

Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Inženjerstvo informacionih sistema

Informacioni sistem za podršku poslovanja salona za pse

Sistemi baza podataka

Natalija Gajić IT17/2017

Novi Sad, 2020.

Sadržaj

[1. Uvod 4](#_Toc41319542)

[2. Analiza programskog domena 5](#_Toc41319543)

[3. ER Model 6](#_Toc41319544)

[4. ER Model podšeme 7](#_Toc41319545)

[5. Tabelarni prikaz obeležja i ograničenja 8](#_Toc41319546)

[6. Relacioni model 10](#_Toc41319547)

[7. DDL 11](#_Toc41319548)

[8. DML 12](#_Toc41319549)

[9. SQL 12](#_Toc41319550)

[10. Objekti 13](#_Toc41319551)

[1. Tabele 13](#_Toc41319552)

[2. Sekvence 16](#_Toc41319553)

[3. Indeksi 16](#_Toc41319554)

[4. Funkcije 19](#_Toc41319555)

[5. Procedure 21](#_Toc41319556)

[6. Trigeri 23](#_Toc41319557)

[11. Upiti 24](#_Toc41319558)

[12. Funkcije 26](#_Toc41319559)

[1. Funkcija *jmg\_validation* 27](#_Toc41319560)

[2. Funkcija *vrednost\_zahteva* 28](#_Toc41319561)

[13. Procedure 29](#_Toc41319562)

[1. Procedura *promeni\_cenu* 29](#_Toc41319563)

[2. Procedura *dodaj\_radnike* 31](#_Toc41319564)

[14. Trigeri 34](#_Toc41319565)

[1. AFTER triger nad tabelom *Radni\_dan* 35](#_Toc41319566)

[2. AFTER triger nad tabelom *Zahteva* 38](#_Toc41319567)

[15. Zaključak 42](#_Toc41319568)

Tabele

[Tabela 1 - Zaposleni 8](#_Toc41319569)

[Tabela 2 - Usluga 8](#_Toc41319570)

[Tabela 3 - Materijal 8](#_Toc41319571)

[Tabela 4 - Radni\_dan 8](#_Toc41319572)

[Tabela 5 - Schema\_dan 9](#_Toc41319573)

[Tabela 6 - Ima\_schemu 9](#_Toc41319574)

[Tabela 7 - Status 9](#_Toc41319575)

[Tabela 8 - Radi 9](#_Toc41319576)

[Tabela 9 - Zahtev za salon 9](#_Toc41319577)

[Tabela 10 - Zahteva 9](#_Toc41319578)

[Tabela 11 - Mat\_za 9](#_Toc41319579)

[Tabela 12 - Se\_koristi 10](#_Toc41319580)

Slike

[Slika 1. ER šema pasniona i salona za pse 6](#_Toc41319581)

[Slika 2. Podšema podnošenja zahteva za salon 7](#_Toc41319582)

[Slika 3 - Rezultat upita za pregled izvršenih usluga zaposlenog 24](#_Toc41319583)

[Slika 4 - Rezultat upita za pregled potrošenog materijala usluga zahethteva 25](#_Toc41319584)

[Slika 5 - Rezultat upita za pregled podataka o zahtevima za svaku od usluga usluga 25](#_Toc41319585)

[Slika 6 - Rezultat upita za pregled profita pruženih usluga 26](#_Toc41319586)

[Slika 7 - Prikaz rezultata poziva funkcije *jmbg\_validation* 27](#_Toc41319587)

[Slika 8 - Informacije o zahtevu sa ID-jem 9 28](#_Toc41319588)

[Slika 9 - Rezultat funkcije vrednost\_zahteva za zahtev sa ID-jem 9 28](#_Toc41319589)

[Slika 10 - Informacije o zahtevu sa ID-jem 3 28](#_Toc41319590)

[Slika 11 - Rezultat funkcije *vrednost\_zahteva* za zahtev sa ID-jem 3 29](#_Toc41319591)

[Slika 12 - Rezultat funkcije *vrednost\_zahteva* za neispravan ID zahteva 29](#_Toc41319592)

[Slika 13 - Cene uluga pre poziva procedure *promeni\_cenu* 30](#_Toc41319593)

[Slika 14 - Cene usluga nakon poziva procedure *promeni\_cenu* kojim se snižava cena 30](#_Toc41319594)

[Slika 15 - Cene usluga nakon poziva procedure *promeni\_cenu* kojim se povišava cena 30](#_Toc41319595)

[Slika 16 - Poruka procedure *promeni\_cenu* kada su parametri koji označavaju granice intervala vrednosti cena neispravni 31](#_Toc41319596)

[Slika 17 - Poruka procedure *promeni cenu* kada parametar koji označava procenat ima nedozvoljenu vrednost 31](#_Toc41319597)

[Slika 18 – Zapisi tabele *Radi* za datum 30.04.2020. i 05.03.2020. nakon poziva procedure *dodaj\_radnike* 32](#_Toc41319598)

[Slika 19 - Zapisima tabele *Radi* za datum 30.04.2020. pre poziva procedure *dodaj\_radnike* 32](#_Toc41319599)

[Slika 20 - Zapisi tabele *Radi* za datum 29.04.2020. i 30.04.2020. nakon poziva procedure *dodaj\_radnike* 33](#_Toc41319600)

[Slika 21 - Zapisi tabele *Radi* za datum 29.04.2020. i 29.04.2021. nakon poziva procedure *dodaj\_radnike* 33](#_Toc41319601)

[Slika 22 - Zapisi tabele *Radi* za datume 28.04.2020. i 29.04.2020. nakon poziva procedure *dodaj\_radnike* 34](#_Toc41319602)

[Slika 23 - Poruka procedure *dodaj\_radnike* kada su prosleđeni parametri neispravni 34](#_Toc41319603)

[Slika 24 - Poruka procedure *dodaj\_radnike* kada prosleđeni datum ne postoji u kalendaru 34](#_Toc41319604)

[Slika 31 - Zapisi tabele *Zahteva* za radni dan sa ID-jem 5 36](#_Toc41319605)

[Slika 32 - Poruka trigera kada nije dozvoljena modifikacija torke tabela *Radni\_dan* 36](#_Toc41319606)

[Slika 33 - Zapisi tabela *Radi* i *Ima\_schemu* pre okidanja trigera 37](#_Toc41319607)

[Slika 34 – Zapisi tabela *Radi* i *Ima\_schemu* nakon okidanja trigera 37](#_Toc41319608)

[Slika 35 - Poruka trigera kada je modifikacija torke tabele *Radni\_dan* dozvoljena i izbrisani zapisi tabela *Radi* i *Ima\_schemu* 37](#_Toc41319609)

[Slika 36 - Poruka trigera kada je modifikacija torke tabele *Radni\_dan* dozvoljena i dodata šema 37](#_Toc41319610)

[Slika 37 - Zapisi tabela *Radni\_dan* i *Ima\_schemu* nakon okidanja trigera 38](#_Toc41319611)

[Slika 38 - Poruka trigera kada je uspešno dodata torka tabele *Radni\_dan* 38](#_Toc41319612)

[Slika 39 - Zapis tabele *Radni\_dan* za neradni dan sa ID-jem 1 39](#_Toc41319613)

[Slika 40 - Poruka trigera prilikom pokušaja zakazivanja pružanja zahtevane uluge za neradni dana 39](#_Toc41319614)

[Slika 41 - Zapis tabele *Zahteva* za uslugu zahteva koja nije zakazana za radan dan 39](#_Toc41319615)

[Slika 42 – Poruka trigera prilikom pokušaja zakazivanja termina pružanja zahtevane usluge kada nije zakazan dan u kom treba pružiti uslugu 39](#_Toc41319616)

[Slika 43 – Zapisi radnog dana tabele *Radni\_dan* i usluge iz tabele *Zahteva* zahtevane za drugi datum 39](#_Toc41319617)

[Slika 44 – Poruka trigera prilikom pokušaja zakazivanja pružanja zahtevane usluge za radni dan koji ne odgovara zahtevanom datumu 39](#_Toc41319618)

[Slika 45 – Zapisi šema radnog dana tabele *Ima\_schemu* i zahtevane usluge za termin koji ne odgovara šemama tabele *Zahteva* 40](#_Toc41319619)

[Slika 46 - Poruka trigera prilikom pokušaja zakazivanja termina izvršavanja usluge koji ne odgovara šemi radnog dana 40](#_Toc41319620)

[Slika 47 - Zapisi zahtevane usluge tabele *Zahteva* i zaposlenog sa neodgovarajućim statusom tabele *Radi* 40](#_Toc41319621)

[Slika 48 - Poruka trigera prilikom pokušaja angažovanja zaposlenog sa neispravnim statusom za izvršavanje usluge 40](#_Toc41319622)

[Slika 49 – Zapis torke *Zahteva* pre i posle ispravnog zakazivanja pružanja usluge 41](#_Toc41319623)

# Uvod

Projekat koji je predstavljen u ovoj dokumentaciji podrazumeva razvoj baze podataka za potrebe preduzeća koje se bavi pružanjem usluga salona. Baza podataka je projektovana da skladišti i pruža podatke o podnetim zahtevima za pružanje usluga, zaposlenima koji ih mogu pružiti, radnim danima u kojima se mogu termini zakazati i raspoloživom materijalu za pružanje usluga. Na ovaj način podržana je aktivnost zakazivanja termina za pružanje traženih usluga na osnovu podataka o raspoloživim terminima, kao i proces dokumentovanja samog izvršenja usluge, što uključuje podatke o utrošenom materijalu i zaposlenima zaduženim za pružanje usluga.

Na osnovu konceptualnog modela baza podataka, datog u vidu ER šeme, kreiran je relacioni model podataka i međurelaciona ograničenja koja nisu rešena na nivou konceptualnog modela i odražavaju pravila poslovanja preduzeća. Na osnovu relacionog modela implementirane su šeme relacije i skup ograničenja i napravljena *Microsoft SQL Server* relaciona baza podataka.

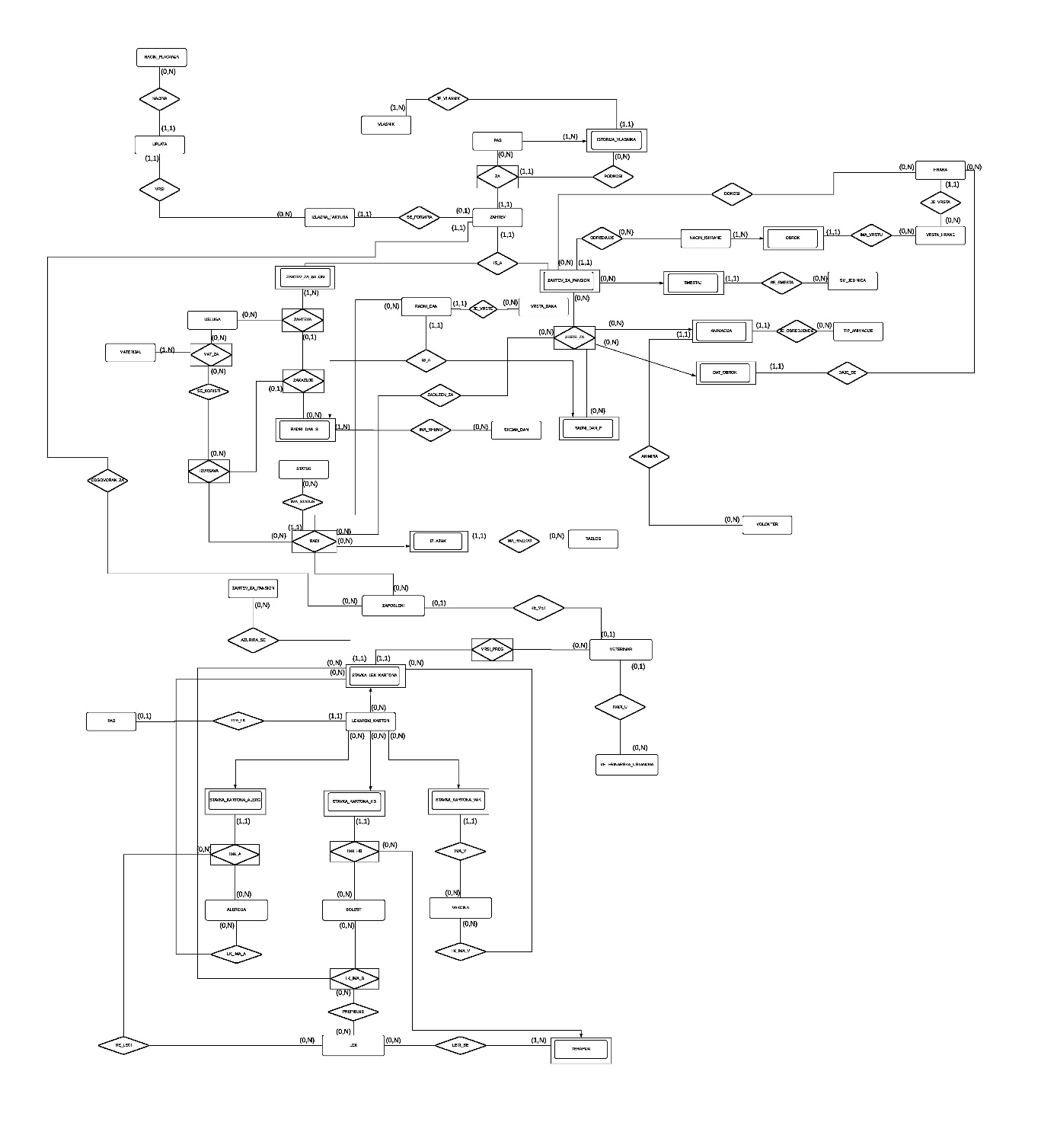
# Analiza programskog domena

Cilj projekta jeste olakšavanje poslovanja i vođenja evidencije o poslovanju preduzeća koje se bavi pružanjem usluga salona. Da bi se ovo postiglo, potreban je informacioni sistem koji će omogućiti skladištenje, pretraživanje i pružanje informacija bitnih za poslovanje preduzeća. Da bi informacioni sistem pružao korisne i ažurne podatke, potrebno je implementirati bazu podataka koja će omogućiti konzistentne podatake i skladištiti ih u skladu sa zahtevima i pravilima poslovanja. Pored toga, da bi se podacima u relativnom kratkom vremenu pristupalo i njima manipulisalo, potrebno je na nivou baze baviti se problemima fizičke organizacije podataka i potrebnih pomoćnih stuktura koje pruža sistem za upravljanje bazom podataka kako bi se postiglo efikasno korišćenje sladištenih podataka.

Baza podataka implementirana je kao relaciona *Microsoft SQL Server* (MS SQL) baza. MS SQL je sistem za upravljanje relacionim bazama podataka razvijen od strane Microsoft-a i predstavlja softverski proizvod sa primarnom funkcijom skladištenja i pružanja podataka zahtevanih od strane drugih softverskih aplikacija, pokrenutih na istom računaru ili udaljenom računaru, putem interneta. *SQL Server* ogranizuje podatke u stranice, najmanje jedinice čitanja i pisanja veličine 8KB, koje pripadaju pojedinačnom objektu baze podataka. Stranice su organizovane u *extent*-e, koje čine osam uzastopnih stranica, i mogu pripadati jednom ili više objekata u baze. Tabele se skladište u vidu serijske datoteke ili balansiranog stabla i u projektu biće razmatrane prednosti jedne strukture podataka nad drugom posmatrajući efikasnost obrade podataka. Na efikasnost obrade podataka značajno utiče korišćenje pomoćnih struktura podataka, stabla pretraživanja, kojima upravlja *SQL Server* i kroz projekat biće detaljnije objašnjeni efekti ovih stuktura i one primenjene na način da poboljšaju kvalitet osnovnih aktivnosti nad bazom podataka u toku poslovanja i korišćenja informacionog sistema.

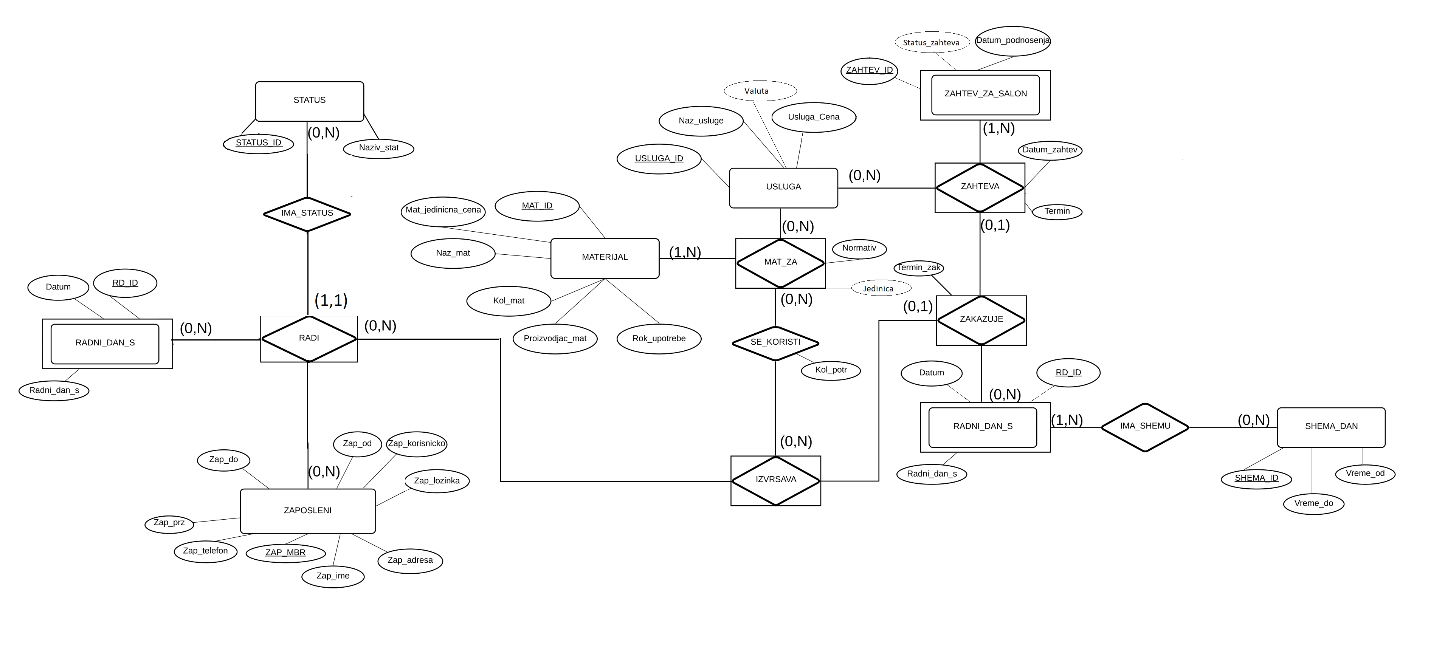
Razmatranjem potreba korisnika informacionog sistema, frekvencije potraživanja i modifikacije različitih podataka iz baze podataka, implementirana je baza podataka koja mehanizmima sistema za upravljanje bazom podataka na najefikasniji način omogućava ažuriranje i pružanje potrebnih podatka.

# ER Model



Slika 1. ER šema pasniona i salona za pse

# ER Model podšeme



Slika 2. Podšema podnošenja zahteva za salon

Podšema podnošenja zahteva za salon obuhvata modelovanje podataka koji su potrebni i prikupljaju se tokom procesa podnošenja, zakazivanja i obavljanja usluga salona zahtevanih zahtevom.

Tip entiteta *Zahtev\_za\_salon* modeluje zahtev za salon koji može podneti vlasnik psa. Jednim zahtevom za salon se zahteva jedna ili više usluga salona, što je izmodelovano tipom poveznika *Zahteva*, i za svaku od usluga zahteva se navodi termin i datum za koji se usluga zahteva. Tip entiteta *Usluga* modeluje usluge koje se mogu pružiti u salonu. Za svaku uslugu se beleži njen naziv i cena. Zahtevana usluga može biti zakazana za jedan radni dan salona u određenom terminu, dok se u jednom radnom danu salona može zakazati više zatraženih usluga zahteva**.** Usluga može biti zakazana u terminu koji odgovara radnom vremenu salona određenog radnog dana**,** tj. shemi radnog dana.

Za svaki radni dan salona se određuje najmanje jedna šema radnog dana. Šema radnog dana izmodelovana je tipom entiteta *Shema\_dan* koji sadrži informacije o radnom vremenu jednog radnog dana, odnosno *vreme\_od* i *vreme\_do*. Jedna šema primenljiva je na više radnih dana. Zakazanu uslugu može da izvrši tačno jedan zaposleni koji radi radnog dana za koji je zakazana usluga zahteva koja se pruža, odnosno ima status, izmodelovan tipom entiteta *Status*, koji govori o tome da može pružati usluge. Zaposleni u jednom radnom danu može da izvrši više zakazanih usluga zahteva.

Tip entiteta *Materijal* modeluje materijal koji može da se koristi pri pružanju više usluga salona i postoji propisan normativ za njegovo korišćenje pri pružanju usluge. Za materijal se evidentira njegov naziv, količina na skladištu, proizvođač, jedinična cena i rok upotrebe. Pri izvršavanju zakazane usluge zahteva se može utrošiti više materijala, i to samo onog koji se koristi baš za pružanje usluge koja se izvršava. Informacija o tome koji materijal se koristi za koju uslugu je izmodelovana tipom poveznika *Mat\_za* Ukoliko se prilikom pružanja usluge koristi materijal, evidentira se količina koja je potrošena, što je izmodelovano tipom poveznika *Se\_koristi*.

# Tabelarni prikaz obeležja i ograničenja

Tabela 1 - Zaposleni

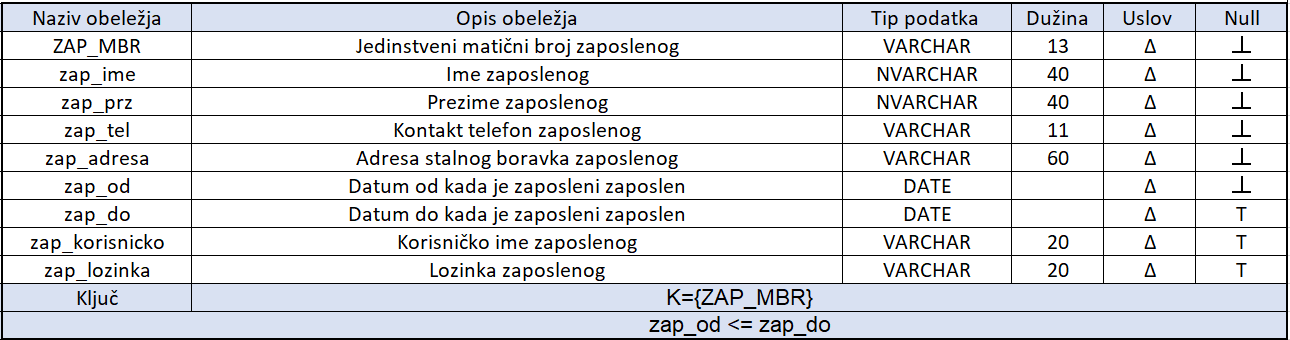


Tabela 2 - Usluga

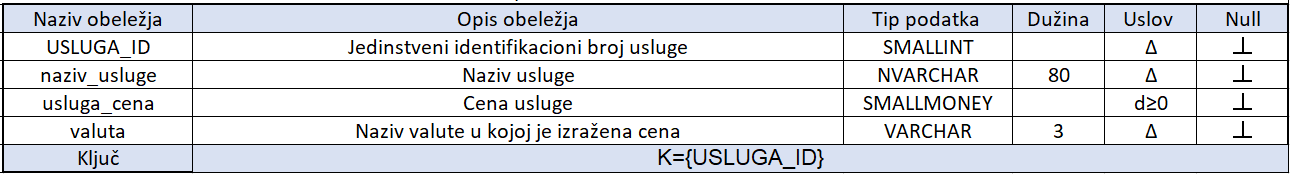


Tabela 3 - Materijal

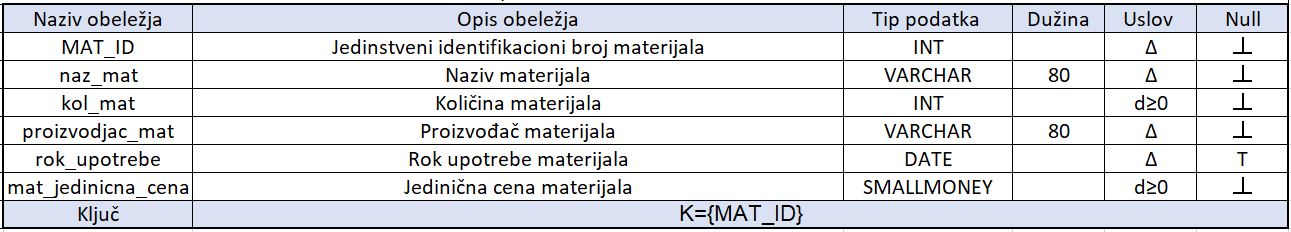


Tabela 4 - Radni\_dan

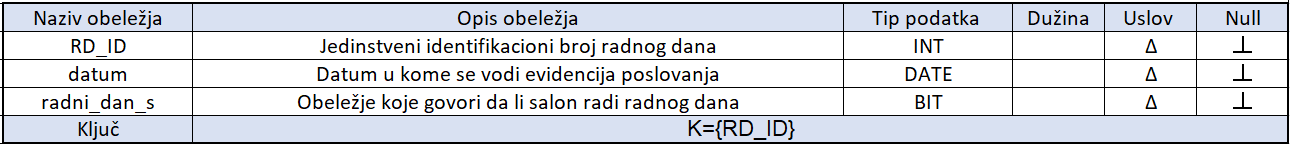


Tabela 5 - Schema\_dan



Tabela 6 - Ima\_schemu



Tabela 7 - Status

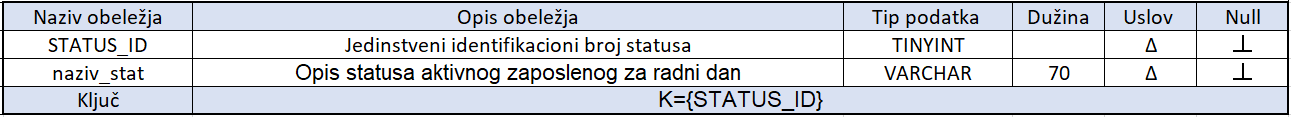


Tabela 8 - Radi



Tabela 9 - Zahtev za salon

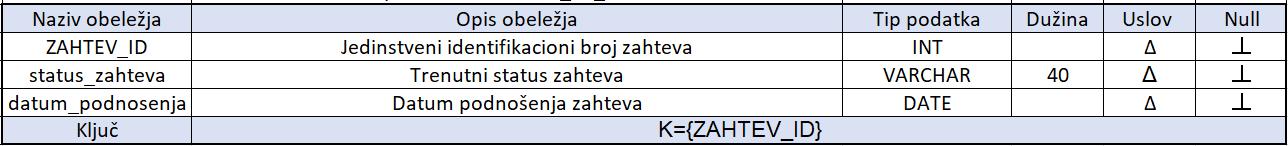


Tabela 10 - Zahteva

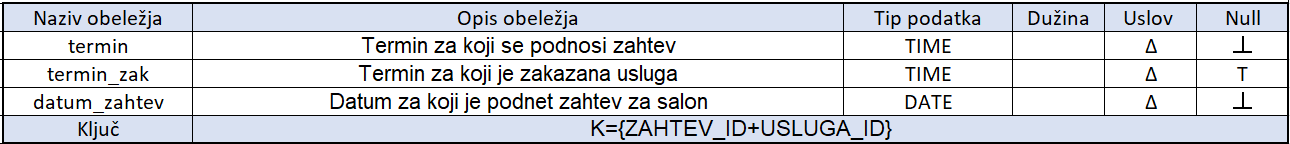


Tabela 11 - Mat\_za

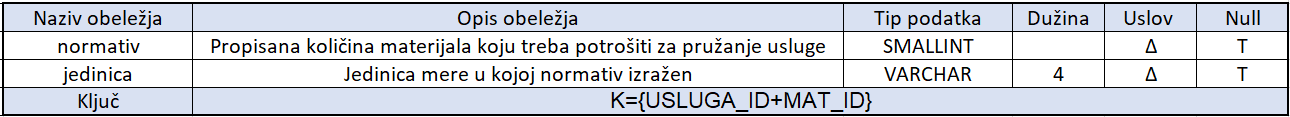
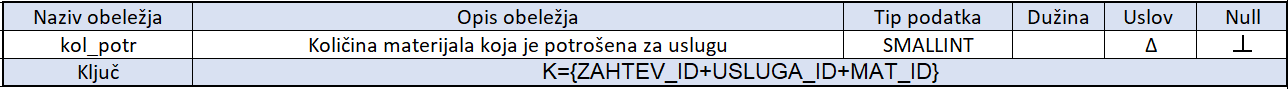


Tabela 12 - Se\_koristi



# Relacioni model

Relacioni model predstavlja implementacioni model podataka sa tipičnom strukturom šeme baze podataka koja ima strukturu tabela. Ovaj model podataka razvijao se od početka 1970-ih godina uz osnovni motiv razvoja koji je predstavljalo otklanjanje nedostataka klasičnih modela podataka, odnosno čvrste povezanosti logičkih i fizičkih aspekata, strukturalne kompleksnosti i nezasnovanosti modela na matematičkim osnovama.

Koncepti strukturalne komponente relacionog modela podataka na nivou intenzije su:

* domen,
* obeležje,
* skup obeležja,
* šema relacije i
* šema baze podataka.

Koncepti strukturalne komponente na nivou ekstenzije su:

* vrednost,
* podatak,
* torka,
* relacija i
* baza podataka.

Uobičajena reprezentacija relacije je pomoću tabele. Operacijska komponenta je deklarativna i sačinjava je jezik za manipulisanje podacima*,* jezik za definiciju podataka i upitni jezik za izražavanje upita nad jednom ili skupom relacija.

Osnovni problem nezavisnosti programa od podataka rešen je potpunim odvajanjem prezentacionog od formata memorisanja. Koncept relacije donosi stuktruralnu jednostavnost i prirodnu upotrebu fundamentalnog matematičkog koncepta. Upotrebu fizičkih adresa u relacionom modelu podataka zamenjuje korišćenje simboličkih adresa, odnosno vrednosti ključa šeme relacije, gde ključ šeme relacije predstavlja minimalni podskup skupa obeležja šeme relacije na osnovu kog se jedinstveno može identifikovati svaka torka posmatrane relacije.

Problem povezivanja podataka u ovom modelu podataka rešen je upotrebom tehnika prostiranja ključa, uvodeći pojam stranog ključa i ograničenja referencijalnog integriteta. Prevođenje simboličkih adresa u fizičke adrese posao je sistema za upravljanje bazom podataka.

Integritetna komponenta relacionog modela definisana je putem tipova ograničenja:

* ograničenje domena,
* ograničenje vrednosti obeležja,
* ograničenje torke,
* integritet entiteta (ograničenje ključa),
* ograničenje jedinstvenosti vrednosti obeležja,
* zavisnost sadržavanja,
* ograničenje referencijalnog integriteta i
* funkcionalna zavisnost.

Relaciona šema baze podataka predstavlja najsloženiji koncept strukturalne komponente na nivou intenzije. Čini je skup šema relacija i skup međurelacionih ograničenja nad tim skupom. Relaciona baza podataka predstavlja pojavu nad zadatom relacionom šemom baze podataka i reprezentuje jedno stanje realnog sistema.

# DDL

DDL je skraćeni naziv za *Data Definition Language,* jezik za definisanje podataka u relacionom modelu podataka. Ovaj jezik pruža operacije za upravljanje šemom baze podataka, odnosno omogućava kreiranje, brisanje i modifikovanje delova šeme baze podataka, tj. objekata baze podataka, kao što su tabele i indeksi. Predstavlja podskup *Structured Query Language*-a (SQL-a) za deklarisanje tabela, kolona, tipova podataka i ograničenja.

Najčešće korišćeni iskazi su CREATE, za kreiranje objekata, ALTER za modifikaciju postojećih objekata i DROP za njihovo brisanje.

Primer DDL naredbi za kreiranje i brisanje tabele :

if object\_id('Production.Radni\_dan','U') is not null

drop table Production.Radni\_dan;

create table Production.Radni\_dan

(

rd\_id int not null,

datum date not null,

radni\_dan\_s bit not null default(0),

constraint PK\_Radni\_dan\_rd\_id PRIMARY KEY(rd\_id)

);

Primer modifikacije tabele specifikacijom *default* ograničenja koje postavlja vrednost kreirane sekvence za podrazumevanu vrednost primarnog ključa:

if OBJECT\_ID('Production.DFT\_radni\_dan\_rd\_id') is not null

alter table Production.Radni\_dan

drop constraint DFT\_radni\_dan\_rd\_id;

if OBJECT\_ID('radni\_dan\_seq') is not null

drop sequence radni\_dan\_seq;

create sequence radni\_dan\_seq as int

start with 1

increment by 1;

alter table Production.Radni\_dan

add constraint DFT\_radni\_dan\_rd\_id default(next value for radni\_dan\_seq) for rd\_id;

# DML

DML je skraćeni naziv za *Data Manipulation Language,* jezik za manipulaciju podacima u relacionom modelu podataka. Ovaj jezik pruža operacije za ažuriranje relacija, odnosno dodavanje torki i brisanje i modifikovanje postojećih torki. Predstavlja podskup *Structured Query Language*-a (SQL-a) za upravljanje podacima baze podataka.

Primer naredbi za unošenje, modifikaciju i brisanje podataka iz tabele *Usluga* šeme *Production*:

insert into Production.Usluga (naziv\_usluge, usluga\_cena, valuta)

values ('Kupanje srednjih rasa', 700, 'din');

update Production.Usluga

set usluga\_cena = 600

where naziv\_usluge = 'Kupanje srednjih rasa';

delete from Production.Usluga

where naziv\_usluge = 'Kupanje srednjih rasa'

# SQL

SQL je skraćeni naziv za *Structured Query Language*, deklarativan jezik dizajniran za upravljanje podacima u relacionim sistemima za upravljanje bazama podataka. SQL je visoko standardizovan jezik i proizvođači sistema za upravljanje bazom podataka podržavaju veći broj ovih standarda. Putem SQL-a ostvaruje se komunikacija sa sistemom za upravljanje bazom podataka i na taj način manipuliše bazom podataka.

SQL omogućava pružanje podataka na uvid operacijama za izražavanje upita nad relacijom ili skupom relacija, pravljenjem spojeva putem JOIN klauzule. Podskup ovog jezika čine DDL i DML jezici, pa SQL omogućava definiciju podataka i manipulaciju podacima (unošenje, modifikacija i brisanje). Pored ovoga, omogućava i kontrolu pristupa podacima.

Primer izražavanja upita nad spojem relacija:

select naziv\_usluge 'Usluga', termin\_zak 'Zakazan termin', datum\_zahtev 'Datum'

from Production.Zahteva z

inner join Production.Radni\_dan r on z.rd\_id = r.rd\_id

inner join Production.Usluga u on u.usluga\_id = z.usluga\_id

where zap\_mbr = '0708992735052' and z.rd\_id = 4

# Objekti

*SQL Server* za implementaciju šeme baze podataka pruža objekte koje je moguće kreirati, modifikovati i brisati, naredbama *create, alter* i *drop.* Informacije o ovim objektima čuvaju se u sistemskim tabelama i moguće je pozivom funkcije *object\_id*, kojoj se prosleđuje naziv i tip objektadobiti vrednost jedinstvenog identifikatora koja se prilikom kreiranja objekta dodeljuje svakom objektu.

Prilikom izrade projekta, kreirani su sledeći objekti:

* Tabele,
* Sekvence,
* Indeksi,
* Funkcije,
* Procedure i
* Trigeri.

## Tabele

Sintaksa za kreiranje, modifikovanje i brisanje tabela:

CREATE TABLE [*database*\_*name*.][*schema*\_*name*.]*table\_name* (

*pk\_column data\_type* PRIMARY KEY*,*

*column\_1 data\_type* NOT NULL*,*

*column\_2 data\_type(size),*

*...,*

*table\_constraints*

);

ALTER TABLE [*database*\_*name*.][*schema*\_*name*.]*table\_name ...*;

DELETE TABLE [*database*\_*name*.][*schema*\_*name*.]*table\_name*;

U naredbi za kreiranje tabele obavezno je navesti naziv tabele, nazive obeležja i naziv njima pridruženih tipova podataka, za koje je potrebno navesti i dužinu ukoliko tip podatka to nalaže. Pored ovoga, moguće je specifirati ograničenje *null* vrednosti, primarnog ključa, ograničenja stranog ključa, vrednosti obeležja, *default* vrednosti i jedinstvene vrednosti obeležja. Ova ograničenja mogu biti zadata na nivou šeme relacije ili obeležja šeme relacije.

*SQL Server* podržava više tipova podataka kojima se može sprecifirati ograničenje vrednosti obeležja i svrstani su u nekoliko kategorija:

* *exact numeric* (INT, BIGINT, SMALLINT, TINYINT, NUMERIC, MONEY, SMALLMONEY),
* *character strings* (CHAR, VARCHAR),
* *Unicode character strings* (NCHAR, NVARCHAR),
* *approximate numeric* (FLOAT, REAL),
* *binary strings* (BINARY, VARBINARY)
* *date and time* (DATE, TIME, DATETIME2, SMALLDATETIME, DATETIME, DATETIMEOFFSET) itd.

Prilikom odabira tipa podatka razmatran je domen tipa podatka i operacije koje nad tipom mogu da budu izvršene kako bi se na najbolji način predstavio podatak iz realnog sistema i njime ispravno manipulisalo. Takođe, razmatrana je veličina i varijabilnost tipa podatka, kako bi se na što efikasniji način iskoristio memorijski prostor.

Podaci koji imaju tekstualnu vrednost skladište se kao vrednosti varijabilnog tipa NVARCHAR, pa se za uneti podatak zauzima onoliko memorije koliko vrednost unosa zauzima, a ne unapred određena veličina memorijskog prostora. NVARCHAR vrednosti zauzimaju više memorije od VARCHAR vrednosti, ali dozvoljavaju *Unicode* karaktere čije unošenje je, u slučaju podataka baze koja se implementira, potrebno podržati. U slučaju projektovane baze se vrednosti obeležja sa ovim tipom retko menjaju i prioritet je performansa operacije čitanja, pa je varijabilan tip poželjniji.

Za tip podatka surogat ključa pojedinih šema relacija postavljen je INT, SMALLINT ili TINYINT, uzimajući u obzir količinu podataka koji će vremenom biti uneta u bazu, odnosno broj pojedinačnih upisa entiteta posmatrane šeme relacije.

Da bi se izbegla implicitna konverzija podataka od strane sistema za upravljanje bazom podataka koja bi uticala na performanse obrade podataka, vođeno je računa o konzistentnosti tipova podataka obeležja koja reprezentuju iste podatake iz realnog sistema, kao što je to u slučaju stranih ključeva.

Da bi se obezbedila tačnost i pouzdanost podataka, nad tabelom se definišu ograničenja, koja mogu biti implementirana deklarativnim ili proceduralnim mehanzmima. Deklaritivnim mehanizmom, pirlikom kreiranja tabele, implementirana su ograničenja referencijalnog integriteta, vrednosti obeležja i torke. Ograničenja referencijalnog integriteta implementirana su definisanjem FOREIGN KEY ograničenja, a ograničenja vrednosti obeležja i ograničenja torke navođenjem CHECK i NOT NULL ograničenja.

Sintaksa za kreiranje i brisanje ograničenja:

CREATE TABLE [*database*\_*name*.][*schema*\_*name*.]*table\_name* (

*pk\_column data\_type* PRIMARY KEY*,*

*column\_1 data\_type* NOT NULL*,*

*column\_2 data\_type(size),*

*...,*

CONSTRAINT CHK\_*constratint\_name* CHECK(<*logical expression*>),

CONSTRAINT FK\_*constraint\_name* FOREIGN KEY (<*referenced\_table\_pk*>) REFERENCES [*database*\_*name*.][*schema*\_*name*.]*table\_name*

);

ALTER TABLE [*database*\_*name*.][*schema*\_*name*.]*table\_name*

DROP CONSTRAINT [*database*\_*name*.][*schema*\_*name*.]*constraint\_name*

Ograničenje integriteta entiteta implementirano je definisanjem ograničenja primarnog ključa izrazom PRIMARY KEY. Prilikom kreiranja ograničenja primarnog ključa tabele, *SQL Server* implicitno kreira *clustered index –* balansirano stablo koje predstavlja modifikaciju B stabla logički uređeno po vrednostima *clustering key*-a, koji u ovom slučaju predstavlja ključ same tabele. Na ovaj način, tabela je organizovana u vidu stabla i u njegovim listovima se nalaze podaci, pa čitanje podatka iz tabele podrazumeva prolazak kroz stablo. Performanse čitanja su bolje ukoliko *SQL Server* za pronalazak podatka treba da pročita manje stranica, odnosno ukoliko ključevi skladišteni u čvorovima stabla koji vode do lista u kojem je podatak zauzimaju manje memorije. Ovo je dodatan razlog zbog čega se prilikom odabira tipa podatka surogat ključeva uzimao u obzir domen tipa podatka koji određuje količinu memorije koju podatak tipa zauzima.

Primarni ključevi određenih šema relacija predstavljaju uniju nekoliko obeležja i prilikom zadavanja ograničenja primarnog ključa je vođeno računa o redosledu navođenja ovih obeležja jer on određuje redosled po kojem se logički organizuju podaci. Ovo znači da je u slučaju redosleda obeležja A, B, C moguće pretraživanje stabla po vrednosti A, AB ili ABC. Osnovna prednost organizacije tabele u vidu stabla naspram organizacije u vidu serijske datoteke je upravo ova pogodnost pretraživanja po logičkom redosledu koji određuju vrednosti ključeva nad kojima je izgrađeno stablo. Mehanizam indeksiranja takođe optimizuje performanse minimizovanjem broja pristupa eksternoj memoriji prilikom obrade upita, jer za čitanje reda iz tabele nije potreban prolazak kroz sve redove tabele, kao što je slučaj kod serijske organizacije.

Kod za kreiranje tabela i ograničenja tabela nalazi se u skripti za kreiranje baze podataka u priloženom fajlu create.sql.

## Sekvence

Sintaksa za kreiranje, modifikovanje i brisanje sekvenci:

CREATE SEQUENCE [*schema*\_*name* . ] *sequence*\_*name*

AS [ *built\_in\_integer\_type* | *user-defined\_integer\_type* ]

START WITH <*constant*>

INCREMENT BY <*constant*>

{ MINVALUE [ <*constant*> ] } | { NO MINVALUE }

{ MAXVALUE [ <*constant*> ] } | { NO MAXVALUE }

CYCLE | { NO CYCLE }

{ CACHE [ <*constant*> ] } | { NO CACHE }

ALTER SEQUENCE [*schema*\_*name* . ] *sequence*\_*name*;

DELETE SEQUENCE [*schema*\_*name* . ] *sequence*\_*name*;

Sekvenca je objekat koji generiše sekvencu numeričkih vrednosti u skladu sa specifiranim ograničenjima prilikom njenog kreiranja, koja se tiču minimalne i startne vrednosti, *cycle* i *chache* funkcionalnosti i vrednosti za koju sekvenca inkrementira. Kako bi se omogućilo automatsko generisanje vrednosti ključa, postavljano je DEFAULT ograničenje nad obeležjem ključa, koje za definisanje podrazumevanevrednosti koristi sekvencu. Za podrazumevanuvrednost ključa u ograničenju se postavlja sledeća neiskorišćena vrednost koju sekvenca generiše pozivom *next value for <sequence\_name>.*

Kod za kreiranje sekvenci nalazi se u skripti za kreiranje baze podataka u priloženom fajlu create.sql.

## Indeksi

Sintaksa za kreiranje i brisanje indeksa:

CREATE [CLUSTERED | NONCLUSTERED] INDEX *index\_name*

ON <[*database*\_*name*.][*schema*\_*name*.]*table\_name* > ( *column* [ ASC | DESC ] [ ,...n ] )

INCLUDE *( column\_name* [ ,...n ] )

[ WHERE <*filter\_predicate*> ]

IF INDEXPROPERTY(OBJECT\_ID(<[*database*\_*name*.][*schema*\_*name*.]*table\_name*)>,

*< index\_name >*, ꞌIndexIdꞌ )

DROP INDEX *index\_name*

ON <[*database*\_*name*.][*schema*\_*name*.]*table\_name* >

Pored grupisanih indeksa (*clustered*) *SQL Server* eksplicitnom naredbom omogućava pravljenje i negrupisanih, *nonclustered,* indeksa. U *SQL Server-u,* indeksi su strukture podataka u vidu balansiranih stabala namenjene za poboljšanje performansi pretraživanja podataka po određenom logičkom redosledu, koji je u slučaju oba indeksa određen vrednostima ključa nad kojim se ovi indeksi izgrađuju. Za razliku od grupisanih indeksa, u listovima stabala negrupisanih indeksa se ne nalaze skladišteni podaci, već pokazivač ka slogu tabele. U slučaju tabela koje su organizovane kao serijske datoteke, vrednost pokazivača je osmobitni zapis koji govori o ID-u datoteke i stranice u kojoj je zapisan slog. Kada je tabela organizovana kao stablo, odnosno kada je nad tabelom kreiran grupisani indeks, pokazivač u listovima stabla negrupisanog indeksa ima vrednost ključa grupisanog indeksa, što znači da je za pretragu podatka potrebno proći stablo negrupisanog, a zatim i grupisanog indeksa. Zbog ovoga, za performanse pretraživanja povoljna je jedinstvena i kratka vrednost ključa grupisanog indeksa. Prednost organizacije tabele u vidu stabla je što pokazivač pokazuje na logičku strukturu, tj. nije od značaja na kojoj se fizičkoj lokaciji nalazi podatak, pa je rezultat toga to da se negrupisani indeks ne ažurira promenom fizičke strukture podataka. Microsoft SQL Server podržava opciju INCLUDE prilikom kreiranja negrupisanih indeksa. Ova opcija omogućava zadavanje obeležja, čije vrednosti će biti uključene u listovima negrupisanih indeksa, pored vrednosti obeležja na osnovu kojeg je negrupisani indeks kreiran. WHERE opcija prilikom kreiranja indeksa omogućava kreiranje filtriranog indeksa, koji se formira samo nad podacima koji zadovoljavaju određeni uslov.

Indeksi su korisni samo ukoliko se upotrebljavaju prilikom obrade upita i neće se koristiti u svim tipovima upita, potrebno je napisati ispravan predikat za filtriranje redova da bi indeks bio upotrebljen. WHERE klauzula će omogućiti upotrebu indeksa samo ako je nad obeležjem klauzule napravljen indeks i upit je dovoljno selektivan. Ukoliko upit vraća veliki broj redova, manja je cena izvršavanja upita koji ne koristi indeks, već se direktno pretražuje tabela. Indeksi su još korisni i u GROUP BY izrazima prilikom korišćenja grupnih funkcija, kao i pri korišćenju ORDER BY klauzule, kako bi se izbeglo sortiranje podataka koje ukoliko se izvršava nad velikom količinom podataka može znatno uticati na performantnost obrade upita. Takođe, optimizator upita u *SQL Serveru* će koristiti indeks samo ukoliko je u upitu naveden ispravan predikat koji omogućava njegovu upotrebu, odnosno optimizator donosi odluku da li je predikat pogodan za pretragu – *searchable,* pa optimizacija podrazumeva i pisanje upita sa prikladnim argumentima pretraživanja (*search arguments* - SARGs).

*SQL Server* koristi *SQL Server Query Optimizer* kao mehanizam za analizu i sprovođenje otpimizacije upita. *SQL Server Query Optimizer* predstavlja statistički optimizator, pa koristi dinamička pravila optimizacije zasnovana na proceni cene izvođenja pojedinačnih operacija. Optimizator najpre definiše *search space* za dat upit, skup mogućih planova izvršavanja upita, pri čemu je u velikom broju slučajeva vremenski zahtevno pronaći sve moguće planove, pa optimizator mora da uspostavi balans između vremena za postizanje optimizacije i kvaliteta plana koji se optimizacijom postiže. Nakon definisanog skupa planova, optimizator procenjuje cenu svakog od planova i bira plan koji ima najmanju cenu. Procena cene podrazumeva procenu svakog operatora upita korišćenjem formula cena koje uključuju upotrebu resursa kao što su I/O operacije, centralna procesorska jedinica i memorija. Procena cene najviše zavisi od algoritma koji operator koristi, kao i od procenjenog broja zapisa koji će biti procesuirani pri izvršavanju upita. Optimizator unapred zna da li je upit dovoljno selektivan i na osnovu ove informacije dososi odluku da li će se pri izvršavanju upita koristiti indeks. Informacije o selektivnosti upita optimizator postiže održavanjem statistika distribucije vrednosti ključeva indeksa, koje čuva u posebnim sistemskim statističkim stranicama. Optimizator koristi ove statistike da bi procenio broj zapisa, odnosno kardinalnost na osnovu kojeg donosi odluku o planu optimizacije koji će biti izvršen.

Indeksi su dinamičke strukture, što znači da se ažuriraju onda kada izmena podataka u bazi to zahteva, pa upotreba velikog broja indeksa nad tabelama sa velikom količinom podataka koja se često menja može značajno, zbog operacija reorganizovanja strukture indeksa, uticati na performanse. S druge strane, indeksi su od velikog značaja za optimizaciju obrade upita nad tabelom sa velikom količinom podataka. Iz ovih razloga, da bi se na najbolji način optimizovala obrada upita, potrebno je uvideti frekvenciju potraživanja određenih podataka od strane korisnika sistema, kako bi se pretraga ovih podataka olakšala upotrebom indeksa. Kako bi se optimizovao rad sa bazom, razmatrani su podaci potrebni za poslovanje preduzeća, kako bi se njhovo potraživanje učilnilo efikasnijim i ekonomičnijim korišćenjem negrupisanih indeksa.

Pregled evidentiranih zahteva u jednom danu, odnosno zahteva koji su podneti određenog datuma i pregled zahtevanih zahteva uređenih po opadajućem redosledu vrednosti datuma.

Pretraživanje ovih podataka optimizovano je kreiranjem negrupisanog indeksa tabele *Zahtev\_za\_salon* koji za ključ uzima vrednosti obeležja koje označava datum podnošenja zahteva, uređen po opadajućem redosledu vrednosti, što se navodi izrazom DESC pri naznačavanju obeležja ključa.

Pregled zakazanih termina u određenom danu poslovanja preduzeća.

Potraživanje ovih podataka je na dnevnom nivou, jer je potrebno proveriti prethodno zakazane termine za određeni dan kako bi se odredilo zakazivanje termina novog zahteva za pružanja usluge. Kako bi se pretraga po određenom danu olakšala, nad obeležjem *rd\_id* tabele *Zahteva* u kojoj treba pronaći ove podatke, kreiran je negrupisani indeks. Pretraživanje je dodatno optimizovano opcijom INCLUDEprilikom kreiranja indeksa. Ovom opcijom su vrednosti obeležja *termin\_zak* i *usluga\_id* dodate u zoni listova, što čini da je za dobijanje podatka o vrednosti ovih obeležja dovoljan prolazak kroz stablo negrupisanog indeksa, odnosno dodatno se ne pretražuje serijska datoteka ili grupisani indeks kako bi se podatak pročitao.

Pregled izvršenih ili zakazanih usluga zahteva za koje je zadužen određeni zaposleni, određenog dana poslovanja ili u određenom periodu određenom početnim i krajnjim datumom radnog dana u kom se posmatra poslovanje zaposlenog.

Pretraga ovih podataka omogućava poslodavcu uvid u angažovanje zaposlenog, kao i pregled zahteva koji su dodeljeni zaposlenom određenog dana kako bi se organizovalo pružanje usluga tog dana. Nad tabelom *Zahteva* kreiran je negrupisan indeks nad obeležjima *zap\_mbr* i *rd\_id*. Za razliku od grupisanih indeksa, nad jednom tabelom može biti kreirano više grupisanih indeksa, kao što je u slučaju tabele *Zahteva*.

*SQL Server* takođe podržava kreiranje *unique* negrupisanih indeksa, koji garantuju da ne postoje duplikati vrednosti ključa, gde ključ predstavlja obeležje nad kojim je indeks kreiran. Validacija jedinstvenosti u slučaju ovakvih indeksa ne razlikuje se zanačajno od validacije koja se postiže kreiranjem UNIQUE ograničenja, ali indeksi pružaju više mogućnosti, kao što je filtriranje podataka čija se jedinstvenost vrednosti obeležja proverava. Filtrirani indeksi, dobijeni korišćenjem klauzule WHERE prilikom kreiranja, mogu se koristiti za implementaciju ograničenja jedinstvene vrednosti obeležja nad obeležjem koje može da bude *null* vrednosti, pa se u tabeli za vrednost ovog obeležja može naći više redova *null* vrednosti, što UNIQUE ograničenje ne dozvoljava. Korisničko ime obeležje je ovog tipa, gde zaposleni ne mora da poseduje korisničko ime (*zap\_korisnicko* ima *null* vrednost), a svako korisničko ime koje je dodeljeno zaposlenom mora biti jedinstveno. Nad tabelom *Zaposleni* kreiran je filtrirani *unique* indeks nad obeležjem *zap\_korisnicko* gde je uslov filtriranja provera *null* vrednosti ovog obeležja.

Sintaksa za kreiranje *unique* indeksa:

CREATE UNIQUE INDEX *index\_name*

ON <[*database*\_*name*.][*schema*\_*name*.]*table\_name* > ( *column* [ ASC | DESC ] [ ,...n ] )

INCLUDE *( column\_name* [ ,...n ] )

[ WHERE <*filter\_predicate*> ]

Kod za kreiranje indeksa nalazi se u skripti za kreiranje baze podataka u priloženom fajlu create.sql.

## Funkcije

Korisnički definisane funkcije enkapsuliraju ponovo upoterbljiv Transact-SQL (T-SQL) kod, gde T-SQL predstavlja proširenje SQL-a namenjeno interakciji sa relacionim bazama podataka koje se koristi od strane *SQL Servera.* Funkcije imaju pristup podacima skladištenim u bazama podataka, ali ne mogu da izvršavaju DDL naredbe, odnosno ne mogu da kreiraju ili modifikuju objekte baze podataka. Pored toga, izvršavanjem funkcija se ne mogu izmeniti podaci skladišteni u bazi DML naredbama. Pozivi funkcija u SQL bloku mogu biti na bilo kom mestu gde se i ugrađene funkcije mogu pozivati i prilikom poziva se mora koristiti poziciona notacija argumenata.

Postoje dva tipa korisnički definisanih funkcija:

* Skalarne funkcije – pozivom vraćaju jednu vrednost i
* *Table*-*valued* funkcije – vraćaju tabelu kao povratnu vrednost.

Shodno podeli korisnički definisanih funkcija, *SQL Server* funkciji dodeljuje tip koji se pored naziva funkcije prosleđuje kao parametar *object\_id* funkcije:

* FN – SQL skalarna funkcija,
* IF – *inline table-valued* funkcija (*table-valued* funkcija sa jednim redom koda) i
* TF – SQL *table-valued* funkcija.

Skalarne funkcije se mogu javiti bilo gde u upitu gde se može pojaviti izraz koji vraća jednu vrednost. Mogu primiti više više parametara.

Sintaksa za kreiranje i brisanje skalarnih funkcija:

CREATE FUNCTION [*database*\_*name*.][*schema*\_*name*.]*function\_name*

(

@*param1\_name* AS *param1\_data\_type*,

@*param2\_name* AS *param2\_data\_type*,

...

)

RETURNS *return\_value\_data\_type*

AS

BEGIN

...

RETURN *return\_value*

END;

GO

IF OBJECT\_ID(*database*\_*name*.][*schema*\_*name*.]*function\_name*, 'FN') IS NOT NULL

DROP FUNCTION *database*\_*name*.][*schema*\_*name*.]*function\_name*

*Inline table-valued* funkcije mogu primiti više parametara i povratna vrednost im je tabela koja nastaje kao rezultat upita u *return* bloku.

Sintaksa za kreiranje i brisanje *inline table-valued* funkcija:

CREATE FUNCTION [*database*\_*name*.][*schema*\_*name*.]*function\_name*

(

@*param1\_name* AS *param1\_data\_type*,

@*param2\_name* AS *param2\_data\_type*,

...

)

RETURNS TABLE

AS

RETURN

(

SELECT ...

);

GO

IF OBJECT\_ID(*database*\_*name*.][*schema*\_*name*.]*function\_name*, 'IF') IS NOT NULL

DROP FUNCTION *database*\_*name*.][*schema*\_*name*.]*function\_name*

*Multi-statement table-valued* funkcije mogu primiti više parametara, a povratna vrednost im je tabela koja nastaje kao rezultat upita u BEGIN ENDbloku. Kod ovih funkcija neophodno je eksplicitno definisati strukturu tabele koja se vraća kao rezultat i eksplicitno je napuniti podacima SELECT naredbom.

Sintaksa za kreiranje i brisanje:

CREATE FUNCTION [*database*\_*name*.][*schema*\_*name*.]*function\_name*

(

@*param1\_name* AS *param1\_data\_type*,

@*param2\_name* AS *param2\_data\_type*,

...

)

RETURNS @*returntable* TABLE

(

*column1* *column1\_data\_type*,

*column2* *column2\_data\_type*,

*column3* *column3\_data\_type*

)

AS

BEGIN

INSERT @*returntable*

SELECT *col1*, *col2*, *col3*

...

RETURN

END;

GO

IF OBJECT\_ID(*database*\_*name*.][*schema*\_*name*.]*function\_name*, 'TF') IS NOT NULL

DROP FUNCTION *database*\_*name*.][*schema*\_*name*.]*function\_name*

Za potrebe projekta kreirane su dve skalarne funkcije, priložene u fajlu funkcije.sql.

## Procedure

Procedure enkapsuliraju T-SQL kod i pozivaju se putem EXECUTE naredbe. Mogu primiti više ulaznih parametara, koji mogu biti obavezni ili opcioni parametri koji imaju definisanu podrazumevanu vrednost. OUTPUT izraz se postavlja pored naziva izlaznih parametara. Za razliku od funkcija, izvršavanjem procedura se mogu modifikovati podaci iz baze i kreirati i modifikovati objekti baze kao što su tabele i indeksi.

Sintaksa za kreiranje i brisanje procedura:

CREATE PROCEDURE [*database*\_*name*.][*schema*\_*name*.]*procedure\_name*

@*param1* AS *param1\_data\_type*,

@*optional\_param2* AS *param2\_data\_type* = <*param2\_value*>

@*output* AS *output\_data\_type*  = <*output\_value*> OUTPUT

AS

BEGIN

SET NOCOUNT ON;

...

SET @*output* = ... ;

...

RETURN;

END

IF OBJECT\_ID([*database*\_*name*.][*schema*\_*name*.]*procedure\_name*, 'P') IS NOT NULL

DROP PROCEDURE [*database*\_*name*.][*schema*\_*name*.]*procedure\_name*;

SET NOCOUNT ON je naredba kojom se isključuje ispisivanje poruka na konzoli prilikom izvršavanja procedure. U slučaju procedura koje se pozivaju često, isključivanje poruka poboljšava performantnost procedura, jer je u tom slučaju manje komunikacije putem mreže kada se poruke ne šalju klijentu. U praksi je korišćenje procedura korisno jer mogu da smanje saobraćaj mreže, pošto se njihovim korišćenjem izbegava slanje međurezultata izvršavanja ili pojedinačnih T-SQL komandi. Takođe, omogućavaju da korisnik ne vidi promene fizičke strukture ili strukture same procedure, jer joj pristupa samo putem poziva, poznavajući njene parametre.

Za preuzimanje vrednosti izlaznog parametra procedure neophodno je deklarisati promenljivu koja će preuzeti vrednost, i u pozivu procedure proslediti ovu promenljivu kao vrednost izlaznog parametra uz novođenje izraza OUTPUT ili OUT.

Sintaksa za izvršavanje procedure:

DECLARE @*get\_output* AS *procedure\_output\_data\_type*;

EXECUTE [*database*\_*name*.][*schema*\_*name*.]*procedure\_name*

@*param1* = *<param1\_value>*,

@*output* = @*get\_output* OUT;

SELECT @*get\_output*;

GO

Za potrebe projekta kreirane su dve uskladištene procedure, priložene u fajlu procedure.sql.

## Trigeri

Triger predstavlja posebnu vrstu skladištene procedure koja je povezana sa određenim DML ili DDL događajem nad tabelom ili pogledom. Specifikaciju trigera čini oblast aktiviranja (tabela nad kojom se definiše), specifikacije operacija koje ga pokreću, uslovi pod kojima se triger aktivira, vreme aktiviranja, frekvencija aktiviranja (jednom za celu operaciju ili za svaku torku koja je predmet operacije, pojedinačno) i aktivnost koju triger treba da realizuje. Triger ne može eksplicitno da bude pozvan, već se aktivira automatski prilikom izvođenja operacije koja može da ga pokrene ako su ostvareni specifirani uslovi za njegovo aktiviranje.

*SQL Server* podržava dve vrste DML trigera, koji su povezani sa INSERT, UPDATE ili DELETE događajima:

* AFTER trigere – okidaju se nakon izvršene DML naredbe i
* INSTEAD OF trigere – izvršava se umesto DML naredbe.

U *SQL Server-*u se triger izvršava samo jednom za jednu DML istrukciju, bez obzira koliko torki je pod uticajem operacije koja okida triger i šema nad kojom je definisan triger mora biti ista kao i šema tabele koja je specifirana kao oblast aktiviranja.

U telu trigera, moguće je pristupiti privremenim tabelama *inserted* i *deleted*. Tabele sadrže torke koje su pod uticajem DML instrukcije koja je dovela do okidanja trigera. U slučaju INSERT i UPDATE događaja *inserted* tabela sadrži novu torku, odnosno vrednost modifikovane torke. U slučaju UPDATE i DELETE događaja *deleted* tabela sadži staro stanje torke, odnosno vrednost torke koja se briše.

Sintaksa za kreiranje i brisanje trigera:

IF OBJECT\_ID([*database*\_*name*.][*schema*\_*name*.]*procedure\_name*, 'TR') IS NOT NULL

DROP TRIGGER [*database*\_*name*.][*schema*\_*name*.]*procedure\_name*;

GO

CREATE TRIGGER [*database*\_*name*.][*schema*\_*name*.]*procedure\_name*

ON [*database*\_*name*.][*schema*\_*name*.]*table\_name*

{AFTER | INSTEAD OF} DELETE, INSERT, UPDATE

AS

BEGIN

--if izraz u slučaju AFTER trigera

IF @@ROWCOUNT = 0 RETURN;

SET NOCOUNT ON;

...

END

GO

Performansa AFTER trigera može biti unapređena proverom vrednosti globalne promenljive @@ROWCOUNT, koja, ukoliko joj se pristupi u prvom redu BEGIN END bloku trigera sadrži broj torki koji je bio pod uticajem instrukcije koja je dovela do okidanja trigera. Ukoliko je ova vrednost nula, RETURN omogućava prekid izvršavanja trigera.

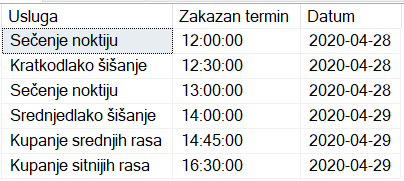
Za potrebe projekta kreirana su dva trigera, priložena u fajlu trigeri.sql.

# Upiti

Pregled izvršenih usluga zahteva od strane određenog zaposlenog u određenom periodu, određenim početnim i krajnjim datumom, u kom se posmatra poslovanje zaposlenog.

Rezultat upita predstavlja selekcija spoja relacija *Radni\_dan* i *Usluga*, gde je uslov selekcije dat u *where* klauzili. Za određivanje perioda u kom se posmatraju izvršene usluge korišćena je ugrađena funkcija *SQL Server-*a *datediff* kojoj se kao parametri prosleđuju datumi i eksplicitno navodi jedinica računanja vremena (*day, month, year, week, hour…*) u kojoj se izražava razlika datuma koju funkcija vraća.

Izvršavanje upita ispisuje torke sa informacijom o uslugama koje je pružio zaposleni sa jedinstvenim matičnim brojem koji je, pored perioda posmatranja poslovanja, naveden kao uslov selekcije u *where* klauzuli.

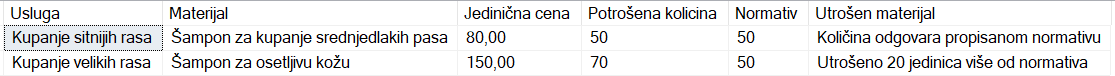


Slika 3 - Rezultat upita za pregled izvršenih usluga zaposlenog

Pregled materijala koji je upotrebljen pri izvršavanju usluga određenog zahteva uz informaciju da li je utrošena količina usklađena sa propisanim normativom korišćenog materijala za pruženu uslugu.

Rezultat upita predstavlja selekcija spoja relacija *Se\_koristi, Materijal, Usluga* i *Mat\_za,* gde je uslov selekcije vrednost jedinstvenog identifikatora zahteva. Da bi se predstavila informacija o odnosu utrošene količine materijala i količine propisane normativom, korišćena je ugrađena funkcija *SQL Server*-a *iif*, koja proverava vrednost logičkog izraza koji predstavlja prvi parametar i vraća drugi parametar ukoliko izraz daje *true* vrednost, u suprotnom, kao rezultat vraća treći prosleđen parametar. Takođe, kako bi se numerička vrednost izraza razlike utrošene količine materijala i normativa prosledila kao deo ispisa kolone *utrošen materijal,* korišćena je funkcija za eksplicitno konvertovanje tipa, *cast,* i funkcija za konkatanaciju stringova, *concat.*

Torke koje se dobijaju izvršavanjem upita daju pregled i precizno poređenje utroška materijala pri pružanju usluga određenog zahteva sa količinom koja je propisana kao normativ za utrošen materijal određene usluge koja je pružena.

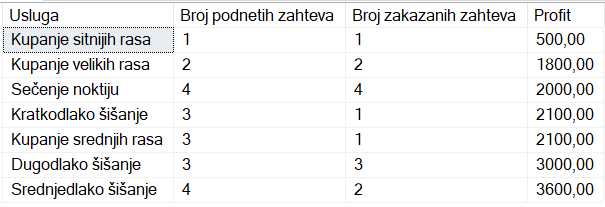


Slika 4 - Rezultat upita za pregled potrošenog materijala usluga zahethteva

Pregled podataka o broju podnetih i broju izvršenih zahteva za svaku od usluga uz informaciju o profitu te usluge uređen po rastućoj vrednosti profita.

Rezulat upita predstavlja spoj relacija *Zahteva* i *Usluga* nad čijim torkama je izvršeno grupisanje *group by* izrazom, koji postavlja kriterijum za grupisanje torki nad kojima se izvršava grupna funkcija. Grupna funkcija se navodi kao jedna od kolona torki koje se dobijaju upitom i prosleđuje joj se obeležje nad čijim vrednostima se grupna funkcija izvršava*.* Upit sa grupnom funkcijom vraća torke sa vrednostima obeležja po kojima je izvršeno grupisanje (ukoliko se ova obeležja navedu u ­*select* naredbi) i vrednost koja je nastala izvršavanjem funkcije nad vrednostima obeležja torki iste grupe.

Izvršavanje upita ispisuje torke gde svaka daje informaciju o ukupnom broju podnetih i izvršenih zahteva (vrednost obeležja *termin\_zak* nije *null*), dobijenih grupnom funkcijom *count*,i profitu usluge, gde je profit proizvod ukupnog broja zahteva za ovu uslugu koji su izvršenii jedinične cene usluge. Podaci su sortirani po rastućoj vrednosti profita izrazom *order by.*

**

Slika 5 - Rezultat upita za pregled podataka o zahtevima za svaku od usluga usluga

Pregled podataka o ukupnom broju podnetih zahteva u danu u nedelji, dana u kojima je ukupan broj zahteva veći od prosečnog broja ukupno podnetih zahteva po danima u nedelji.

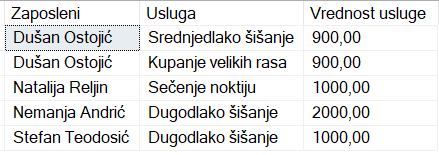
Rezultat upita predstavlja relacija *Zahteva* nad kojom je izvršeno grupisanje podnetih zahteva po danima u nedelji, kako bi se sračunao ukupan broj podnetih zahteva svakog dana korišćenjem grupne funkcije *count.* Da bi se dobio dan u nedelji koristi se ugrađena funkcija *datepart* koja za prosleđeni datum, ukoliko je prvi parametar funkcije postavljen na *dw*, vraća broj koji daje informaciju o danu u nedelji koji odgovara datumu, gde je 0 nedelja, 1 ponedeljak, 2 utorak i, sledstveno tome, 6 subota. Kako bi se u torkama predstavila jasna informacija o danu u nedelji kreirana je funkcija *danUNedelji* koja za prosleđenu povratnu vrednost funkcije *datepart* vraća string koji odgovara danu u nedelji posmatranog datuma.

Izrazom *having* izvršena je selekcija torki dobijenih grupisanjem, tako da su rezultat upita samo torke koje daju informaciju o danu u nedelji u kojem je veći ukupan broj podnetih zahteva od prosečnog ukupnog broja podnetih zahteva po danima u nedelji. Ova prosečna vrednost dobijena je upitom nad istom tabelom, uz isti kriterijum grupisanja, putem grupne funkcije *avg,* koja vraća vrednost koja je operand *having* izraza.

Pregled ukupnog profita pružene usluge od strane zaposlenog za svakog od zaposlenih. Ukoliko je zaposleni pružao više različitih usluga, prikaz usluge čiji je profit najveći kod zaposlenog.

Rezultat upita je grupisanje nad spojom relacija *Zahteva, Usluga* i *Zaposleni.* Grupisanjem se postiže da se proizvodom grupne funkcije *count,* koja u ovom upitu daje za zaposlenog ukupan broj zahteva određene usluge koju je zaposleni pružio, i jedinične cene pružene usluge, dobije profit koji je postignut pružanjem određene usluge od strane određenog zaposlenog.

Kako bi se u rezultatu upita prikazale za svakog zaposlenog informacije o profitu usluge kojom je ostvaren najveći profit, korišćen je upit u *from*-u, koji omogućava da se iz grupisanih podataka funkcijom *max* sračunaju usluge sa maksimalnim profitom za svakog od zaposlenog. Ovaj podatak navodi se u *having* izrazu, i na taj način izvršavanje upita ispisuje torke koje prikazuju za svakog zaposlenog uslugu čijim pružanjem je zaposleni postigao najveći profit posmatrajući sve usluge koje je zaposleni pružao.



Slika 6 - Rezultat upita za pregled profita pruženih usluga

# Funkcije

Za potrebe projekta napravljene su dve skalarne funkcije:

* *jmbg\_validation* i
* *vrednost\_zahteva*.

T-SQL kod funkcija priložen je u fajlu funkcije.sql.

## Funkcija *jmg\_validation*

Funkcija *jmbg\_validation* proverava da li je vrednost jedinstvenog matičnog broja zaposlenog validna. Pri pozivu joj se prosleđuje parametar tipa *varchar* koji predstavlja jmbg čiju ispravnost treba proveriti. Ukoliko funkcija vrati bitvrednosti nula, jmbg nije ispravan, u suprotnom, validacija je uspešna.

Prvi korak u proveri jmbg-a predstavlja deklarisanje vrednosti jmbg-a koje u zavisnosti od svoje pozicije u stringu koji predstavlja jmbg imaju svoje značenje i dozvoljene vrednosti. Izdvajanje delova jmbg-a urađeno je pomoću ugrađene funkcije SUBSTRING, koja za parametre prima string nad kojim se vrši operacija, zatim celi broj koji označava pozciju početnog indeksa od koga se uzima broj karaktera označen trećim parametarom. Izdvojeni delovi jmbg-a dodeljeni su varijablama deklarisanim putem izraza DECLARE i vrednosti ovih varijabli su prosleđene kao parametar TRY\_CAST funkcije, kako bi se string dobijen iz jmbg-a pretvorio u numerički tip podatka, nad kojim se dalje vrše operacije poređenja. Ova funkcija vratiće *null* vrednost ukoliko pretvaranje vrednosti nije moguće, i za svaki od delova jmbg-a koje je dalje potrebno posmatrati, neuspešno pretvaranje označava da jmbg nije ispravan i izvršavanje funkcije se prekida.

Nakon uspešnog preuzimanja vrednosti iz jmbg-a, u ostatku funkcije se proverava da li ove vrednosti odgovaraju dozvoljenim. Korišćena je ugrađena funkcija EOMONTH koja za prosleđeni datum vraća datum koji odgovara poslednjem danu meseca prosleđenog datuma. Pomoću funkcije DATEPART koja izdvaja deo iz prosleđenog datuma na osnovu prvog parametra koji označava deo datuma koji funkcija vraća, nakon provere prestupnosti godine, proverava se da li je dan u mesecu dobijen iz jmbg-a ispravan. Nakon provere ostalih vrednosti, računa se vrednost kontrolnog broja, koja se računa nad ciframa jmbg-a po utvrđenom pravilu. Ukoliko je vrednost ovog broja tačna, funkcija vraća bit vrednosti jedan.

U nastavku je dat kod kojim se ispituje povratna vrednost poziva funkcije. Prosleđen je jmbg koji iz neispravan različitih razloga, ispisanih na Slici 7, na šta upućuje vrednost funkcije.

select iif(dbo.jmbg\_validation('0708998735052')=1, concat('valid ','0708998735052'), concat('invalid ','0708998735052')) as 'Ispravan unos',

iif(dbo.jmbg\_validation('1301998735052')=1, concat('valid ','1301998735052'), concat('invalid ','1301998735052')) as 'Neispravan mesec',

iif(dbo.jmbg\_validation('0631998735052')=1, concat('valid ','0631998735052'), concat('invalid ','0631998735052')) as 'Neispravan dan u mesecu',

iif(dbo.jmbg\_validation('07089g8695052')=1, concat('valid ','07089g8695052'), concat('invalid ','07089g8695052')) as 'Neispravna godina',

iif(dbo.jmbg\_validation('0708998695052')=1, concat('valid ','0708998695052'), concat('invalid ','0708998695052')) as 'Neispravne RR cifre',

iif(dbo.jmbg\_validation('0708998735053')=1, concat('valid ','0708998735053'),

concat('invalid ','0708998735053')) as 'Neispravan kontrolan broj'



Slika 7 - Prikaz rezultata poziva funkcije *jmbg\_validation*

## Funkcija *vrednost\_zahteva*

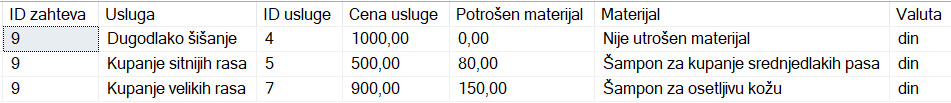
Funkcija *vrednost\_zahteva* računa ukupnu cenu izvršenog zahteva, koja se računa kao suma cena pruženih usluga i jediničnih cena materijala koji su utrošeni za pružanje ovih usluga.

Funkciji se za parametar prosleđuje *int* vrednost koja označava jedinstveni identifikator zahteva čija cena, tipa *smallmoney*,predstavlja povratnu vrednost funkcije. Informacije o pruženim uslugama nalaze se u tabeli *Zahteva*, koju treba spojiti sa tabelom *Usluga* kako bi se dobio podatak o ceni pružene usluge. Kako se jednim zahtevom može zahtevati više usluga, selekcija spoja pomenutih tabela koja za kriterijum selekcije uzima šifru zahteva može vratiti više torki, pa je, da bi se pristupilo vrednostima cene usluge iz ovih torki, potrebno koristiti kursor.

Kursor je objekat koji omogućava prolazak kroz torke iz skupa koji se dobija rezultatom upita koji se navodi prilikom njegovog kreiranja. Nakon otvaranja kursora, torkama se pristupa naredbom FETCH NEXT FROM koja se poziva u WHILE petlji gde je uslov za izlazak iz petjle da je vrednost globalne promenljive @@FETCH\_STATUS različita od nule, gde nula predstavlja informaciju o uspešnom preuzimanju vrednosti torke kursora. Nakon izlaska iz petlje, kursor se zatvara i uklanja referenca na njega naredbama CLOSE i DEALLOCATE.

Kako bi se dobila cena materijala koji je utrošen za pružanje usluga, kreiran je još jedan kursor koji prolazi kroz torke spoja tabela *Se\_koristi* i *Materijal,* prolazeći kroz WHILE petlju sa identičnim uslovom izlaska iz petlje.

Vrednosti cena usluga i jediničnih cena utrošenih materijala, kojima se pristupa putem kreiranih kursora, dodaju se na vrednost promenljive koja čuva vrednost ukupne cene zahteva. U nastavku je dat poziv ove funkcije i kod kojim se proveravaju usluge zahteva čija je ukupna cena zahteva računata.



Slika 8 - Informacije o zahtevu sa ID-jem 9

select dbo.vrednost\_zahteva(9) as 'Ukupna cena zahteva'



Slika 9 - Rezultat funkcije vrednost\_zahteva za zahtev sa ID-jem 9



Slika 10 - Informacije o zahtevu sa ID-jem 3

select dbo.vrednost\_zahteva(3) as 'Ukupna cena zahteva'



Slika 11 - Rezultat funkcije *vrednost\_zahteva* za zahtev sa ID-jem 3

U slučaju zahteva sa neispravnom vrednošću šifre zahteva koja se prosleđuje, funkcija će vratiti vrednost 0.

select dbo.vrednost\_zahteva(-13) as 'Ukupna cena zahteva'



Slika 12 - Rezultat funkcije *vrednost\_zahteva* za neispravan ID zahteva

# Procedure

Za potrebe projekta kreirane su dve uskladištene procedure:

* *promeni\_cenu* i
* *dodaj\_radnike.*

Kod procedura priložen je u fajlu procedure.sql.

## Procedura *promeni­\_cenu*

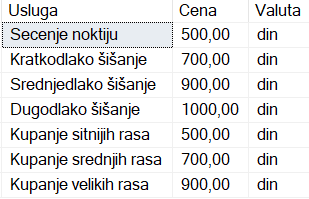
Procedura *promeni\_cenu* snižava ili povišava cenu usluga za određeni procenat. U slučajevima kada na nivou preduzeća bude doneta odluka o poskupljenju ili pojeftinjenju usluga, procedura olakšava postupak sprovođenja ovakve odluke, jer nije potrebno pojedinačno modifikovati cenu svake od usluga.

Procedura dodatno ograničava set usluga koje će biti modifikovane parametrima *cena\_od* i *cena\_do*, koji označavaju granice intervala vrednosti cena usluga koje mogu biti modifikovane. Proceduri se pored ovih *smallmoney* vrednosti, prosleđuje bit čijom interpretacijom vrednosti se dobija informacija da li je potrebno sniziti ili povisiti cenu usluga. Poslednji parametar procedure je procenat za koji je potrebno sniziti ili povisiti cenu i mora biti pozitivan broj manji od broja sto.

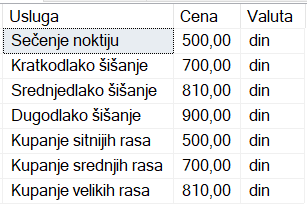
Ukoliko vrednosti parametara cene ne predstavljaju pozitivne vrednosti ili je vrednost *cena\_od* veća od vrednosti *cena\_do*, završava se izvršavanje procedure. Nakon provere ispravnosti vrednosti prosleđenih parametara, proverava se vrednost parametara koji daje informaciju o tome da li treba povisiti ili sniziti cenu. Putem kursora kreiranog nad tabelom *Usluga* pristupa se vrednostima torki usluga čija cena se nalazi unutar intervala određenog ulaznim parametrima procedure. Nakon preuzimanja vrednosti obeležja primarnog ključa torke, u WHILE petlji se za svaku od torki koja je prosleđena kursoru modifikuje vrednost cene za naznačen procenat UPDATE naredbom.

Poziv procedure kojim se za 10% snižava cena usluga čija je cena između 800 i 1000:

execute dbo.promeni\_cenu 800, 1000, 0, 10;



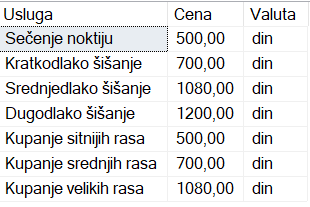
Slika 13 - Cene uluga pre poziva procedure *promeni\_cenu*



Slika 14 - Cene usluga nakon poziva procedure *promeni\_cenu* kojim se snižava cena

Poziv procedure kojim se za 20% povišava cena usluga čija je cena između 800 i 1000:

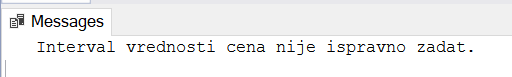
execute dbo.promeni\_cenu 800, 1000, 1, 20;



Slika 15 - Cene usluga nakon poziva procedure *promeni\_cenu* kojim se povišava cena

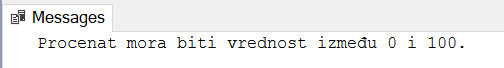
Poziv procedure sa neispravno prosleđenim parametrima vratiće poruku koja obaveštava o grešci i procedura neće biti izvršena.

execute dbo.promeni\_cenu 1000, 800, 1, 20;



Slika 16 - Poruka procedure *promeni\_cenu* kada su parametri koji označavaju granice intervala vrednosti cena neispravni

execute dbo.promeni\_cenu 800, 1000, 1, 120;



Slika 17 - Poruka procedure *promeni cenu* kada parametar koji označava procenat ima nedozvoljenu vrednost

## Procedura *dodaj\_radnike*

Procedura *dodaj\_radnike* tabelu *Radi* popunjava informacijama o statusu aktivnih radnika za određeni dan poslovanja preduzeća. Pošto su česti slučajevi kada se statusi aktivnih radnika poklapaju za dva dana poslovanja, ova procedura omogućava da se za određeni radni dan poslovanja unesu radnici sa statusima koji su za te iste radnike važili nekog prethodnog dana.

Procedura za parametre prima dva datuma. Prvi datum, koji se čuva u promenljivoj *stari\_datum,* označava datum dana poslovanja čije zapise iz tabele *Radi* (radnike i njihove statuse) treba uneti u istu tabelu, ali za dan poslovanja određen drugim datumom, *novi\_datum*. Procedura dozvoljava samo prepisivanje od starijih datuma, odnosno prvi parametar procedure mora biti stariji datum od drugog datuma. U slučaju da *novi\_datum* nije upisan u radni kalendar poslovanja preduzeća u tabeli *Radni\_dan*, ovaj datum se unosi i postavlja za radan dan preduzeća. Pre samog unošenja zapisa statusa radnika, brišu se prethodno uneti zapisi tabele *Radi* gde je vrednost datuma jednaka vrednosti *novi\_datum,* uz pomoć kreiranog kursora nad ovom tabelom (provera nije potrebna ukoliko datum nije postojao u tabeli *Radni\_dan*). Brisanje se vrši samo ukoliko u tabeli *Zahteva* zahtevima zakazanim za dan datuma *novi\_datum* nisu dodeljeni radnici koji te zahteve treba da izvrše, jer u ovom slučaju brisanje ovih radnika sa njihovim statusima iz tabele *Radi* ne bi bilo moguće.

Kreira se kursor nad tabelom *Radi* selektovanjem torki unetih za vrednost datuma *stari\_datum,* i preuzimaju vrednosti jedinstvenog matičnog broja zaposlenog i statusa tog zaposlenog. Ove vrednosti unose se u istu tabelu za vrednosti zaposlenih i njihovih statusa za dan određen datumom *novi\_datum.* Na ovaj način aktivni radnici sa statusima za radni dan datuma *stari\_datum* uneti su kao aktivni radnici sa istim statusima za radni dan datuma *novi\_datum.*

Poziv procedure kojim se radnici i njihovi statusi dana 30.04.2020. prepisuju u tabelu *Radi* za dan 03.05.2020, gde nisu prethodno postojali zapisi za ovaj dan:

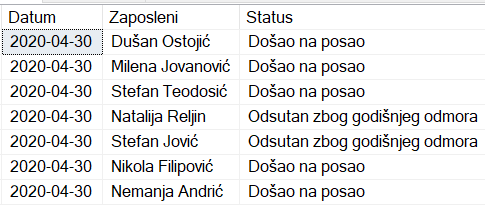
execute dbo.dodaj\_radnike '04-30-2020', '05-03-2020';



Slika 18 – Zapisi tabele *Radi* za datum 30.04.2020. i 05.03.2020. nakon poziva procedure *dodaj\_radnike*

Poziv procedure kojim se radnici i njihovi statusi dana 29.04.2020. prepisuju u tabelu *Radi* za dan 30.04.2020, gde su prethodno postojali zapisi za dan 30.04.2020., ali zaposleni nisu dodeljeni zahtevima za ovaj dan u tabeli *Zahteva*:

execute dbo.dodaj\_radnike '04-29-2020', '04-30-2020';



Slika 19 - Zapisima tabele *Radi* za datum 30.04.2020. pre poziva procedure *dodaj\_radnike*



Slika 20 - Zapisi tabele *Radi* za datum 29.04.2020. i 30.04.2020. nakon poziva procedure *dodaj\_radnike*

Poziv procedure kojim se radnici i njihovi statusi dana 29.04.2020. prepisuju u tabelu *Radi* za dan 29.04.2021, gde u radni kalendar nije prethodno unet datum 29.04.2021:

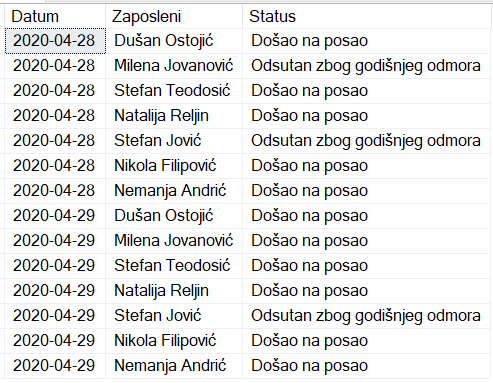
execute dbo.dodaj\_radnike '04-29-2020', '04-29-2021';



Slika 21 - Zapisi tabele *Radi* za datum 29.04.2020. i 29.04.2021. nakon poziva procedure *dodaj\_radnike*

U nastavku je dat poziv procedure kojim se radnici i njihovi statusi dana 28.04.2020. prepisuju u tabelu *Radi* za dan 29.04.2020. Pošto su prethodno postojali zapisi za dan 29.04.2020. i zaposleni sa svojim statusima iz tabele *Radi* su dodeljeni zahtevima za ovaj dan u tabeli *Zahteva*, procedura neće biti izvršena do kraja.

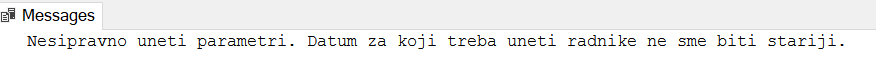
execute dbo.dodaj\_radnike '04-28-2020', '04-29-2020';



Slika 22 - Zapisi tabele *Radi* za datume 28.04.2020. i 29.04.2020. nakon poziva procedure *dodaj\_radnike*

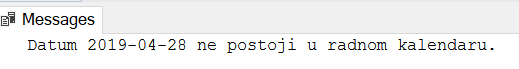
Ukoliko se proceduri kao vrednost *stari\_datum* prosledi datum koji ne postoji u radnom kalendaru, ili datum koji nije stariji od datuma *novi\_datum*, završava se izvršavanje procedure i prosleđuje poruka o grešci.

execute dbo.dodaj\_radnike '04-29-2020', '04-28-2020';



Slika 23 - Poruka procedure *dodaj\_radnike* kada su prosleđeni parametri neispravni

execute dbo.dodaj\_radnike '04-28-2019', '04-28-2020';



Slika 24 - Poruka procedure *dodaj\_radnike* kada prosleđeni datum ne postoji u kalendaru

# Trigeri

Ograničenja podšeme baze podataka koja nije moguće implementirati deklarativnim mehanizmima implementirana su proceduralnim, kreiranjem trigera. U ova ograničenja spadaju međurelaciona ograničenja i pravila poslovanja koja ne rezultuju u ograničenjima šeme baze podataka. Ograničenja koja proizilaze iz pravila poslovanja tiču se unapred definisanog redosleda i uslovljenosti izvođenja operacija nad bazom podataka, pod određenim uslovima definisanim putem odnosa vrednsoti obeležja.

Za potrebe projekta kreirana su dva AFTER trigera:

* trigernad tabelom *Radni\_dan* i
* trigernad tabelom *Zahteva*.

*SQL Server* ne podržava specifiranje frekvencije aktiviranja, već se triger aktivira jednom za celu operaciju koja ga je izazvala nezavisno od broja torki koje su predmet operacije koja je izazvala izvršavanje trigera. Zbog ovoga, potrebno je proceduralno obezbediti da aktivnost trigera bude primenjena za svaku od torki koja je bila predmet operacije koja je okinula triger. Da bi se ovo postiglo, u trigerima se koristi kursor koji omogućava prolazak kroz ovaj set torki. Vrednosti obeležja ovih torki nalaze se u torkama privremenih tabela *inserted* i *deleted* kojima se može pristupiti u trigeru. U nastavku je dat opis kreiranih trigera pri čemu se podrazumeva da je upotrebom kursora obezbeđeno da se aktivnost trigera primenjuje za svaku torku koja je bila predmet okidajuće operacije.

## AFTER triger nad tabelom *Radni\_dan*

Dani poslovanja preduzuća čija se evidencija vodi u tabeli *Radni\_dan* mogu biti radni dani ili neradni. Potrebno je obezbediti da svaki radni dan preduzeća ima barem jednu dodeljenu šemu radnog dana po kojoj se posluje u tom danu. Da bi se ovo postiglo, nad tabelom *Radni\_dan* kreiran je AFTER triger koji nakon izvšene INSERT ili UPDATE operacije proverava da li je dan radan. Ukoliko je dan radan, potrebno je u tabeli *Ima\_schemu* za odgovarajući radni dan naznačiti šemu po kojoj se u tom danu posluje. Da bi ograničenje bilo u potpunosti implemenirano potrebno je uzeti u obzir sve moguće načine njegovog narušavanja, pa je potrebno prilikom modifikacije dodatno proveriti određene uslove.

U slučaju pokušaja unošenja nove torke u tabelu, nakon izvršavanja INSERT operacije, vrednosti nove torke koja je uneta se dobijaju iz *inserted* tabele. Ukoliko je vrednost obeležja *radni\_dan\_s* jedinica, što ukazuje da je dan poslovanja radan, u tabeli *Ima\_schemu* se radnom danu dodeljuje izabrana podrazumevana šema.

Modifikacija torke tabele *Radni\_dan* koja podrazumeva promenu vrednosti obeležja *radni\_dan\_s* kojom se naznačava radni dan može biti poništena ukoliko se vrši promena vrednosti ovog obeležja sa jedinice na na nulu. U svim ostalim slučajevima, modifikacija se dozvoljava bez dodatne provere uslova, samo se u slučaju modifikacije neradnog kojom on postaje radan u tabelu *Ima\_schemu* unosi zapis za ovaj dan.

Ukoliko se za dan poslovanja, koji je prethodno bio označen kao radan, pokušava sprovesti modifikacija tako da postaje neradan, potrebno je, ukoliko ti zapisi postoje, obrisati zapise iz tabele *Radi* za ovaj dan, budući da se u ovoj tabeli ne mogu naći informacije za neradan dan. Međutim, brisanje ovih torki tabele *Radi* nije moguće ukoliko su torke referencirane u tabeli *Zahteva*. Ukoliko su vrednosti iz tabele *Radi* za određeni dan referencirane u tabeli *Zahteva*, to znači da su tog dana zakazane i pružene usluge određenih zahteva, pa ne treba dozvoliti da se takav dan modifikacijom označi kao neradan. Ukoliko za radni dan koji pokušava da se modifikuje tako da postaje neradni postoje torke u tabeli *Zahteva*, izvršavanje trigera se završava slanjem poruke koja ukazuje da modifikacija nije dozvoljena i modifikacija se poništava. Ukoliko torke u tabeli *Zahteva* ne postoje, brišu se zapisi iz tabele *Radni\_dan* uneti za dan koji se modifikuje, kao i zapisi iz tabele *Ima\_schemu* i zadržavaju modifikovane vrednosti torke.

Pokušaj modifikacije radnog dana poslovanja promenom vrednosti obležja *radni\_dan\_s* na nulu, kada za radni dan koji treba modifikovati postoje zapisi u tabeli *Zahteva* (Slika 31):

update Production.Radni\_dan

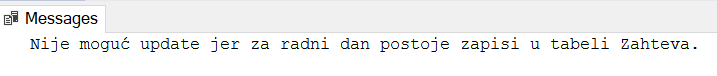
set radni\_dan\_s = 0

where rd\_id = 5;



Slika 31 - Zapisi tabele *Zahteva* za radni dan sa ID-jem 5

Poruka koja se dobija pri pokušaju modifikacije radnog dana poslovanja promenom vrednosti obležja *radni\_dan\_s* na nulu, kada za radni dan koji treba modifikovati postoje zapisi u tabeli *Zahteva*, data na je na Slici 32.



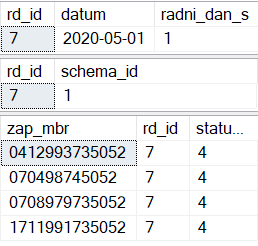
Slika 32 - Poruka trigera kada nije dozvoljena modifikacija torke tabela *Radni\_dan*

U nastavku je dat pokušaj modifikacije radnog dana poslovanja promenom vrednosti obležja *radni\_dan\_s* na nulu, kada za radni dan koji treba modifikovati ne postoje zapisi u tabeli *Zahteva*, ali postoje u tabeli *Radi* i *Ima\_schemu* (Slika 33). U ovom slučaju, brišu se podaci iz tabela *Radi* i *Ima\_schemu*(Slika 34).

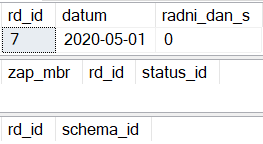
update Production.Radni\_dan

set radni\_dan\_s = 0

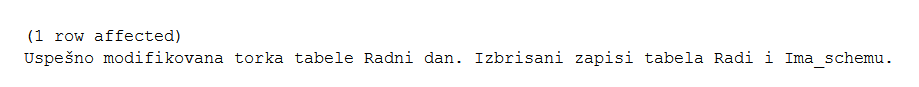
where rd\_id = 7;



Slika 33 - Zapisi tabela *Radi* i *Ima­\_schemu* pre okidanja trigera

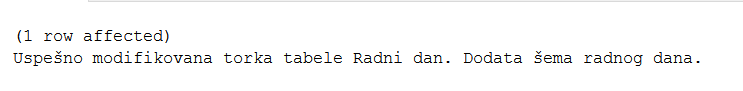


Slika 34 – Zapisi tabela *Radi* i *Ima­\_schemu* nakon okidanja trigera



Slika 35 - Poruka trigera kada je modifikacija torke tabele *Radni\_dan* dozvoljena i izbrisani zapisi tabela *Radi* i *Ima\_schemu*

Ukoliko se modifikuje neradan dan poslovanja promenom vrednosti obeležja *radni\_dan\_s* na jedinicu, u tabelu *Ima\_schemu* se dodaje zapis za ovaj dan (Slika 36).

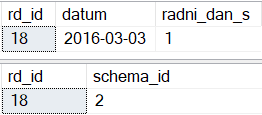


Slika 36 - Poruka trigera kada je modifikacija torke tabele *Radni\_dan* dozvoljena i dodata šema

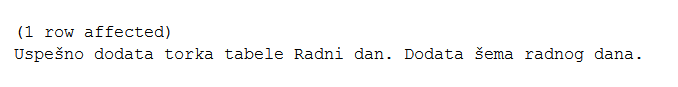
Nakon unošenja dana poslovanja koji je radan u tabelu *Ima\_schemu* se unosi zapis kojim se radnom danu dodeljuje šema (Slika 37).

insert into Production.Radni\_dan (datum, radni\_dan\_s)

values('03-03-2016',1);



Slika 37 - Zapisi tabela *Radni\_dan* i *Ima­­\_schemu* nakon okidanja trigera



Slika 38 - Poruka trigera kada je uspešno dodata torka tabele *Radni\_dan*

## AFTER triger nad tabelom *Zahteva*

AFTER trigerom nad tabelom *Zahteva* implementirana su ograničenja koja se odnose na aktivnosti zakazivanja termina izvršavanja usluga podnetih zahteva. Uslovi koji moraju biti zaodovoljeni prilikom zakazivanja su sledeći:

1. Izvršavanje usluge se zakazuje za datum koji je u radnom kalendaru označen kao radni dan (vrednost *radni\_dan\_s* obeležja tabele *Radni\_dan* je jedinica). Ukoliko izvršavanje usluge nije zakazano za određeni radni dan, ne može biti zakazan ni termin.
2. Datum radnog dana za koji se zakazuje izvršavanje usluge odgovara datumu za koji je usluga zahtevana.
3. Termin za koji se izvršavanje usluge zakazuje se uklapa u neku od šema radnog dana koje su dodeljene radnom danu za koji se usluga zakazuje (*termin\_zak* pripada intervalu čije su granice određene vrednostima obeležja *vreme\_od* i *vreme\_do* tabele *Schema \_dan*).
4. Zaposleni koji se postavlja za izvršioca usluge zakazane za određeni radni dan tog radnog dana ima status koji ukazuje na to da može da pruži uslugu (ograničenje stranog ključa nad ovom tabelom ne dozvoljava da se za izvršavanje usluge postavi zaposleni ukoliko usluga nije zakazana za određeni radni dan).

Ukoliko vrednosti torke koju treba uneti u tabelu *Zahteva*, odnosno modifikovane vrednosti torke u slučaju modifikacije, zadovoljavaju potrebne uslove, operacija unosa, odnosno modifikacije, se ne poništava. U slučaju da neki od uslova nije zadovoljen, RAISEROR vraća informaciju o grešci i izvršena operacija se poništava. U slučaju operacije dodavanja, operacija se poništava brisanjem torke, a u slučaju modifikacije vraćanjem vrednosti obeležja torke na stare.

U nastavku su date poruke o greškama, uslovi pod kojima su one izazvane i kod kojim se triger okida prilikom pokušaja modifikacije. Iste poruke dobijaju se i u slučaju unosa novih torki sa vrednostima koje iz istih razloga ne zadovoljavaju iste uslove koji dovode do grešaka pri modifikaciji.

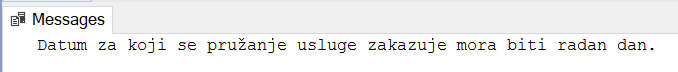
update Production.Zahteva

set rd\_id = 1

where zahtev\_id = 12 and usluga\_id = 3;



Slika 39 - Zapis tabele *Radni\_dan* za neradni dan sa ID-jem 1

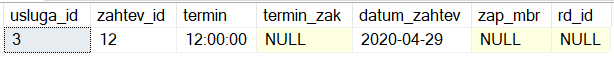


Slika 40 - Poruka trigera prilikom pokušaja zakazivanja pružanja zahtevane uluge za neradni dana

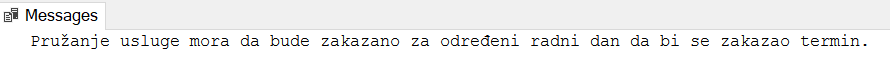
update Production.Zahteva

set termin\_zak = '12:00'

where zahtev\_id = 12 and usluga\_id = 3;



Slika 41 - Zapis tabele *Zahteva* za uslugu zahteva koja nije zakazana za radan dan

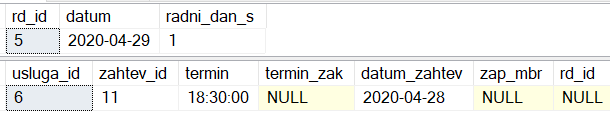


Slika 42 – Poruka trigera prilikom pokušaja zakazivanja termina pružanja zahtevane usluge kada nije zakazan dan u kom treba pružiti uslugu

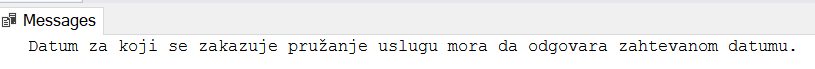
update Production.Zahteva

set rd\_id = 5

where zahtev\_id = 11 and usluga\_id = 6;



Slika 43 – Zapisi radnog dana tabele *Radni\_dan* i usluge iz tabele *Zahteva* zahtevane za drugi datum



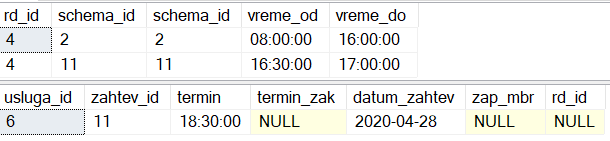
Slika 44 – Poruka trigera prilikom pokušaja zakazivanja pružanja zahtevane usluge za radni dan koji ne odgovara zahtevanom datumu

update Production.Zahteva

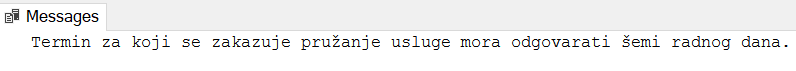
set rd\_id = 4,

termin\_zak = '18:30'

where zahtev\_id = 11 and usluga\_id = 6;



Slika 45 – Zapisi šema radnog dana tabele *Ima\_schemu* i zahtevane usluge za termin koji ne odgovara šemama tabele *Zahteva*



Slika 46 - Poruka trigera prilikom pokušaja zakazivanja termina izvršavanja usluge koji ne odgovara šemi radnog dana

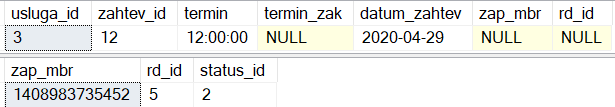
update Production.Zahteva

set rd\_id = 5,

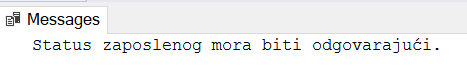
termin\_zak = '12:00',

zap\_mbr = '1408983735452'

where zahtev\_id = 12 and usluga\_id = 3;



Slika 47 - Zapisi zahtevane usluge tabele *Zahteva* i zaposlenog sa neodgovarajućim statusom tabele *Radi*



Slika 48 - Poruka trigera prilikom pokušaja angažovanja zaposlenog sa neispravnim statusom za izvršavanje usluge

Rezultat ispravnog zakazivanja pružanja usluge zahteva za određeni radni dan koji odgovara zahtevanom datumu, za termin koji odgovara nekoj od šema radnog dana i za čije je izvršavanje dodeljen zaposleni sa odgovarajućim statusom dat je na Slici 49.

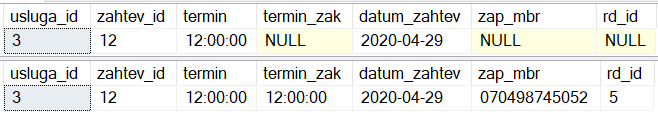
update Production.Zahteva

set rd\_id = 5,

termin\_zak = '12:00',

zap\_mbr = '070498745052'

where zahtev\_id = 12 and usluga\_id = 3;



Slika 49 – Zapis torke *Zahteva* pre i posle ispravnog zakazivanja pružanja usluge

# Zaključak

Na osnovu implementacione relacione šeme baze podataka, implementirana je relaciona baza podataka, koristeći komponente i mehanizme sistema za upravljanje relacionim bazama podataka, *Microsoft SQL Server-*a, koji ovo omogućavaju.

Prilikom implementacije, potrebno je obezbediti određen nivo performantnosti rada sistema za upravljanje bazom podataka, što podrazumeva obezbeđenje zadovoljavajućeg vremena odziva sistema na unapred predviđene funkcionalne zahteve. Funkcionalni zahtevi definisani su u fazi projektovanja šeme baze podaka i oni su od značaja i u fazi implementacije, jer se razmatranjem frekvencije potraživanja određenih podataka ili prioritetnih operacija nad podacima, donose odluke prilikom implementacije. Performantan rad sistema uslovljen je, između ostalog, karakteristikama samog sistema, kao i karakteristikama instalirane računarsko-komunikacione infrastrukture i instaliranih operativnih sistema. Karakteristike sistema koje treba imati u vidu radi obezbeđenja performantnosti predstavljaju ograničenja u pogledu implementacionih mehanizama i načina fizičkog organizovanja podataka koje je pod kontrolom sistema za upravljanje.

Jedan od parametara performatnog rada sistema za upravljanje bazom podataka predstavlja efikasnost korišćenja memorijskog prostora koji je dodeljen bazi podataka i koji je usko povezan sa efikasnom obradom podataka. Efikasna obrada podataka u velikoj meri zavisi od načina fizičkog organizovanja podataka i memorijskog prostora koji zauzimaju podaci. Sistem za upravljanje bazom podataka poseduje mehanizme kojima omogućava praćenje i podešavanje parametara fizičke organizacije datoteka i potrebno je poznavati način fizičke organizacije koji se implicitno postiže prilikom kreiranja tabela u kojima se podaci skladište. Takođe, potrebno je na najbolji način predstaviti podatak iz realnog sistema i razmatrati domen unapred definisanih tipova podataka i operacije koje nad tipom mogu da budu izvršene, kako bi zazuzeće memorijskog prostora bilo optimalno, ali ujedno i omogućeno ispravno manipulisanje podacima i njihovo skladištenje u potrebnom obimu.

Da bi implementacija bila sprovedena na način koji obezbeđuje tačnost i pouzdanost podataka, potrebno je poznavati prirodu deklarativnih i proceduralnih mehanizmama za implementaciju ograničenja šeme baze podataka i poslovnih pravila. Takođe, sistemi za upravljanje bazom podataka poseduju mehanizme za postizanje određenog nivoa nezavisnosti programa koji koriste bazu podataka od samih podataka, pa se tako logička nezavisnost programa postiže kreiranjem procedura i funkcija koje programi koriste.

Na efikasnost obrade podataka i realizaciju upita nad njima značajno utiče korišćenje pomoćnih struktura podataka i mehanizama optimizacije obrade upita koji veliki broj sistema poseduje. Kako bi ovi mehanizmi bili korisni, potrebno je poznavati uslove u kojima je njihovo korišćenje korisno i uzeti u obzir definisane funkcionalne zahteve kao i prioritet i učestalost izvršavanja operacija nad određenim podacima, u pogledu ažuriranja i izražavanja upita nad podacima.

Faza implementacije baze podataka se u velikoj meri oslanja na fazu projektovanja i analizom i razmatranjem utvrđenih funkcionalnih zahteva implementacija je sprovedena sa ciljem postizanja efikasnog korišćenja baze podataka. Radi postizanja najboljeg implementacionog rešenja, uzimane su u obzir prednosti i ograničenja sistema za upravljanje bazom podataka čijim mehanizmima je baza implementirana. Efikasno korišćenje baze podataka preduslov je za postizanje ciljeva koje treba postići implementacijom informacionog sistema čiji je baza podataka deo. Na ovaj način, mehanizmima sistema za upravljanje bazom podataka, implementirana je baza podataka kojom se ostvaruje potreban nivo pouzdanosti i performantnost rada nad podacima, odnosno olakšavanje poslovanja i vođenje evidencije o poslovanju preduzeća koje se bavi pružanjem usluga salona.