Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Departman za industrijsko inženjerstvo i menadžment
Inženjerstvo informacionih sistema

Informacioni sistem za praćenje poslovanja preduzeća za pružanje vulkanizerskih usluga

- SISTEMI BAZA PODATAKA -

Isidora Japundža IT 34/2020

Novi Sad, jun 2023.

Sadržaj

1. Uvod

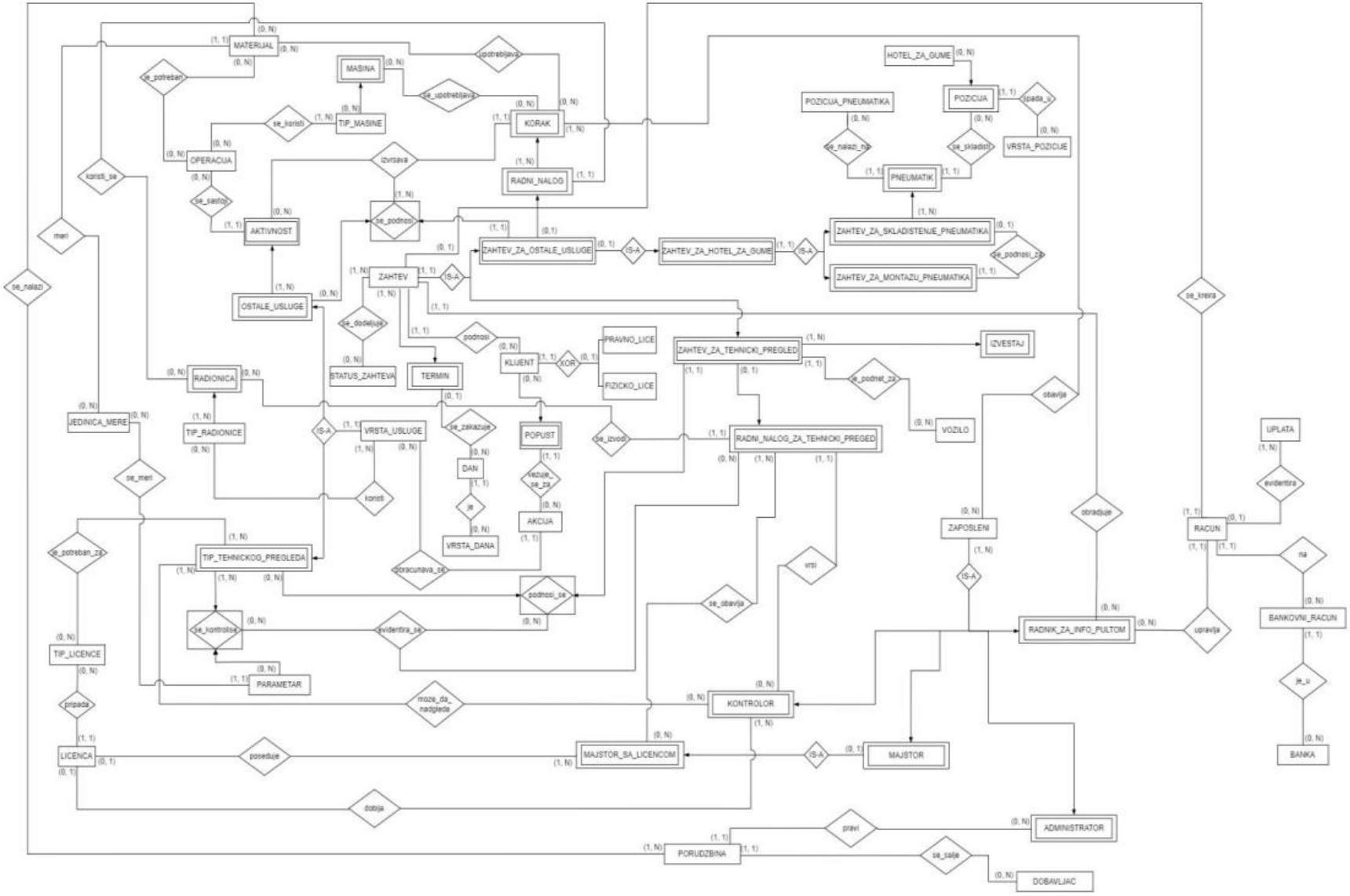
Projekat u okviru ove dokumentacije podrazumeva razvoj baze podataka za potrebe poslovanja jednog dela vulkanizerske radnje. Deo na koji je baza podataka fokusirana jeste slanje klijentskih zahteva za uslugu. Baza podataka korisnicima pruža različite manipulacije i operacije nad podacima i interakcije korisnika preko tog sadržaja.

U daljem tekstu biće predstavljen ER model podataka, relacioni model podataka, tabelarni prikaz ograničenja pojave tipa, integriteta tipa i jedinstvenih vrednosti. Prevođenjem ER modela podataka kreiran je relacioni model podataka, a na osnovu njega, izvršena je implementacija šeme baze podataka.

2. Analiza programskog domena

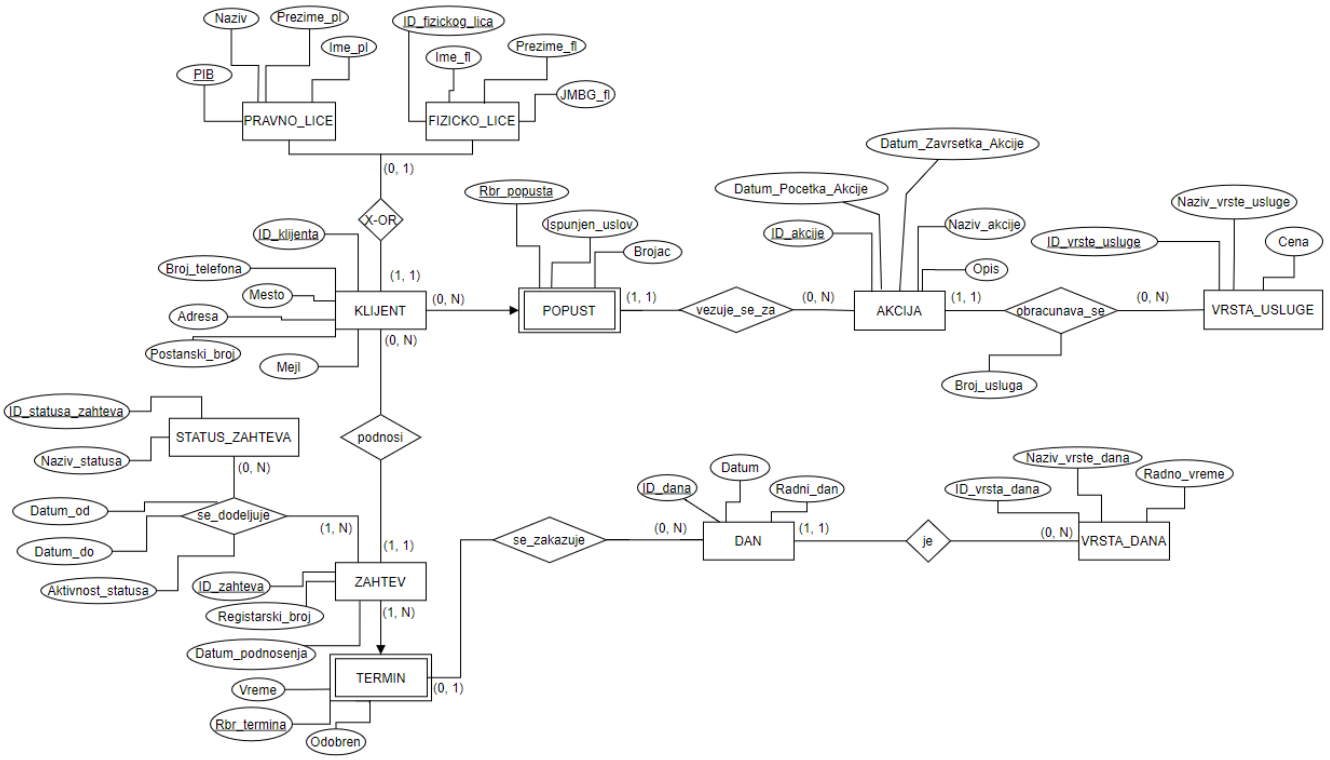
Cilj projektnog zadatka jeste unapređenje poslovanja vulkanizerske radnje što dalje omogućava ispunjenje svih funkcionalnosti obuhvaćenih u specifikaciji zahteva. Takođe, predstavlja oslonac za upravljanje i samo organizovanje jednog složenog sistema kao što je vulkanizerska radnja sa svim svojim uslugama te samim tim olakšava kontrolu i upravljanje njime. Osnovni cilj projekta čini skraćivanje vremena i povećanje kvaliteta u obavljanju svih aktivnosti koje su vezane za realni sistem. Detaljnom analizom svih činilaca realnog sistema kao i pažljivim prikupljanjem informacija potrebno je omogućiti kreiranje baze podataka koja olakšava poslovanje svim korisnicima sistema. Ova baza podataka omogućava beleženje i organizaciju podataka o klijentima, zahtevima, popustima, akcijama, vrsti usluga, terminima i radnom kalendaru. Konačno, projekat predstavlja lako prilagodljivu i pre svega fleksibilnu strukturu poslovanja vulkanizerske radnje.

3. ER model čitave šeme



Slika 1. ER model čitave šeme

4. ER model podšeme



Slika 2. ER model podšeme

Opis podšeme:

Podšema koja je prikazana na slici 2 prikazuje proces podnošenja zahteva za uslugu. Klijent može, ali i ne mora da podnese zahteve za uslugu. Za njega se beleže podaci kao što su adresa i mesto, poštanski broj, kao i broj telefona. U slučaju da je klijent fizičko lice za njega se beleže osnovni podaci kao što su ime i prezime, dok se JMBG beleži u izuzetnim slučajevima. Sa druge strane, za pravno lice potrebno je obezbediti podatak naziva pravnog lica. Neophodno je odrediti i osobu za kontakt za koju se takođe beleže ime i prezime.

Klijent može da ostvari određene popuste koji su vezani za akciju važeću u tom momentu. Akcija se beleži za tačno jednu vrstu usluge. Karakteriše je identifikacioni broj, naziv, opis, kao i datum početka odnosno završetka akcije. Akcija se obračunava za tačno jednu vrstu usluge.

Za svaki zahtev beleži se datum podnošenja, kao i registarski broj ukoliko je to potrebno. Svaki zahtev sadrži i određeni broj predviđenih termina. U okviru termina postoji podatak o njegovom vremenu, kao i o potvrdi odobrenosti istog. Svaki termin se može zakazati za jedan dan ukoliko je to u skladu sa radnim kalendarom. Jedan zahtev prolazi kroz više statusa zahteva, ali u određenom trenutku može biti samo u jednom.

Prati se istorija statusa zahteva. Istorija statusa obuhvata datum kada je zahtev dobio određeni naziv statusa dok datum završetka ne mora biti naznačen. Ukoliko postoji prethodni aktivan status za

zadati zahtev prvobitno je potrebno njega zaključiti. Potom se može dodeliti novi status za koji takođe nije potrebno da se zna datum završetka.

U projektu će biti obrađena podšema *Dostavljanje klijentskog zahteva za uslugu* (u daljem tekstu *Projekat_IT34_2020*) u kojoj učestvuju sledeći:

1. Tipovi entiteta:
    - klijent,
    - pravno_lice,
    - fizičko_lice,
    - popust,
    - akcija,
    - vrsta_usluge,
    - zahtev,
    - status_zahteva
    - termin,
    - dan i
    - vrsta_dana.

2. Tipovi poveznika:
    - vezuje_se_za,
    - obračunava_se,
    - podnosi,
    - se_zakazuje,
    - se_dodeljuje i
    - je

5. Tabelarni prikaz obeležja i ograničenja

U tabelama 1–17 tabelarno su prikazana ograničenja jedinstvene vrednosti obeležja.

Tabela 1. Tip entiteta *PRAVNO_LICE*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
PIB	Poreski identifikacioni broj	Integer	9	⊥	length(d) = 9
Naziv	Naziv pravnog lica	Varchar	20	⊥	Δ
Ime_pl	Ime kontakt osobe	Varchar	12	⊥	Δ
Prezime_pl	Prezime kontakt osobe	Varchar	20	⊥	Δ
Ključ	K = {*PIB*}				

Tabela 2. Tip entiteta *FIZICKO_LICE*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_fizickog_lica	Jedinstveni identifikacioni broj fizičkog lica	Integer	8	⊥	d > 0
Ime_fl	Ime fizičkog lica	Varchar	12	⊥	Δ
Prezime_fl	Prezime fizičkog lica	Varchar	20	⊥	Δ
JMBG_fl	JMBG fizičkog lica	Integer	13	T	length(d) = 13
Ključ	K = {*ID_fizickog_lica*}				
Unique(*FIZICKO_LICE*, *JMBG_fl*) = T					

Tabela 3. Tip entiteta *KLIJENT*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_klijenta	Jedinstveni identifikacioni broj klijenta	Integer	8	⊥	d > 0
Broj_telefona	Broj telefona klijenta	Varchar	14	⊥	∆
Adresa	Adresa klijenta	Varchar	30	T	∆
Mesto	Mesto prebivališta klijenta ili sedišta pravnog lica	Varchar	12	T	∆
Postanski_broj	Poštanski broj mesta	Integer	5	T	length(d) = 5
Mejl	Mejl klijenta	Varchar	30	T	∆
Ključ	K = {*ID_klijenta*}				

Tabela 4. Tip entiteta *POPUST*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Rbr_popusta	Redni broj popusta	Integer	3	⊥	d > 0
Ispunjen_uslov	Potvrda ispunjenja uslova za obračunavanje popusta	Boolean	1	⊥	d ∈{0, 1}
Brojac	Evidencija broja zahteva određenog klijenta, inicijalno je 0, ali se uvećava sa	Integer	8	⊥	d ≥ 0

	svakim zahtevom				
Ključ	K = {*ID_klijenta* + *Rbr_popusta*}				

Tabela 5. Tip poveznika *VEZUJE_SE_ZA* (*POPUST* i *AKCIJA*)

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Ključ	K = {*ID_klijenta* + *Rbr_popusta*}				

Tabela 6. Tip entiteta *AKCIJA*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_akcije	Jedinstveni identifikacioni broj akcije	Integer	8	⊥	d > 0
Naziv_akcije	Naziv određene akcije	Varchar	40	⊥	∆
Opis	Opis pogodnosti u okviru određene akcije	Varchar	100	⊥	∆
Datum_pocetka_akcije	Datum kada počinje određena akcija	Date		⊥	∆
Datum_zavrsetka_akcije	Datum završavanja određene akcije	Date		⊥	∆
Ključ	K = {*ID_akcije*}				

Tabela 7. Tip poveznika *OBRACUNAVA_SE* (*VRSTA_USLUGE* i *AKCIJA*)

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Broj_usluga	Broj usluga koji predstavlja uslov za ostvarenje prava na popust	Integer	2	⊥	d > 0
Ključ	K = {*ID_akcije*}				

Tabela 8. Tip entiteta *VRSTA_USLUGE*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_vrste_usluge	Jedinstveni identifikacioni broj vrste usluge	Integer	8	⊥	d > 0
Naziv_vrste_usluge	Naziv vrste usluge	Varchar	30	⊥	∆
Cena	Cena za datu vrstu usluge	Integer	8	⊥	d > 0
Valuta	Valuta plaćanja	Varchar	10	⊥	∆
Ključ	K = {*ID_vrste_usluge*}				
Unique(*VRSTA_USLUGE*, *Naziv_vrste_usluge*) = T					

Tabela 9. Tip poveznika *PODNOSI* (*ZAHTEV* i *KLIJENT*)

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Ključ	K = {*ID_zahteva*}				

Tabela 10. Tip entiteta *ZAHTEV*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_zahteva	Jedinstveni identifikacioni broj zahteva	Integer	8	⊥	d > 0
Registarski_broj	Registarski broj vozila	Varchar	10	T	∆
Datum_podnosenja	Datum podnošenja zahteva za uslugu	Date		⊥	∆
Ključ	K = {*ID_zahteva*}				

Tabela 11. Tip poveznika *SE_DODELJUJE* (*ZAHTEV* i *STATUS_ZAHTEVA*)

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Datum_od	Datum početka aktivnosti statusa	Date		⊥	∆
Datum_do	Datum završetka aktivnosti statusa	Date		T	∆
Aktivnost_statusa	Potvrda da je jedan od statusa aktivan u datom trenutku	Boolean	1	⊥	d ∈{0, 1}
Ključ	K = {*ID_zahteva* + *ID_statusa_zahteva*}				

Tabela 12. Tip entiteta *STATUS_ZAHTEVA*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_statusa_zahteva	Jedinstveni identifikacioni broj statusa	Integer	8	⊥	d > 0
Naziv_statusa	Naziv statusa	Varchar	12	⊥	∆
Ključ	K = {*ID_statusa_zahteva*}				

Tabela 13. Tip entiteta *TERMIN*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Rbr_termina	Redni broj predloženog termina	Integer	2	⊥	d > 0
Vreme	Vreme predloženog termina	Time		⊥	∆
Odobren	Potvrđenost predloženog termina	Boolean	1	⊥	d ∈{0, 1}
Ključ	K = {*ID_zahteva* + *Rbr_termina*}				

Tabela 14. Tip poveznika *SE_ZAKAZUJE* (*TERMIN* i *DAN*)

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Ključ	K = {*ID_zahteva* + *Rbr_termina*}				

Tabela 15. Tip entiteta *DAN*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_dana	Jedinstveni identifikacioni broj dana	Integer	5	⊥	d > 0
Datum	Konkretan datum (dan, mesec i godina)	Date		⊥	Δ
Radni_dan	Potvrda da li je radni dan	Boolean	1	⊥	d ∈{0, 1}
Ključ	K = {*ID_dana*}				

Tabela 16. Tip poveznika *JE* (*DAN* i *VRSTA_DANA*)

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
Ključ	K = {*ID_dana*}				

Tabela 17. Tip entiteta *VRSTA_DANA*

Naziv obeležja	Opis obeležja	Tip podatka	Dužina podatka	*Null*	Uslov
ID_vrste_dana	Jedinstveni identifikacioni broj vrste dana	Integer	3	⊥	d > 0
Naziv_vrste_dana	Naziv određene vrste dana	Varchar	25	⊥	Δ
Radno_vreme	Radno vreme određene vrste dana	Varchar	15	⊥	Δ
Ključ	K = {*ID_vrste_dana*}				
Unique(*VRSTA_DANA*, *Naziv_vrste_dana*) = T					

6. Relacioni model podšeme

Relacioni model baze podataka nastaje s ciljem otklanjanja nedostataka uočenih kod prethodnih modela podataka, mrežnog i hijerarhijskog. Njihovi glavni nedostaci bilu su: čvrsta povezanost programa i fizičke strukture podataka, strukturalna kompleksnost i proceduralno orijentisani jezici za manipulaciju podacima. Problem čvrste povezanosti rešen je potpunim odvajanje prezentacionog od formata memorisanja, kao i uvođenjem pojma šema relacije. Problem strukturalne kompleksnosti rešen je uvođenjem pojma relacije, dok je navigacioni jezik zamenjen *SQL* (*Structured Query Language*) jezikom, deklarativnim jezikom koji je zasnovan na relacionom računu nad torkama.

Predstavlja model koji se koristi za organizaciju podataka u obliku tabela i definisanje veza između tih tabela. Osnovna ideja relacionog modela jeste da podaci budu organizovani u relacijama, gde svaka relacija predstavlja tabelu sa redovima i kolonama (atributima).

Relacioni model podataka uključuje nekoliko osnovnih i složenih koncepata za organizaciju podataka.

Osnovni koncepti relacionog modela podataka su:

1. **Tabela**: Osnovna struktura podataka u relacionom modelu. Predstavlja kolekciju srodnih podataka organizovanih u redove i kolone (atribute). Svaki red u tabeli predstavlja jedan entitet ili instancu, dok svaka kolona predstavlja atribut tog entiteta.
2. **Atribut:** Predstavlja karakteristike ili svojstva entiteta u tabeli. Svaki atribut ima ime i domen vrednosti koji definiše tip podataka koji atribut može sadržati (npr. celobrojni broj, datum itd.).
3. **Ključ**: Koristi se za jedinstvenu identifikaciju redova u tabeli. Primarni ključ je jedinstveni identifikator svakog reda u tabeli i obezbeđuje jedinstvenost i jednoznačnost. Sporedni ključevi (sekundarni ključevi) su ključevi koji se koriste za identifikaciju redova u kontekstu specifičnih zahteva.

Složeni koncepti relacionog modela podataka uključuju:

1. **Veze**: Veze se koriste za uspostavljanje povezanosti između tabela u bazi podataka. Veze se uspostavljaju preko ključeva i definišu način na koji su tabele međusobno povezane. Veze mogu biti jedan prema jedan, jedan prema više ili više prema više.
2. **Ograničenja integriteta***:* Ograničenja se koriste za definisanje pravila koja moraju biti ispunjena prilikom unošenja, ažuriranja ili brisanja podataka u tabelama. Ograničenja mogu uključivati primarne ključeve, sporedne ključeve, jedinstvenost, provere integriteta podataka i druge uslove.
3. **Upiti** (*Querying*): Upiti su način dobijanja podataka iz baze podataka. Korisnici mogu koristiti upitni jezik (npr. *SQL - Structured Query Language*) za izvršavanje upita koji vraćaju određene informacije iz tabela ili kombinuju podatke iz više tabela.

Ti koncepti omogućavaju organizaciju i upravljanje podacima u relacionom modelu podataka, pružajući fleksibilnost, efikasnost i konzistentnost u manipulaciji podacima.

U nastavku teksta prikazana je relaciona šema baze podataka koja je dobijena transformacijom ER šeme baze podataka na osnovu koje je kreirana baza podataka.

PRAVNO_LICE({*PIB, Naziv, Ime_Pl, Prezime_Pl*}, {*PIB*})

FIZICKO_LICE({*ID_fizickog_lica, Ime_fl, Prezime_fl, JMBG_fl*}, {*ID_fizickog_lica*})

KLIJENT({*ID_klijenta, Broj_telefona, Adresa, Mesto, Postanski_broj, Mejl, PIB, ID_fizickog_lica*}, {*ID_klijenta*})

KLIJENT[*PIB*] ⊆ *PRAVNO_LICE*[*PIB*]

KLIJENT[*ID_fizickog_lica*] ⊆ *FIZICKO_LICE*[*ID_fizickog_lica*]

(*PIB* = ω ∧ *ID_fizickog_lica* ≠ ω) ∨ (*PIB* ≠ ω ∧ *ID_fizickog_lica* = ω)

Unique(*KLIJENT, PIB*) = T

Unique(*KLIJENT, ID_fizickog_lica*) = T

AKCIJA({*ID_akcije, Naziv_akcije, Opis, Broj_usluga, Datum_pocetka_akcije, Datum_zavrsetka_akcije, ID_vrste_usluge*}, {*ID_akcije*})

AKCIJA[*ID_vrste_usluge*] ⊆ *VRSTA_USLUGE*[*ID_vrste_usluge*]

Null(*AKCIJA, ID_vrste_usluge*) = ⊥

POPUST({*Rbr_popusta, Ispunjen_uslov, Brojac, ID_klijenta, ID_akcije*}, {*ID_klijenta + Rbr_popusta*})

POPUST[*ID_klijenta*] ⊆ *KLIJENT*[*ID_klijenta*]

POPUST[*ID_akcije*] ⊆ *AKCIJA*[*ID_akcije*]

Null(*POPUST, ID_akcije*) = ⊥

VRSTA_USLUGE({*ID_vrste_usluge, Naziv_vrste_usluge, Cena*}, {*ID_vrste_usluge*})

ZAHTEV({*ID_zahteva, Registarski_broj, Datum_podnosenja, ID_klijenta*},{*ID_zahteva*})

ZAHTEV[*ID_klijenta*] ⊆ *KLIJENT*[*ID_klijenta*]

Null(*ZAHTEV, ID_klijenta*) = ⊥

STATUS_ZAHTEVA({*ID_statusa_zahteva, Naziv_statusa*}, {*ID_statusa_zahteva,*})

SE_DODELJUJE({*Datum_od, Datum_do, Aktivnost_statusa, ID_zahteva,*

ID_statusa_zahteva}, {*ID_zahteva* + *ID_statusa_zahteva*})

SE_DODELJUJE[*ID_zahteva*] ⊆ *ZAHTEV*[*ID_zahteva*]

ZAHTEV[*ID_zahteva*] ⊆ *SE_DODELJUJE*[*ID_zahteva*]

SE_DODELJUJE[*ID_statusa_zahteva*] ⊆ *STATUS_ZAHTEVA*[*ID_statusa_zahteva*]

VRSTA_DANA({*ID_vrste_dana, Naziv_vrste_dana, Radno_vreme*}, {*ID_vrste_dana*})

DAN({*ID_dana, Datum, Radni_dan, ID_vrste_dana*}, {*ID_dana*})

DAN[*ID_vrste_dana*] ⊆ *VRSTA_DANA*[*ID_vrste_dana*]

Null(*DAN, ID_vrste_dana*) = ⊥

TERMIN({*Rbr_termina, Vreme, Odobren, ID_zahteva, ID_dana*},

{*ID_zahteva* + *Rbr_termina*})

TERMIN[*ID_zahteva*] ⊆ *ZAHTEV*[*ID_zahteva*]

ZAHTEV[*ID_zahteva*] ⊆ *TERMIN*[*ID_zahteva*]

TERMIN[*ID_dana*] ⊆ *DAN*[*ID_dana*]

7. Data Definition Language DDL

DDL (*Data Definition Language*) predstavlja skup naredbi u *SQL*-u koje se koriste za definisanje i upravljanje strukturom baze podataka. *DDL* se koristi za definisanje objekata baze podataka kao što su tabele, sekvence, funkcije, procedure, trigeri, pogledi, uloge itd. Najznačajnije *DDL* naredbe su *CREATE, ALTER, DROP* i *TRUNCATE* koje služe, redom, za kreiranje, modifikaciju i brisanje objekata baze podataka. U nastavku je opisano nekoliko osnovnih naredbi koje obuhvata DDL:

1. **CREATE**

Naredba *CREATE* koristi se za kreiranje objekata baze podataka. Na slici 3 prikazana je *DDL* naredba za kreiranje tabele *Klijent*. Prilikom kreiranja tabele potrebno je navesti naziv tabele, zatim skup obeležja koja opisuju tu tabelu, kao i njihove tipove podataka i dužinu podataka. Pri definisanju obeležja tabele navodi se i podatak o tome da li obeležje dozvoljava *NULL* vrednosti. Zatim se navode sva ograničenja koja je potrebno definisati za određenu tabelu. Na slici 3, u okviru tabele *Klijent*, definisana su različita ograničenja.

```
CREATE TABLE Projekat_IT34_2020.Klijent
(
    ID_klijenta NUMERIC(8) NOT NULL,
    Broj_telefona VARCHAR(14) NOT NULL,
    Adresa VARCHAR(30),
    Mesto VARCHAR(12),
    Postanski_broj NUMERIC(5),
    Mejl VARCHAR(30),
    PIB NUMERIC(9),
    ID_fizickog_lica NUMERIC(8),
    CONSTRAINT PK_Klijent PRIMARY KEY (ID_klijenta),
    CONSTRAINT FK_Klijent_PravnoLice FOREIGN KEY (PIB)
        REFERENCES Projekat_IT34_2020.PravnoLice(PIB),
    CONSTRAINT FK_Klijent_FizickoLice FOREIGN KEY (ID_fizickog_lica)
        REFERENCES Projekat_IT34_2020.FizickoLice(ID_fizickog_lica),
    CONSTRAINT CHK_Klijent_ID_klijenta CHECK (len(ID_klijenta) > 0),
    CONSTRAINT CHK_Klijent_Postanski_broj CHECK (len(Postanski_broj) = 5),
    CONSTRAINT CHK_Klijent_PIB_ID_fizickog_lica CHECK ((PIB IS NULL AND ID_fizickog_lica IS NOT NULL)
                                                OR (PIB IS NOT NULL AND ID_fizickog_lica IS NULL))
);
```

Slika 3. Primer korišćenja naredbe CREATE za kreiranje tabele Klijent

2. ALTER

Naredba *ALTER* koristi se za modifikaciju objekata baze podataka. Modifikacija tabele može da podrazumeva dodavanje kolona u tabelu, brisanje postojećih kolona ili izmenu definicije kolone. Na slici 4 prikazana je *ALTER* naredba za dodavanje ograničenja nad tabelom *PravnoLice*, konkretno dodavanje sekvence *Seq_PravnoLice.*

```
ALTER TABLE Projekat_IT34_2020.PravnoLice
    ADD CONSTRAINT DFT_PravnoLice_PIB DEFAULT (next value for Projekat_IT34_2020.Seq_PravnoLice) for PIB;
```

Slika 4. Primer korišćenja naredbe ALTER za dodavanje ograničenja tabele PravnoLice

3. DROP

Naredba *DROP* se koristi za brisanje objekata baze podataka. Na slici 5 je prikazana *DROP* naredba brisanje tabele *Klijent* i sekvence *Seq_PravnoLice.*

```
IF OBJECT_ID('Projekat_IT34_2020.Klijent', 'U') IS NOT NULL
    DROP TABLE Projekat_IT34_2020.Klijent;
GO

IF OBJECT_ID('Projekat_IT34_2020.Seq_PravnoLice', 'SO') IS NOT NULL
    DROP SEQUENCE Projekat_IT34_2020.Seq_PravnoLice;
GO
```

Slika 5. Primer korišćenja naredbe DROP za brisanje tabele Klijent i brisanje ograničenja tabele PravnoLice

Pored ovih naredbi, u projektu su korišćene još neke *DDL* naredbe koje omogućavaju definisanje i upravljanje strukturom baze podataka, što je ključno za organizaciju i manipualaciju podacima.

8. Data Manipulation Language DML

DML (*Data Manipulation Language*) predstavlja jezik koji se koristi za manipulaciju podataka u tabelama baze podataka. *DML* se koristi za unos, modifikaciju, brisanje i pretraživanje torki u tabeli. Najznačajnije *DML* naredbe su *SELECT*, *INSERT*, *UPDATE* i *DELETE* koje služe, redom, za pretraživanje, unos, modifikaciju i brisanje torki u tabelama baze podataka.

1. SELECT

Naredba *SELECT* se koristi za pretraživanje i izlistavanja podataka iz tabela. Omogućava izbor određenih kolona i uslova za filtriranje podataka. Na slici 6 prikazana je upotreba jednostavnog *SELECT* upita.

```
SELECT * FROM Projekat_IT34_2020.PravnoLice
SELECT * FROM Projekat_IT34_2020.FizickoLice
SELECT * FROM Projekat_IT34_2020.Klijent
```

Slika 6. Primer korišćenja naredbe SELECT za ispisivanje podataka iz tabela

2. INSERT

Naredba *INSERT* se koristi za unos novih torki u tabelu u bazu podataka. Nakon ključne reči *INSERT INTO* navodi se naziv tabele u koju se unose torke i može se navesti lista obeležja tabele, zatim sledi ključna reč *VALUES* i navode se vrednosti koje se unose u tabelu. Na slici 7 prikazana je naredba *INSERT* za unos nove torke u tabelu *Klijent*.

```
INSERT INTO Projekat_IT34_2020.Klijent (ID_klijenta, Broj_telefona, Adresa, Mesto, Postanski_broj, Mejl, PIB, ID_fizickog_lica)
    VALUES (
        5,
        '025486259',
        'Glavna 159',
        'Ruma',
        22400,
        'aleksic@gmail.com',
        null,
        1
    )
```

Slika 7. Primer korišćenja naredbe INSERT INTO za dodavanje novih vrednosti u tabelu Klijent

3. UPDATE

Naredba *UPDATE* se koristi za modifikaciju vrednosti obeležja postojeće torke u tabeli u bazi podataka. Nakon ključne reči *UPDATE* navodi se naziv tabele u okviru koje se vrši modifikacija, zatim ključna reč *SET* zatim obeležje čija se vrednost modifikuje i nakon toga

ključna reč *WHERE* i *ID* te torke čija se vrednost obeležja modifikuje. Na slici 8 prikazana je naredba *UPDATE* za modifikaciju obeležja *Cena* u tabeli *VrstaUsluge.*

```
UPDATE VrstaUsluge
SET Cena = @novaCena
WHERE ID_vrste_usluge = (SELECT ID_vrste_usluge FROM inserted)
```

Slika 8. Primer korišćenja naredbe *UPDATE* za izmenu postojeće vrednosti obeležja *Cena* u tabeli *VrstaUsluge*

4. **DELETE**

Naredba *DELETE* koristi se za brisanje postojeće torke u okviru tabele u bazi podataka. Nakon ključnih reči *DELETE FROM* navodi se naziv tabele iz koje se briše torka, zatim ključna reč *WHERE* i ID te torke koja se briše. Ukoliko se ne navede *WHERE* uslov u okviru *DELETE* naredbe, obrisaće se sve torke iz navedene tabele. Na slici 9 prikazana je naredba *DELETE* za brisanje svih torki iz tabela *PravnoLice*, *FizickoLice* i *Klijent*.

```
DELETE FROM Projekat_IT34_2020.PravnoLice
DELETE FROM Projekat_IT34_2020.FizickoLice
DELETE FROM Projekat_IT34_2020.Klijent
```

Slika 9. Primer korišćenja naredbe DELETE

5. **COMMIT**

Naredba *COMMIT* se koristi za potvrđivanje promena u bazi podataka i trajno čuvanje izvršenih operacija.

6. **ROLLBACK**

Naredba *ROLLBABCK* se koristi za poništavanje izmena i vraćanje baze podataka u prethodno stanje. Koristi se u slučaju greške ili potrebe za odustajanjem od promena.

Ove su samo neke od naredbi koje spadaju u *DML* u okviru *SQL*-a. One omogućavaju manipulaciju podacima u tabelama baze podataka, što je ključno za izvršavanje različitih operacija i održavanje konzistentnosti podataka.

9. Structured Query Langugage SQL

SQL (*Structured Query Language*) predstavlja deklarativni jezik koji je zasnovan na relacionom računu nad torkama i koji se koristi za rad sa skupovima podataka tj. sa torkama. *SQL* jeste standardni jezik za relacione baze podataka, za upravljanje i manipulaciju bazama podataka. Koristi se za definisanje strukture baze podataka, unošenje, izmenu, brisanje i pretraživanje podataka, upravljanje korisnicima i privilegijama, kontrolu transakcija i mnogo toga. Jedna od najznačajnijih upotreba *SQL*-a jeste za izvšavanje upita nad bazom podataka. *SQL* obuhvata nekoliko ključnih oblasti i funkcionalnosti.

Osnovni oblik naredbe za *SQL* upite jeste:

```
SELECT  <lista obeležja>
FROM  <lista relacija>
WHERE  <logički izraz>.
```

Pored osnovnog oblika naredbe za *SQL* upite, u *SQL* upitima moguće je koristiti i klauzule (*group by*, *having*, *order by*), podupite, ugrađene funkcije (*DATEPART*(), *CAST*(), *LEFT*(), *CONCAT*() itd.), kao i grupne funkcije (*SUM*, *MAX*, *MIN*, *AVG*, *COUNT*).

1. Definisanje objekata:
   - CREATE DATABASE – kreira novu bazu podataka,
   - CREATE TABLE – definiše novu tabelu i njene kolone,
   - CREATE INDEX – kreira indeks na određenoj koloni za poboljšanje performansi i
   - CREATE VIEW – definiše virtuelnu tabelu baziranu na postojećim tabelama.

2. Manipulacija podacima:
   - SELECT – izvršava upit za pretraživanje i dohvatanje podataka iz tabela,
   - INSERT INTO – unosi nove redove (torke) u tabelu,
   - UPDATE – ažurira postojeće podatke u tabeli i
   - DELETE – briše redove iz tabele.

3. Upravljanje korisnicima i privilegijama:
   - CREATE USER – kreira novog korisnika u bazi podataka,
   - GRANT – dodeljuje privilegije korisnicima nad određenim objektima i
   - REVOKE – oduzima privilegije korisnicima.

4. Transakcije:
   - BEGIN TRANSACTION – počinje transakciju,
   - COMMIT – potvrđuje izvršene promene i trajno čuva transakciju i
   - ROLLBACK – poništava izmene i vraća bazu podataka u prethodno stanje.

5. Ograničenja:
    - PRIMARY KEY – definiše primarni ključ za jedinstvenu identifikaciju redova u tabeli,
    - FOREIGN KEY – definiše spoljni ključ koji uspostavlja vezu sa drugom tabelom,
    - UNIQUE – ograničava vrednosti kolone da budu jedinstvene i
    - CHECK – definiše uslov koji mora biti zadovoljen za unos podataka.

6. Pogledi i funkcije:
    - CREATE VIEW – definiše virtuelnu tabelu baziranu na postojećim tabelama i
    - CREATE FUNCTION – kreira korisnički definisanu funkciju koja vraća vrednost.

7. Indeksi:
    - CREATE INDEX – kreira indeks na određenoj koloni za poboljšanje performansi pretrage.

8. Agregatne funkcije:
    - COUNT – vraća broj redova koji zadovoljavaju određeni uslov,
    - SUM – vraća zbir vrednosti izabrane kolone,
    - AVG – vraća prosečnu vrednost izabrane kolone,
    - MAX – vraća najveću vrednost izabrane kolone i
    - MIN – vraća najmanju vrednost izabrane kolone.

9. Podupiti:
    - EXISTS – proverava postojanje rezultata podupita,
    - IN – proverava da li vrednost pripada rezultatima podupita i
    - LIKE – proverava da li vrednost odgovara određenom obrascu.

10. Grupisanje i sortiranje:
    - GROUP BY – grupiše rezultate po određenoj koloni,
    - HAVING – postavlja uslov za grupisanje i
    - ORDER BY – sortira rezultate po određenoj koloni.

SQL je moćan jezik koji omogućava efikasno upravljanje podacima u relacionim bazama podataka. Ove naredbe omogućavaju definisanje strukture, unošenje, izmenu i brisanje podataka, upravljanje korisnicima i privilegijama, kontrolu transakcija, implementaciju ograničenja i još mnogo toga, čime se omogućava efikasno upravljanje podacima u bazi podataka. Na slici 10 prikazan je jedan od SQL upita u okviru projekta.

```
SELECT Zahtev.ID_zahteva as 'Redni broj zahteva', Registarski_broj as 'Registarska oznaka vozila', count(Termin.Rbr_termina) as 'Broj termina', max(Dan.Datum) as 'Poslednji datum',
       sum(CASE WHEN Termin.Odobren = 1 THEN 1 ELSE 0 END) as 'Broj odobrenih termina'
            FROM Projekat_IT34_2020.Zahtev LEFT JOIN Projekat_IT34_2020.Termin on (Zahtev.ID_zahteva = Termin.ID_zahteva)
                                           LEFT JOIN Projekat_IT34_2020.Dan on (Termin.ID_dana = Dan.ID_dana)
WHERE Zahtev.Datum_podnosenja >= DATEADD(MONTH, -3, GETDATE()) --AND Dan.Radni_dan = 0
GROUP BY Zahtev.ID_zahteva, Zahtev.Registarski_broj
HAVING count(Termin.Rbr_termina) > 0 AND sum(CASE WHEN Termin.Odobren = 1 THEN 1 ELSE 0 END) > 0
ORDER BY count(Termin.Rbr_termina) desc;
```

Slika 10. Primer SQL upita u okviru projekta

10. Objekti

U okviru poglavlja OBJEKTI dat je opis svih objekata koji su kreirani u toku izrade projektnog zadatka. Uz projektnu dokumentaciju priloženi su i:

- *script* za kreiranje i brisanje scheme, tabela i sekvenci,
- *script* za *insert* podataka,
- *script* sa SQL upitima,
- *script* za kreiranje funkcija,
- *script* za kreiranje procedura i
- *script* za kreiranje trigera.

U okviru projekta za praćenje poslovanja preduzeća za pružanje vulkanizerskih usluga korišćeni su sledeći objekti:

1. **Schema**

Predstavlja logičku grupu objekata baze podataka, odnosno kolekciju svih objekata koji dele isti prostor imenovanja. Nad *schemom* mogu biti kreirani različiti objekti: tabele, sekvence, funkcije, procedure, trigeri, pogledi itd.

Kreiranje *scheme* obavlja se pomoću naredbe CREATE SCHEMA.

```
CREATE SCHEMA <naziv_scheme>
```

Sintaksa za brisanje *scheme* putem naredbe *DROP*

```
DROP SCHEMA <naziv_scheme>
```

ili:

```
IF SCHEMA_ID('<naziv_scheme>) IS NOT NULL
DROP SCHEMA <naziv_scheme>;
GO
```

U okviru projekta kreirana je *schema* pod nazivom *Projekat_IT34_2020* i u okviru nje kreirane su sve tabele, sekvence, funkcije, procedure i trigeri (prikazano na slici 11).

```
-------------------- DROP SCHEMA --------------------
IF SCHEMA_ID('Projekat_IT34_2020') IS NOT NULL
    DROP SCHEMA Projekat_IT34_2020;
GO

-------------------- CREATE SCHEMA --------------------
GO
CREATE SCHEMA Projekat_IT34_2020;
GO
```

Slika 11. Primer brisanja i kreiranja scheme

2. Tabele

Predstavljaju objekte koji se koriste za čuvanje i prikaz podataka. Sastoje se iz kolona i redova, pri čemu kolone tabele predstavljaju obeležja, dok su redovi tabele torke. Za obeležja u okviru tabele potrebno je definisati naziv obeležja, tip podatka, dužinu podatka, kao i da li dozvoljavaju *NULL* vrednosti. Takođe, u okviru tabela se definišu razna ograničenja. Definiše se pomoću naredbe CREATE TABLE koja navodi naziv tabele, kolone i njihove tipove podataka. U okviru projekta kreirano je 12 tabela u okviru *Projekat_IT34_2020* scheme.

Sintaksa za kreiranje tabele putem naredbe CREATE:

```
CREATE TABLE <naziv_scheme>.<naziv tabele>
(<naziv obeležja1> <tip podatka> [NULL/NOT NULL],
<naziv obeležja 2> <tip podatka> [NULL/NOT NULL]
CONSTRAINT <naziv ograničenja> …)
```

Sintaksa za brisanje tabele putem naredbe *DROP*:

```
DROP TABLE <naziv_scheme>.<naziv tabele>
```

ili:

```
IF OBJECT_ID (' <naziv_scheme>.<naziv tabele>', 'U') IS NOT NULL
DROP TABLE <naziv_scheme>.<naziv tabele>;
```

```
CREATE TABLE Projekat_IT34_2020.PravnoLice
(
    PIB NUMERIC(9) NOT NULL,
    Naziv VARCHAR(20) NOT NULL,
    Ime_pl VARCHAR(12) NOT NULL,
    Prezime_pl VARCHAR(20) NOT NULL,
    CONSTRAINT PK_PravnoLice PRIMARY KEY (PIB),
    CONSTRAINT CHK_PravnoLice_PIB CHECK(PIB BETWEEN 100000001 AND 999999999 and len(PIB) = 9)
);

IF OBJECT_ID('Projekat_IT34_2020.PravnoLice', 'U') IS NOT NULL
    DROP TABLE Projekat_IT34_2020.PravnoLice;
GO
```

Slika 12. Primer kreiranja i brisanja tabele

3. Sekvenca

Predstavlja objekat koji služi za automatsko generisanje numeričkih vrednosti. Nakon generisanja numeričke vrednosti, inkrementira se za određenu vrednost u zavisnosti od toga kako je definisana. Najčešće se koristi za generisanje vrednosti primarnog ključa. Definiše se pomoću naredbe *CREATE SEQUENCE* koja navodi naziv sekvence, početnu vrednost i korak. U okviru projekta napravljene su sekvence za generisanje vrednosti primarnog ključa u odgovarajućim tabelama.

Osnovna sintaksa za kreiranje sekvence putem naredbe CREATE:

```
CREATE SEQUENCE <naziv_scheme>.<naziv_sekvence>
```

Pored osnovne sintakse za kreiranje sekvence moguće je dodati određeni broj opcija:

- INCREMENT BY – za koliko se inkrementira vrednost sekvence,

- MINVALUE – minimalna vrednost sekvence,
- MAXVALUE – maksimalna vrednost sekvence,
- START WITH – broj od kojeg počinje sekvenca i
- CYCLE/NO CYCLE – da li je sekvenca ciklična ili ne.

Sintaksa za brisanje sekvenci putem naredbe DROP:

```
DROP SEQUENCE <naziv_scheme>.<naziv_sekvence>
```

```
IF OBJECT_ID('Projekat_IT34_2020.Seq_PravnoLice', 'SO') IS NOT NULL
    DROP SEQUENCE Projekat_IT34_2020.Seq_PravnoLice;
GO


CREATE SEQUENCE Projekat_IT34_2020.Seq_PravnoLice as INT
start with 100000001
increment by 1
minvalue 1
no cycle;
```

Slika 13. Primer brisanja i kreiranja sekvence

4. Funkcija

Objekat koji prihvata ulazne parametre i izvršava određenu akciju definisanu u telu funkcije. Omogućava izvršavanje određenih operacija nad podacima. Koriste se za izračunavanje vrednosti, manipulaciju podacima, formatiranje rezultata i obavljanje različitih logičkih ili matematičkih operacija. *SQL* funkcije mogu biti ugrađene (*built-in*) ili korisnički definisane.

Ugrađene funkcije su već uključene u sam sistem za upravljanje bazom podataka (*DBMS*). One su široko podržane i omogućavaju izvršavanje različitih operacija nad podacima. Sledi nekoliko primera ugrađenih funkcija:

- COUNT – vraća broj redova koji zadovoljavaju određeni uslov,
- SUM – izračunava zbir vrednosti u određenoj koloni,
- AVG – izračunava prosečnu vrednost u određenoj koloni i
- CONCAT – spaja dve ili više nizova karaktera u jedan.

Funkcije se u SQL-u mogu pozivati na bilo kom mestu, gde se mogu pozivati i gde se mogu pozivati i ugrađene funkcije:

- u listi *SELECT* objekata,
- u uslovu kod *WHERE* ili *HAVING* klauzula,
- *CONNECT BY*, *START WITH*, *ORDER BY* i *GROUP BY* klauzula,
- *VALUES* klauzule u *INSERT* naredbi ili
- *SET* klauzule u *UPDATE* naredbi.

Korisnički definisane funkciju su funkcije koje korisnici mogu kreirati kako bi zadovoljili svoje specifične potrebe. One se definišu pomoću naredbe *CREATE FUNCTION* i mogu prihvatiti ulazne parametre i vraćati određenu vrednost. Korisnički definisane funkcije omogućavaju ponovno korišćenje određene logike ili operacija u različitim delovima upita ili skripti, čime se poboljšava efikasnost i čitljivost koda.

U zavisnosti od toga da li kao rezultat vraćaju vrednost ili tabelu, u okviru SQL-a postoje tri vrste funkcija:

- FN – SQL skalarne funkcije
- IF – SQL inline table – valued funkcije
- TF – SQL multi statement table – valued funkcije

Skalarne funkcije su funkcije koje mogu da prihvate više parametara, ali vraćaju jednu vrednost. Sintaksa za kreiranje skalarne funkcije putem naredbe *CREATE*:

```
CREATE FUNCTION <naziv_scheme>.<naziv_funkcije> (
	--definisanje ulaznih parametara funkcije
	@parametar1 as varchar(20),
	@parametar2 as int,
	@parametar3 as date, ...
) RETURNS VARCHAR(20) --tip podatka koji vraća funkcija
AS
	--telo funkcije
BEGIN
	RETURN --povratna vrednost funkcije
END;
GO;
```

Sintaksa za brisanje skalarne funkcije putem naredbe *DROP*:

```
IF OBJECT_ID('<naziv_scheme>.<naziv_funkcije>, 'FN') IS NOT NULL
	DROP FUNCTION <naziv_scheme>.<naziv_funkcije>;
```

Inline table-valued funkcije su funkcije koje mogu da prihvate više parametara, a čija je povratna vrednost tabela koja nastaje kao rezultat upita u *RETURN* bloku, pri čemu nije potrebno definisati strukturu tabele.

Sintaksa za kreiranje inline table-valued funkcije je slična sintaksi skalarnih funkcija, a razlikuje se po povratnom tipu vrednosti u *RETURN* bloku. U okviru *RETURN* bloka navodi se upit, a povratna vrednost je tabela. U nastavku je prikazana sintaksa *RETURN* bloka kod inline table-valued funkcija.

```
...
RETURNS TABLE AS RETURN (
	<SELECT ...>
);
```

Sintaksa za brisanje inline table-valued funkcije putem naredbe *DROP*:

```
IF OBJECT_ID('<naziv_scheme>.<naziv_funkcije>, 'IF') IS NOT NULL

	DROP FUNCTION <naziv_scheme>.<naziv_funkcije>;
```

Multi statment table-valued funkcije su funkcije koje mogu da prihvate više parametara, a čija je povratna vrednost tabela koja nastaje kao rezultat upita u *BEGIN-END* bloku, pri čemu je

neophodno definisati strukturu tabele koja se vraća kao rezultat funkcije i napuniti je eksplicitno.

Sintaksa za kreiranje multi statment table-valued funkcije putem naredbe *CREATE:*

```
CREATE FUNCTION <naziv_scheme>.<naziv_funkcije>
(
	@parametar1 as varchar(20),
	@ parametar2 as int,
	@ parametar3 as date, ...
)
RETURNS @tabela TABLE
AS (
	-- definisanje strukture tabele
	 obeležje1 as int,
	obeležje2 as varchar(20),
	obeležje3 as numeric(10, 2), ...
)
AS
BEGIN
	INSERT @tabela  --eksplicitno punjenje tabele koja se vraća kao rezultat
		SELECT  obeležje1, obeležje2, obeležje3, ...
		FROM <naziv_scheme>.<naziv_tabele>
		WHERE <obeležje1 > = @parametar2
	RETURN -- izlazi iz funkcije, ali ne vraća vrednost
END;
GO
```

Sintaksa za brisanje multi statment table-valued funkcije putem naredbe *DROP*:

```
IF OBJECT_ID('<naziv_scheme>.<naziv_funkcije>, 'TF') IS NOT NULL

	DROP FUNCTION<naziv_scheme>.<naziv_funkcije>;
```

5. Procedura

Predstavlja objekat baze podataka koji sadrži kod koji se izvršava kao zasebna celina. Procedure su slične metodama u drugim programskim jezicima. Koristi se za izvršavanje određene logike ili operacija. Kreira se pomoću naredbe *CREATE PROCEDURE*. Procedura se, za razliku od funkcije, poziva putem naredbe *EXEC* tj. *EXECUTE* nakon koje se navodi sam naziv procedure.

Sintaksa za kreiranje procedure putem naredbe CREATE:

```
CREATE PROC <naziv_scheme>.<naziv_procedure>
	-- procedura ne mora imati ulazne parametre
	-- parametri mogu biti obavezni i opcioni, opcioni su oni koji imaju definisanu podrazumevanu vrednost
	-- parametri mogu biti ulazni i izlazni, izlazni se definišu putem ključne reči OUTPUT
	@parametar1 AS VARCHAR(20),
	@ parametar2 AS INT OUTPUT,
```

```
      @ parametar3 AS DATE, ...
AS
BEGIN
      SET NOCOUNT ON; -- isključuje ispisivanje poruke na konzoli kada se izvršava
      procedura
      SELECT *
      FROM <naziv_scheme>.<naziv_tabele>
      WHERE obeležje1=@parametar1; -- primer blok naredbe u BEGIN-END bloku
      procedure
      RETURN; -- prekida izvršavanje procedure
END
GO
```

Sintaksa za poziv procedure:

```
exec <naziv_scheme>.<naziv_procedure> @parametar1, @parametar2, @parametar3 ;
```

Sintaksa za brisanje procedure putem naredbe *DROP*:

```
IF OBJECT_ID('<naziv_scheme>.<naziv_procedure>, 'p') IS NOT NULL

      DROP PROC <naziv_scheme>.<naziv_procedure>;
```

6. Kursor

Koristi se za iteraciju kroz rezultate upita u bazi podataka. Omogućava prolazak kroz redove tabele koja nastaje kao rezultat nekog *SQL* upita. Pristup jednom redu podataka u svakoj iteraciji. Kursor se kreira pomoću naredbe *DECLARE CURSOR*. Kursor se prvo deklariše, zatim se otvara, koristi i na kraju zatvara i uklanja se referenca na njega.

Sintaksa za deklaraciju kursora:

```
-- deklaracija kursora
declare <ime>_cursor cursor fast_forward read_only for
select obeležje1, obeležje2,...
      from <naziv_scheme>.<naziv_tabele>;

-- otvaranje kursora
open <ime>_cursor;
-- zahvatanje prve vrste iz kursora

fetch next from <ime>_cursor into @varijabla1, @varijabla2;
while @@fetch_status = 0
 -- 0 - uspešno izvršen fetch, -1 - neuspešno izvan result seta, -2 - neuspešno red nedostaje
      begin
            -- operacije nad vrstom
            ...
            -- preuzimanje sledeće vrste iz kursora
            fetch next from <ime>_cursor into @varijabla1, @varijabla2;
      end;
```

```
-- zatvaranje kursora, ali referenca na kursor ostaje
close <ime>_cursor;

-- uklanjanje i reference na kursor
deallocate <ime>_cursor;
go
```

7. Triger

Predstavlja specijalnu uskladištenu proceduru koja se vezuje za *DML* događaje: *INSERT*, *UPDATE*, *DELETE* (ili *DDL* događaje), pri čemu ti događaji predstavljaju okidač za izvršavanje trigera. To je *SQL* blok koji se izvršava kao odgovor na određeni događaj ili akciju nad tabelom. Koriste se za implementaciju poslovnih pravila ili za automatsko ažuriranje podataka. U okviru *SQL*-a postoje dve vrste trigera: *AFTER* triger koji se okida nakon izvršene *DML* naredbe i *INSTEAD OF* triger koji se izvršava umesto *DML* naredbe. Normalan izlazak iz trigera je poziv *RETURN*, kao i iz uskladištene procedure. U telu trigera, moguće je pristupiti privremenim tabelama inserted i deleted. Tabele sadrže torke koje su pod uticajem *DML* događaja koji je doveo do okidanja trigera. U slučaju *INSERT* i *UPDATE* događaja, inserted tabela sadrži novo stanje odnosno nove torke. U slučaju *UPDATE* i *DELETE* događaja, deleted tabela sadrži staro stanje odnosno stare torke. Triger se kreira pomoću naredbe *CREATE TRIGGER*

Sintaksa za kreiranje trigera putem naredbe *CREATE*:

```
create trigger <naziv_scheme>.<naziv_trigera>
on <naziv_scheme>.<naziv_tabele>
after/ instead of <insert/ update/ delete>
as
begin
	-- telo trigera
end;
go
```

Sintaksa za brisanje trigera putem naredbe *DROP*:

```
if object_id('<naziv_scheme>.<naziv_trigera>, 'tr') is not null
	drop trigger <naziv_scheme>.<naziv_trigera>;
```

Ovo su neki od objekata koji su korišćeni prilikom izrade projekta u SQL-u. Sintaksa i upotreba ovih objekata mogu se prilagoditi specifičnim zahtevima projekta i platformi za upravljanje bazom podataka.

11. Upiti

U projektu je kreirano pet *SQL* upita čije detaljno objašnjenje sledi u nastavku.

Prvi SQL upit

Izlistati: ID_vrste_usluge kao 'Šifru vrste usluge', Naziv_vrste_usluge kao 'Naziv vrste usluge', cenu, valutu, broj klijenata koji su koristili svaku vrstu usluge kao i 'Ukupan broj korišćenja usluge'. Sortirati po ceni rastuće.

Objašnjenje upita:

Ovaj *SQL* upit koristi se za izlistavanje informacija o vrstama usluge, broju klijenata koju su koristili svaku vrstu usluge kao i ukupan broj korišćenja usluge.

Upit kombinuje podatke iz nekolika tabela: „VrstaUsluge", „Akcija" i „Popust". Kroz korišćenje LEFT JOIN operacija tabela „VrstaUsluge" povezuje se sa ostalim tabelama na osnovu odgovarajućih ključeva.

Rezultat upita obuhvata sledeće kolone:

- Šifra vrste usluge – obuhvata identifikacioni broj vrste usluge,
- Naziv vrste usluge – naziv vrste usluge pod određenim identifikacionim brojem,
- Cena – cena usluge,
- Valuta – valuta za koju važi cena usluge,
- Broj klijenata – broj klijenata koji su koristili određenu vrstu usluge i
- Ukupan broj korišćenja usluge – ukupan broj korišćenja određene vrste usluge.

Preko *GROUP BY* grupisaćemo sve ono što smo prethodno napisali u select delu. Funkcija *COUNT* služi za brojanje skupova vrednosti onog što se nalazi u zagradi, u ovom slučaju *ID_klijenta*.

Konačno, rezultati se sortiraju prema ceni u rastućem redosledu, što se postiže korišćenjem klauzule *ORDER BY* sa parametrom „Cena asc".

Rezultat upita:

	Šifra vrste usluge	Naziv vrste usluge	Cena	Valuta	Broj klijenata	Ukupan broj korišćenja usluge
1	4	Skladištenje pneumatika	500	RSD	1	3
2	2	Provera pritiska	1200	RSD	1	3
3	2	Provera pritiska	1200	RSD	0	NULL
4	3	Krpljenje pneumatika	1500	RSD	1	1
5	3	Krpljenje pneumatika	1500	RSD	1	2
6	1	Zamena pneumatika	2000	RSD	1	0
7	5	Balansiranje pneumatika	3500	RSD	0	NULL

Slika 14. Rezultat prvog SQL upita

Drugi SQL upit

Izlistati ID_klijenta kao 'Redni broj klijenta', Ime_fl i Prezime_fl kao 'Ime i prezime klijenta', Broj_telefona kao 'Broj telefona', Mesto, ID_zahteva kao 'Identifikacioni broj', Naziv_vrste_usluge za koju se zahtev podnosi kao 'Vrsta usluge'. Za zahteve klijenata (samo fizičkih lica) koji su podneti dana '2023-10-01' i čija je Aktivnost_statusa != 1. Sortirati po mestu rastuće, a prezimenu opadajuće.

Objašnjenje upita:

Ovaj *SQL* upit koristi se za izlistavanje informacija o klijentima (fizičkim licima) i izabranoj vrsti usluge prema zahtevu podnetom dana '2023-10-01' čiji status nije aktivan.

Upit kombinuje podatke iz nekolika tabela: „Klijent", „FizičkoLice", „VrstaUsluge", „Akcija", „Popust" i „SeDodeljuje". Kroz korišćenje JOIN operacija tabela „Klijent" povezuje se sa ostalim tabelama na osnovu odgovarajućih ključeva.

Rezultat upita obuhvata sledeće kolone:

- ID klijenta – obuhvata identifikacioni broj klijenta,
- Ime i prezime klijenta – ime i prezime fizičkog lica sa određenim identifikacionim brojem,
- Broj telefona – broj telefona klijenta,
- Mesto – mesto (prebivalište) klijenta,
- Identifikacioni broj zahteva – identifikacioni broj zahteva koji klijent podnosi i
- Vrsta usluge – vrsta usluge za koju je klijent podneo zahtev.

Uz to, upit koristi WHERE klauzulu za filtriranje rezultata prema određenom uslovu. U ovom slučaju, uslov je da je datum podnošenja zahteva'2023-10-01'i da status nije aktivan. Ovo omogućava da se prikažu samo zahtevi podneti određenog datuma čiji status nije aktivan.

Preko *GROUP BY* grupisaćemo sve ono što smo prethodno napisali u select delu. Funkcija *COUNT* služi za brojanje skupova vrednosti onog što se nalazi u zagradi.

Konačno, rezultati se sortiraju prema mestu u rastućem, a prema prezimenu fizičkog lica u opadajućem redosledu, što se postiže korišćenjem klauzule *ORDER BY* sa parametrom „Mesto asc, Prezime_fl desc".

Rezultat upita:

	ID_klijenta	Ime i prezime klijenta	Broj telefona	Mesto	Identifikacioni broj zahteva	Vrsta usluge
1	1	Marko Markovic	0600464776	Temerin	1	Krpljenje pneumatika

Slika 15. Rezultat drugog SQL upita

Treći SQL upit

Izlistati : Naziv_vrste_usluge, Broj_zahteva kao 'Broj zahteva', Broj odobrenih termina, Broj klijenata gde je broj odobrenih termina veći od 0 i sortirati rastuće prema odobrenim terminima i nazivu vrste usluge.

Objašnjenje upita:

Ovaj SQL upit koristi se za izlistavanje informacija o nazivima vrsta usluge, odnosno o broju zahteva za određenu uslugu, broju odobrenih termina i broju klijenata.

Upit kombinuje podatke iz nekolika tabela: „Klijent", „FizičkoLice", „VrstaUsluge", „Akcija", „Popust" i „SeDodeljuje". Kroz korišćenje LEFT JOIN operacija tabela „VrstaUsluge" povezuje se sa ostalim tabelama na osnovu odgovarajućih ključeva.

Rezultat upita obuhvata sledeće kolone:

- Naziv vrste usluge – obuhvata nazive različitih vrsta usluga,
- Broj zahteva – ukupan broj zahteva za određenu uslugu,
- Broj odobrenih termina – ukupan broj odobrenih termina za određeni zahtev i
- Broj klijenata – ukupan broj klijenata za vrstu usluge.

Uz to, upit koristi WHERE klauzulu za filtriranje rezultata prema određenom uslovu. U ovom slučaju, uslov je da termin bude odobren, odnosno da vrednost obeležja *Termin.Odobren* = 1. Ovo omogućava da se prikažu samo termini koji su odobreni.

Preko *GROUP BY* grupisaćemo sve ono što smo prethodno napisali u select delu. Funkcija *COUNT* služi za brojanje skupova vrednosti onog što se nalazi u zagradi. Klauzula *HAVING* koristi se zajedno sa *GROUP BY* klauzulom za filtriranje rezultata grupisanih upita. Funkcioniše na sličan način kao *WHERE* klauzula, ali se primenjuje nakon grupisanja.

Konačno, rezultati se sortiraju prema broju odobrenih termina i prema nazivu vrste usluge u rastućem redosledu, što se postiže korišćenjem klauzule *ORDER BY* sa parametrom „count(Termin.Odobren) asc, Naziv_vrste_usluge asc".

Rezultat upita:

	Naziv vrste usluge	Broj zahteva	Broj odobrenih termina	Broj klijenata
1	Zamena pneumatika	1	1	1
2	Krpljenje pneumatika	4	4	2

Slika 16. Rezultat trećeg SQL upita

Četvrti SQL upit

Kreirati upit koji vraća informacije o zahtevima koji su podneti u poslednja tri meseca. Izlistati: ID_zahteva kao 'Redni broj zahteva', Registarski_broj vozila za koje se podnosi zahtev kao 'Registarska oznaka vozila', 'Ukupan broj termina za svaki zahtev', 'Poslednji datum' na koji se odnosi termin kao i 'Broj odobrenih termina'. Prikazati samo zahteve koji imaju bar jedan termin i bar jedan odobren termin. Sortirati u opadajućem redosledu prema broju termina.

Objašnjenje upita:

Ovaj SQL upit koristi se za izlistavanje informacija o zahtevima koji su podneti u poslednja tri meseca iščitavajući redom podatke o rednom broju zahteva, registarskom broju vozila, ukupnom broju termina, poslednji datum za koji važo termin kao i broju odobrenih termina.

Upit kombinuje podatke iz nekolika tabela: ,,Zahtev", ,,Termin" i ,,Dan". Kroz korišćenje LEFT JOIN operacija tabela ,,Zahtev" povezuje se sa ostalim tabelama na osnovu odgovarajućih ključeva.

Rezultat upita obuhvata sledeće kolone:

- Redni broj zahteva – obuhvata identifikacioni broj zahteva,
- Registarska oznaka vozila - registarski broj vozila za koji se podnosi zahtev,
- Broj termina – ukupan broj termina za zadati zahtev,
- Poslednji datum – poslednji datum za zahtev i
- Broj odobrenih termina – broj odobrenih termina za zahtev.

Uz to, upit koristi WHERE klauzulu za filtriranje rezultata prema određenom uslovu. U ovom slučaju, uslov je da je datum podnošenja zahteva prosleđen u poslednja tri meseca. Ovo omogućava da se prikažu samo zahtevi podnešeni u prethodna tri meseca.

Preko *GROUP BY* grupisaćemo sve ono što smo prethodno napisali u select delu. Funkcija *COUNT* služi za brojanje skupova vrednosti onog što se nalazi u zagradi (broja termina), a funkcija *MAX* kako bismo dobili poslednji datum. Klauzula *HAVING* koristi se zajedno sa *GROUP BY* klauzulom za filtriranje rezultata grupisanih upita. Funkcioniše na sličan način kao *WHERE* klauzula, ali se primenjuje nakon grupisanja.

Konačno, rezultati se sortiraju prema opadajućem ukupnom broju termina, što se postiže korišćenjem klauzule ORDER BY sa parametrom ,,count(Termin.Rbr_termina) desc".

Rezultat upita:

	Redni broj zahteva	Registarska oznaka vozila	Broj termina	Poslednji datum	Broj odobrenih termina
1	1	NS 465 MF	4	2023-02-15	3
2	5	NI 298 MJ	1	2023-06-13	1

Slika 17. Rezultat četvrtog SQL upita

Peti SQL upit

Izlistati podatke o klijentima: ID_klijenta kao 'Redni broj klijenta', Broj_telefona kao 'Broj telefona', Adresa, Mesto, Postanski_broj as 'Postanski broj', Naziv trenutno aktivnog statusa za određeni zahtev i trenutno aktuelne akcije. Izlistati samo one klijente čiji broj telefona počinje sa 060 ili 069. Sortirati ih po ID_statusa_zahteva rastuće.

Objašnjenje upita:

Ovaj SQL upit koristi se za izlistavanje informacija o klijentima (broju njihovog telefona, adresi i mestu stanovanja, kao i poštanskom broju). Takođe dostupne su informacije o trenutno aktivnom status njihovog zahteva i akcije koja je trenutno u toku.

Upit kombinuje podatke iz nekolika tabela: „Klijent", „Zahtev", „SeDodeljuje", „StatusZahteva", „Popust" i „Akcija". Kroz korišćenje JOIN operacija tabela „Klijent" povezuje se sa ostalim tabelama na osnovu odgovarajućih ključeva.

Rezultat upita obuhvata sledeće kolone:

- Redni broj klijenta – identifikaciona oznaka klijenta,
- Broj telefona – broj telefona klijenta,
- Adresa – adresa stanovanja klijenta,
- Mesto – mesto (prebivalište) klijenta,
- Poštanski broj – poštanski broj mesta klijenta,
- Trenutno aktivan status – status koji je trenutno aktivan za zahtev klijenta i
- Akcija koja je trenutno u toku – naziv akcije koja je trenutno u toku.

Uz to, upit koristi WHERE klauzulu za filtriranje rezultata prema određenom uslovu. U ovom slučaju, uslov je da broj telefona klijenta počinje sa '060' ili '069'. Ovo omogućava da se prikažu samo klijenti sa određenim brojevima telefona. Za ovakav tip poređenja koristi se operator *OR*.

Konačno, rezultati se sortiraju prema mestu u rastućem, a prema prezime fizičkog lica u opadajućem redosledu, što se postiže korišćenjem klauzule ORDER BY sa parametrom „ID_statusa_zahteva asc".

Rezultat upita:

	Redni broj klijenta	Broj telefona	Adresa	Mesto	Poštanski broj	Trenutno aktivan status	Akcija koja je trenutno u toku
1	2	0607523564	Pavla Papa	Novi Sad	21000	Novi	Tri u jedan
2	1	0600464776	Adi Endrea 1/4	Temerin	21235	Završen	Krpljenje za minut!
3	3	069452758	Partizanska 29	Backi Jarak	21234	Prihvacen	Dva u jedan

Slika 18. Rezultat petog SQL upita

12. Funkcije

U projektu su kreirane dve funkcije čije detaljno objašnjenje sledi u nastavku.

Prva funkcija

Kreirana je funkcija FunkcijaZahtev koja za prosleđeni parametar @klijentID vraća ukupan broj zahteva koje je taj klijent podneo.

Objašnjenje funkcije:

Funkcija je deklasirana pod nazivom „Projekat_IT34_2020.FunkcijaZahtev“. Ova SQL funkcija je skalarna i kao ulazni parametar prihvata identifikacioni broj klijenta (@klijentID) tipa numeric, kao rezultat vraća broj zahteva koje je taj klijent uputio. Rezultat druge funkcije jeste tipa int.

U telu funkcije, u okviru BEGIN – END bloka definiše se promenljiva @brojZahteva i njena vrednost će biti setovana na vrednost koja se dobija iz SQL upita koji izlistava broj zahteva na osnovu prosleđenog ID-ja klijenta.

U okviru upita neophodno je izvršiti spajanje tabele *Klijent* sa tabelom *Zahtev* po *ID_klijenta.* Funkcija kao rezultat vraća varijablu @brojZahteva koja sadrži podatak o broju zahteva koje je klijent podneo.

Testiranje funkcije prikazano je na sledećem listingu:

```
SELECT distinct(Klijent.ID_klijenta) as 'Rbr klijenta', Mejl, Broj_telefona as 'Broj telefona', Projekat_IT34_2020.FunkcijaZahtev(Zahtev.ID_klijenta) as 'Broj klijentskih zahteva'
	FROM Projekat_IT34_2020.Klijent LEFT JOIN Projekat_IT34_2020.Zahtev on (Klijent.ID_klijenta = Zahtev.ID_klijenta)
```

Na slici ispod se može videti rezultat pozivanja funkcije Projekat_IT34_2020.FunkcijaZahtev.

	Rbr klijenta	Mejl	Broj telefona	Broj klijentskih zahteva
1	1	markovic@gmail.com	0600464776	3
2	2	lukovic@gmail.com	0607523564	1
3	3	bojanic@gmail.com	069452758	2
4	4	hashelena@heliant....	068245789	1
5	5	aleksic@gmail.com	025486259	4

Slika 19. Rezultat pozivanja druge funkcije

Druga funkcija

Kreirana je funkcija FunkcijaKlijentInfo koja za prosleđeni parametar @klijentID vraća informacije o klijentu kao i njegovom tipu.

Objašnjenje funkcije:

Ova SQL funkcija definisana je kako bi pružila mogućnost dobijanja informacija o klijentu na osnovu njegovog identifikacionog broja.

Funkcija je deklasirana pod nazivom „Projekat_IT34_2020.FunkcijaKlijentInfo“ i prihvata jedan parametar @klijentID tipa numeric(8) koji predstavlja identifikacioni broj klijenta.

Funkcija vraća tabelu kao rezultat. Tabela sadrži osam kolona:

- ID_klijenta – identifikacioni broj klijenta,
- Naziv – naziv klijenta pod određenim identifikacionim brojem,
- Broj_telefona – broj telefona klijenta pod određenim identifikacionim brojem,
- Adresa – adresa klijenta pod određenim identifikacionim brojem,
- Mesto – mesto klijenta pod određenim identifikacionim brojem,
- Postanski_broj – postanski broj klijenta pod određenim identifikacionim brojem,
- Mejl – mejl klijenta pod određenim identifikacionim brojem i
- Tip klijenta – tip klijenta pod određenim identifikacionim brojem.

Logika funkcije sastoji se od SELECT upita koji uključuje proveru da li je klijent fizičko ili pravno lice. Ukoliko identifikacioni broj fizičkog lice nije NULL onda se ispisuje njegovo ime i prezime. U suprotnom ispisuje se naziv pravnog lica. Takođe ispisuju se ostali podaci o klijentu čiji identifikacioni broj je jednak prosleđenom broju.

Na taj način, ova funkcija omogućava korisnicima da pozovu funkciju sa prosleđenim identifikacionim brojem klijenta i dobiju rezultat koji sadrži informacije o klijentu, uključujući njegov tip i naziv, mejl, broj telefona, adresu i mesto stanovanja sa poštanskim brojem.

Testiranje funkcije:

Na slici ispod se može videti poziv funkcije sa prosleđenim identifikacionim brojem 2, a potom i povratna vrednost odnosno tabela koja sadrži prethodno navedene podatke.

```
SELECT * FROM Projekat_IT34_2020.FunkcijaKlijentInfo(2);
```

	ID_klijenta	Naziv	Broj_telefona	Adresa	Mesto	Postanski_broj	Mejl	Tip klijenta
1	2	Levi9	0607523564	Pavla Papa	Novi Sad	21000	lukovic@gmail.com	Pravno lice

Slika 20. Rezultat druge funkcije FunkcijaKlijentInfo

13. Procedure

U projektu su kreirane dve procedure čije detaljno objašnjenje sledi u nastavku.

Prva procedura

Procedura koja za prosleđen datum podnošenja zahteva (Datum_podnosenja) ispisuje podatke o klijentima, konkretno o fizičkim licima, koji su podneli zahtev i to u formi: „Datuma: <Datum_podnosenja> postoje sledeći zahtevi: Rbr: <rbr>, ID klijenta: <ID_klijenta>, ime i prezime: <Ime_fl> <Prezime_fl>, čiji je broj telefona: <Broj_telefona>, iz <Mesto> podnosi zahtev sa brojem: <ID_zahteva>. Ukupno zahteva: <ukupnoZahteva>". U slučaju da prosleđen datum ne postoji ispisuje odgovarajuću poruku: „Ne postoji prosleđen datum!"

Objašnjenje procedure:

Ova procedura prihvata jedan parametar *@datum* tipa date. Procedura ima za cilj da proveri da li postoji zahtev podnet na unet datum.

Najpre se deklarišu lokalne promenljive: @idKlijenta, @imePrezime, @brojTelefona, @mesto, @brojZahteva, @ukupnoZahteva i @rbr.

Zatim se koristi *SET* naredba za dodeljivanje vrednosti promenljivoj @ukupnoZahteva na osnovu podataka iz tabele *Projekat_IT34_2020.Zahtev*. Dodeljuje vrednost na osnovu *COUNT*(*) gde je uslov *Datum_podnosenja = @datum*.

Nakon toga, koristi se *IF* uslov da bi se proverilo da li *@datum* postoji u rezultatu podupita (*SELECT Datum_podnosenja FROM Projekat_IT34_2020.Zahtev*). Ako se ne nalazi, ispisuje se poruka 'Ne postoji prosleđen datum!'.

U suprotnom, koristi se *PRINT* naredba da bi se prikazala poruka koja sadrži datum koji je unet i ispisuje zahteve koji postoje tog dana.

Zatim se deklariše kursor *zahtev_cursor* koji se koristi za iteraciju kroz redove koji zadovoljavaju uslov da je *Datum_podnosenja = @datum*. *SELECT* naredba unutar kursora uzima podatke iz više tabela povezanih putem *JOIN* naredbi.

Nakon otvaranja kursora, koristi se *FETCH NEXT* naredba za preuzimanje prvog reda rezultata iz kursora i dodeljivanje vrednosti lokalnim promenljivama @idKlijenta, @imePrezime, @brojTelefona, @mesto i @brojZahteva.

Zatim se koristi *WHILE* petlja da se iterira kroz preostale redove rezultata. Unutar petlje se koristi *PRINT* naredba za prikazivanje poruke koja sadrži informacije o svakom zahtevu koji je podnet određenog datuma. *FETCH NEXT* naredba koristi se za preuzimanje sledećeg reda iz kursora i ažuriranje lokalnih promenljivih. Promenljiva @rbr koristi se za numerisanje redova. Primenom *SET* naredbe potrebno je promenljivu @rbr inkementirati za jedan pri svakom prolazu.

Nakon što se svi redovi obrade, koriste se *CLOSE* i *DEALLOCATE* naredbe za zatvaranje kursora. Potom se primenom *PRINT* naredbe ispisuje ukupan broj zahteva tog datuma.

Testiranje procedure vrši se pozivanjem *EXEC* naredbe i prosleđivanjem parametra.

```
exec Projekat_IT34_2020.ProceduraZahtev '2023-02-01'
exec Projekat_IT34_2020.ProceduraZahtev '2025-02-01'
```

Slika 21. Testiranje prve procedure

```
Datuma: 2023-02-01 postoje sledeci zahtevi:
Rbr: 1, ID klijenta: 3, ime i prezime: Bojan Bojanic ciji je broj telefona: 069452758 iz Backi Jarak podnosi zahtev sa brojem: 10.
Rbr: 2, ID klijenta: 1, ime i prezime: Marko Markovic ciji je broj telefona: 0600464776 iz Temerin podnosi zahtev sa brojem: 11.
Rbr: 3, ID klijenta: 1, ime i prezime: Marko Markovic ciji je broj telefona: 0600464776 iz Temerin podnosi zahtev sa brojem: 12.
Ukupno zahteva: 3

Ne postoji prosleden datum!
```

Slika 22. Rezultat prve procedure

Druga procedura

Kreirana procedura *ProceduraStatusAktivnosti* izvlači neaktivne zahteve iz tabele zahtev i prikazuje određene detalje kao što su: identifikacioni broj zahteva, datum njegovog podnošenja i status zahteva. Neaktivni zahtevi su oni čiji je datum završetka poznat. Za svaki zahtev ispisuje se sledeće: Identifikacioni broj zahteva: <ID_zahteva> Datum: <Datum_podnosenja> Status zahteva: <Naziv_statusa>

Objašnjenje procedure:

Ova procedura ne prihvata nijedan parametar. Procedura ima za cilj da prikaže neaktivne zahteve i određene podatke o njima. Najpre se deklarišu lokalne promenljive: @zahtevID, @zahtevDatum i @zahtevStatus.

Zatim se deklariše kursor *status_cursor* koji se koristi za iteraciju kroz redove koji zadovoljavaju uslov da je datum trajanja statusa unet, odnosno *SeDodeljuje.Datum_do IS NOT NULL*. *SELECT* naredba unutar kursora uzima podatke iz više tabela povezanih putem *INNER JOIN* naredbi.

Nakon otvaranja kursora, koristi se *FETCH NEXT* naredba za preuzimanje prvog reda rezultata iz kursora i dodeljivanje vrednosti lokalnim promenljivama @zahtevID, @zahtevDatum i @zahtevStatus.

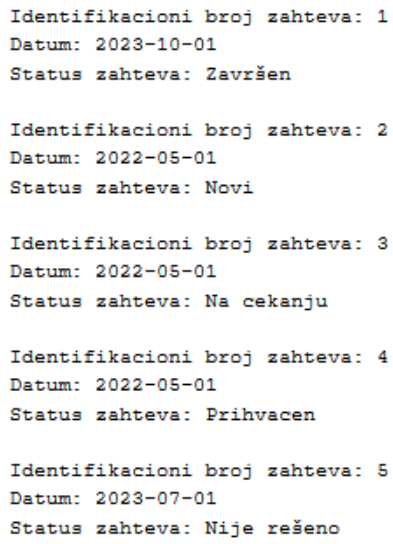
Zatim se koristi *WHILE* petlja da se iterira kroz preostale redove rezultata. Unutar petlje se koristi *PRINT* naredba za prikazivanje poruke koja sadrži informacije o svakom zahtevu koji nije aktivan, njegovom datumu podnosenja i nazivu statusa. *FETCH NEXT* naredba koristi se za preuzimanje sledećeg reda iz kursora i ažuriranje lokalnih promenljivih.

Nakon što se svi redovi obrade, koriste se *CLOSE* i *DEALLOCATE* naredbe za zatvaranje kursora.

Testiranje procedure vrši se pozivanjem *EXEC* naredbe i prosleđivanjem parametra.

```
exec Projekat_IT34_2020.ProceduraStatusAktivnosti;
```

Slika 23. Testiranje druge procedure



Slika 24. Rezultata druge procedure

14. Trigeri

U projektu su kreirana dva trigera čije detaljno objašnjenje sledi u nastavku.

Prvi triger

Omogućiti da se prilikom unosa novog statusa zahteva za određeni zahtev datum prestanka važenja prethodnog bude postavljen na datum kada je dodeljen novi i da se aktuelnog starog stavi na 0, a aktuelnost novog na 1.

Objašnjenje trigera:

Triger je definisan nad tabelom *Projekat_IT34_2020.SeDodeljuje* i aktivira se kada se vrši operacija (*INSERT*) nad datom tabelom. Umesto da se podaci direktno ubace u tabelu, ovaj triger menja ponašanje umetanja i izvršava određene akcije.

Prvobitno je potrebno kreirati i definisati naziv trigera, kao i navesti nad kojom tabelom je on definisan.

```
CREATE TRIGGER Projekat_IT34_2020.TrigerStatusZahteva ON
Projekat_IT34_2020.SeDodeljuje
```

`ON Projekat_IT34_2020.SeDodeljuje` definiše da je triger vezan za tabelu SeDodeljuje.

INSTEAD OF INSERT označava da će triger umesto običnog umetanja podataka izvršiti navedene akcije.

AS BEGIN označava početak bloka koda trigera.

Prva akcija trigera je ažuriranje postojećeg reda u tabeli Projekat_IT34_2020.SeDodeljuje. Konkretno, `UPDATE Projekat_IT34_2020.SeDodeljuje` izvršava ažuriranje nad tabelom *Projekat_IT34_2020.SeDodeljuje*.

`SET Datum_do = (SELECT Datum_od FROM inserted)` postavlja vrednost kolone *Datum_do* na vrednost iz kolone *Datum_od* iz tabele inserted. Tabela *inserted* sadrži redove koji su dodati.

Takođe, `Aktivnost_statusa = 0` postavlja vrednost kolone *Aktivnost_statusa* na 0.

`WHERE Aktivnost_statusa = 1 AND ID_statusa_zahteva = (SELECT ID_statusa_zahteva FROM inserted)` postavlja uslov za ažuriranje. Ovo znači da će samo redovi koji ispunjavaju uslov biti ažurirani. Uslov jeste da je *Aktivnost_statusa* jednaka 1 i da je *ID_statusa_zahteva* isti kao *ID_statusa_zahteva* iz tabele *inserted*.

Nakon toga sledi operacija (*INSERT*) u istu tabelu *Projekat_IT34_2020.SeDodeljuje*. Podaci koji se umeću dolaze iz tabele *inserted*.

SELECT deo navodi koje kolone treba popuniti novim vrednostima. Ove vrednosti uzimaju se iz tabele inserted.

NULL postavlja vrednost kolone *Datum_do* na *NULL* za status zahteva, dok 1 postavlja vrednost kolone *Aktivnost_statusa* na 1 za status zahteva.

`FROM inserted` označava se podaci umeću iz tabele *inserted*.

Dakle, ovaj triger prvo ažurira postojeći red u tabeli Projekat_IT34_2020.SeDodeljuje, postavljajući određene vrednosti kolona. Zatim vrši umetanje novog reda u istu tabelu, pri čemu se neke kolone popunjavaju sa vrednostima iz tabele inserted.

Testiranje trigera prikazano je na narednim slikama.

```
INSERT INTO Projekat_IT34_2020.SeDodeljuje(ID_statusa_zahteva, ID_zahteva, Datum_od, Datum_do, Aktivnost_statusa)
VALUES (1, 3, '2023-06-01', null, 1);

INSERT INTO Projekat_IT34_2020.SeDodeljuje(ID_statusa_zahteva, ID_zahteva, Datum_od, Datum_do, Aktivnost_statusa)
VALUES (3, 5, '2021-03-03', null, 1);
```

Slika 25. Testiranje prvog trigera

	ID_statusa_zahteva	ID_zahteva	Datum_od	Datum_do	Aktivnost_statusa
1	3	1	2023-02-01	2023-02-07	0
2	1	2	2023-10-01	1900-01-01	1
3	5	3	2022-05-01	1900-01-01	1
4	7	4	2023-07-01	2023-08-01	0
5	9	5	2021-02-01	2023-02-15	1

Slika 26. Tabela SeDodeljuje pre izvršavanja trigera

	ID_statusa_zahteva	ID_zahteva	Datum_od	Datum_do	Aktivnost_statusa
1	3	1	2023-02-01	2023-02-07	0
2	1	2	2023-10-01	2023-06-01	0
3	1	3	2023-06-01	NULL	1
4	5	3	2022-05-01	1900-01-01	1
5	7	4	2023-07-01	2023-08-01	0
6	3	5	2021-03-03	NULL	1
7	9	5	2021-02-01	2023-02-15	1

Slika 27. Rezultat prvog trigera

Drugi triger

Napisati triger koji implementira poslovno pravilo: „ako se unese akcije cena se smanjuje za 10%".

Objašnjenje trigera:

Triger je definisan nad tabelom *Projekat_IT34_2020.VrstaUsluge* i aktivira se nakon što se izvrši operacija unosa (*INSERT*) nad datom tabelom.

Prvobitno je potrebno kreirati i definisati naziv trigera, kao i navesti nad kojom tabelom je on definisan.

`ON Projekat_IT34_2020.VrstaUsluge` definiše da je triger vezan za tabelu VrstaUsluge.

AFTER INSERT označava da će triger nakon umetanja podataka izvršiti navedene akcije.

AS BEGIN označava početak bloka koda trigera (tela trigera).

Prva akcija trigera jeste provera da li je broj unetih redova *(@@ROWCOUNT*) jednak nuli. Ukoliko je to slučaj triger se zaustavlja i ne izvršava dalje korake. Zatim je potrebno postaviti opciju *NOCOUNT* na *ON*, što znači da se neće vraćati broj redova koji su pogođeni *UPDATE* operacijom.

Nakon toga, potrebno je deklarisati promenljive. Definišu se tri promenljive @trenutnaCena za čuvanje trenutne cene koja se nalazi u tabeli VrstaUsluge kada je ID_vrste_usluge jednak prosleđenom id-ju.

```
SELECT Cena FROM Projekat_IT34_2020.VrstaUsluge WHERE ID_vrste_usluge  = @id
```

Potrebno je definisati i promenljivu za čuvanje nove cene (@novaCena), kao i @id za čuvanje vrednosti ID_vrste_usluge koja se preuzima iz *inserted* tabele. Tabela *inserted* sadrži redove koji su dodati.

Potrebno je dodeliti vrednosti prethodno definisanim promenljivama. Trenutna cena se popunjava vrednošću cene za odgovarajuću vrstu usluge, dok se nova cena računa kao 90% od trenutne.

Potom se pomoću *PRINT* naredbe ispisuje stara i nova cena i *UPDATE* tabele *VrstaUsluge*, odnosno njene kolone *Cena* gde se unosi vrednost @novaCene iz tabele *inserted*.

Na slici je prikazano testiranje trigera unosom novog broja statusa zahteva …. i pregled kako izgleda tabela nakon operacije INSERT.

	ID_vrste_usluge	Naziv_vrste_usluge	Cena	Valuta
1	1	Zamena pneumatika	2000	RSD
2	2	Provera pritiska	1200	RSD
3	3	Krpljenje pneumatika	1500	RSD
4	4	Skladištenje pneum...	500	RSD
5	5	Balansiranje pneu...	3500	RSD

Slika 28. Tabela VrstaUsluge pre izvršavanja trigera

```
Stara cena = 565 RSD.
Nova cena = 508 RSD.

(1 row affected)

Completion time: 2023-06-04T13:33:07.3589191+02:00
```

Slika 29. Rezultat drugog trigera

	ID_vrste_usluge	Naziv_vrste_usluge	Cena	Valuta
1	1	Zamena pneumatika	2000	RSD
2	2	Provera pritiska	1200	RSD
3	3	Krpljenje pneumatika	1500	RSD
4	4	Skladištenje pneumatika	500	RSD
5	5	Balansiranje pneumatika	3500	RSD
6	6	Provera trigera	508	RSD

Slika 30. Tabela VrstaUsluge nakon izvršavanja trigera

15. Zaključak

Cilj izrade projektnog zadatka bio je razvoj baze podataka za poslovanje vulkanizerskog preduzeća. Baza podataka omogućava efikasno beleženje i organizaciju podataka o klijentima, njihovim zahtevima, popustima, akcijama, vrstama usluge, statusima zahteva i radnom kalendaru. Olakšava praćenje podnetih klijentskih zahteva za uslugu i njihovih statusa. Takođe, omogućava efikasno upravljanje informacijama o poslovanju vulkanizerskog preduzeća. Implementacija šeme baze podataka omogućava unapređenje organizacije i efikasnosti poslovanja vulkanizerskog preduzeća.

Korisnici projektovane baze podataka u svakom momentu imaju uvid u najznačanije informacije ovog realnog sistema. Implementacija baze podataka predstavlja ključan korak unapređenja poslovnih procesa vulkanizerskog preduzeća. Ima potencijal da značajno poboljša efikasnost, produktivnost i kvalitet poslovanja u oblasti vulkanizerskog preduzeća.