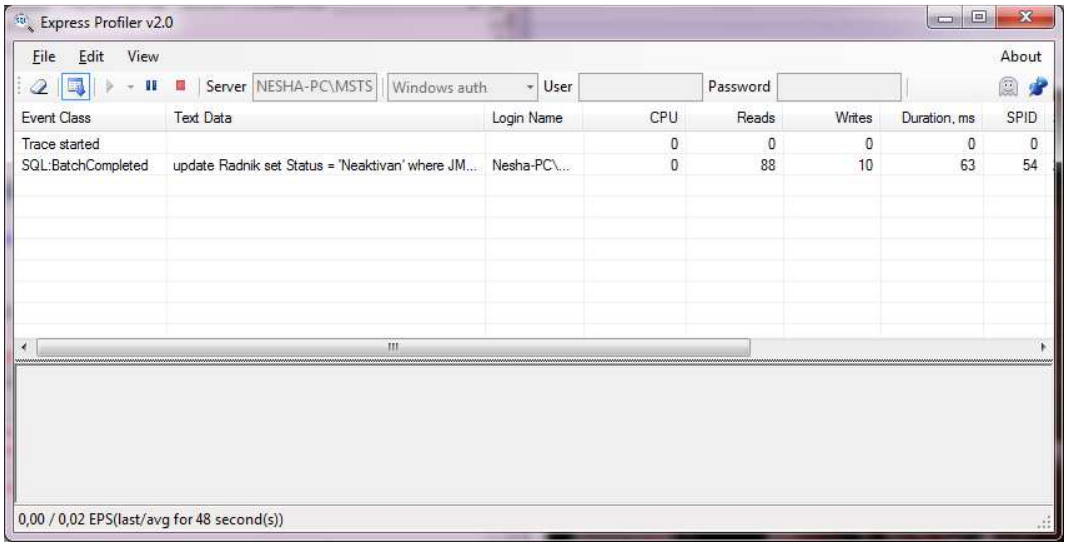
Ograničenje nad tabelom Radnik

U tabeli Radnik je definisano ograničenje da nije moguće dodeliti status „Neaktivan“ radniku koji je rukovodilac nekom OJ.



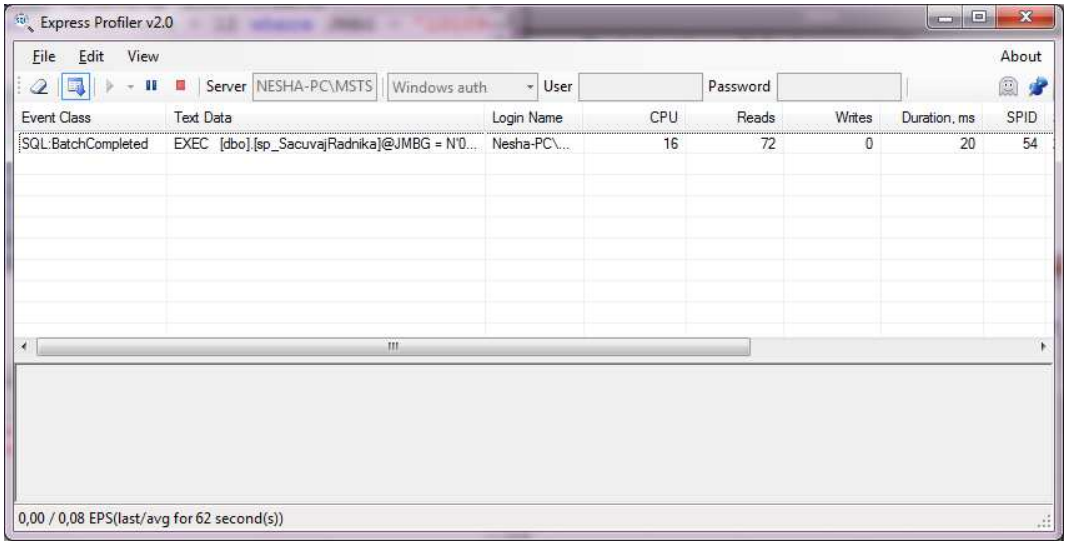
Slika 4.6.17. MOV tabela Radnik i OJ

Kada se ovo ograničenje implementira putem *triggera*, dobija se sledeći rezultat



Slika 4.6.18. Ograničenje nad kolonama više tabela putem *trigger*-a

Ukoliko se implementacija ograničenja vrši putem *stored procedure*, rezultat je:



Slika 4.6.19. Ograničenje nad kolonama više tabela putem *stored procedure-e*

Tabelarni prikaz dobijenih rezultata:

	Procesorsko vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
Trigger	0	88	10	63
Stored Procedure	16	72	0	20

Tabela 4.6.4. Rezultati merenja složenih vrednosnih ograničenja

Grafički prikaz rezultata:

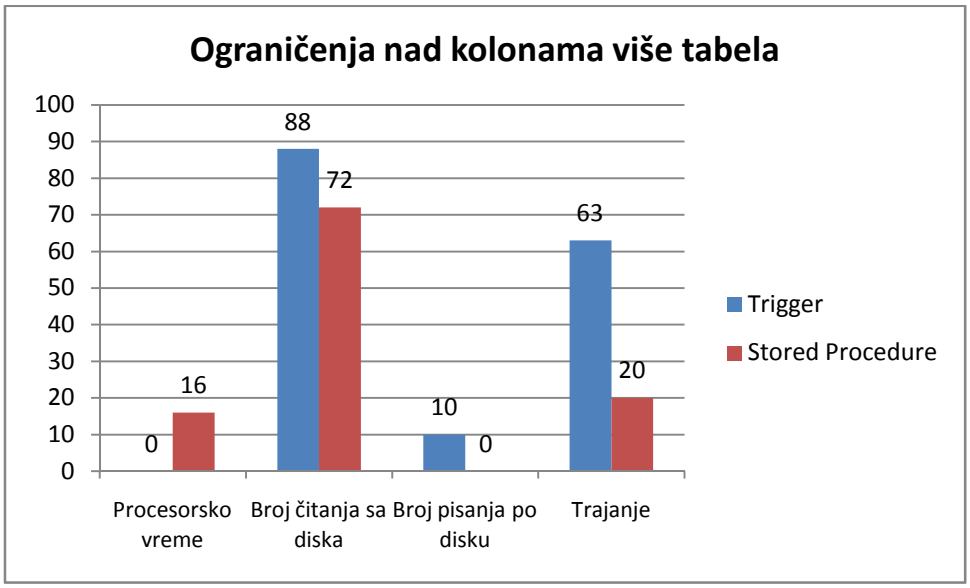


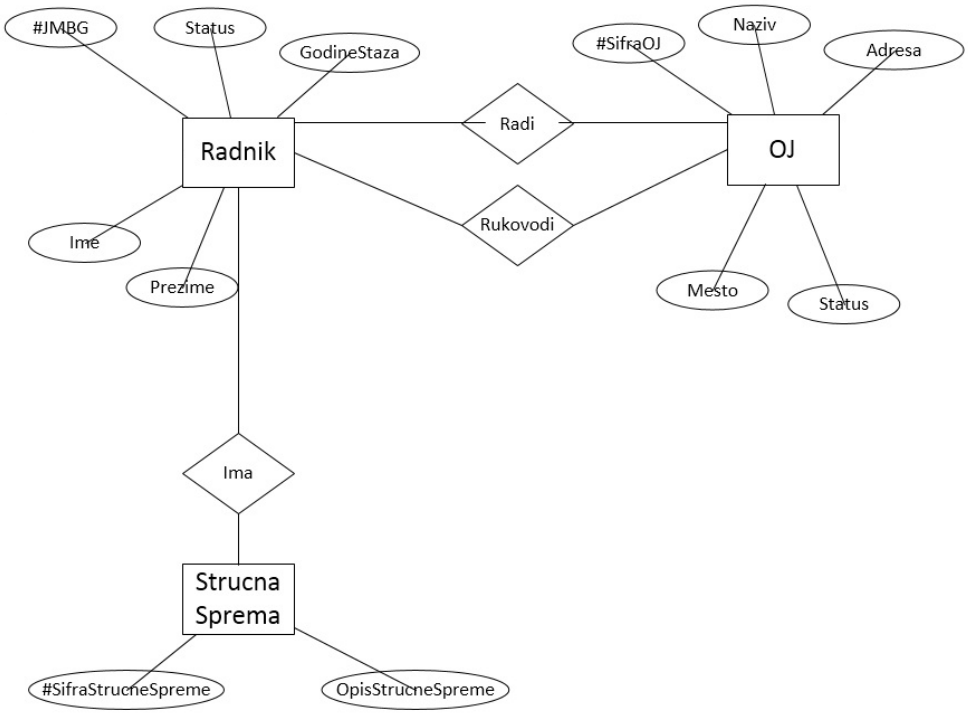
Tabela 4.6.20. Rezultati merenja različitih implementacija ograničenja nad kolonama više tabela

Prilikom implementacije ovog ograničenja putem *triggera*-a procesoru treba znatno manje vremena nego ako se ograničenje implementira putem *stored procedure*. Broj čitanja sa diska je za 16 veći ukoliko se koristi *Trigger*. Broj pisanja po disku je 10 ukoliko se koristi *Trigger*, a 0 ukoliko se koristi *stored procedure*-a . Trajanje je tri puta veće ako se ograničenje radi putem *Trigger*-a.

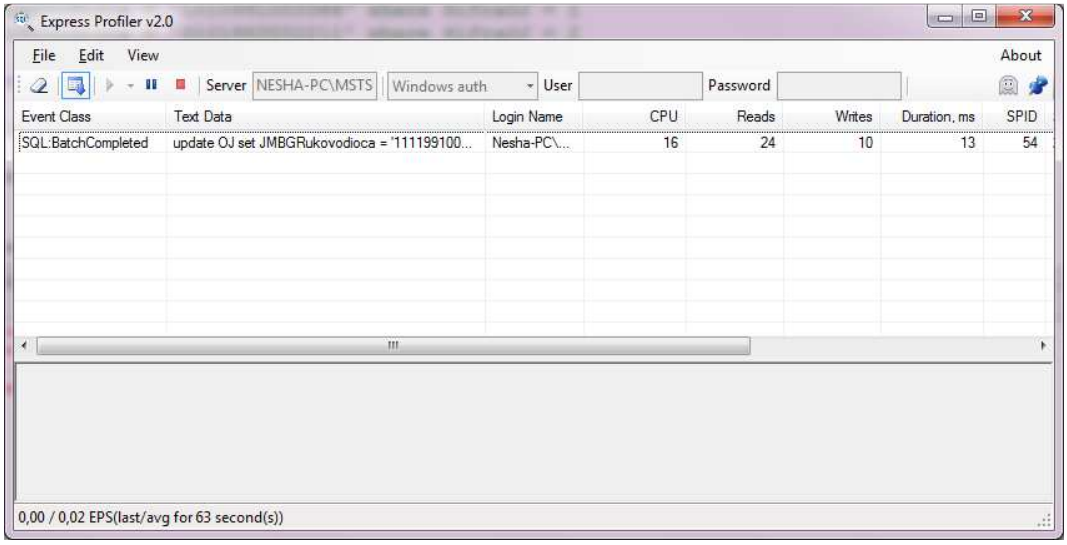
Nakon dobijenih rezultata može se zaključiti da je po jednom kriterijuma (procesorskom vremenu) bolje koristiti *Trigger*, a po preostala tri kriterijuma (broj čitanja sa diska, broj pisanja po disku i trajanju) bolje koristititi *Stored Procedure.*

Ograničenje nad tabelom OJ

Nije moguće da radnik bude rukovodilac OJ ukoliko mu stručna sprema nije veća od 3 (minimum diploma fakulteta) i ako mu status nije „Aktivan“.

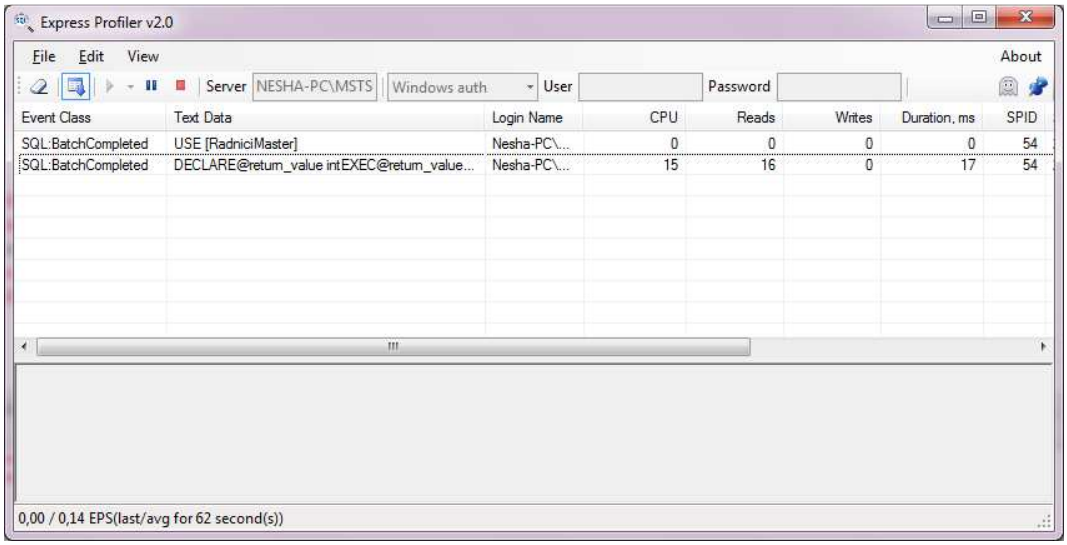


Slika 4.6.21. MOV tabela Radnik,OJ i StrucnaSprema



Slika 4.6.22. Ograničenje nad kolonama više tablea putem *triggera*

Ukoliko se implementacija ograničenja vrši putem *stored procedure*, rezultat je:



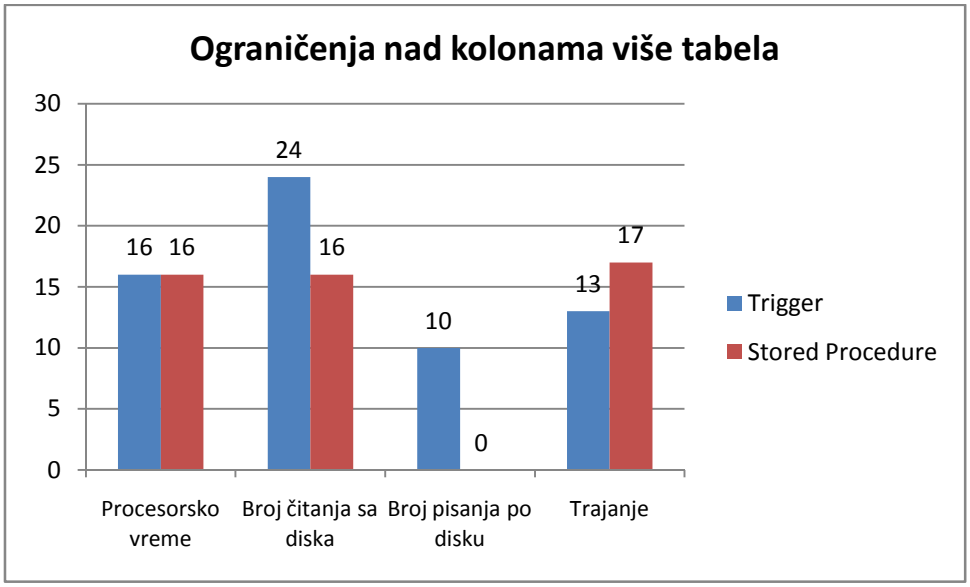
Slika 4.6.23. Ograničenje nad kolonama više tabela putem *stored procedure*

Tabelarni prikaz rezultata:

	Procesorsko vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
Trigger	16	24	10	13
Stored Procedure	16	16	0	17

Tabela 4.6.5. Rezultati merenja različitih implementacija ograničenja nad kolonama više tabela

Grafički prikaz rezultata:



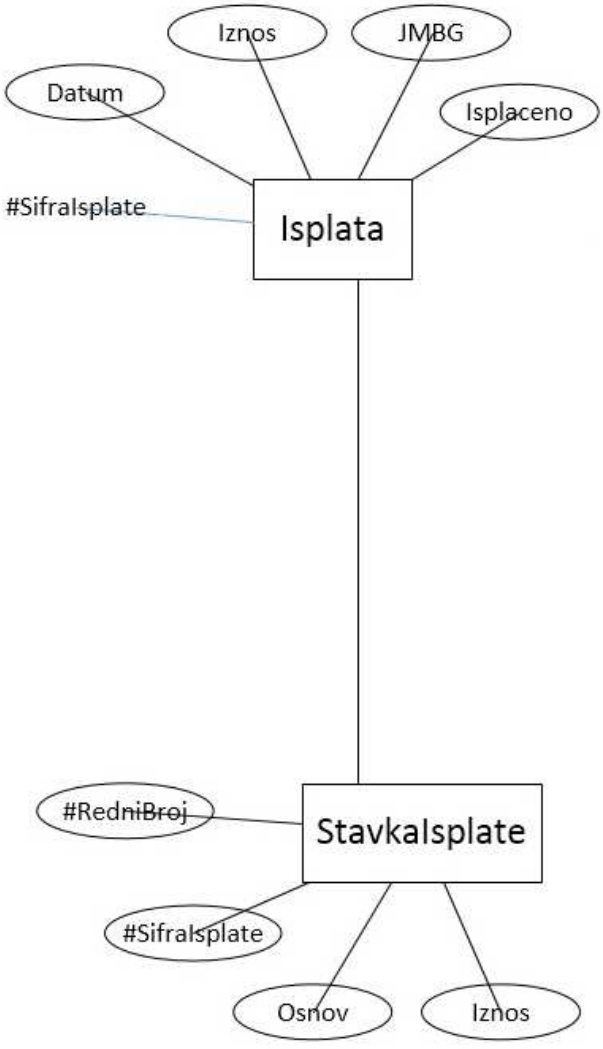
Slika 4.6.24. Rezultati merenja različitih implementacija ograničenja nad kolonama više tabela

Kod implementacije nad tabelom OJ, procesorsko vreme je isto, broj čitanja sa diska je veće za 8 kod *trigger*-a, broj pisanja po disku je 10 puta veći kod *trigger*-a, a ukupno trajanje je veće za 4ms.

I za ovaj slučaj ograničenja nad kolonama više tabela se može reći da je implementacija putem *trigger-a* bolja po broj čitanja sa diska, a za druga dva kriterijuma je bolja *stored procedure*.

Ograničenje nad tabelom Isplata

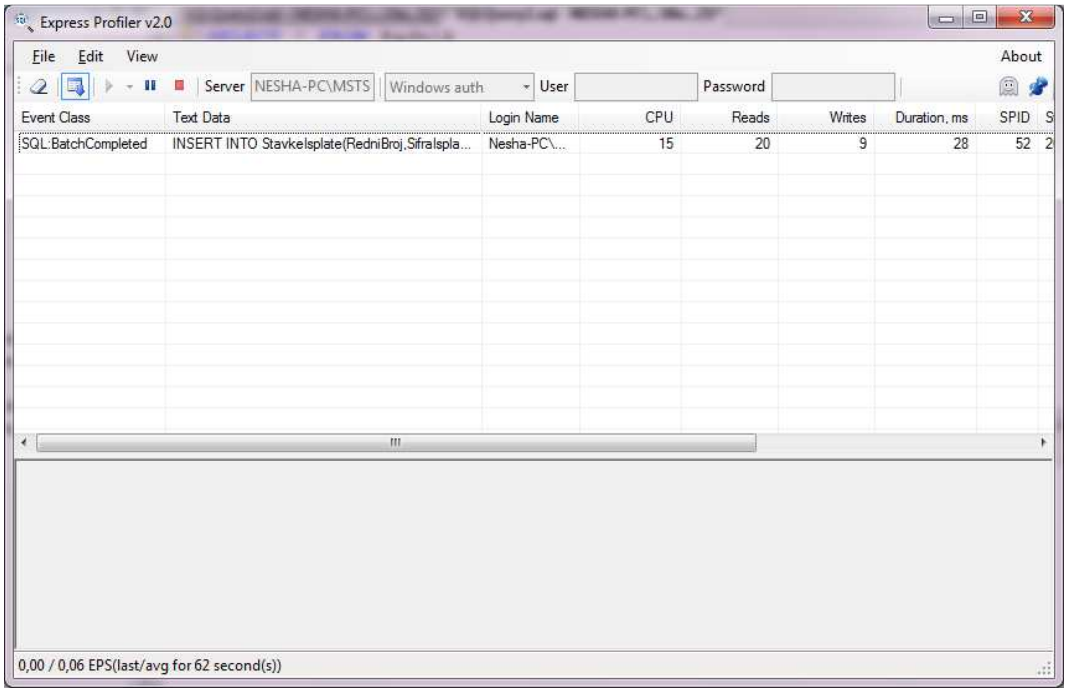
U tabeli Isplata iznos mora biti jednak sumi svih stavki koje odgovaraju toj isplati.



Slika 4.6.25. MOV tabela Isplata i StavkaIsplate

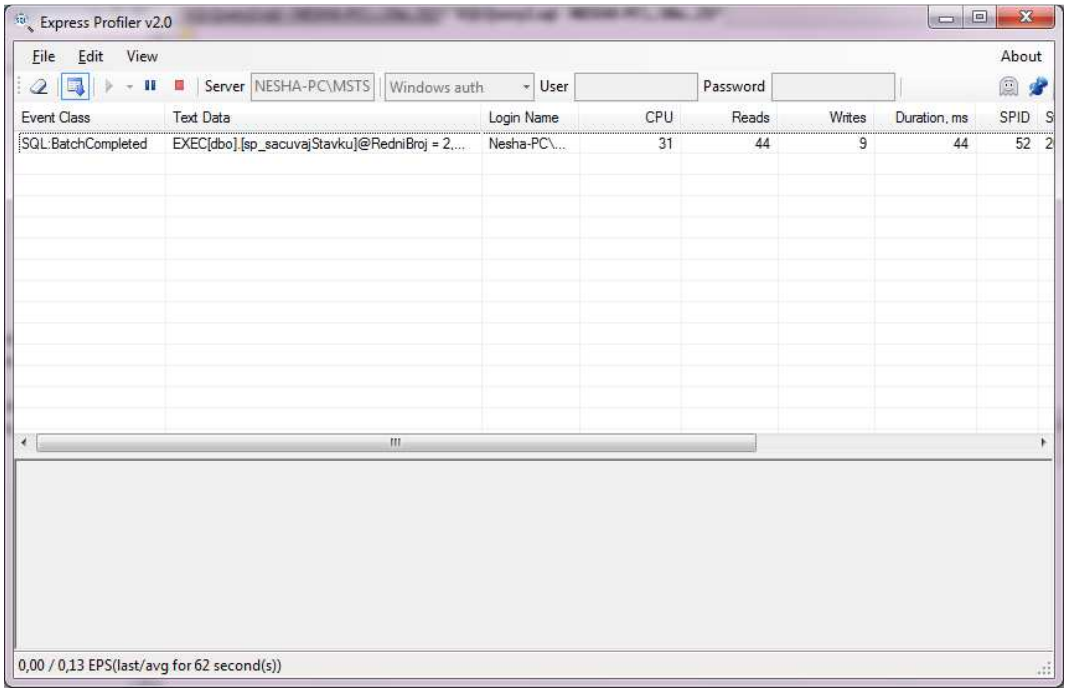
Ovaj tip ograničenja je nešto drugačiji od ostalih, do sada implementiranih, ograničenja. Ovde se integritet podataka održava tako što se nakon ažuriranja jednog podatka ažurira neki drugi podatak.

Ukoliko se ograničenje implementira putem *trigger*-a, rezultat izvršavanja izgleda ovako:



Slika 4.6.26. Ograničenje nad kolonama više tabela putem *trigger*-a

Ukoliko se ograničenje implementira putem *stored procedure*, rezultat će u profileru izgledati ovako:



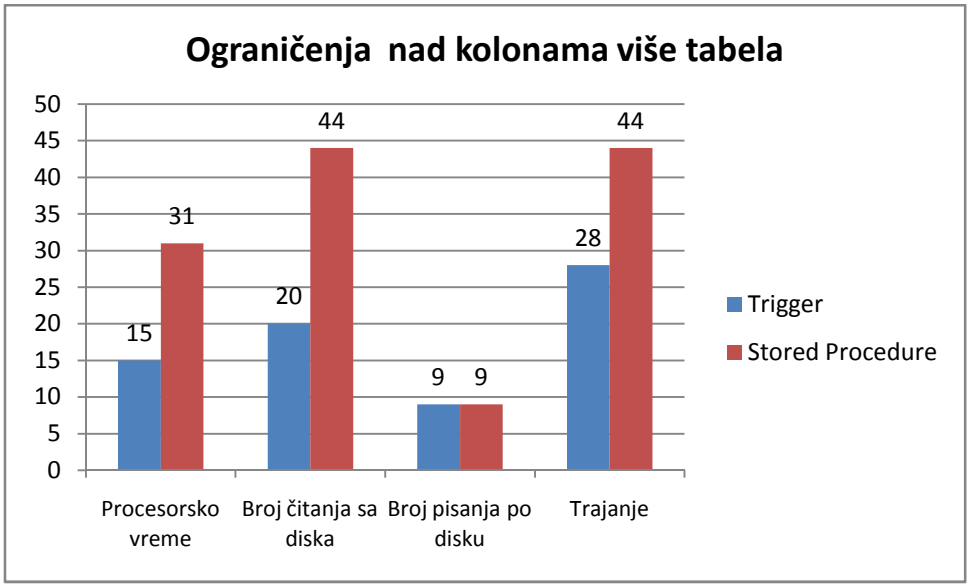
Slika 4.6.27. Ograničenje nad kolonama više tabela putem *stored procedure*

Tabelarni prikaz rezultata:

	Procesorsko vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
Trigger	15	20	9	28
Stored Procedure	31	44	9	44

Tabela 4.6.6. Rezultati merenja različitih implementacija ograničenja nad kolonama više tabela

Grafički prikaz rezultata:



Slika 4.6.28. Rezultati merenja različitih implementacija ograničenja nad kolonama više tabela

Prilikom ove implementacije *trigger* pokazuje znatno bolje rezultate. Znatno su bolje performanse za procesorsko vreme koje je znatno manje nego kod *Stored Procedure*. A takođe bolji su rezultati i kod broja čitanja sa diska i ukupno trajanje. Dok je broj pisanja po disku isti.

Iz gore navedenih činjenica se može zaključiti da je bolje koristiti *trigger* po većini kriterijuma.

4.6.4. Analiza rezultata merenja

Da bi se detaljno izmerilo koji je način bolji, vršena su merenja za svaki tip ograničenja pojedinačno. Dobijani su različiti rezultati, u nekim situacijama za neke kriterijume je *stored procedure* bila performantnija, dok je za neke bolje rezultate davao *trigger*.

U sledećoj tabeli dat je tabelarni prikaz za koji tip ograičenja[1] i za koji kriterijum se preporučuje *trigger*, a za koji *stored procedure*.

[1] Tipovi ograničenja su numerisani respektivno, kako su bili analizirani u radu

	Trigger				*Stored Procedure*			
	Proc. vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje	Proc. vreme	Broj čitanja sa diska	Broj pisanja po disku	Trajanje
I tip		√					√	√
II tip		√		√			√	
III tip		√		√	√			
IV tip	√					√	√	√
V tip				√		√	√	
VI tip	√	√		√				

Tabela 4.6.7. Tabelarni prikaz rezultata merenja po tipovima i kriterijumima

Kada je u pitanju procesorsko vreme i kada se radi prostim ograničenjima (I i II tip) situcija je identična, ali kada se posmatraju složena ograničenja pokazuje se da *trigger* uzima manje procesorkog vremena za izvršavanje upita.

Što se tiče broja čitanja sa diska bolje performanse pokazuje *trigger*. Broj čitanja sa diska predstavlja broj logičkih čitanja koje server izvršava u ime događaja. Pošto je testiranje vršeno na relativno maloj bazi, maloj u smislu broja zapisa, *trigger* će pokazivati bolje rezultate od *stored procedure* jer mu je lakše da obradi manji broj zapisa.

Upravo zbog ove konstatacije *trigger* pokazuje bolje rezultate kada je ukupno vreme izvršavanja u pitanju, što je u mnogim slučajevima od presudnog značaja.

Kriterijum u kojoj je *stored procedura* znatno bolja je broj pisanja po disku. Broj pisanja po disku predstavlja broj fizičkih zapisa koje server izvršava u ime događaja. Iako je baza mala *stored procedure* daju bolje performanse u ovom slučaju jer one predstavljaju skup T-SQL upita, a u tom slučaju je zapis po disku dosta jednostavniji

Na osnovu tabele 4.6.7., vidi se da je situacija šarenolika ali da preporuke idu ka *trigger-u*. Jedini kriterijum gde je situacija jasna je da je mnogo bolje koristiti *stored procedure-u* je kod broja pisanja po disku.

Ono što bi svakako trebalo razmotriti pre odluke o načinu implementacije ograničenja je i bezbednost. Ukoliko se ograničenja implementiraju u *stored procedure*-ama, korisnik i dalje nije sprečen da izvrši operaciju održavanja baze podatka bez korišćenja *stored procedure*-, čime se može desiti da integritet baze podataka bude narušen.

U današnjem IT-u ima sve manje i manje statičkih veb stranica, i dinamičke stranice postaju jedan globalni standard. A iza svake dinamičke veb stranice se nalazi uglavnom baza podaataka, tako da ona postaje veoma važan deo. Ovo istraživanje pokazuje da su *trigger*-i bolji kada je u pitanju manji sajt i manja baza jer mogu znatno brže da obrađuju podatke što je veoma važan, ako ne i najvažniji deo svakom korisniku. Da za što manje vremena dobije traženi rezultat. Međutim, pošto je baza i sajt koji su prezentovani u ovom radu zamišljeni za evidenciju velikog broja radnika, kada se želi manipulacija podacima bolje je koristiti *stored procedure*. Svaki veb sajt i svaka baza je projektovana na svoj način i mora se naći balans između veličine baze podatka i performansi.

Na osnovu analiza, *stored procedure*-a jeste bolja, ali je preporuka da se one koriste samo ukoliko nema drugih korisnika nad bazom koji ih neće koristiti.

5. ZAKLJUČAK

Baze podataka su nezaobilazni deo svakog informacionog sistema. Većina baza podataka su složene baze u kojima je potrebno definisati razna ograničenja.

Cilj ovog rada je bio predstaviti različite načine na koje se mogu implementirati ograničenja nad bazom podataka i održati njen integritet. Nije svaki način implementacije isti, naročito po pitanju performansi. Za velike sisteme, performanse su ključan faktor.

Kroz spoj teoretskog opisa i studijskog primera u kome je detaljno razrađena implementacija ograničenja na dva načina, generalno gledano može se reći da su stored procedure efikasnije. U nekim situacijama je *trigger* bolji, ali je u većini slučajeva rad putem *stored procedure* davao bolje rezultate.

Dat je i kratak osvrt na to kada implementacija putem *stored procedure* nije poželjan način održavanja integriteta baze podataka. Koji način implementaicije ograničenja treba izabrati zavisi od samog sistema, korisnika, od tipova ograničenja, veličine baze.

Što se tiče implementacije kod dinamičkih veb sajtova poželjnije je koristiti *trigger*-e jer je kompleksnost programskog koda baza podatka znatno manji nego kod veb aplikacija ili desktop aplikacija, a za njih je bolje koristiti *stored procedure*. Na osnovu tabele 4.6.7. vidi se da su *trigger*-i znatno bolji što se tiče trajanja. Tako da ako se želi automatska reaacija na događaj iz naše baze uvek je bolje koristiti *trigger*-e nego *stored procedure*. One se trebaju koristiti ako se želi izbeći redudantnost koda.

Treba istaći da je ovo istraživanje rađeno u SQL Serveru, sistemu za upravljanje bazama podataka kompanije Microsoft. Na koji način je bolje vršiti implementaciju u nekim drugim sistemima za upravljanje bazama podataka bi mogla biti tema za neko naredno istraživanje i poređenje razlika između implementacija u različitim sistema za upravljanje bazama podataka.

LITERATURA

[1] http://infolab.stanford.edu

[2] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

[3] http://php.net

[4] SQL Server 2014: Unlocking Real-Time Insights http://blogs.technet.com/

[5] Planning and Architecture (Database Engine) http://msdn.microsoft.com/en-us/library/cc280361(v=sql.105).aspx

[6] SQL Server Profiler http://msdn.microsoft.com/en-us/library/ms181091(v=sql.120).aspx,

[7] https://www.petri.com/sql-server-2014-news