# Sustav za upravljanje restoranom

## Dokumentacija tima 3 za projekt kolegija "Baze podataka 2"

***Nositelj kolegija:*** doc. dr. sc. Goran Oreški

***Izvođač kolegija:*** Romeo Šajina, mag. inf

***Tim:*** Tim 3

***Autori:***

***Luka Vilagoš:*** 0303114421

***Patricia Lazić:*** 0303114869

***Dinko Nađ:*** 0303113989

***Paula Vorih:*** 0303114507

***Alma Reka:*** 0303116340

***Sara Maljić:*** 0009094343

# Sadržaj

# 1. Uvod

## 1.1. Opis odabranog procesa

Tema koju smo odabrali za projekt za kolegij Baze Podataka 2 je "Sustav za upravljanje restoranom". Njegova svrha je omogućavanje njegovim korisnicima lakše upravljanje restoranom i njegovim evidencija. Korisnici izrađuju jelovnike te u njega upisuju stavke jelovnika, izrađuju i prate informacije o narudžbama, pregledavaju i izrađuju račune, izrađuju recepte i uređuju njihove sastojke. Također omogućuje korisnicima da uređuju informacije o restoranu, i

## 1.2. Minimalni zahtjevi

**Baza podataka:** struktura baze podataka mora podržavati sve ključne entitete i njihove međusobne odnose kao što su restorani, zaposlenici, stolovi, računi, jelovnici, sastojci, recepti, narudžbe, velike i male nezgode. Neki entiteti trebaju uključivati osnovne podatke kao što su vrijeme kreiranja, izmjene i logičko brisanje (zapis se ne briše fizički iz baze podataka, nego se označava kao obrisan - u našem slučaju "deleted_at").

**Veze među entitetima:** jasno definirane veze između entiteta, uključujući vezu između restorana i povezanih entiteta poput skladišta, zaposlenika i stolova, validaciju da su sastojci povezani s ispravnim skladištem, a stavke s relevantnim jelovnikom i restoranom te validaciju da računi uključuju stavke koje pripadaju odabranom restoranu.

**Upravljanje financijama:** omogućavanje upravljanja financijskim transakcijama za restorane i zaposlenike, uključujući praćenje stanja računa, plaća zaposlenika, mjesečnih troškova te troškova vezanih za nezgode.

**Rezervacije:** Mogućnost korisnika da kreiraju rezervacije koje uključuju validaciju dostupnosti stolova unutar odabranog restorana.

**Praćenje zaliha:** mehanizam za upravljanje skladištem i zalihama, uključujući praćenje trenutnog stanja sastojaka, potrebnih narudžbi i povezanih troškova.

**Jelovnik i recepti:** struktura za kreiranje i upravljanje jelovnicima, stavkama i receptima, uz praćenje povezanih sastojaka i njihovih količina.

**Nezgode:** mehanizam za bilježenje malih i velikih nezgoda, uključujući utjecaj na financije restorana i upravljanje korištenim resursima ili izgubljenim sastojcima.

**Funkcionalnost računa:** generiranje računa koji uključuju stavke naručene od strane korisnika povezanih s relevantnim stolom, restoranom i zaposlenikom.

**Narudžbe sastojaka:** sustav za kreiranje narudžbi potrebnih sastojaka uz praćenje njihovog statusa i povezanosti sa skladištem.

**Automatizacija procesa:** omogućavanje automatskog ažuriranja povezanih entiteta poput ažuriranja stanja skladišta prilikom kreiranja narudžbi ili obračuna troškova nakon nezgoda.

## 1.3. Cilj

Cilj ovog sustava je omogućiti jednostavan i učinkovit način upravljanja restoranima, uključujući administraciju zaposlenika, vođenje skladišta, upravljanje financijama, praćenje rezervacija i narudžbi, te vođenje evidencije o računima i jelovnicima. Sustav treba osigurati visoku razinu integriteta podataka kroz validaciju transakcija, povezanost stavki računa s jelovnicima, te praćenje stanja skladišta. Također, sustav omogućava transparentno upravljanje troškovima, analizu prihoda i troškova, kao i evidenciju nezgoda kako bi korisnici mogli donositi informirane odluke o poslovanju.

# 2. Opis poslovnog procesa

Poslovni proces koji smo ovdje modelirali temelji se na sustavu za upravljanje restoranom. Unatoč složenosti procesa, naš model je pojednostavljen kako bi bio pregledan, ali i dovoljno detaljan za prikaz kompetentnosti sustava.

Relacija **zaposlenik** pohranjuje osnovne informacije o zaposlenicima restorana, uključujući ime, prezime, email, datum rođenja, iznos plaće, tip zaposlenika (npr. vlasnik, kuhar, konobar) i sliku. Svaki zaposlenik je povezan s točno jednim restoranom putem atributa "restoran_id". Relacija **zaposlenik_placa** prati isplate plaća zaposlenicima, uključujući iznos, mjesec isplate i referencu na zaposlenika putem atributa "zaposlenik_id".

Relacija **restoran** pohranjuje osnovne informacije o restoranima, uključujući naziv, adresu, broj telefona i sliku. Svaki restoran je povezan s više zaposlenika, skladišta, stolova i računa putem odgovarajućih atributa. Relacija **skladiste** prati stanje zaliha svakog restorana, dok atribut "stanje" bilježi status skladišta (puno, prazno, kritično, normalno). Sastojci u skladištu pohranjeni su u relaciji **sastojak**, koja uključuje cijenu, potrebnu i trenutnu količinu te vrstu mjere (grami, litre, komadi). Relacija **recept_sastojak** prati potrošnju sastojaka u receptima.

Relacija **restoran_racun** upravlja financijama svakog restorana, bilježeći stanje računa u određenoj valuti. Detaljne transakcije računa restorana evidentiraju se u relaciji **transakcija_restoran**, dok se troškovi povezani sa zaposlenicima evidentiraju u relaciji **transakcija_zaposlenik**. Operativni troškovi restorana dodatno su evidentirani u relaciji **trosak**, koja prati iznos, naziv troška i je li riječ o mjesečnom trošku.

Relacija **stol** pohranjuje informacije o stolovima u restoranu, uključujući broj mjesta, lokaciju (unutra, vani, VIP) i identifikacijski broj stola. Relacija **rezervacija** evidentira rezervacije stolova, uključujući ime gosta i vrijeme rezervacije. Računi gostima evidentiraju se u relaciji **racun**, koja uključuje povezanost sa stolom, restoranom, zaposlenikom, napojnicom i ukupnim iznosom.

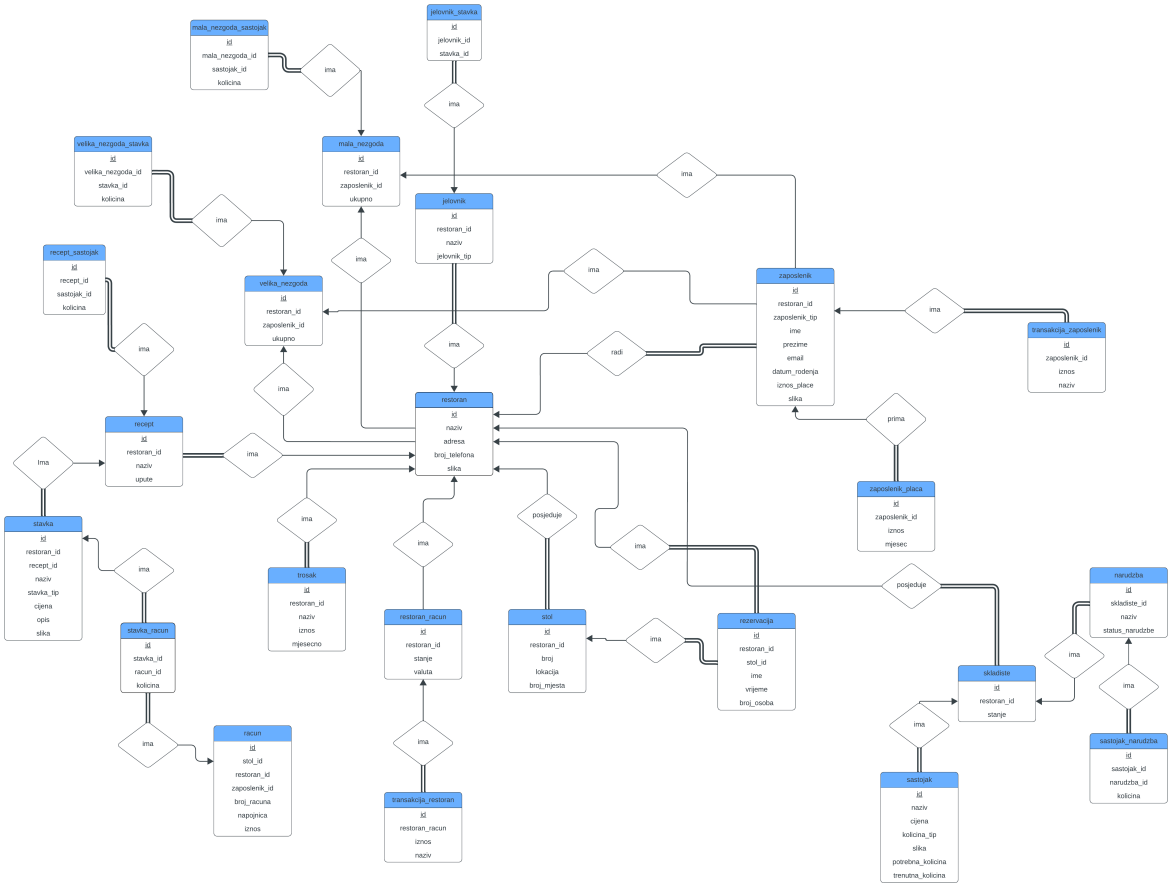
Relacija **recept** pohranjuje recepte restorana, uključujući naziv i upute. Stavke na jelovniku restorana evidentiraju se u relaciji **stavka**, koja uključuje tip (riba, meso, piće), cijenu, opis i referencu na recept. Relacija **jelovnik** organizira stavke u kategorije poput pića, jela i deserta, dok relacija **jelovnik_stavka** povezuje stavke s određenim jelovnicima.

Relacija **narudzba** prati narudžbe za popunjavanje zaliha skladišta, uključujući status narudžbe (završeno, poništeno, neriješeno). Povezanost sastojaka s narudžbom evidentirana je u relaciji **sastojak_narudzba**, koja bilježi količinu naručenog sastojka.

Relacije **mala_nezgoda** i **velika_nezgoda** bilježe nezgode u restoranu i povezane troškove. Mala nezgoda povezuje sastojke kroz relaciju **mala_nezgoda_sastojak**, dok velika nezgoda povezuje stavke kroz relaciju **velika_nezgoda_stavka**.

Ova struktura osigurava cjelovito praćenje ključnih aspekata poslovanja restorana, omogućavajući stabilnost i prilagodbu za potencijalna proširenja.

# 3. ER dijagram



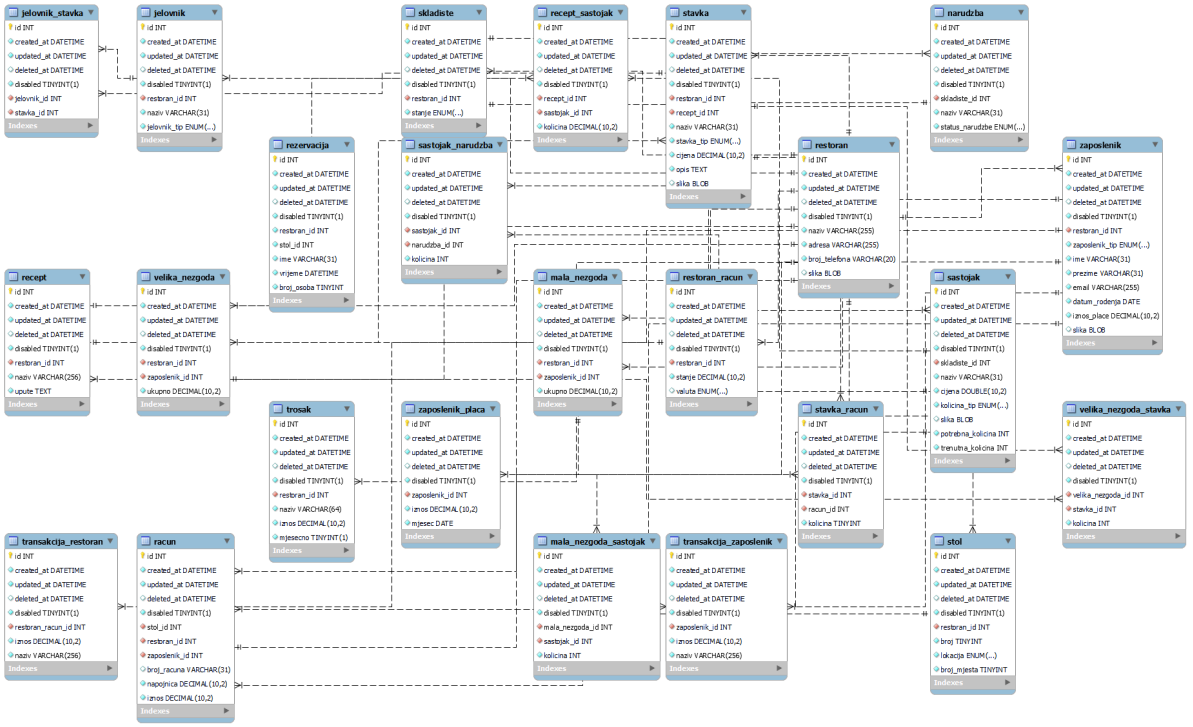

# 4. Shema relacijskog modela

restoran(id, created_at, updated_at, deleted_at, disabled, naziv, adresa, broj_telefona, slika) zaposlenik(id, created_at, updated_at, deleted_at, disabled, restoran_id, zaposlenik_tip, ime, prezime, email, datum_rodenja, iznos_place, slika) zaposlenik_placa(id, created_at, updated_at, deleted_at, disabled, zaposlenik_id, iznos, mjesec) skladiste(id, created_at, updated_at, deleted_at, disabled, restoran_id, stanje) restoran_racun(id, created_at, updated_at, deleted_at, disabled, restoran_id, stanje, valuta) transakcija_restoran(id, created_at, updated_at, deleted_at, disabled,

restoran_racun_id, iznos, naziv) transakcija_zaposlenik(id, created_at, updated_at, deleted_at, disabled, zaposlenik_id, iznos, naziv) trosak(id, created_at, updated_at, deleted_at, disabled, restoran_id, naziv, iznos, mjesecno) stol(id, created_at, updated_at, deleted_at, disabled, restoran_id, broj, lokacija, broj_mjesta) rezervacija(id, created_at, updated_at, deleted_at, disabled, restoran_id, stol_id, ime, vrijeme, broj_osoba) racun(id, created_at, updated_at, deleted_at, disabled, stol_id, restoran_id, zaposlenik_id, broj_racuna, napojnica, iznos) recept(id, created_at, updated_at, deleted_at, disabled, restoran_id, naziv, upute) stavka(id, created_at, updated_at, deleted_at, disabled, restoran_id, recept_id, naziv, stavka_tip, cijena, opis, slika) stavka_racun(id, created_at, updated_at, deleted_at, disabled, stavka_id, racun_id, kolicina) jelovnik(id, created_at, updated_at, deleted_at, disabled, restoran_id, naziv, jelovnik_tip) jelovnik_stavka(id, created_at, updated_at, deleted_at, disabled, jelovnik_id, stavka_id) sastojak(id, created_at, updated_at, deleted_at, disabled, skladiste_id, naziv, cijena, kolicina_tip, slika, potrebna_kolicina, trenutna_kolicina) recept_sastojak(id, created_at, updated_at, deleted_at, disabled, recept_id, sastojak_id, kolicina) narudzba(id, created_at, updated_at, deleted_at, disabled, skladiste_id, naziv, status_narudzbe) sastojak_narudzba(id, created_at, updated_at, deleted_at, disabled, sastojak_id, narudzba_id, kolicina) mala_nezgoda(id, created_at, updated_at, deleted_at, disabled, restoran_id, zaposlenik_id, ukupno) mala_nezgoda_sastojak(id, created_at, updated_at, deleted_at, disabled, mala_nezgoda_id, sastojak_id, kolicina) velika_nezgoda(id, created_at, updated_at, deleted_at, disabled, restoran_id, zaposlenik_id, ukupno) velika_nezgoda_stavka(id, created_at, updated_at, deleted_at, disabled, velika_nezgoda_id, stavka_id, kolicina) ```

# 5. Logička shema baze podataka

Ova logička shema generirana je u MySQL workbench-u te prikazuje detaljan prikaz baze podataka, njenih relacija i veza.



# 6. Tablice

## 6.1. Paula Vorih

### 6.1.1. Tablica zaposlenik

**Tablica "zaposlenik"** sadržava svo osoblje zaposleno u restoranu. U restoranu postoji pet različitih zaposleničkih uloga.

```
CREATE TABLE zaposlenik(
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    zaposlenik_tip ENUM('vlasnik', 'kuhar', 'glavni_konobar', 'konobar',
'cistac') DEFAULT 'kuhar' NOT NULL,
    ime VARCHAR (31) NOT NULL,
    prezime VARCHAR (31) NOT NULL,
    email VARCHAR (255) NOT NULL,
    datum_rodenja DATE NOT NULL,
    iznos_place DECIMAL NOT NULL,
    slika BLOB
);
```

**Atribut "id"** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut "restoran_id"** je tipa INT. Ne može sadržavati NULL vrijednost.

**Atribut "zaposlenik_tip"** je tipa ENUM i u njega mogu biti zapisane samo one vrijednosti iz navedenog skupa vrijednosti u zagradama, u ovom slučaju zaposlenik može biti vlasnik, kuhar, glavni konobar, konobar ili čistač. Ako ne navedemo niti jednu vrijednost, sa svojstvom DEFAULT ćemo osigurat da se u takvom slučaju zaposleniku dodjeli uloga "kuhar". Ne može sadržavati NULL vrijednost i to smo izbjegli tako da smo postavili defaultni slučaj.

**Atribut "ime"** je tipa VARCHAR(31). U njega se pohranjuje ime zaposlenika. Ne može sadržavati NULL vrijednost.

**Atribut "prezime"** je tipa VARCHAR(31). U njega se pohranjuje prezime zaposlenika. Ne može sadržavati NULL vrijednost.

**Atribut "datum_rodenja"** je tipa DATE. Datum je formata YYYY-MM-DD. U njega se pohranjuje datum rođenja pojedinog zaposlenika. Ne može sadržavati NULL vrijednost.

**Atribut "iznos_place"** je tipa DECIMAL(10,2) koji može pohraniti maksimalno 8 znamenki lijevo od decimalne točke i dvije znamenke desno od decimalne točke. U ovom atributu se pohranjuje novčani iznos plaće koja će se isplatiti korisniku na dan uplate plaće. Taj iznos je defaultno ponuđen pojedinom zaposleniku, ali se može mijenjati na dan uplate. Dodatno objašnjenje: zaposlenik.iznos je default kolika je plaća nekog zaposlenika. zaposlenik_placa.iznos je iznos koji mu je zapravo bio plaćen taj mjesec. Defaultno je zaposlenik_placa.iznos = zaposlenik.iznos ali može se razlikovati ako onaj koji isplaćuje plaće tako odluči.

## 6.1.2. Tablica zaposlenik_placa

**Tablica "zaposlenik_placa"** služi za evidentiranje dobitka pojedinog zaposlenika.

```
CREATE TABLE zaposlenik_placa (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    zaposlenik_id INT NOT NULL,
    iznos DECIMAL (10, 2) NOT NULL,
    mjesec DATE DEFAULT (CURRENT_DATE()) NOT NULL,

    FOREIGN KEY (zaposlenik_id) REFERENCES zaposlenik (id)
);
```

**Atribut "id"** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut "zaposlenik_id"** je tipa INT i ujedno je i strani ključ. Vrijednosti su jednake id-u u tablici zaposlenik. Ne može sadržavati NULL vrijednost.

**Atribut "iznos"** je tipa DECIMAL(10,2) koji može pohraniti maksimalno 8 znamenki lijevo od decimalne točke i dvije znamenke desno od decimalne točke. Služi za evidentiranje iznosa plaće. Ne može sadržavati NULL vrijednost.

**Atribut "mjesec"** je tipa DATE sa defaultnom vrijednosti današnjeg datuma. Datum je formata YYYY-MM-DD. Ne može sadržavati NULL vrijednost.

## 6.1.3. Tablica restoran

**Tablica "restoran"** služi za evidentiranje podataka o pojedinom restoranu. U aplikaciji može postojati jedan ili više restorana.

```
CREATE TABLE restoran (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    naziv VARCHAR (255) NOT NULL,
    adresa VARCHAR (255) NOT NULL,
    broj_telefona VARCHAR (20) UNIQUE NOT NULL,
    slika BLOB
);
```

**Atribut "id"** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut "naziv"** je tipa VARCHAR(255). U njega se pohranjuje ime restorana. Ne može sadržavati NULL vrijednost.

**Atribut "adresa"** je tipa VARCHAR(255). U njega se pohranjuje adresa restorana. Ne može sadržavati NULL vrijednost.

**Atribut "broj_telefona"** je tipa VARCHAR(20) sa svojstvom UNIQUE što znači da različiti restorani ne mogu imati isti broj telefona. Telefonski broj je spremljen kao varijabilni karakter jer prva znamenka može biti 0. Ne može sadržavati NULL vrijednost.

**Atribut "slika"** je tipa BLOB. U njega se pohranjuje ikona restorana. Dodavanje slike nije obavezno.

## 6.1.4. Tablica restoran_račun

**Tablica "restoran_racun"** prati financijsko stanje pojedinih restorana.

```
CREATE TABLE restoran_racun (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    stanje DECIMAL (10, 2) DEFAULT 0 NOT NULL,
    valuta ENUM ('EUR', 'USD', 'CAD')
);
```

**Atribut "id"** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut "restoran_id"** je tipa INT. Ne može sadržavati NULL vrijednost.

**Atribut "stanje"** je tipa DECIMAL(10,2) koji može pohraniti maksimalno 8 znamenki lijevo od decimalne točke i dvije znamenke desno od decimalne točke. To možemo objasniti kao stanje restorana na bankovnom računu. Defaultna vrijednost stanja je 0. Ne može sadržavati NULL vrijednost.

**Atribut "valuta"** je tipa ENUM i u njega mogu biti zapisane samo one vrijednosti iz navedenog skupa vrijednosti u zagradama, u ovom slučaju EUR (euro), USD (američki dolar) i CAD (kanadski dolar). To je u ovom slučaju glavna valuta koju restoran koristi.

# 6.2. Dinko Nađ

## 6.2.1. Tablica skladiste

**Tablica „skladiste"** koristi se za pohranjivanje stanja skladišta za svaki restoran. Jedan restoran može imati više skladišta.

```
CREATE TABLE skladiste (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    stanje ENUM('puno', 'prazno', 'kritično', 'normalno'),

    FOREIGN KEY (restoran_id) REFERENCES restoran (id)
);
```

**Atribut „id"** tipa je INT s AUTO_INCREMENT svojstvom, što znači da se vrijednost ovoga atributa automatski povećava za svako novo skladište. Ovaj atribut definiram je kao PRIMARY KEY jer je on jedinstveni identifikator za svaki zapis u tablici.

**Atribut „restoran_id"** je tipa INT i prikazuje kojem restoranu pripada skladište. Ograničenje NOT NULL znači da skladište mora biti povezano sa nekim restoranom. Atribut također ima FOREIGN KEY koji označava da se „restoran_id" zapravo referencira na „id" u tablici restoran.

**Atribut „stanje"** nam pokazuje kakvo je stanje toga skladišta. ENUM nam osigurava da se vrijednost može postaviti na samo jedno od unaprijed definiranih vrijednosti. Na početku skladište se postavlja na „prazno" jer se ne zna kakvo je skladište u trenutku stvaranja skladišta. Ograničenje NOT NULL osigurava da stanje ne smije biti null. Ako vrijednost nije zadana, znači da u skladištu nema ništa.

### 6.2.2. Tablica trosak

**Tablica "trosak"** koristi se za evidenciju troškova restorana.

```
CREATE TABLE trosak (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    naziv VARCHAR (64) NOT NULL,
    iznos DECIMAL (10, 2) NOT NULL,
    mjesecno BOOLEAN DEFAULT FALSE NOT NULL,

    FOREIGN KEY (restoran_id) REFERENCES restoran (id)
);
```

**Atribut „id“** tipa je INT s AUTO_INCREMENT svojstvom, što znači da se vrijednost ovoga atributa automatski povećava za svaki novi trošak. Ovaj atribut definiram je kao PRIMARY KEY jer je on jedinstveni identifikator za svaki zapis u tablici.

**Atribut „restoran_id“** je tipa INT i prikazuje kojem restoranu pripada koji trošak. Ograničenje NOT NULL znači da trošak mora biti povezan sa nekim restoranom. Atribut također ima FOREIGN KEY koji označava da se „restoran_id“ zapravo referencira na „id“ u tablici restoran.

**Atribut „naziv“** je tipa VARCHAR(64) i odnosi se na naziv troška koji se dogodio. On ima ograničenje NOT NULL što znači da je naziv obavezan i da polje ne smije ostati samo. Maksimalna duljina naziva je 64, ali zauzima samo upisani dio u memoriju.

**Atribut „Iznos“** je tipa DECIMAL(10,2) koji nam pokazuje koliki je trošak koji želimo evidentirati. Ovaj zapis može imati 10 znamenki sveukupno, od kojih 2 znamenke predstavljaju decimalna mjesta. Ima ograničenje NOT NULL jer se iznos ne može izostaviti.

**Atribut „mjesecno“** tipa je BOOLEAN koji označava jel se taj trošak odnosi na sve mjesece ili samo na 1 mjesec. Ako je TRUE onda se taj trošak mora svaki mjesec plaćati, a ako je FALSE, onda se mora samo jednom platiti. DEFAULT FALSE osigurava da je zadana vrijednost FALSE, što znači da nije mjesečna obaveza.

### 6.2.3. Tablica stavka

```
CREATE TABLE stavka (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    recept_id INT NOT NULL,
```

```
    naziv VARCHAR (31) NOT NULL,
    stavka_tip ENUM ('riba', 'salata', 'meso', 'alkohol', 'gazirano', 'kava',
'prilog', 'juha') NOT NULL,
    cijena DECIMAL (10, 2) NOT NULL,
    opis TEXT NOT NULL,
    slika BLOB,

    FOREIGN KEY (restoran_id) REFERENCES restoran (id),
    FOREIGN KEY (recept_id) REFERENCES recept (id)
);
```

**Atribut „id“** tipa je INT s AUTO_INCREMENT svojstvom, što znači da se vrijednost ovoga atributa automatski povećava za svaku novu stavku. Ovaj atribut definiram je kao PRIMARY KEY jer je on jedinstveni identifikator za svaki zapis u tablici.

**Atribut „restoran_id“** je tipa INT i prikazuje kojem restoranu pripada koja stavka. Ograničenje NOT NULL znači da trošak mora biti povezan sa nekim restoranom. Atribut također ima FOREIGN KEY koji označava da se „restoran_id“ zapravo referencira na „id“ u tablici restoran.

**Atribut „recept_id“** je tipa INT i povezuje stavku sa receptom. Ograničenje NOT NULL znači da trošak mora biti povezan sa nekim receptom. Atribut također ima FOREIGN KEY koji označava da se „recept_id“ zapravo referencira na „id“ u tablici recept.

**Atribut „naziv“** je tipa VARCHAR(31) koji predstavlja naziv stavke. Maksimalna duljina naziva je 31, ali zauzima samo upisani dio u memoriju. Ograničenje NOT NULL osigurava da stavka mora imati svoj naziv.

**Atribut „stavka_tip“** je tipa ENUM i pokazuje nam u koju kategoriju će spadati da stavka. ENUM nam osigurava da se vrijednost može postaviti na samo jedno od unaprijed definiranih vrijednosti. Ograničenje NOT NULL osigurava da stavka mora imati svoju kategoriju/tip.

**Atribut „cijena“** je tipa DECIMAL koji nam pokazuje kolika je cijena neke stavke. Ovaj zapis može imati 10 znamenki sveukupno, od kojih 2 znamenke predstavljaju decimalna mjesta. Ima ograničenje NOT NULL jer se iznos ne može izostaviti.

**Atribut „opis“** je tipa TEXT koji sadrži detaljnije informacije o stavci npr. Sastojci, način pripreme. To je tekstualno polje koje može sadržavati velike količine teksta. Ograničenje NOT NULL osigurava da svaka stavka mora imati svoj opis.

**Atribut „slika“** je tipa BLOB koje služi za pohranjivanje slike jela/pića. Ovo polje može pohraniti binarne podatke, poput slike u PNG formatu.

## 6.2.4. Tablica sastojak

```
CREATE TABLE sastojak (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    skladiste_id INT NOT NULL,
    naziv VARCHAR (31) NOT NULL,
```

```
    cijena DOUBLE (10, 2) NOT NULL,
    kolicina_tip ENUM ('g', 'mg', 'kg', 'l', 'ml', 'kol', 'tsp', 'tbsp') NOT
NULL,
    slika BLOB,

    potrebna_kolicina INT DEFAULT 0 NOT NULL,
    trenutna_kolicina INT DEFAULT 0 NOT NULL,

    FOREIGN KEY (skladiste_id) REFERENCES skladiste (id)
);
```

**Atribut „id“** tipa je INT s AUTO_INCREMENT svojstvom, što znači da se vrijednost ovoga atributa automatski povećava za svaku novu stavku. Ovaj atribut definiram je kao PRIMARY KEY jer je on jedinstveni identifikator za svaki zapis u tablici.

**Atribut „skladiste_id“** je tipa INT i prikazuje kojem skladištu pripada koji sastojak. Ograničenje NOT NULL znači da sastojak mora biti povezan sa nekim skladištem. Atribut također ima FOREIGN KEY koji označava da se „skladiste_id“ zapravo referencira na „id“ u tablici skladište.

**Atribut „naziv“** tipa je VARCHAR(31) i predstavlja naziv sastojka koji se mogu nalaziti u skladištu. Maksimalna duljina naziva je 31, ali zauzima samo upisani dio u memoriju. Ograničenje NOT NULL osigurava da stavka mora imati svoj naziv.

**Atribut „cijena“** je tipa DECIMAL koji nam pokazuje kolika je cijena za svaki sastojak. Ovaj zapis može imati 10 znamenki sveukupno, od kojih 2 znamenke predstavljaju decimalna mjesta. Ima ograničenje NOT NULL jer se iznos ne može izostaviti.

**Atribut „kolicina_tip“** je vrsta mjerenja količine sastojaka. ENUM nam osigurava da se vrijednost može postaviti na samo jedno od unaprijed definiranih vrijednosti koje smo mi zadali. Ograničenje NOT NULL osigurava da svaka stavka mora imati tip.

**Atribut „slika“** je tipa BLOB koje služi za pohranjivanje slike jela/pića. Ovo polje može pohraniti binarne podatke, poput slike u PNG formatu.

**Atribut „potrebna_kolicina“** je tipa INT i predstavlja minimalnu potrebnu količinu sastojaka u skladištu. Cijeli broj koji označava koliko sastojaka treba biti na zalihama. DEFAULT 0 postavlja vrijednost na 0 ako vrijednost nije definirana. Ograničenje NOT NULL osigurava da svaka stavka mora imati tip.

**Atribut „trenutna_kolicina“** je tipa INT i predstavlja trenutnu količinu sastojaka u skladištu. Cijeli broj koji označava koliko sastojaka je dostupno za korištenje. DEFAULT 0 postavlja vrijednost na 0 ako vrijednost nije definirana. Ograničenje NOT NULL osigurava da svaka stavka mora imati tip.

## 6.3. Luka Vilagoš

### 6.3.1. Tablica transakcija_restoran

Tablica “zaposlenik_placa” služi za evidentiranje dobitka pojedinog zaposlenika.

```
CREATE TABLE transakcija_restoran (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_racun_id INT NOT NULL,
    iznos DECIMAL (10, 2) DEFAULT 0 NOT NULL,
    naziv VARCHAR (256) DEFAULT 'Transakcija restoran' NOT NULL,

    FOREIGN KEY (restoran_racun_id) REFERENCES restoran_racun (id)
);
```

**Atribut “id”** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut “restoran_racun_id”** je tipa INT i ujedno je i strani ključ. Vrijednosti su jednake id-u u tablici restoran_racun. Ne može sadržavati NULL vrijednost.

**Atribut “iznos”** je tipa DECIMAL(10,2) koji može pohraniti maksimalno 8 znamenki lijevo od decimalne točke i dvije znamenke desno od decimalne točke. Služi za evidentiranje iznosa transakcije. Ne može sadržavati NULL vrijednost i defaultno je 0.

**Atribut “naziv”** je tipa VARCHAR(256) i u njega se pohranjuje opis transakcije. Ako ništa nije navedeno, onda je opis “Transakcija restoran”. Ne može biti NULL.

### 6.3.2. Tablica transakcija_zaposlenik

Tablica “transakcija_zaposlenik” prati transakcije koje je proveo pojedini zaposlenik.

```
CREATE TABLE transakcija_zaposlenik (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    zaposlenik_id INT NOT NULL,
    iznos DECIMAL (10, 2) DEFAULT 0 NOT NULL,
    naziv VARCHAR (256) DEFAULT 'Transakcija restoran' NOT NULL,

    FOREIGN KEY (zaposlenik_id) REFERENCES zaposlenik (id)
);
```

**Atribut “id”** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut "zaposlenik_id"** je tipa INT i ujedno je i strani ključ koji referencira id u tablici zaposlenik. Ne može biti NULL.

**Atribut "iznos"** je tipa DECIMAL(10,2) koji može pohraniti maksimalno 8 znamenki lijevo od decimalne točke i dvije znamenke desno od decimalne točke. To je iznos transakcije koju je proveo zaposlenik. Ne može biti NULL. Defaultni iznos te transakcije je 0.

**Atribut "naziv"** je tipa VARCHAR(256) i u njega se pohranjuje opis transakcije. Ako ništa nije navedeno, onda je opis "Transakcija restoran". Ne može biti NULL.

## 6.3.3. Tablica narudzba

Tablica "restoran" služi za evidentiranje podataka o pojedinom restoranu. U aplikaciji može postojati jedan ili više restorana.

```
CREATE TABLE narudzba (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    skladiste_id INT NOT NULL,
    naziv VARCHAR (31) NOT NULL,
    status_narudzbe ENUM ('ZAVRSENO', 'PONISTENO', 'NERIJESENO') DEFAULT
'NERIJESENO' NOT NULL,

    FOREIGN KEY (skladiste_id) REFERENCES skladiste (id)
);
```

**Atribut "id"** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom

svakog novog umetanja id vrijednosti.

**Atribut "skladiste_id"** je tipa INT i ujedno je strani ključ koji referencira id u tablici skladiste. Ne može sadržavati NULL vrijednost.

**Atribut "naziv"** je tipa VARCHAR(31) i u njega se pohranjuje naziv narudžbe. Ne može biti NULL.

**Atribut "status_narudzbe"** je tipa ENUM i u njega mogu biti zapisane samo one vrijednosti iz navedenog skupa vrijednosti u zagradama, u ovom slučaju narudžba može imati status ZAVRSENO, PONISTENO ili NERIJESENO. Ako ne navedemo niti jednu vrijednost, sa svojstvom DEFAULT ćemo osigurat da se u takvom slučaju status_narudzbe postavi na NERIJESENO. Ne može sadržavati NULL vrijednost i to smo izbjegli tako da smo postavili defaultni slučaj.

## 6.3.4. Tablica sastojak_narudzba

Tablica "zaposlenik" sadržava svo osoblje zaposleno u restoranu. U restoranu postoji pet različitih zaposleničkih uloga.

```
CREATE TABLE sastojak_narudzba (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    sastojak_id INT NOT NULL,
    narudzba_id INT NOT NULL,
    kolicina INT UNSIGNED NOT NULL,

    FOREIGN KEY (sastojak_id) REFERENCES sastojak (id),
    FOREIGN KEY (narudzba_id) REFERENCES narudzba (id)
);
```

**Atribut "id"** je tipa INT koji se svojstvom AUTO_INCREMENT povećava prilikom svakog novog umetanja id vrijednosti.

**Atribut "sastojak_id"** je tipa INT i ujedno je strani ključ koji referencira id u tablici sastojak. Ne može sadržavati NULL vrijednost.

**Atribut "narudzba_id"** je tipa INT i ujedno je strani ključ koji referencira id u tablici narudzba. Ne može sadržavati NULL vrijednost.

**Atribut "kolicina"** je tipa INT UNSIGNED što znači da može pohranjivati samo pozitivne iznose integera. Simbolizira količinu sastojka koji se naručuje. Ne može biti NULL.

# 6.4. Sara Maljić

## 6.4.1. Tablica stol

**Tablica "stol"** predstavlja stolove koji se nalaze u restoranima kao i informacije o njima, poput u kojem se restoranu nalazi koji stol, na kojoj lokaciji je smješten pojedinačni stol te koliko sjedećih mjesta ima. Njezini atributi su id, created_at, updated_at, deleted_at, disabled, restoran_id, broj, lokacija, broj_mjesta.

```
CREATE TABLE stol (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

restoran_id INT NOT NULL,
broj TINYINT UNSIGNED NOT NULL,
lokacija ENUM ('unutra', 'vani', 'vip') NOT NULL,
broj_mjesta TINYINT UNSIGNED NOT NULL,

FOREIGN KEY (restoran_id) REFERENCES restoran (id)
```

```
);
```

**Atribut "id"** je PRIMARY KEY te služi kao jedinstvena identifikacija svakog stola te je tipa INT s postavkom AUTO_INCREMENT, a služi tome da se broj automatski povećava za svaki novi zapis.

**Atribut "created_at"** je tipa DATETIME i bilježi datum i vrijeme kada je stol, točnije zapis o stolu napravljen. CURRENT_TIMESTAMP je zadana vrijednost koja omogućava automatsko postavljanje trenutnog datuma i vremena pri stvaranju novog stola te ima ograničenje NOT NULL koje osigurava da vrijednost u tom stupcu ne bude prazna.

**Atribut "updated_at"** je tipa DATETIME i bilježi datum i vrijeme posljednje izmjene zapisa o stolu. CURRENT_TIMESTAMP je zadana vrijednost koja omogućava automatsko postavljanje trenutnog datuma i vremena pri izmjeni zapisa o stolu. Ima automatsku nadogradnju ON UPDATE CURRENT_TIMESTAMP koja osigurava da se pri naredbi UPDATE vrijednost tog stupca automatski ažurira na trenutni datum i vrijeme. Također sadržava ograničenje NOT NULL koje osigurava da vrijednost u tom stupcu ne bude prazna.

**Atribut "deleted_at"** je tipa DATETIME i bilježi datum i vrijeme kada je zapis o stolu izbrisan. Ako zapis nije obrisan, to polje može ostati NULL.

**Atribut "disabled"** je tipa BOOLEAN, što znači da može sadržavati samo 2 vrijednosti, TRUE ili FALSE, a služi za označavanje je li stol onemogućen ili aktivan. Ovaj atribut ima početnu vrijednost FALSE, što znači da je stol aktivan. Također ima ograničenje NOT NULL, što znači da vrijednost ne ostane prazna.

**Atribut "restoran_id"** je tipa INT te označava id restorana kojem pripada pojedinačni stol. Sadrži ograničenje NOT NULL koje omogućuje da vrijednost ne ostane prazna. Definiran je kao FOREIGN KEY koji referencira stupac id u tablici "restoran", što znači da se na taj način uspostavlja veza između stola i restorana.

**Atribut "broj"** je tipa TINYINT UNSIGNED te označava broj stolova unutar restorana. Tip TINYINT UNSIGNED služi kako bismo mogli pohraniti cijele brojeve u rasponu od 0 do 255, no pošto također ima ograničenje NOT NULL taj raspon počinje od broja 1.

**Atribut "lokacija"** je tipa ENUM te može sadržavati samo jednu od slijedećih vrijednosti: unutra, vani, vip. Ovavrijednost označava lokaciju stola unutar restorana te također ima ograničenje NOT NULL.

**Atribut "broj_mjesta"** je tipa TINYINT UNSIGNED te označava broj sjedećih mjesta na stolovima. U njega možemo upisati raspon brojeva od 1 do 255 jer također ima ograničenje NOT NULL.

## 6.4.2. Tablica rezervacija

**Tablica "rezervacija"** se koristi za pohranu rezervacija u restoranima, a njezini atriburti su id, created_at, updated_at, deleted_at, disabled, restoran_id, stol_id, ime, vrijeme.

```
CREATE TABLE rezervacija (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

restoran_id INT NOT NULL,
stol_id INT NOT NULL,
ime VARCHAR (31) NOT NULL,
vrijeme DATETIME NOT NULL
);
```

**Atribut "id"** je tipa INT s postavkom AUTO_INCREMENT, a služi tome da se automatski povećava za svaku novu rezervaciju. Definiran je kao PRIMARY KEY te služi kao jedinstvena identifikacija svake rezervacije.

**Atribut "created_at"** je tipa DATETIME sa zadanom vrijednosti CURRENT_TIMESTAMP i bilježi datum i vrijeme kada je rezervacija napravljenja, te zbog zadane vrijednosti automatski se postavlja trenutni datum i vrijeme pri stvaranju iste. Ima ograničenje NOT NULL što osigurava da polje ne ostane prazno.

**Atribut "updated_at"** je tipa DATETIME s automatskom nadogradnjom ON UPDATE CURRENT_TIMESTAMP te kao i za atribut "created_at" bilježi datum i vrijeme kada je rezervacija izmjenjena s automatskim postavljanjem trenutnog datuma i vremena. Također ima ograničenje NOT NULL.

**Atribut "deleted_at"** je tipa DATETIME te pohranjuje datum i vrijeme kada je rezervacija obrisana. Ne sadrži ograničenje NOT NULL, dakle polje će biti prazno ukoliko rezervacija nije obrisana.

**Atribut "disabled"** je tipa BOOLEAN s početnom vrijednosti FALSE, što znači da može prihvatit samo vrijednosti TRUE ili FALSE. U ovom slučaju FALSE predstavlja aktivnu rezervaciju, a TRUE predstavlja da je rezervacija onemogućena. Ima ograničenje NOT NULL, jer je rezervacija ili aktivna ili neaktivna.

**Atribut "restoran_id"** je tipa INT te označava id restorana povezanog s rezervacijom. Također sadrži ograničenje NOT NULL jer svaka rezervacija mora imati svoj restoran.

**Atribut "stol_id"** je tipa INT te označava id stola povezanog s rezervacijom. Također ima ograničenje NOT NULL jer svaka rezervacija mora imati stol.

**Atribut "ime"** je tipa VARCHAR(31) te pohranjuje ime osobe koja je izvršila rezervaciju. Maksimalna duljina unosa je naznačena u tipu podataka, a to je u ovom slučaju 31 te ima ograničenje NOT NULL, jer za svaku rezervaciju je potrebno ime na koje glasi.

**Atribut "vrijeme"** je tipa DATETIME te označava točno vrijeme rezervacije. Sadržava ograničenje NOT NULL jer svaka rezervacija mora imati termin kada će se održati.

### 6.4.3. Tablica jelovnik

**Tablica "jelovnik"** se koristi za pohranu podataka o jelima, pićima i desertima dostupnih u restoranu, vrsti i imena jelovnika i sadrži slijedeće atribute: id, created_at, updated_at, deleted_at, disabled, restoran_id, naziv, jelovnik_tip.

```
CREATE TABLE jelovnik (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

restoran_id INT NOT NULL,
naziv VARCHAR (31) NOT NULL,
jelovnik_tip ENUM ('pica', 'jela', 'desert') NOT NULL,

FOREIGN KEY (restoran_id) REFERENCES restoran (id)
);
```

**Atribut "id"** je tipa INT sa AUTO_INCREMENT svojstvom koje služi za automatsko povećanje za svaki novi zapis. Definiran je kao PRIMARY KEY jer je potrebno identificirati svaki jelovnik jedinstveno.

**Atribut "created_at"** je tipa DATETIME i služi za bilježenje datuma i vremena kada je jelovnik stvoren. Zbog zadane vrijednosti koju ima, CURRENT_TIMESTAMP, automatski se postavlja trenutni datum i vrijeme pri stvaranju novog zapisa. Ima ograničenje NOT NULL.

**Atribut "updated_at"** je tipa DATETIME sa zadanom vrijednosti ON UPDATE CURRENT_TIMESTAMP koja omogućuje automatsko postavljanje datuma i vremena pri svakoj izmjeni zapisa. Također ima ograničenje NOT NULL.

**Atribut "deleted_at"** je tipa DATETIME te pohranjuje datum i vrijeme kada je zapis obrisan. Nema ograničenje NOT NULL jer ne mora svaki zapis biti obrisan.

**Atribut "disabled"** je tipa BOOLEAN sa zadanom vrijednosti FALSE, što znači da prima vrijednosti TRUE ili FALSE, a zbog zadane vrijednosti FALSE znači da je aktivan, a ako upišemo FALSE jelovnik će se prikazati kao onemogućen. Ovo polje ima ograničenje NOT NULL.

**Atribut "restoran_id"** je tipa INT te označava id restorana kojem jelovnik pripada. Definiran je kao PRIMARY KEY koji referencira stupac id u tablici "restoran" kako bi se jelovnik povezao s restoranom. Ima ograničenje NOT NULL jer svaki jelovnik mora imati svoj restoran.

**Atribut "naziv"** je tipa VARCHAR(31) te pohranjuje naziv jelovnika. Zbog ograničenja 31 maksimalna duljina unosa, u ovom slučaju imena jelovnika, je 31 znak. Ovo polje ima ograničenje NOT NULL jer svaki jelovnik mora imati svoj naziv.

**Atribut "jelovnik_tip"** je tipa ENUM jer može sadržavati samo jednu od slijedećih vrijednosti: pica, jela, desert. Služi označavanju vrste jelovnika i također ima ograničenje NOT NULL jer svaki jelovnik mora biti nekog od navedenih tipova.

### 6.4.4. Tablica jelovnik_stavka

**Tablica "jelovnik_stavka"** se koristi za uspostavljanje veze između jelovnika i pojedinačnih stavki unutar jelovnika. Svaki zapis u tablici povezuje jelovnik i stavku, a sadrži slijedeće atribute: id, created_at, updated_at, deleted_at, disabled, jelovnik_id, stavka_id.

```
CREATE TABLE jelovnik_stavka (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

jelovnik_id INT NOT NULL,
stavka_id INT NOT NULL,

FOREIGN KEY (jelovnik_id) REFERENCES jelovnik (id),
FOREIGN KEY (stavka_id) REFERENCES stavka (id)
);
```

**Atribut "id"** je tipa INT sa svojstvom AUTO_INCREMENT, a služi za automatsko povećavanje za svaki novi zapis te je definiran kao PRIMARY KEY jer je potrebno da se jedinstveno identificira svaki zapis u ovoj tablici.

**Atribut "created_at"** je tipa DATETIME sa zadanom vrijednosti CURRENT_TIMESTAMP, a pohranjuje datum i vrijeme kada je zapis stvoren. Zbog zadane vrijednosti automatski se postavlja trenutni datum i vrijeme pri stvaranju novog zapisa te ima ograničenje NOT NULL.

**Atribut "updated_at"** je tipa DATETIME sa zadanom vrijednosti ON UPDATE CURRENT_TIMESTAMP, a služi za pohranu trenutnog datuma i vremena svaki put kada se zapis izmijeni. Također ima ograničenje NOT NULL.

**Atribut "deleted_at"** je tipa DATETIME te pohranjuje datum i vrijeme kada je zapis izbrisan. Nema ograničenje NOT NULL jer ako zapis nije obrisan, ostaje prazno.

**Atribut "disabled"** je tipa BOOLEAN sa zadanom vrijednosti FALSE, a služi za označavanje je li veza između jelovnika i stavke onemogućena. Pošto je zadana vrijednost FALSE, veze se prikazuju kao aktivne, no ukoliko se stavi vrijednost TRUE veza bi bila onemogućena. Ovo polje ima ograničenje NOT NULL.

**Atribut "jelovnik_id"** je tipa INT te je definiran kao FOREIGN KEY koji referencira stupac id u tablici "jelovnik", a označava id jelovnika kojemu stavka pripada. Ima ograničenje NOT NULL.

**Atribut "stavka_id"** je tipa INT te je definiran kao FOREIGN KEY koji referencira stupac id u tablici "stavka", a označava id stavke koje pripada jelovniku. Također ima ograničenje NOT NULL.

## 6.5. Alma Reka

### 6.5.1. Tablica racun

**Tablica "racun"** pohranjuje informacije o financijskim transakcijama u restoranu.

```
CREATE TABLE racun (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    stol_id INT NOT NULL,
    restoran_id INT NOT NULL,
    zaposlenik_id INT NOT NULL,
    broj_racuna VARCHAR (31),
    napojnica DECIMAL (10, 2) DEFAULT 0 NOT NULL,
    iznos DECIMAL (10, 2) DEFAULT 0 NOT NULL,

    FOREIGN KEY (stol_id) REFERENCES stol (id),
    FOREIGN KEY (restoran_id) REFERENCES restoran (id),
    FOREIGN KEY (zaposlenik_id) REFERENCES zaposlenik (id)
);
```

**id** (INT AUTO_INCREMENT): Primarni ključ tablice. Jedinstveno identificira svaki račun.

**created_at** (DATETIME): Datum i vrijeme kada je račun stvoren. Postavljen na trenutačno vrijeme prilikom stvaranja.

**updated_at** (DATETIME): Datum i vrijeme zadnje izmjene računa. Ažurira se automatski na trenutačno vrijeme kad se račun izmijeni.

**deleted_at** (DATETIME): Datum i vrijeme kad je račun označen kao obrisan.

**disabled** (BOOLEAN): Logička vrijednost koja pokazuje je li račun deaktiviran.

**stol_id** (INT): Strani ključ koji referencira stol na kojem je račun otvoren.

**restoran_id** (INT): Strani ključ koji referencira restoran u kojem je račun izdan.

**zaposlenik_id** (INT): Strani ključ koji referencira zaposlenik koji je izdao račun.

**broj_racuna** (VARCHAR(31)): Jedinstveni broj računa.

**napojnica** (DECIMAL(10,2)): Iznos napojnice dodan na račun.

**iznos** (DECIMAL(10,2)): Ukupni iznos računa.

### 6.5.2. Tablica recept

**Tablica "recept"** pohranjuje recepte koje koriste restorani.

```
CREATE TABLE recept (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    naziv VARCHAR (256) NOT NULL,
    upute TEXT NOT NULL,

    FOREIGN KEY (restoran_id) REFERENCES restoran (id)
);
```

**id** (INT AUTO_INCREMENT): Primarni ključ tablice. Jedinstveno identificira svaki recept.

**created_at** (DATETIME): Datum i vrijeme kada je recept stvoren.

**updated_at** (DATETIME): Datum i vrijeme zadnje izmjene recepta.

**deleted_at** (DATETIME): Datum i vrijeme kad je recept označen kao obrisan.

**disabled** (BOOLEAN): Pokazuje je li recept deaktiviran.

**restoran_id** (INT): Strani ključ koji referencira restoran za koji je recept kreiran.

**naziv** (VARCHAR(256)): Naziv recepta.

**upute** (TEXT): Detaljne upute za pripremu jela.

### 6.5.3. Tablica stavka_racun

**Tablica "stavka_racun"** pohranjuje informacije o pojedinim stavkama koje se nalaze na računima.

```
CREATE TABLE stavka_racun (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    stavka_id INT NOT NULL,
    racun_id INT NOT NULL,
    kolicina TINYINT UNSIGNED DEFAULT 1 NOT NULL,

    FOREIGN KEY (stavka_id) REFERENCES stavka (id),
    FOREIGN KEY (racun_id) REFERENCES racun (id)
);
```

**id** (INT AUTO_INCREMENT): Primarni ključ tablice. Jedinstveno identificira svaku stavku na računu.

**created_at** (DATETIME): Datum i vrijeme kada je stavka dodana na račun.

**updated_at** (DATETIME): Datum i vrijeme zadnje izmjene stavke na računu.

**deleted_at** (DATETIME): Datum i vrijeme kada je stavka označena kao obrisana.

**disabled** (BOOLEAN): Logička vrijednost koja pokazuje je li stavka deaktivirana.

**stavka_id** (INT): Strani ključ koji referencira tablicu stavka iz koje stavka dolazi.

**racun_id** (INT): Strani ključ koji referencira tablicu racun kojem stavka pripada.

**kolicina** (TINYINT UNSIGNED): Količina stavke naručena na računu. Postavljeno na 1 kao zadana vrijednost.

### 6.5.4. Tablica recept_sastojak

**Tablica "recept_sastojak"** povezuje recepte s njihovim sastojcima.

```
CREATE TABLE recept_sastojak (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    recept_id INT NOT NULL,
    sastojak_id INT NOT NULL,
    kolicina DECIMAL (10, 2) NOT NULL DEFAULT 1,
    kolicina_tip ENUM ('g', 'mg', 'kg', 'l', 'ml', 'kol', 'tsp', 'tbsp'),

    FOREIGN KEY (recept_id) REFERENCES recept (id),
    FOREIGN KEY (sastojak_id) REFERENCES sastojak (id)
);
```

**id** (INT AUTO_INCREMENT): Primarni ključ tablice.

**created_at** (DATETIME): Datum i vrijeme kada je veza stvorena.

**updated_at** (DATETIME): Datum i vrijeme zadnje izmjene veze.

**deleted_at** (DATETIME): Datum i vrijeme kad je veza označena kao obrisan.

**disabled** (BOOLEAN): Pokazuje je li veza deaktivirana.

**recept_id** (INT): Strani ključ koji referencira recept.

**sastojak_id** (INT): Strani ključ koji referencira sastojak u receptu.

**kolicina** (DECIMAL(10,2)): Količina sastojka potrebna za recept.

**kolicina_tip** (ENUM('g', 'mg', 'kg', 'l', 'ml', 'kol', 'tsp', 'tbsp')): Tip mjere za količinu sastojka.

## 6.6. Patricia Lazić

### 6.6.1. Tablica mala_nezgoda

**Tablica „mala_nezgoda“** služi za bilježenje malih nezgoda to jest incidenata u restoranu, zajedno s povezanim podacima. Uključuje atribute: ID, created_at, updated_at, deleted_at, disabled, restoran_id, zaposlenik_id i ukupno.

```
CREATE TABLE mala_nezgoda (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    zaposlenik_id INT NOT NULL,
    ukupno DECIMAL (10, 2) NOT NULL,

    FOREIGN KEY (restoran_id) REFERENCES restoran (id),
    FOREIGN KEY (zaposlenik_id) REFERENCES zaposlenik (id)
);
```

**Atribut „ID“** primarni je ključ tablice, jedinstveni identifikator svake male nezgode. Atribut sadrži AUTO_INCREMENT svojstvo čime automatski generira jedinstvenu vrijednost za svaki redak umetnut u tablicu.

**Atribut „created_at** (DATETIME)“ automatski pohranjuje trenutni datum i vrijeme u trenutku kada se zapis male nezgode dodaje u tablicu. Pomoću funkcije CURRENT_TIMESTAMP vrijednost se definira prema trenutnom vremenu kada se novi redak dodaje u tablicu. Sadržava NOT NULL ograničenje koje osigurava da stupac ne može imati praznu vrijednost.

**Atribut „updated_at** (DATETIME)“ automatski pohranjuje datum i vrijeme zadnjeg ažuriranja zapisa male nezgode. U slučaju ažuriranja, updated_at automatski se osvježava s novim datumom i vremenom. DEFAULT CURRENT_TIMESTAMP omogućuje da prilikom prvog unosa retka, polje automatski dobiva trenutni datum i vrijeme kao početnu vrijednost. ON UPDATE CURRENT_TIMESTAMP omogućuje da svaki put kad se redak ažurira, vrijednost ovog polja automatski se osvježava na trenutno vrijeme. Atribut sadržava NOT NULL ograničenje.

**Atribut „deleted_at** (DATETIME)“ sadržava datum i vrijeme zapisa male nezgode koji je označen kao „obrisan“. Vrijednost ostaje NULL ako mala nezgoda nije izbrisana, a kada je izbrisana se popunjava s datumom i vremenom. deleted_at atribut je koristan u praćenju povijesti promjena malih nezgoda.

**Atribut „disabled“** djeluje kao logički indikator koji određuje je li zapis male nezgode aktivan ili disabled (onemogućen). BOOLEAN pohranjuje logičnu vrijednost, TRUE označuje da je zapis disabled, a FALSE da je aktivan. DEFAULT FALSE omogućuje da je zadana vrijednost atributa aktivna to jest FALSE.

**Atribut „restoran_id“** je tipa integer i namijenjen za držanje referenci na primarni ključ druge relacije putem ograničenja stranog ključa, u ovom slučaju referencira restoran u kojem se dogodila mala nezgoda. Sadržava NOT NULL ograničenje.

**Atribut „zaposlenik_id“** je tipa integer i strani je ključ koji referencira zaposlenika koji je odgovoran za malu nezgodu. Sadržava NOT NULL ograničenje.

**Atribut „ukupno“** označava iznos ukupne štete uzrokovanim malom nezgodom. Koristi DECIMAL(10, 2) tip podatka koji omogućuje preciznu pohranu decimalnih brojeva. Broj može sadržavati ukupno 10 znamenki i 2. znamenke su za decimalni dio. Sadržava NOT NULL ograničenje.

## 6.6.2. Tablica mala_nezgoda_stavka

**Tablica „mala_nezgoda_sastojak“** sadržava specifične sastojke povezane s malom nezgodom. Uključuje atribute: ID, created_at, updated_at, deleted_at, disabled, mala_nezgoda_id, sastojak_id i količina.

```
CREATE TABLE mala_nezgoda_sastojak (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    mala_nezogda_id INT NOT NULL,
    sastojak_id INT NOT NULL,
    kolicina INT UNSIGNED NOT NULL,


    FOREIGN KEY (mala_nezogda_id) REFERENCES mala_nezgoda (id),
    FOREIGN KEY (sastojak_id) REFERENCES sastojak (id)
);
```

**Atribut „ID“** je primarni ključ tablice i jedinstveni identifikator svakog sastojka male nezgode. Sadržava AUTO_INCREMENT svojstvo.

**Atribut „created_at (DATETIME)“** automatski pohranjuje trenutni datum i vrijeme u trenutku kada se sastojak male nezgode dodaje u tablicu. Atribut sadržava NOT NULL ograničenje.

**Atribut „updated_at (DATETIME)“** automatski se ažurira na trenutačno vrijeme kada se sastojak male nezgode izmijeni.

**Atribut „deleted at (DATETIME)“** označava datum i vrijeme zapisa sastojka male nezgode označen kao „izbrisan“.

**Atribut „disabled“** djeluje kao logička vrijednost koja određuje je li sastojak male nezgode aktivan ili disabled (onemogućen). Koristi boolean tip podatka.

**Atribut „mala_nezgoda_id“** je strani ključ koji povezuje sastojak s određenom malom nezgodom to jest referencira stupac id u tablici mala_nezgoda.

**Atribut „sastojak_id“** je strani ključ koji označava specifičan sastojak povezan s malom nezgodom. Pomoću ID-a dobijemo informacije o sastojcima iz relacije sastojak. Drugim riječima svakim unosom u tablicu mala_nezgoda_sastojak specificira se koji sastojak je povezan s malom nezgodom.

**Atribut „kolicina“** je tipa integer i sadržava količinu sastojka koji je prilikom male nezgode izgubljen ili oštećen.

### 6.6.3. Tablica velika_nezgoda

**Relacija „velika_nezgoda„** služi za bilježenje većih nezgoda to jest incidenata u restoranu. Sadrži atribute: ID, created_at, updated_at, deleted_at, disabled, restoran_id, zaposlenik_id i ukupno.

```
CREATE TABLE velika_nezgoda (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    restoran_id INT NOT NULL,
    zaposlenik_id INT NOT NULL,
    ukupno DECIMAL (10, 2) NOT NULL,

    FOREIGN KEY (restoran_id) REFERENCES restoran (id),
    FOREIGN KEY (zaposlenik_id) REFERENCES zaposlenik (id)
);
```

**Atribut „ID“** označuje primarni ključ tablice i jedinstveni identifikator svake velike nezgode.

**Atribut „created_at (DATETIME)“** automatski pohranjuje trenutni datum i vrijeme u trenutku kada se velika nezgoda dogodila.

**Atribut „updated at (DATETIME)“** automatski se ažurira na trenutačno vrijeme kada se velika nezgoda izmijeni.

**Atribut „deleted_at (DATETIME)“** označava datum i vrijeme zapisa velike nezgode označen kao „izbrisan“.

**Atribut „disabled“** djeluje kao logički indikator koji određuje je li velika nezgoda aktivna ili onemogućena. Koristi boolean tip podatka to jest TRUE ili FALSE.

**Atribut „restoran_id“** je strani ključ koji referencira restoran u kojemu se velika nezgoda dogodila. Ovaj atribut omogućuje povezivanje tablicu „velika_nezgoda“ s tablicom „restoran“ pomoću čega dobivamo osnovne informacije o restoranu i možemo odrediti u kojem restoranu se dogodila velika nezgoda.

**Atribut „zaposlenik_id“** je strani ključ koji označava koji je zaposlenik odgovoran za veliku nezgodu. Atribut povezuje tablicu „velika_nezgoda“ s tablicom „zaposlenik“ koji sadrži podatke o svim zaposlenicima.

**Atribut „ukupno“** označava iznos ukupne štete uzrokovano velikom nezgodom. Koristi DECIMAL tip podatka koji omogućuje preciznu pohranu decimalnih brojeva.

### 6.6.4. Tablica velika_nezgoda_stavka

**Relacija „velika_nezgoda_stavka“** bilježi specifične stavke povezane s velikom nezgodom. Sadrži atribute: ID, created_at, updated_at, deleted_at, disabled, velika_nezgoda_id, stavka_id, količina.

```
CREATE TABLE velika_nezgoda_stavka (
    id INT AUTO_INCREMENT PRIMARY KEY,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT
NULL,
    deleted_at DATETIME,
    disabled BOOLEAN DEFAULT FALSE NOT NULL,

    velika_nezgoda_id INT NOT NULL,
    stavka_id INT NOT NULL,
    kolicina INT UNSIGNED NOT NULL,


    FOREIGN KEY (velika_nezgoda_id) REFERENCES velika_nezgoda (id),
    FOREIGN KEY (stavka_id) REFERENCES stavka (id)
);
```

**Relacija „velika_nezgoda_stavka“** bilježi specifične stavke povezane s velikom nezgodom. Sadrži atribute: ID, created_at, updated_at, deleted_at, disabled, velika_nezgoda_id, stavka_id, količina.

**Atribut „ID“** označuje primarni ključ tablice i jedinstveni je identifikator svake stavke velike nezgode.

**Atribut „created_at (DATETIME)“** automatski pohranjuje trenutni datum i vrijeme u trenutku kada je stavka prvi put unesena u relaciju.

**Atribut „updated_at (DATETIME)“** bilježi datum i vrijeme posljednjeg ažuriranja stavke velike nezgode u relaciji.

**Atribut „deleted_at (DATETIME)“** označava datum i vrijeme stavke velike nezgode označene kao „obrisano“.

**Atribut „disabled“** djeluje kao logički indikator koji određuje je li stavka velika nezgode active ili disabled.

**Atribut „velika_nezgoda_id“** je strani ključ koji povezuje tablicu „velika_nezgoda_stavka“ s tablicom „velika_nezgoda“ na temelju ID-a. Označava kojoj velikoj nezgodi pripadaju stavke iz tablice „velika_nezgoda_stavka“ čime pomaže u lakšoj analizi detalja štete velike nezgode.

**Atribut „stavka_id“** je strani ključ koji povezuje tablicu velika_nezgoda_stavka s tablicom stavka i time omogućuje praćenje stavki uključenih u veliku nezgodu.

**Atribut „količina“** tipa je integer i označava količinu stavke koja je izgubljena ili oštećena radi velike nezgode.

# 7. Upiti

# 7.1. Paula Vorih

## 7.1.1. Upit - Prikaz svih zaposlenika s iznosom njihovih mjesečnih plaća u 1. i 6. mjesecu

Iz tablice zaposlenik koja je join-ana s tablicom zaposlenik_placa tako da zaposlenik.id bude jednak kao zaposlenik_placa.zaposlenik.id, odabiremo stupce zaposlenik.ime, zaposlenik.prezime, zaposlenik_placa.iznos preimenovan u “mjesecna_placa” i zaposlenik_placa.mjesec. Na kraju postavljamo uvjet da MONTH u zaposlenik_placa.mjesec bude unutar skupa vrijednosti (1, 6).

```
SELECT zaposlenik.ime,
zaposlenik.prezime,
zaposlenik_placa.iznos AS mjesecna_placa,
zaposlenik_placa.mjesec
FROM zaposlenik
JOIN zaposlenik_placa ON zaposlenik.id = zaposlenik_placa.zaposlenik_id
WHERE MONTH(zaposlenik_placa.mjesec) IN (1, 6);
```

## 7.1.2. Upit - Prikaz restorana koji ima najmanje novca na računu

Iz tablice restoran koja je join-ana s tablicom restoran_racun tako da restoran.id bude jednak restoran_racun.restoran_id, odabiremo stupce restoran.naziv kao “restoran”, restoran_racun.stanje kao “iznos_na_racunu” i restoran_racun.valuta. Nakon toga podatke u restoran_racun.stanje poredajemo uzlazno i limitiramo broj izlaznih rezultata na 1.

```
SELECT restoran.naziv AS restoran,
restoran_racun.stanje AS iznos_na_racunu,
restoran_racun.valuta
FROM restoran
JOIN restoran_racun ON restoran.id = restoran_racun.restoran_id
ORDER BY restoran_racun.stanje ASC
LIMIT 1;
```

## 7.1.3. Upit - Prikazujemo broj zaposlenih po restoranima i ukupan iznos izdanih mjesečnih plaća

Iz tablice restoran koja je lijevim outer join-om povezana s tablicom zaposlenik na temelju da je restoran.id jednak zaposlenik.restoran_id, koja je također lijevim outer join-om povezana s tablicom zaposlenik_placa na temelju da je zaposlenik.id jednak zaposlenik_placa.zaposlenik_id, odabiremo stupce restoran.nativ preimenovan kao “restoran”, brojimo distinktne zaposlenik id-eve i to preimenujemo u “broj_zaposlenika” te sumiramo sve iznose zaposleničkih plaća i preimenujemo u “ukupne_place”. Na kraju se sve grupira po nazivu restorana kako bi u rezultatu dobili ime resotrana, broj zaposlenika navedenog restorana i ukupni iznos isplaćenih plaća u tom restoranu.

```
SELECT restoran.naziv AS restoran,
       COUNT(DISTINCT zaposlenik.id) AS broj_zaposlenika,
       SUM(zaposlenik_placa.iznos) AS ukupne_place
FROM restoran
LEFT JOIN zaposlenik ON restoran.id = zaposlenik.restoran_id
LEFT JOIN zaposlenik_placa ON zaposlenik.id = zaposlenik_placa.zaposlenik_id
GROUP BY restoran.naziv;
```

## 7.2. Dinko Nađ

### 7.2.1. Upit - računa trošak svakoga restorana na mjesečnoj bazi i nemjesečnoj bazi te računa ukupan trošak i slaže ih po veličini počevši od restorana koji je naviše potrošio

```
SELECT restoran.naziv,
       SUM(CASE WHEN mjesecno = TRUE THEN iznos ELSE 0 END) AS mjesecni_trosak,
       SUM(CASE WHEN mjesecno = FALSE THEN iznos ELSE 0 END) AS
nemjesecni_trosak,
       SUM(iznos) AS ukupni_trosak
FROM trosak join restoran on restoran_id=restoran.id
GROUP BY restoran.naziv
ORDER BY ukupni_trosak DESC;
```

U FROM dijelu izabiremo tablicu na kojoj ćemo raditi obradu podataka. U ovom slučaju to je tablica „trosak“ i pridružujemo JOIN operacijom tablicu „restoran“ putem stranog ključa „restoran_id“, koji povezuje te dvije tablice na temelju navedenog uvjeta(restoran_id=restoran.id). Zatim operacijom GROUP BY grupiramo sve retke iz tablice „trosak“ prema „restoran.naziv“. Nakon grupiranja slijedi operacija SELECT koja ispisuje „restoran.naziv“, te se određuje koje stupce i izraze treba izračunati za svaku grupu. U prvom dijelu upita izračunava se zbroj „iznosa“ za sve retke unutar svake grupe gdje je „mjesecno“ TRUE. Ako uvjet nije ispunjen, rezultat za taj redak je 0, na kraju se dodaje naziv stupca „mjesecni_troskovi“ pomoću naredbe AS. U drugom dijelu izračunava zbroj „iznos“ za sve retke unutar svake grupe gdje je „mjesecno“ FALSE. Ako uvjet nije zadovoljen, rezultat za taj redak je 0, na kraju dodaje naziv stupca „nemjesecni_troskovi“ pomoću naredbe AS. U trećem dijelu se izračunava ukupan zbroj „iznos“ za sve retke unutar svake grupe. Na kraju, operacijom ORDER BY sortiraju se rezultati prema „ukupni_trosak“ u opadajućem redosljedu, tako da restorani sa najvećim ukupnim troškovima budu prikazani prvi.

### 7.2.2. Upit - Pronalazi najskuplju stavku u svakom restoranu i zatim sortira te stavke prema cijeni u opadajućem rasporedu

```
SELECT restoran.naziv AS naziv_restorana,
       stavka.naziv AS najskuplja_stavka,
       stavka.cijena AS najvisa_cijena
FROM stavka
    JOIN restoran ON stavka.restoran_id = restoran.id
WHERE (stavka.restoran_id, stavka.cijena) IN (
    SELECT restoran_id, MAX(cijena)
    FROM stavka
    GROUP BY restoran_id
)
ORDER BY najvisa_cijena DESC;
```

U FROM dijelu upita izabiremo tablicu stavka, koja sadrži podatke o stavkama u restoranima. Zatim pridružujemo JOIN operacijom tablicu restoran putem stranog ključa restoran_id, koji povezuje te dvije tablice na temelju uvjeta stavka.restoran_id = restoran.id. Na taj način povezujemo stavke s odgovarajućim restoranima. U WHERE dijelu upita koristimo podupit koji se koristi za pronalaženje najskuplje stavke u svakom restoranu. U podupitu koristimo SELECT restoran_id, MAX(cijena), koji pronalazi najvišu cijenu za svaki restoran. Podupit grupira podatke prema restoran_id pomoću GROUP BY, kako bi za svaki restoran dobio maksimalnu cijenu. Zatim, u glavnom upitu, pomoću IN operacije, filtriramo samo one stavke koje imaju cijenu jednaku najvišoj cijeni za svaki restoran, što nam omogućuje da dobijemo najskuplju stavku u svakom restoranu. U SELECT dijelu upita, uzimamo naziv restorana iz tablice restoran i naziv najskuplje stavke te njezinu cijenu. Na kraju, ORDER BY operacija sortira rezultate prema cijeni stavke u opadajućem redoslijedu , tako da restorani s najskupljim stavkama budu prikazani prvi.

### 7.2.3. Upit - Prikaz najčešće korištenih sastojaka u svim skladištima

```
SELECT naziv, SUM(trenutna_kolicina) AS ukupna_kolicina
    FROM sastojak
        GROUP BY naziv
            ORDER BY ukupna_kolicina DESC;
```

U FROM dijelu izabiremo tablicu na kojoj ćemo raditi obradu podataka. U ovom slučaju to je tablica „sastojak“. Zatim se izvršava GROUP BY koji grupira sve retke prema stupcu „naziv“, što znači da svi redci s istim nazivom sastojka grupiraju se zajedno. Zatim pomoću naredbe SELECT odabirem koje stupce treba izračunati i prikazati u konačnom rezultatu. U konačnom rezultatu će se prikazivati „naziv“ i suma trenutnih količina u svim skladištima pomoću naredbe SUM(trenutna_kolicina). Nakon što su se svi izrazi izračunali, pomoću naredbe ORDER BY sortirali smo rezultate prema „ukupna_kolicina“ u opadajućem redoslijedu. Sastojci sa najvećom ukupnom količinom bit će prikazani na vrhu.

## 7.3. Luka Vilagoš

Upiti su navedeni u Flask aplikaciji.

## 7.4. Sara Maljić

### 7.4.1. Upit - Dohvaćanje broja rezervacija po vrsti lokacije stola za svaki restoran

Ovaj upit služi za dohvaćanje broj rezervacija za svaku vrstu lokacije stola unutar pojedinog restorana.

```
SELECT s.restoran_id, s.lokacija, COUNT(r.id) AS broj_rezervacija
FROM rezervacija r
JOIN stol s ON r.stol_id = s.id
GROUP BY s.restoran_id, s.lokacija
ORDER BY s.restoran_id, s.lokacija;
```

Za ovaj upit koristimo naredbu *SELECT* kako bismo odabrali stupce *s.restoran_id* i *s.lokacija*, a pomoću naredbe *COUNT(r.id)* prebrojavamo rezervacije koje se odnose na svaku kombinaciju restorana i lokacije stola.

S *FROM rezervacija r* definiramo početnu tablicu iz koje dohvaćamo podatke za upit te joj dajemo nadimak *r*. Taj nadimak omogućuje da se tablica u ostatku upita poziva tim nadimkom.

Pomoću JOIN-a povezujemo tablice "rezervacija" i "stol" na temelju zajedničkog stupca *stol_id*, čemu služi *s.id*. On je primarni ključ tablice stol, što znači da jedinstveno identificira svaki red u toj tablici. *r.stol_id* je vanjski ključ u tablici "rezervacija" koji pokazuje na primarni ključ *s.id* u tablici "stol". Dakle, *s.id* služi kao jedinstveni identifikator stola koji se koristi u naredbi JOIN koja povezuje zapise iz tablica "rezervacija" i "stol" putem zajedničkog odnosa između stupca *r.stol_id* i *s.id*.

Naredba GROUP BY se koristi za grupiranje rezultata prema specifičnim stupcima, a u ovom slučaju su to kombinacija stupaca *s.restoran_id* i *s.lokacija*. Zbog toga za svaki jedinstveni par iz te kombinacije (restoran_id i lokacija) izračunava se broj rezervacija pomoću naredbe COUNT, a GROUP BY osigurava da se rezultati grupiraju po navedenim kombinacijama.

Naredba ORDER BY služi za sortiranje rezultata po jednom ili više stupaca nakon izvedbe GROUP BY. U našem slučaju prvo su rezultati sortirani prema stupcu *s.restoran_id*, a potom unutar svakog *restoran_id* prema stupcu *s.lokacija*.

Kao rezultat dobivamo stupce id restorana, lokaciju i broj rezervacija.

### 7.4.2. Upit - Prikazivanje naziva svih jelovnika i broj stavki na svakom od njih

Ovaj upit koristimo za dohvaćanje naziva svih jelovnika i broja stavki koje pripadaju svakom od njih.

```
SELECT j.naziv AS naziv_jelovnika, COUNT(js.stavka_id) AS broj_stavki
FROM jelovnik j
LEFT JOIN jelovnik_stavka js ON j.id = js.jelovnik_id
GROUP BY j.id
ORDER BY broj_stavki DESC;
```

Prvo koristimo naredbu *SELECT* za dohvaćanje naziva jelovnika iz stupca *j.naziv*, koji je označen nadimkom *naziv_jelovnika*, i broj stavki na svakom jelovniku pomoću funkcije *COUNT(js.stavka_id)*, koja je označena nadimkom *broj_stavki*, a označava broj stavki povezanih s određenim jelovnikom.

Naredba *FROM jelovnik j* određuje tablicu *jelovnik* kao početnu tablicu iz koje se dohvaćaju podaci i dodjeljuje joj se nadimak *j*.

Pomoću *LEFT JOIN* povezujU se tablice *jelovnik* i *jelovnik_stavka* kojoj dajemo nadimak js. To nam omogućuje dohvaćanje svih redaka iz tablice *jelovnik*, čak i ako ne postoje odgovarajući zapisi u tablici *jelovnik_stavka*. Povezujemo ih putem zajedničkog odnosa između stupca *j.id*, koji je primarni ključ tablice *jelovnik*, i stupca *js.jelovnik_id* koji je vanjski ključ u tablici "jelovnik stavka".

*GROUP BY* grupira rezultate prema stupcu *j.id*, dakle izračun funkcije *COUNT(js.stavka_id)* se primjenjuje za svaki jedinstveni jelovnik. Zbog korištenja *LEFT JOIN*, ukoliko neki jelovnik nema stavki funkcija *COUNT* će za taj zapis vratiti vrijednost 0.

*ORDER BY* sortira rezultate prema broju stavki iz stupca *broj_stavki* po silaznom poretku, *DESC*. Zbog toga jelovnici s najvećim brojem stavki biti će prikazani prvi.

Kao rezultat dobivamo popis naziva svih jelovnika i broj stavki povezanih s njima, sortiran od jelovnika s najvećim prema jelovnika s najmanjim brojem stavki.

## 7.4.3. Upit - Dohvaćanje svih stavki koje su dodane na jelovnik i koje pripadaju kategoriji "jela"

Ovaj upit koristimo za dohvaćanje svih stavki koje su dodane na jelovnik i pripadaju kategoriji *jela*.

```
SELECT st.naziv AS naziv_stavke, st.cijena AS cijena, j.naziv AS naziv_jelovnika
 FROM stavka st
 JOIN jelovnik_stavka js ON st.id = js.stavka_id
 JOIN jelovnik j ON js.jelovnik_id = j.id
 WHERE j.jelovnik_tip = 'jela'
 ORDER BY st.cijena DESC;
```

Naredbom *SELECT* dohvaćaju se naziv stavke iz stupca *st.naziv*, kojem se daje alias *naziv_stavke*, cijena stavke iz stupca *st.cijena*, kojem se daje alias *cijena*, te naziv jelovnika iz stupca *j.naziv*, kojem se daje alias *naziv_jelovnika*.

Podaci se preuzimaju iz tri tablice: *stavka*, *jelovnik_stavka* i *jelovnik*. Početna tablica je *stavka*, kojoj je dodijeljen alias *st*, a povezivanje s ostalim tablicama vrši se pomoću dviju naredbi *JOIN*. Prvo se tablica *stavka* povezuje s tablicom *jelovnik_stavka*, kojoj je dodijeljen alias *js*, koristeći uvjet *st.id = js.stavka_id*. Ovo omogućuje povezivanje svake stavke s odgovarajućim zapisima u tablici *jelovnik_stavka* putem primarnog ključa tablice *stavka* (*st.id*) i vanjskog ključa tablice *jelovnik_stavka* (*js.stavka_id*). Zatim se tablica *jelovnik_stavka* povezuje s tablicom *jelovnik*, kojoj je dodijeljen alias *j*, koristeći uvjet *js.jelovnik_id = j.id*. Ovaj uvjet povezuje svaki zapis u tablici *jelovnik_stavka* s

odgovarajućim jelovnikom putem vanjskog ključa tablice *jelovnik_stavka* (*js.jelovnik_id*) i primarnog ključa tablice *jelovnik* (*j.id*).

Filtriranje podataka provodi se pomoću uvjeta *WHERE j.jelovnik_tip = 'jela'*, kojim se odabiru samo oni jelovnici čiji je tip označen kao "jela". Na kraju, rezultati se sortiraju naredbom *ORDER BY st.cijena DESC*, čime se stavke prikazuju u silaznom redoslijedu prema njihovoj cijeni, s najskupljim stavkama na vrhu.

Rezultat upita uključuje naziv, cijenu i pripadajući jelovnik za sve stavke koje su kategorizirane kao dio jelovnika tipa "jela", organizirane od najskuplje prema najjeftinijoj.

## 7.5. Alma Reka

### 7.5.1. Upit - Ispiši ukupni prihod ostvaren od svakog recepta

```
SELECT
    r.naziv AS Recept,
    SUM(s.cijena * sr.kolicina) AS UkupniPrihod
FROM
    recept r
JOIN
    recept_sastojak rs ON r.id = rs.recept_id
JOIN
    sastojak s ON rs.sastojak_id = s.id
JOIN
    stavka_racun sr ON s.id = sr.stavka_id
GROUP BY
    r.naziv;
```

Ovaj upit prvo povezuje tablicu recept s recept_sastojak kako bi se dobila veza između recepta i njegovih sastojaka. Zatim se povezuje s tablicom sastojak da se dobiju pojedinosti o svakom sastojku, uključujući cijenu. Konačno, sastojci su povezani s stavka_racun što omogućava pristup informacijama o količinama koje su se koristile u transakcijama. Korištenje SUM funkcije s množenjem cijene sastojka i količine iz računa omogućava izračun ukupnog prihoda po receptu.

### 7.5.2. Upit - Ispiši ukupnu količinu svakog sastojka koja se koristi u receptima, poredano od najviše korištene količine prema nižoj

```
SELECT
    s.naziv AS Sastojak,
    SUM(rs.kolicina) AS UkupnaKolicina
FROM
    recept_sastojak rs
JOIN
    sastojak s ON rs.sastojak_id = s.id
GROUP BY
    s.naziv
ORDER BY
    UkupnaKolicina DESC;
```

U ovom upitu koristimo recept_sastojak za povezivanje sastojaka s receptima u kojima se koriste. Sumiranjem količina svakog sastojka dobijemo ukupnu količinu koja se koristi za sve recepte. Rezultati su grupirani po nazivu sastojka i poredani od najveće prema najmanjoj korištenoj količini, što nam pokazuje koje su sastojke najčešće korišteni.

### 7.5.3. Upit - Ispiši prihod svakog recepta za specifičan račun

```
SELECT
    r.naziv AS Recept,
    SUM(s.cijena * sr.kolicina) AS Prihod
FROM
    recept r
JOIN
    recept_sastojak rs ON r.id = rs.recept_id
JOIN
    sastojak s ON rs.sastojak_id = s.id
JOIN
    stavka_racun sr ON rs.sastojak_id = sr.stavka_id
WHERE
    sr.racun_id = 1
GROUP BY
    r.naziv;
```

Ovdje se koristi slična logika kao u prvom upitu, ali s dodatnim uvjetom filtriranja (WHERE sr.racun_id = 1) koji se odnosi na specifičan račun. To nam omogućuje da vidimo koliki je prihod generiran po svakom receptu unutar odabranog računa. Opet se koristi grupiranje po nazivu recepta kako bi se prihodi sumirali za svaki recept posebno.

## 7.5.4. Transakcija - NovaRacunTransakcija

Transakcija za stvaranje novog računa s pripadajućim stavkama u sustavu restorana.

```
DROP PROCEDURE IF EXISTS NovaRacunTransakcija;

DELIMITER $$
CREATE PROCEDURE NovaRacunTransakcija (
    IN restoranID INT,
    IN zaposlenikID INT,
    IN napojnica DECIMAL(10, 2),
    IN stolID INT,
    IN stavke JSON
)
BEGIN
    DECLARE racunID INT;
    DECLARE stavkaID INT;
    DECLARE kolicina INT;
    DECLARE iterator INT DEFAULT 0;

    START TRANSACTION;

    INSERT INTO racun (restoran_id, zaposlenik_id, napojnica, stol_id, iznos)
    VALUES (restoranID, zaposlenikID, napojnica, stolID, 0);

    SET racunID = LAST_INSERT_ID();

    WHILE iterator < JSON_LENGTH(stavke) DO
        SET stavkaID = JSON_UNQUOTE(JSON_EXTRACT(stavke, CONCAT('$[', iterator,
'].stavka_id')));
        SET kolicina = JSON_UNQUOTE(JSON_EXTRACT(stavke, CONCAT('$[', iterator,
'].kolicina')));

        -- Ispravljen naziv tablice
        INSERT INTO stavka_racun (racun_id, stavka_id, kolicina)
        VALUES (racunID, stavkaID, kolicina);

        SET iterator = iterator + 1;
    END WHILE;

    COMMIT;
END$$
DELIMITER ;

SET @stavke = '[
    {"stavka_id": 1, "kolicina": 2},
    {"stavka_id": 2, "kolicina": 1}
]';

CALL NovaRacunTransakcija(1, 2, 15.00, 1, @stavke);
```

Ova procedura NovaRacunTransakcija omogućuje stvaranje novog računa s detaljima o restoranu, zaposleniku, napojnici, stolu i stavkama računa koje su definirane u JSON formatu. Transakcija osigurava da se svi koraci izvrše kao jedna atomična operacija, čime se povećava pouzdanost i smanjuje rizik od nekonzistentnosti podataka uslijed potencijalnih pogrešaka ili prekida.

**Paremetri procedure**

1. **restoranID (INT)** - Identifikator restorana.
2. **zaposlenikID (INT)** - Identifikator zaposlenika koji je kreirao račun.
3. **napojnica (DECIMAL(10, 2))** - Iznos napojnice.
4. **stolID (INT)** - Identifikator stola na kojem je račun kreiran. 5. **stavke (JSON)** - Lista stavki u JSON formatu s atributima:

   a. stavka_id: Identifikator stavke.

   b. kolicina: Količina stavke.

   **Detaljan rad procedure:**

   1. **Početak Transakcije:** Osigurava da se sve ili nijedna operacija neće izvršiti.
   2. **Insert računa:** Unosi osnovne informacije o računu u tablicu racun.
   3. **Postavljanje racunID:** Sprema ID novostvorenog računa za daljnje reference.
   4. **Iteriranje kroz stavke:** Koristi JSON objekt za iteriranje kroz stavke koje su zadane, izvlačeći ID i količinu svake stavke.
   5. **Unos stavki računa:** Za svaku stavku unosi povezanu količinu u tablicu stavka_racun.
   6. **Commit Transakcije:** Potvrđuje sve izmjene u bazi podataka ako nema grešaka.

```
UPDATE racun  SET iznos = ukupna_vrijednost  WHERE id = racunID;
 COMMIT;
```

**Ažuriranje iznosa računa:** Dio koda koji postavlja vrijednost stupca iznos u tablici racun na temelju kalkulirane vrijednosti ukupna_vrijednost. Ovaj dodatak osigurava da se ukupan iznos računa reflektira nakon unosa svih povezanih stavki.

```
SET @stavke = '[    {"stavka_id": 1, "kolicina": 2},    {"stavka_id": 2,
"kolicina": 1} ]';
```

Ovo postavlja primjer JSON niza koji sadrži informacije o stavkama koje će biti dodane na novi račun, koristeći ih kao primjer za testiranje i demonstraciju funkcionalnosti procedure.

```
CALL NovaRacunTransakcija(1, 2, 15.00, 1, @stavke);
```

Ova naredba izvršava proceduru NovaRacunTransakcija s određenim parametrima za stvaranje novog računa. Demonstrira kako korisnik može jednostavno kreirati novi račun s definiranim stavkama.

# 7.6. Patricia Lazić

## 7.6.1. Upit - Prikaz ukupnog iznosa malih nezgoda po restoranu

```
SELECT
     mn.restoran_id,
     r.naziv AS restoran_naziv,
        SUM(mn.ukupno) AS ukupno_mala_nezgoda
FROM
    mala_nezgoda mn
JOIN
        restoran r ON mn.restoran_id = r.id
GROUP BY
     mn.restoran_id, r.naziv;
```

Ovaj upit prikazuje podatke o ukupnom iznosu iz tablice mala_nezgoda grupirane po restoranima, zajedno s nazivima restorana.

Odabiremo mn.restoran_id što je ID restorana iz tablice mala_nezgoda, naziv restorana iz tablice restoran (r.naziv) i koristimo agregacijsku funkciju SUM(mn.ukupno) koja računa ukupni trošak malih nezgoda za svaki restoran.

Tablicu mala_nezgoda povezujemo sa tablicom restoran koristeći strani ključ restoran_id i grupiramo po mn.restoran_id i r.naziv što znači da će sve male nezgode za isti restoran biti grupirane zajedno. U rezultatu dobivamo 3 stupca: ID restorana, naziv restorana i ukupni trošak male nezgode. Pomoću ovog upita pratimo lakše troškove malih nezgoda.

## 7.6.2. Upit - Popis sastojaka korištenih u "malim nezgodama" s ukupnim količinama

```
SELECT
    s.naziv AS sastojak_naziv,
    SUM(mns.kolicina) AS ukupna_kolicina,
    SUM(mns.kolicina * s.cijena) AS ukupni_trosak
FROM
    mala_nezgoda mn
JOIN
    mala_nezgoda_sastojak mns ON mn.id = mns.mala_nezgoda_id
JOIN
    sastojak s ON mns.sastojak_id = s.id
WHERE
    mn.deleted_at IS NULL
GROUP BY
    s.id, s.naziv
ORDER BY
    ukupna_kolicina DESC, ukupni_trosak DESC;
```

Odaberemo naziv sastojka i koristimo funkciju SUM(mns.količina) koja zbraja ukupnu količinu sastojaka iz svih malih nezgoda i SUM(mns.količina * s.cijena) koja računa ukupni trošak za sastojak. Tablicu mala_nezgoda smo povezali s tablicom mala_nezgoda_sastojak i zatim povezali s tablicom sastojak na temelju ID-a i time spajamo male nezgode s korištenim sastojcima i njihovim detaljima. „mn.deleted_at IS NULL“ isključuje male nezgode koje su označene kao izbrisane, zatim grupiramo rezultate prema svakom sastojku (s.id, s.naziv) i sortiramo rezultate prema ukupnoj količini i ukupnom trošku DESC. U rezultatu dobivamo 3 stupca: naziv sastojka, ukupna količina i ukupni trošak Ovaj upit nam omogućuje lakši uvid u resurse koji su potrošeni ili izgubljeni prilikom male nezgode.

## 7.6.3. Upit - Prikaz ukupne štete po restoranu za sve nezgode (male i velike)

```
SELECT
    restoran_id,
    SUM(ukupno) AS ukupna_steta
FROM (
    SELECT restoran_id, ukupno FROM mala_nezgoda
    UNION ALL
    SELECT restoran_id, ukupno FROM velika_nezgoda
) AS sve_nezgode
GROUP BY restoran_id
ORDER BY ukupna_steta DESC;
```

Unutar podupita spajamo podatke male i velike nezgode. Prvo izvlačimo podatke iz male nezgode to jest ID restorana gdje se mala nezgoda dogodila i povezujemo rezultate s velikom nezgodom koristeći UNION ALL koji omogućuje kombinaciju rezultata iz obje tablice bez uklanjanja duplikata. Rezultat ovog podupita je tablica s dva stupca: ID restorana i ukupno koju spajamo u privremenu tablicu s nazivom sve_nezgode. Koristimo agregacijsku funkciju SUM koja zbraja štete svih nezgoda za svaki restoran i grupiramo podatke po ID-u restorana. Sortiramo po ukupnoj šteti DESC što znači da je najveća šteta na vrhu. U rezultatu dobivamo tablicu s 2 stupca: ID restorana i ukupna šteta velikih i malih nezgoda. Ova tablica prikazuje ukupnu štetu po restoranima sortiranu od najveće prema najmanjoj.

## 7.6.4. Upit - Prikaz svih velikih nezgoda s detaljima o stavkama, uključujući količine i cijene stavki detaljno

```
SELECT
    vn.id AS velika_nezgoda_id,
    vn.restoran_id,
    vn.zaposlenik_id,
    vn.ukupno AS ukupna_steta,
    vs.id AS stavka_id,
    vs.naziv AS stavka_naziv,
    vns.kolicina,
    vs.cijena,
    (vns.kolicina * vs.cijena) AS ukupna_cijena
FROM
    velika_nezgoda vn
JOIN
```

```
    velika_nezgoda_stavka vns ON vn.id = vns.velika_nezgoda_id
JOIN
    stavka vs ON vns.stavka_id = vs.id
ORDER BY
    velika_nezgoda_id, stavka_id;
```

Odaberemo osnovne podatke iz velike nezgode kao što su ID velike nezgode, ID restorana i zaposlenika i ukupne štete. Koristeći JOIN povezujemo s tablicom velika_nezgoda_stavka i s tablicom stavka na temelju ID-a. Time prikupljamo podatke o ID-u stavke koja je povezana s velikom nezgodom kao i količina oštećene stavke i naziv i cijene stavke. Množimo količinu oštećene stavke (vns.količina) s cijenom pojedinačne stavke (vs.cijena) i time dobivamo ukupnu cijenu svake stavke. Rezultat sortiramo prema ID-u velike nezgode i ID-u svake stavke što u konačnici prikazuje sve velike nezgode i detalje o stavkama. Ovaj upit omogućava detaljan uvid u troškove koji su povezani s pojedinačnim stavkama.

## 7.6.5. Transakcija za ažuriranje količine sastojaka u maloj nezgodi

```
START TRANSACTION;

UPDATE mala_nezgoda_sastojak
SET kolicina = 4
WHERE mala_nezgoda_id = 2 AND sastojak_id = 3;

UPDATE mala_nezgoda
SET ukupno = (SELECT SUM(s.kolicina * sastojak.cijena)
              FROM mala_nezgoda_sastojak s
              JOIN sastojak ON s.sastojak_id = sastojak.id
              WHERE s.mala_nezgoda_id = 2)
WHERE id = 2;

COMMIT;
```

Pokrećemo transakciju koristeći naredbu START TRANSACTION koja osigurava da su svi koraci u transakciji izvršeni kao jedinstvena cjelina što znači da ako neki korak ne uspije, svi izvršeni koraci bit će vračani. UPDATE mijenja podatke u tablici mala_nezgoda_sastojak i pomoću naredbe SET određujemo novu vrijednost količine sastojaka na 4. WHERE mala_nezgoda_id = 2 AND sastojak_id = 3; je uvjet koji filtrira prema vrijednosti mala_nezgoda_id pa će se ažurirati samo oni redci koji imaju mala_nezgoda_id jednaka 2 kao i prema sastojak_id gdje će se ažurirati samo redci kojima je sastojak_id jednak 3. AND je logički operator koji kombinira dva uvjeta. Nakon ovog koraka količina sastojka s ID = 3 u nezgodi ID = 2 je 4. UPDATE mijenja podatke u tablici mala_nezgoda i SET postavlja novi iznos u stupac ukupno na temelju izračuna podupita. Unutar podupita koristimo agregacijsku funkciju SUM(s.kolicina * sastojak.cijena) koja računa ukupnu cijenu svih sastojaka u maloj nezgodi s ID = 2 (kojeg smo postavili pomoću WHERE uvjeta), tako da množimo količinu svakog sastojka s njegovom cijenom i  zatim zbrajamo te vrijednosti. Povezujemo tablice mala_nezgoda_sastojak s tablicom sastojak na temelju ID-a kako bismo dobili cijenu svakog sastojka. Ovim korakom će ukupni iznos nezgode s ID = 2 biti ažuriran u tablici mala_nezgoda prema novim cijenama i količinama sastojaka. COMMIT završava transakciju i potvrđuje sve promjene koje su napravljene unutar transakcije što znači da će promjene biti trajno pohranjene u

bazi podataka. Ako se neki korak ne uspije izvršiti, koristi se ROLLBACK koji vraća sve promjene u početno stanje. Ova transakcija omogućuje da se ukupan iznos male nezgode automatski prilagođava i također osigurava očuvanje točnosti podataka, čak i ako dođe do greške tijekom izvršenja upita.

# 8. Pogledi

## 8.1. Paula Vorih

### 8.1.1. Pogled - Prikazivanje prosječnih iznosa izdanih plaća po restoranima

```
CREATE VIEW ProsjecnePlaceRadnikaPoRestoranu AS
SELECT
    restoran.id AS restoran_id,
    restoran.naziv,
    AVG(zaposlenik_placa.iznos) AS prosjecna_placa
FROM restoran
JOIN zaposlenik ON zaposlenik.restoran_id = restoran.id
JOIN zaposlenik_placa ON zaposlenik.id = zaposlenik_placa.zaposlenik_id
GROUP BY restoran.id;
```

Kreiramo pogled ProsjecnePlaceRadnikaPoRestoranu u kojemu odabiremo iz tablice restoran koja je join-ana s tablicom zaposlenik na temelju da je zaposlenik.restoran_id jednak restoran.id i koja je također join-ana s tablicom zaposlenik_placa na temelju da je zaposlenik.id jednak zaposlenik_placa.zaposlenik_id, odabiremo stupce restoran.id preimenovan u “restoran_id”, restoran.naziv i prosječnu vrijednost iz zaposlenik_placa.iznos kojoj dajemo alias prosjecna_placa. Na kraju se sve grupira po id-u restorana kako bi u rezultatu id-eve, imena i prosječne iznose plaća svih restorana koji isplaćuju plaće svojim radnicima.

### 8.1.2. Pogled - Prikazivanje svih plaća po zaposlenicima

```
CREATE VIEW SvePlace AS
SELECT zaposlenik.ime,
       zaposlenik.prezime,
       zaposlenik_placa.iznos AS placa,
       zaposlenik_placa.mjesec,
       restoran.naziv AS restoran
FROM zaposlenik
JOIN zaposlenik_placa ON zaposlenik.id = zaposlenik_placa.zaposlenik_id
JOIN restoran ON zaposlenik.restoran_id = restoran.id;
```

Kreiramo pogled SvePlace u kojemu iz tablice zaposlenik koja je join-ana sa zaposlenik_placa na temelju da su zaposlenik.id i zaposlenik_placa.zaposlenik_id jednaki, koja je također join-ana s tablicom restoran na temelju da su zaposlenik.restoran_id i restoran.id jednaki, odabiremo stupce zaposlenik.ime, zaposlenik.prezime, zaposlenik_placa.iznos preimenovan u “placa”, zaposlenik_placa.mjesec i restoran.naziv preimenovan u “restoran”. Kao rezultat dobivamo svaku plaću te iznos svake plaće koja je ikad izdana bilo kojem zaposleniku.

## 8.2. Dinko Nađ

### 8.2.1. Pogled - Prikazuje koliko broj stavki po tipu i restoranu

```
DROP VIEW IF EXISTS broj_stavki_po_restoranu;
CREATE VIEW broj_stavki_po_restoranu AS
SELECT restoran.naziv AS naziv_restorana,
       stavka.stavka_tip,
       COUNT(*) AS broj_stavki
FROM stavka
JOIN restoran ON stavka.restoran_id = restoran.id
    GROUP BY restoran.naziv, stavka.stavka_tip
        ORDER BY restoran.naziv, broj_stavki DESC;

SELECT * FROM broj_stavki_po_restoranu;
```

Prvo, u FROM dijelu izabiremo iz koje tablice želimo uzimati podatke. U ovom slučaju, to je tablica „stavka“. Zatim, pomoću naredbe JOIN, pridružujemo tablicu „restoran“ na temelju stranog ključa stavka.restoran_id = restoran.id, čime povezujemo stavke s odgovarajućim restoranima. Naredbom GROUP BY grupiramo retke prema kombinaciji restoran.naziv i stavka.stavka_tip. Svaka jedinstvena kombinacija ovih stupaca predstavlja jednu grupu, tj. broj stavki za svaki tip stavke unutar svakog restorana. Nakon grupiranja, slijedi naredba SELECT koja odabire stupce restoran.naziv i stavka.stavka_tip, te COUNT(*), koji izračunava broj redaka. Nakon toga, pomoću naredbe ORDER BY, sortiramo podatke prema restoran.naziv i broj_stavki u opadajućem rasporedu, tako da restorani s najvećim brojem stavki budu prikazani prvi. Na kraju, kreiramo pogled pod nazivom broj_stavki_po_restoranu pomoću naredbe CREATE VIEW. Ovaj pogled sadrži rezultate koje smo upravo odabrali. Nakon kreiranja pogleda, izvršava se SELECT upit koji dohvaća sve podatke iz pogleda. Naredba DROP VIEW omogućuje da obrišemo pogled broj_stavki_po_restoranu kada ga više ne trebamo.

## 8.3. Luka Vilagoš

Upiti su napisani u Flask aplikaciji.

## 8.4. Sara Maljić

### 8.4.1. Pogled - Popis rezervacija s detaljima o stolu

Korištenjem pogleda *pogled_rezervacije_stol* pojednostavljuje se rad s podacima jer omogućuje lako dohvaćanje specifičnih informacija bez potrebe za ponovnim pisanjem složenih JOIN uvjeta u svakom upitu.

```
CREATE VIEW pogled_rezervacije_stol AS
SELECT
    r.id AS rezervacija_id,
    r.ime AS ime_gosta,
    r.vrijeme AS vrijeme_rezervacije,
    s.broj AS broj_stola,
    s.lokacija AS lokacija_stola
FROM rezervacija r
JOIN stol s ON r.stol_id = s.id;
```

Pogled pod nazivom *pogled_rezervacije_stol* kreiran je kako bi omogućio jednostavniji i pregledniji način dohvaćanja podataka o rezervacijama i povezanim stolovima. Ovaj pogled koristi podatke iz tablica *rezervacija* i *stol* kako bi spojio informacije o gostima, vremenu rezervacije i stolovima na koje se te rezervacije odnose. U ovom pogledu, naziv stupca *r.id* iz tablice *rezervacija* označen je aliasom *rezervacija_id*, dok je stupac *r.ime* označen kao *ime_gosta*, a *r.vrijeme* kao *vrijeme_rezervacije*. Slično tome, iz tablice *stol* preuzimaju se podaci o broju stola (*s.broj*) i njegovoj lokaciji (*s.lokacija*), koji su označeni aliasima *broj_stola* i *lokacija_stola*.

Podaci za pogled se dohvaćaju pomoću naredbe *FROM rezervacija r*, gdje je tablica *rezervacija* osnovna tablica i ima alias *r*. Kako bi se povezale rezervacije s pripadajućim stolovima, koristi se naredba *JOIN stol s ON r.stol_id = s.id*, koja povezuje tablice putem zajedničkih ključeva: vanjskog ključa *stol_id* iz tablice *rezervacija* i primarnog ključa *id* iz tablice *stol*.

## 8.4.2. Pogled - Broj rezervacija po lokaciji stola

Korištenjem pogleda *pogled_rezervacije_po_lokaciji*, dohvaćanje i analiza podataka o rezervacijama po lokacijama stolova znatno su pojednostavljeni, eliminirajući potrebu za ponovnim pisanjem složenih JOIN i GROUP BY uvjeta u svakom upitu.

```
CREATE VIEW pogled_rezervacije_po_lokaciji AS
SELECT s.lokacija AS lokacija_stola, COUNT(r.id) AS broj_rezervacija
FROM stol s
LEFT JOIN rezervacija r ON s.id = r.stol_id
GROUP BY s.lokacija;
```

Pogled pod nazivom *pogled_rezervacije_po_lokaciji* kreiran je kako bi omogućio prikaz broja rezervacija za stolove grupiranih prema njihovoj lokaciji. Ovaj pogled koristi podatke iz tablica *stol* i *rezervacija*, gdje se informacije o lokacijama stolova povezuju s brojem rezervacija za te stolove.

U pogledu, stupac *s.lokacija* iz tablice *stol* označen je aliasom *lokacija_stola*, a funkcija *COUNT(r.id)*, koja prebrojava rezervacije povezane s pojedinim stolovima, označena je aliasom *broj_rezervacija*. Time se u rezultatu jasno prikazuje svaka jedinstvena lokacija stola i ukupan broj rezervacija za stolove na toj lokaciji.

Podaci za pogled dohvaćaju se iz tablice *stol*, kojoj je dodijeljen alias *s*, a povezivanje s tablicom *rezervacija* vrši se naredbom *LEFT JOIN rezervacija r ON s.id = r.stol_id*. Ovaj JOIN omogućuje dohvaćanje svih stolova iz tablice *stol*, čak i ako za te stolove ne postoje odgovarajuće rezervacije u tablici *rezervacija*. U tom slučaju, funkcija *COUNT(r.id)* za takve stolove vraća vrijednost 0.

Korištenjem naredbe *GROUP BY s.lokacija*, rezultati se grupiraju prema svakoj jedinstvenoj lokaciji stola. Na taj način, za svaku lokaciju izračunava se ukupan broj rezervacija povezanih s tom lokacijom.

### 8.4.3. Pogled - Prikaz svih aktivnih rezervacija s informacijama o stolu

```
CREATE VIEW aktivne_rezervacije AS
SELECT r.id AS rezervacija_id, r.ime, r.vrijeme, s.broj AS broj_stola, s.lokacija
FROM rezervacija r
JOIN stol s ON r.stol_id = s.id
WHERE r.deleted_at IS NULL AND r.disabled = FALSE;

SELECT * FROM aktivne_rezervacije WHERE lokacija = 'unutra';
```

Pogled pod nazivom *aktivne_rezervacije* kreiran je kako bi omogućio jednostavan prikaz svih aktivnih rezervacija u sustavu. Aktivne rezervacije definirane su kao one koje nisu označene za brisanje (*r.deleted_at IS NULL*) i nisu onemogućene (*r.disabled* = *FALSE*). Cilj ovog pogleda je olakšati upravljanje i filtriranje aktivnih rezervacija na temelju ključnih informacija.

Pogled dohvaća podatke iz tablica *rezervacija* i *stol*, gdje se povezuju informacije o rezervacijama s podacima o stolu na kojem su rezervacije napravljene. U stupcima rezultata prikazuju se:

- **rezervacija_id**: jedinstveni identifikator rezervacije (alias za *r.id*),
- **ime**: ime gosta koji je napravio rezervaciju (preuzeto iz *r.ime*),
- **vrijeme**: vrijeme rezervacije (preuzeto iz *r.vrijeme*),
- **broj_stola**: broj stola na kojem je rezervacija napravljena (alias za *s.broj*),
- **lokacija**: lokacija stola (preuzeto iz *s.lokacija*).

Podaci se dohvaćaju iz tablice *rezervacija*, kojoj je dodijeljen alias **r**, a povezuju se s tablicom *stol*, kojoj je dodijeljen alias **s**, koristeći naredbu *JOIN stol s ON r.stol_id = s.id*. Ovo osigurava da se svaka rezervacija povezuje s točnim stolom putem odnosa između primarnog ključa tablice *stol* (*s.id*) i vanjskog ključa tablice *rezervacija* (*r.stol_id*).

Pogled dodatno filtrira podatke kako bi uključio samo rezervacije koje nisu obrisane niti onemogućene, koristeći uvjete:

- **r.deleted_at IS NULL**, koji osigurava da su obrisane rezervacije isključene,
- **r.disabled = FALSE**, koji osigurava da su onemogućene rezervacije također isključene

# 8.5. Alma Reka

## 8.5.1. Pogled - RacunUkupnaVrijednost

Pogled koji pruža ukupnu vrijednost svakog računa, uključujući napojnicu.

```
DROP VIEW IF EXISTS RacunUkupnaVrijednost;
CREATE VIEW RacunUkupnaVrijednost AS
SELECT
    r.id AS RacunID,
    r.iznos + r.napojnica AS UkupnaVrijednost
FROM
    racun r;

UPDATE racun
SET iznos = 300, napojnica = 20
WHERE id = 1;
SELECT * FROM racun;
SELECT * FROM RacunUkupnaVrijednost;
```

Ovaj pogled RacunUkupnaVrijednost izrađuje se kako bi se lako vidjelo koliko je ukupno plaćeno po svakom računu, uzimajući u obzir i osnovni iznos i dodanu napojnicu. Osnovna ideja je da se iz svake stavke računa racun uzme iznos i napojnica, te se ti iznosi zbroje da bi se dobila konačna, ukupna vrijednost plaćanja.

```
SET iznos = 300, napojnica = 20
WHERE id = 1;
```

Ova naredba ažurira specifičan račun s ID-jem 1, postavljajući njegov iznos na 300 i napojnicu na 20. Služi za demonstraciju kako promjene u osnovnim podacima utječu na podatke prikazane u pogledu.

```
SELECT * FROM racun;
SELECT * FROM RacunUkupnaVrijednost;
```

Ovi upiti su primjeri kako dohvatiti podatke iz izvorne tablice racun te kako vidjeti kako izgledaju podaci u kreiranom pogledu RacunUkupnaVrijednost nakon izvršenih promjena.

### 8.5.2. Pogled - Pogled koji prikazuje sve sastojke potrebne za svaki recept, uključujući potrebne količine

```
CREATE VIEW SastojciPoReceptu AS
SELECT
    rec.naziv AS Recept,
    sastojak.naziv AS Sastojak,
    rs.kolicina AS PotrebnaKolicina
FROM
    recept rec
JOIN
    recept_sastojak rs ON rec.id = rs.recept_id
JOIN
    sastojak ON rs.sastojak_id = sastojak.id;

SELECT * FROM SastojciPoReceptu;
```

Pogled SastojciPoReceptu formira se kako bi se jednostavno moglo pratiti koji sastojci, i u kojim količinama, su potrebni za svaki recept. Tablica recept se povezuje s recept_sastojak radi identifikacije sastojaka koji ulaze u svaki recept, a zatim se povezuje sa sastojak kako bi se dobili nazivi i količine tih sastojaka.

```
SELECT * FROM SastojciPoReceptu;
```

Ovaj upit služi za prikaz svih podataka iz pogleda SastojciPoReceptu, omogućujući pregled potrebnih sastojaka za recepte navedene u bazi podataka.

## 8.6. Patricia Lazić

### 8.6.1. Pogled - Pregled svih nezgoda po zaposleniku s ukupnim troškovima

```
CREATE VIEW pregled_nezgoda_po_zaposleniku AS
SELECT
    z.id AS zaposlenik_id,
    z.ime AS zaposlenik_ime,
    z.prezime AS zaposlenik_prezime,
    COALESCE(SUM(mn.ukupno), 0) AS ukupno_mala_nezgoda,
    COALESCE(SUM(vn.ukupno), 0) AS ukupno_velika_nezgoda,
    COALESCE(SUM(mn.ukupno), 0) + COALESCE(SUM(vn.ukupno), 0) AS ukupno_trosak
FROM
    zaposlenik z
LEFT JOIN
    mala_nezgoda mn ON z.id = mn.zaposlenik_id
LEFT JOIN
    velika_nezgoda vn ON z.id = vn.zaposlenik_id
GROUP BY
    z.id, z.ime, z.prezime;
```

Odaberemo atribute ID, ime i prezime iz tablice zaposlenik i spajamo ju s tablicama mala nezgoda i velika nezgoda koristeći left join koji osigurava da će svi zaposlenici biti prikazani čak i ako nemaju povezane male i velike nezgode. Spoj se temelji na tome da ID zaposlenika se podudara sa ID-em zaposlenika iz velike i male nezgode. Koristimo funkciju SUM() kako bi zbrojili ukupno iz tablice male i velike nezgode za svakog zaposlenika. Ako zaposlenik nema veliku ili malu nezgodu, rezultat sume bi bio NULL. Pomoću funkcije COALESCE() NULL vrijednost zamjenjuje s 0. Zatim računamo ukupan trošak za velike i male nezgode tako što zbrajamo rezultate male nezgode i velike nezgode kako bismo dobili ukupni trošak svih nezgoda po zaposleniku. Grupiramo rezultate po zaposlenikovom ID-u, imenu i prezimenu te u konačnici dobivamo tablicu s općim podacima zaposlenika, iznos male i velike nezgode i ukupan trošak svih nezgoda. Ovaj pogled omogućuje detaljan uvid u troškove povezane s nezgodama za svakog zaposlenika.

# 9. Indeksi

## 9.1. Paula Vorih

### 9.1.1. Indeks

```
CREATE INDEX indx_restoran_naziv ON restoran(naziv);
```

### 9.1.2. Indeks

```
CREATE INDEX indx_zaposlenik_placa_iznos ON zaposlenik_placa(iznos);
```

## 9.2. Luka Vilagoš

### 9.2.1. Indeks

```
CREATE INDEX idx_transakcija_zaposlenik ON transakcija_zaposlenik(zaposlenik_id);
```

### 9.2.2. Indeks

```
CREATE INDEX idx_transakcija_restoran ON transakcija_restoran(restoran_racun_id);
```

## 9.3. Sara Maljić

### 9.3.1. Indeks - Brže pretraživanje rezervacija prema vremenu

```
CREATE INDEX idx_rezervacija_vrijeme ON rezervacija (vrijeme);
```

Ovaj izraz kreira indeks pod nazivom idx_rezervacija_vrijeme na stupcu vrijeme u tablici rezervacije. Glavna svrha ovoga indeksa je ubrzati pretraživanje i filtriranje rezervacija prema stupcu vrijeme. Indeks će omogućiti brže pronalaženje odgovarajućih redaka jer baza može brzo pristupiti i sortirati podatka na temelju vrijednosti vrijeme.

### 9.3.2. Indeks - Pretraživanje stolova prema lokaciji

```
CREATE INDEX idx_stol_lokacija ON stol (lokacija);
```

Ovaj izraz kreira indeks pod nazivom idx_stol_lokacija na stupcu lokacija u tablici stol. Glavna svrha ovoga indeksa je

ubrzati pretraživanje i filtriranje rezervacija prema stupcu lokacija. Indeks će omogućiti brže pronalaženje odgovarajućih redaka jer baza može brzo pristupiti i sortirati podatka na temelju vrijednosti vrijeme.

## 9.4. Alma Reka

### 9.4.1. Indeks - idx_racun_zaposlenik

Indeks za poboljšanje performansi upita koji filtriraju podatke na temelju ID-a zaposlenika u tablici racun.

```
CREATE INDEX idx_racun_zaposlenik ON racun(zaposlenik_id);

SELECT * FROM racun
WHERE zaposlenik_id = 2;
EXPLAIN SELECT * FROM racun WHERE zaposlenik_id = 2;
```

Ovaj indeks, nazvan idx_racun_zaposlenik, stvara se na koloni zaposlenik_id unutar tablice racun. Glavni cilj je ubrzati proces pretraživanja i izvlačenja podataka koji se odnose na određene zaposlenike. Indeks omogućava bazi podataka da efikasnije locira i dohvaća zapise vezane za specifične zaposlenike, što je posebno korisno u situacijama s velikim volumenom podataka i čestim upitima koji filtriraju po ID-u zaposlenika.

```
EXPLAIN SELECT * FROM racun WHERE zaposlenik_id = 2;
```

Naredba EXPLAIN koristi se za prikaz plana izvršenja za upit, što uključuje detalje o korištenju indeksa. Ovo je korisno za razumijevanje kako baza podataka obrađuje upit i provjeravanje da li se stvoreni indeks idx_racun_zaposlenik aktivno koristi za optimizaciju upita. Očekuje se da će EXPLAIN pokazati da upit koristi indeks za pristup podacima, što rezultira smanjenjem vremena potrebnog za dohvaćanje podataka.

## 9.5. Patricia Lazić

### 9.5.1. Indeks

```
CREATE INDEX idx_velika_nezgoda_restoran_zaposlenik ON velika_nezgoda
(restoran_id, zaposlenik_id);
```

Koristimo naredbu CREATE INDEX koja stvara indeks na jednoj ili više kolona u tablici baze podataka. Indeks nam omogućuje ubrzanje pretrage podataka u tablici što je korisno kada se radi o velikoj količini podataka. Dali smo naziv indeksa „idx_velika_nezgoda_restoran_zaposlenik“. ON velika_nezgoda označava da indeks kreiramo na tablici „velika_nezgoda“. Restoran_id i zaposlenik_id su kolone na kojima je stvoren indeks to jest restoran_id označava strani ključ prema tablici restoran, a zaposlenik_id označava strani ključ prema tablici zaposlenik. Indeks je koristan ako na primjer tražimo sve velike nezgode koje je prijavio određeni zaposlenik u određenom restoranu.

# 10. Korisnici

## 10.1. Paula Vorih

### 10.1.1. Voditelj_skladista:

```
CREATE USER 'voditelj_skladista'@'localhost' IDENTIFIED BY 'Voditelj_042';
```

Ako kreiramo korisnika na localhost-u, to znači da će se pristup bazi moći vršiti samo sa tog računala. Korisnik se identificira, to jest, njegova lozinka je Voditelj_042.

### 10.1.2. Dodjela određenih prava ovom korisniku

```
GRANT SELECT, INSERT, UPDATE ON skladiste TO 'voditelj_skladista'@'localhost';
GRANT SELECT ON narudzba TO 'voditelj_skladista'@'localhost';
GRANT SELECT ON sastojak TO 'voditelj_skladista'@'localhost';
```

Korisniku dozvoljavamo SELECT, INSERT i UPDATE operacije na tablici “skladiste” i samo SELECT operaciju nad tablicama “narudzba” i “sastojak”.

## 10.2. Dinko Nađ

### 10.2.1. Korisnik - kuhar

Ima samo pravo vidjeti podatke o jelovniku, receptima, sastojcima i stavkama.

```
#DROP ROLE IF EXISTS kuhar;
#DROP USER IF EXISTS 'Dinko_nad'@'localhost';create role kuhar;

GRANT SELECT ON sustav_za_upravljanje_restoranom.jelovnik TO kuhar;
GRANT SELECT ON sustav_za_upravljanje_restoranom.recept TO kuhar;
GRANT SELECT ON sustav_za_upravljanje_restoranom.sastojak TO kuhar;
GRANT SELECT ON sustav_za_upravljanje_restoranom.stavka TO kuhar;
CREATE USER 'Dinko_nad'@'localhost' IDENTIFIED BY 'bravo';
GRANT kuhar TO 'Dinko_nad'@'localhost';
SET DEFAULT ROLE kuhar TO 'Dinko_nad'@'localhost';
```

Pomoću naredbe CREATE ROLE kreiramo novu ulogu pod imenom „kuhar“. To nam omogućuje grupiranje privilegija, pa ih možemo dodijeliti drugim korisnicima. Pomoću naredbi GRANT SELECT ON.....TO kuhar dodaju se privilegije SELECT na četiri tablice (jelovnik, recept ,sastojak, stavka) unutar baze podataka sustav_za_upravljanje_restoranom ulozi kuhar. Dodavanjem privilegije SELECT, ograničavamo korisnika na samo čitanje podataka iz tablica, ali ne mogu ih mijenjati. Nakon dodjele privilegija, kreiramo korisnika pomoću naredbe CREATE USER. Naredba stvara novog korisnika „Dinko_nad“ sa lozinkom „bravo“ koji se može prijaviti samo s „localhost“. Pomoću naredbe „GRANT kuhar TO 'Dinko_nad'@'localhost';“ dodajemo ulogu „kuhar“ korisniku Dinko_nad. Korisnik nasljeđuje sve privilegije koje su dodijeljene ulozi „kuhar“, a to uključuje privilegiju SELECT na tablicama jelovnik,recept,sastojak,stavka. Na samom kraju postavljamo „kuhar“ kao zadanu ulogu za korisnika Dinko_nad. Ta naredba osigurava da svaki puta kada se prijavi korisnik „Dinko_nad“, automatski će dobiti privilegije koje su povezane sa „kuhar“. DROP naredbe služe da se ukloni role ili user.

## 10.3. Luka Vilagoš

### 10.3.1. Korisnik - vlasnik

```
CREATE ROLE vlasnik;
CREATE USER 'vlasnik'@'localhost' IDENTIFIED BY 'password';
GRANT vlasnik TO 'vlasnik'@'localhost';
SET DEFAULT ROLE vlasnik TO 'vlasnik'@'localhost';
```

Kreiramo ulogu vlasnik te ju dajemo korisniku vlasnik@localhost. To je također defaultna uloga za ovog korisnika.

Ako kreiramo korisnika na localhost-u, to znači da će se pristup bazi moći vršiti samo sa tog računala. Korisnik se identificira, to jest, njegova lozinka je ‘password’.

### 10.3.1. Dodjela određenih prava ovom korisniku

```
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.restoran
TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.zaposlenik TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.zaposlenik_placa TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.skladiste TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.restoran_racun TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.trosak
TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.stol TO
vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.rezervacija TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.racun TO
vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.recept
TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.stavka
TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.jelovnik
TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.jelovnik_stavka TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON sustav_za_upravljanje_restoranom.sastojak
TO vlasnik;
GRANT SELECT, INSERT, UPDATE, DELETE ON
sustav_za_upravljanje_restoranom.recept_sastojak TO vlasnik;

GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.narudzba TO vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.sastojak_narudzba TO
vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.mala_nezgoda TO vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.mala_nezgoda_sastojak TO
vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.velika_nezgoda TO
vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.velika_nezgoda_stavka TO
vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.transakcija_restoran TO
vlasnik;
GRANT SELECT, INSERT ON sustav_za_upravljanje_restoranom.transakcija_zaposlenik
TO vlasnik;
```

# 10.4. Sara Maljić

## 10.4.1. Korisnik - tajnik

Tajnik izdaje plače zaposlenicima, ima uvid u povijest transakcija, zapisuje troškove (mjesečne i jednokratne).

```
CREATE USER 'tajnik'@'localhost' IDENTIFIED BY 'tajnik_password';
GRANT SELECT, INSERT, UPDATE ON zaposlenik_placa TO 'tajnik'@'localhost';
GRANT SELECT, INSERT, UPDATE ON restoran TO 'tajnik'@'localhost';
GRANT SELECT, INSERT, UPDATE ON restoran_racun TO 'tajnik'@'localhost';
```

Ovaj SQL kod kreira korisnika u bazi podataka pod nazivom "tajnik" koji ima pristup određenim tablicama u bazi podataka. Korisnik "tajnik" ima lozinku "tajnik_password " i može se prijaviti s lokalnog hosta . Nadalje, privilegije se daju za odabrane operacije na tablicama : zaposlenik_placa, restoran i restoran_racun . Tajniku je odobreno SELECT , da vidi podatke , INSERT , za dodavanje novih podataka i UPDATE , za izmjene postojećih podataka u sve tri tablice . To znači da tajnik može pregledavati i unositi nove transakcije, kao i ažurirati podatke vezane uz mjesto zaposlenika, poslovanje restorana i račune, ali neće imati prava na brisanje podataka.

# 10.5. Alma Reka

## 10.5.1. Korisnik glavni_kuhar

Postavljanje korisničkih prava za glavnog kuhara koji ima pristup i mogućnost mijenjanja informacija o stavkama, receptima, jelovnicima i sastojcima. Korisnik glavni kuhar koji ima pravo na uvid u stavke, recepte, jelovnike, sastojke. Može mijenjati jelovnike i recepte.

```
CREATE USER 'glavni_kuhar'@'localhost' IDENTIFIED BY 'password';

GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.stavka TO
'glavni_kuhar'@'localhost';

GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.recept TO
'glavni_kuhar'@'localhost';

GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.jelovnik TO
'glavni_kuhar'@'localhost';

GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.sastojak TO
'glavni_kuhar'@'localhost';

DROP USER 'glavni_kuhar'@'localhost';
```

Ova naredba briše postojećeg korisnika glavni_kuhar na serveru localhost. Ovo je korisno u slučajevima kada želite resetirati korisničke postavke ili ažurirati prava pristupa iz sigurnosnih ili organizacijskih razloga.

```
CREATE USER 'glavni_kuhar'@'localhost' IDENTIFIED BY 'password';
```

Ovime se kreira novi korisnički račun za glavnog kuhara na localhost s lozinkom password. Ova naredba postavlja osnovu za dodjelu specifičnih prava i pristupa unutar sustava za upravljanje restoranom.

```
GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.stavka TO
'glavni_kuhar'@'localhost';
GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.recept TO
'glavni_kuhar'@'localhost';
GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.jelovnik TO
'glavni_kuhar'@'localhost';
GRANT SELECT, INSERT, UPDATE ON sustav_za_upravljanje_restoranom.sastojak TO
'glavni_kuhar'@'localhost';
```

GRANT naredbe dodjeljuju prava SELECT, INSERT i UPDATE za glavnog kuhara na tablicama stavka, recept, jelovnik i sastojak unutar baze podataka sustav_za_upravljanje_restoranom. Ovo omogućava glavnom kuharu da pregleda podatke, unosi nove informacije i ažurira postojeće unose u svim navedenim kategorijama, što je ključno za učinkovito upravljanje kuhinjom i jelovnicima.

# 10.6. Patricia Lazić

## 10.6.1. Korisnik - konobar

```
CREATE ROLE konobar;
GRANT INSERT, UPDATE ON sustav_za_upravljanje_restoranom.mala_nezgoda TO konobar;
GRANT INSERT, UPDATE ON sustav_za_upravljanje_restoranom.velika_nezgoda TO
konobar;
GRANT INSERT, UPDATE ON sustav_za_upravljanje_restoranom.mala_nezgoda_sastojak TO
konobar;
GRANT INSERT, UPDATE ON sustav_za_upravljanje_restoranom.velika_nezgoda_stavka TO
konobar;
CREATE USER 'Patricia_konobar'@'localhost' IDENTIFIED BY 'patricia12345';
GRANT konobar TO 'Patricia_konobar'@'localhost';
SET DEFAULT ROLE konobar TO 'Patricia_konobar'@'localhost';
```

Koristeći naredbu CREATE ROLE stvaramo novu ulogu pod imenom „konobar“. Pomoću naredbi GRANT INSERT, UPDATE ON.....TO konobar dodaju se privilegije INSERT i UPDATE na tablice mala_nezgoda, velika_nezgoda, mala_nezgoda_sastojak i velika_nezgoda_stavka unutar baze podataka sustav_za_upravljanje_restoranom. To omogućuje umetanje i ažuriranje podataka specificiranih tablica. Pomoću naredbe CREATE USER kreiramo korisnika „Patricia_konobar“ koji ima lozinku „patricia12345“ koji se može prijaviti samo sa localhost-a što znači s istog računala na kojem je pokrenut MySQL server. Pomoću GRANT konobar TO korisniku se dodjeljuju privilegije definirane u roli „konobar“ i SET DEFAULT ROLE konobar postavlja rolu konobar kao zadanu rolu za korisnika Patricia_konobar.

# 11. Funkcije

## 11.1. Paula Vorih

### 11.1.1. Kreiranje funkcije za računanje ukupne zarade zaposlenika

```
DELIMITER //
CREATE FUNCTION UkupnaZarada(p_zaposlenik_id INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE ukupno DECIMAL(10, 2);
    SELECT SUM(iznos) INTO ukupno
    FROM zaposlenik_placa
    WHERE zaposlenik_id = p_zaposlenik_id;
    RETURN IFNULL(ukupno, 0);
END;
//
DELIMITER ;
```

Funkcija kao argument prima id zaposlenika koji je tipa INT i rezultat koji vraća je tipa DECIMAL(10, 2). Unutar BEGIN i END naredbi nalazi se tijelo funkcije gdje prvo deklariramo lokalnu varijablu “ukupno” koja je tipa DECIMAL(10, 2) i u nju spremamo sumirani iznos svih plaća iz tablice zaposlenik_placa, gdje je id zaposlenika jednak id-u zaposlenika za kojeg pozivamo funkciju. Kada vraćamo rezultat, koristimo IFNULL(ukupno, 0) kako bi u slučaju da se vrati NULL vrijednost (npr. ako zaposleniku još nije isplaćena niti jedna plaća), umjesto te NULL vrijednosti jednostavno prikazali broj 0.

**Primjer korištenja ove funkcije sa SELECT naredbom (računanje ukupne zarade za zaposlenika s id-em 1):**

SELECT UkupnaZarada(1);

# 11.2. Dinko Nađ

## 11.2.1. Funkcija - Izračunava ukupnu vrijednost svih namjernica koje se trenutno nalaze u skladištu za odabrani restoran

```
#DROP FUNCTION IF EXISTS cijena_skladista_restorana;

DELIMITER //
CREATE FUNCTION cijena_skladista_restorana(f_restoran_id INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE ukupno DECIMAL(10, 2);
        SELECT SUM(cijena * trenutna_kolicina) INTO ukupno
            FROM sastojak
        JOIN skladiste ON skladiste.id = skladiste_id
        WHERE skladiste.restoran_id = f_restoran_id;
    RETURN ukupno;
END //

DELIMITER ;

SELECT cijena_skladista_restorana(8) FROM dual;
```

Na samom početku pomoću naredbe DROP FUNCTION brišemo funkciju ako već postoji. DELIMITER // je standardni naredba s kojom započinjemo pisanje svake funkcije. CREATE FUNCTION definira novu funkciju pod nazivom „cijena_skladista_restorana“. Ta funkcija prima jedan parametar „f_restoran_id“ tipa int, koji predstavlja ID restorana za koji želimo izračunati ukupnu cijenu sastojaka u skladištu. RETURNS predstavlja tip podataka koji vračamo nakon izvršavanja funkcije, u ovome slučaju je to DECIMAL(10,2) sa 10 znamenki i 2 decimalna mjesta jer očekujemo decimalni broj. Naredba DETERMINISTIC znači da će funkcija uvijek vratiti isti rezultat za isti ulaz. BEGIN označava početak tijela funkcije. Na početku funkcije uz pomoć naredbe DECLARE deklariramo lokalnu varijablu „ukupno“ koja će pohraniti ukupnu vrijednost svih sastojaka u skladištu. Tip varijable je DECIMAL(10,2) jer treba spremiti i decimalne zapise. Nakon deklaracije slijedi upit. Prvo se u FROM dijelu uzima tablica iz koje se čitaju podatci, u ovom slučaju to je tablica „sastojak“. JOIN označava da želimo povezati podatke iz tablice „sastojak“ s podatcima iz tablice „skladiste“. Spajanje se vrši na temelju uvjete „ON skladiste.id=skladiste_id“. U WHERE dijelu filtriramo podatke tako da samo skladišta koja pripadaju restoranu sa specifičnim „restoran_id“ budu uključena u daljnji izračun. Uspoređuje se sa parametrom funkcije „f_restoran_id“ koji mi kasnije pozivom funkcije određujemo i onda izbacuje samo restorane sa predanim id-em. U SELECT množenjem „cijene“ i „trenutna_kolicina“ izračunavamo ukupnu cijenu za pojedini sastojak, a funkcijom SUM zbrajamo ove cijene za sve sastojke koji su prethodno filtrirani. INTO „ukupno“ pohranjuje rezultat u varijablu „ukupno“. I na samom kraju pomoću naredbe RETURN vračamo rezultat izračuna i završavamo funkciju sa END // odnosno DELIMITER;. Kako bismo provjerili funkciju pozivamo „SELECT cijena_skladista_restorana(8) FROM dual;“ i tražimo da nam izračuna vrijednost skladišta za restoran 8.

### 11.2.2. Funkcija - Provjerava razliku između potrebne i trenutne količine sastojaka te vraća "TREBA NARUČITI" ako je razlika negativna, ili "NE TREBA NARUČITI" ako nije

```
drop function treba_naruciti;
DELIMITER //
CREATE FUNCTION treba_naruciti(f_skladiste INT)
RETURNS varchar(20)
DETERMINISTIC
BEGIN
    DECLARE ukupno int;
    DECLARE rezultat Varchar(20);
        select sum(trenutna_kolicina-potrebna_kolicina) INTO ukupno
            from sastojak group by id
                HAVING id=f_skladiste;
        If ukupno >=0 then
            set rezultat="NE TREBA NARUCITI";
        else
        set rezultat="TREBA NARUCITI";
            end if;
            return rezultat;

END //

DELIMITER ;
select id,naziv, treba_naruciti(id) as TREBA_LI_NARUCITI from sastojak;
```

Na samom početku pomoću naredbe DROP FUNCTION brišemo funkciju ako već postoji. DELIMITER // je standardni naredba s kojom započinjemo pisanje svake funkcije. CREATE FUNCTION definira novu funkciju pod nazivom „treba_naruciti“. Ta funkcija prima jedan parametar „f_skladiste“ tipa int. RETURNS predstavlja tip podataka koji vračamo nakon izvršavanja funkcije, u ovome slučaju je to VARCHAR(20). Naredba DETERMINISTIC znači da će funkcija uvijek vratiti isti rezultat za isti ulaz. BEGIN označava početak tijela funkcije. Naredbom DECLARE deklariraju se dvije lokalne varijable. Varijabla „ukupno“ tipa je int i služi za spremanje međurezultata. Varijabla „rezultat“ tipa je varchar(20) i to je ono što se će se kasnije ispisivati. Nakon deklaracije, u FROM dijelu se odabire tablica iz koje će se uzimati podatci. Pomoću GROUP BY grupiramo podatke prema id-u. Nakon grupiranja, računa se razlika između trenutne količine i potrebne količine kako bi dobili informaciju jel treba više nego što ima. To se sprema u varijablu „ukupno“ naredbom INTO. Nakon selekcije, naredbom HAVING tražimo gdje je id jednak parametru koji smo predali.

U IF dijelu ispitujemo jel varijabla ukupno veća ili jednaka 0, ako je onda se u varijablu rezultat sprema „NE TREBA NARUCITI", ako uvjet nije zadovoljen, ide se u ELSE dio i tu se u rezultat sprema "TREBA NARUCITI". Sa END IF završavamo IF naredbu. Na samom kraju vračamo varijablu rezultat i završavamo funkciju sa END // odnosno DELIMITER;. Kako bismo pozivamo SELECT upit koji poziva funkciju za svaki red u tablici sastojak i vraća „id“, „naziv“ sastojka, te rezultat funkcije kao „TREBA_LI_NARUCITI“ koji označava treba li naručiti sastojak ovisno o njegovim količinama.

# 11.3. Luka Vilagoš

## 11.3.1. Funkcija - Kreiranje funkcije za računanje ukupne cijene narudžbe

```
DELIMITER //

CREATE FUNCTION ukupna_cijena_narudzbe(narudzba_id INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE ukupna_cijena DECIMAL(10, 2) DEFAULT 0;

    SELECT SUM(s.cijena * sn.kolicina)
    INTO ukupna_cijena
    FROM sastojak_narudzba sn
    JOIN sastojak s ON sn.sastojak_id = s.id
    WHERE sn.narudzba_id = narudzba_id;

    RETURN ukupna_cijena;
END;
//

DELIMITER ;
```

Funkcija kao argument prima id narudžbe koji je tipa INT i rezultat koji vraća je tipa DECIMAL(10, 2). Unutar BEGIN i END naredbi nalazi se tijelo funkcije gdje prvo deklariramo lokalnu varijablu “ukupna_cijena” koja je tipa DECIMAL(10, 2) sa defaultnom vrijednosti 0 i u nju spremamo sumirani iznos cijene iz tablice sastojak (s) pomnožene sa količinom iz tablice sastojak_narudzba (sn). Podatke odabiremo iz tablice sastojak_narudzba sn koja je join-ana s tablicom sastojak s, gdje je sn.narudzba_id jednak id-u sastojka i sn.narudzba_id mora biti jednak id-u narudžbe za koju pozivamo funkciju. Kada vraćamo rezultat, vraćamo ono što je pohranjeno u varijabli ukupna_cijena.

**Primjer korištenja ove funkcije sa SELECT naredbom (računanje ukupne zarade za narudžbu s id-em 2):**

```
SELECT  ukupna_cijena_narudzbe(2);
```

# 11.4. Sara Maljić

## 11.4.1. Funkcija - Provjera dostupnosti stola u određenom vremenskom razdoblju

```
DROP FUNCTION IF EXISTS provjeri_dostupnost_stola;
DELIMITER //
CREATE FUNCTION provjeri_dostupnost_stola(p_stol_id INT, p_vrijeme DATETIME)
RETURNS BOOLEAN
DETERMINISTIC
```

```
BEGIN
    DECLARE dostupno BOOLEAN;
    SELECT NOT EXISTS (
        SELECT 1
        FROM rezervacija
        WHERE stol_id = p_stol_id
        AND vrijeme BETWEEN p_vrijeme AND DATE_ADD(p_vrijeme, INTERVAL 2 HOUR)
        AND deleted_at IS NULL
        AND disabled = FALSE
    )
    INTO dostupno;
    RETURN dostupno;
END;
//
DELIMITER ;
```

U SQL funkciji *provjeri_dostupnost_stola*, cilj je utvrditi je li određeni stol dostupan za rezervaciju u zadanom vremenskom razdoblju. Funkcija prihvaća dva ulazna argumenta:

- **p_stol_id** (*INT*): identifikator stola za provjeru,
- **p_vrijeme** (*DATETIME*): datum i vrijeme kada se provjerava dostupnost.

Funkcija prvo deklarira lokalnu varijablu *dostupno* tipa *BOOLEAN* koja će pohraniti rezultat provjere dostupnosti.

Unutar tijela funkcije koristi se naredba *SELECT NOT EXISTS* koja provjerava postoji li bilo koja rezervacija za određeni stol (**stol_id**) unutar zadanog vremenskog intervala. Taj vremenski interval definiran je kao razdoblje između početnog vremena (**p_vrijeme**) i dva sata kasnije (*DATE_ADD(p_vrijeme, INTERVAL 2 HOUR)*). Uz to, uvjeti osiguravaju:

- **deleted_at IS NULL**: rezervacija nije označena kao obrisana,
- **disabled = FALSE**: rezervacija nije onemogućena.

Rezultat upita (*TRUE* ili *FALSE*) unosi se u varijablu *dostupno*.

- Ako **NOT EXISTS** vrati *TRUE* (tj. ne postoji nijedna rezervacija koja zadovoljava uvjete), funkcija određuje da je stol dostupan i vraća *TRUE*.
- Ako **NOT EXISTS** vrati *FALSE* (tj. postoji barem jedna rezervacija), funkcija određuje da stol nije dostupan i vraća *FALSE*.

Naredba **DELIMITER //** koristi se kako bi se omogućilo definiranje tijela funkcije, budući da standardni separator **;** može uzrokovati probleme pri definiranju složenih SQL objekata.

Ključni dijelovi funkcije:

- **DROP FUNCTION IF EXISTS**: osigurava da se funkcija kreira iznova, uklanjajući staru verziju ako postoji.
- **CREATE FUNCTION**: definira novu funkciju s nazivom *provjeri_dostupnost_stola*.
- **DECLARE dostupno BOOLEAN**: deklarira varijablu koja pohranjuje rezultat provjere.
- **SELECT NOT EXISTS (... INTO dostupno)**: provodi upit i rezultat pohranjuje u varijablu.
- **RETURN dostupno**: vraća konačan rezultat (*TRUE* ili *FALSE*).

Ova funkcija omogućuje jednostavnu i učinkovitu provjeru dostupnosti stolova za rezervaciju u zadanim vremenskim okvirima i služi kao koristan alat za upravljanje rezervacijama u sustavu.

### 11.4.2. Funkcija - Pronalazak prikladnog stola prema broju mjesta

```
DELIMITER //
CREATE FUNCTION pronadi_prikladan_stol(restoran_id INT, broj_mjesta INT,
lokacija_preference ENUM('unutra', 'vani', 'vip'))
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE prikladan_stol INT;
    SELECT id
    INTO prikladan_stol
    FROM stol
    WHERE restoran_id = restoran_id AND broj_mjesta >= broj_mjesta AND lokacija =
lokacija_preference AND deleted_at IS NULL AND disabled = FALSE
    LIMIT 1;
    RETURN prikladan_stol;
END;
//
DELIMITER ;
```

Funkcija pronadi_prikladan_stol služi za pronalaženje stola u restoranu koji zadovoljava određene uvjete. Funkcija uzima tri argumenta: restoran_id, broj_mjesta i lokacija_preference. Funkcija vraća ID stola koji zadovoljava ove uvjete ili NULL ako takav stol nije pronađen.

Unutar funkcija, prvo se deklarira varijabla prikladan_stol tipa INT koja će pohraniti ID stola koji odgovara uvjetima pretrage. Zatim se izvršava SELECT upit koji traži stol u tablici stol prema zadanim uvjetima: stol mora biti u odgovarajućem restoranu, imati dovoljan broj mjesta, biti smješten na odgovarajućoj lokaciji, ne smije biti obrisan

i ne smije biti onemogućen. Upit koristi LIMIT 1 kako bi vratio samo prvi odgovarajući stol koji zadovoljava ove uvjete. Na kraju, funkcija vraća ID pronađenog stola.

Kao rezultat, ova funkcija omogućuje pronalaženje stola u restoranu koji odgovara preferencijama korisnika vezanim uz broj mjesta i lokaciju stola, uz dodatne provjere da stol nije obrisan ili onemogućen.

## 11.5. Alma Reka

### 11.5.1. Funkcija - UkupnaVrijednostRacuna

Funkcija za izračunavanje ukupne vrijednosti računa na temelju osnovnog iznosa i napojnice.

```
DELIMITER $$

CREATE FUNCTION UkupnaVrijednostRacuna(racunID INT)
```

```
RETURNS DECIMAL(10, 2)

DETERMINISTIC

BEGIN

  DECLARE ukupno DECIMAL(10, 2);

  SELECT iznos + napojnica INTO ukupno FROM racun WHERE id = racunID;

  RETURN ukupno;

END$$

DELIMITER ;



SHOW FUNCTION STATUS WHERE Name = 'UkupnaVrijednostRacuna';

SELECT UkupnaVrijednostRacuna(1) AS UkupnaVrijednost;
```

Ova SQL funkcija UkupnaVrijednostRacuna stvorena je za izračun ukupnog iznosa računa, uključujući i napojnicu, na temelju ID-a računa. Funkcija prima racunID kao parametar, koji koristi za pronalazak specifičnog računa u tablici racun. Kombinira iznos računa i napojnicu kako bi dobila konačnu ukupnu vrijednost, koja se pohranjuje u lokalnu varijablu ukupno deklariranu unutar funkcije. Vraćena vrijednost je decimalni broj s dva decimalna mjesta, što osigurava preciznost u financijskim izračunima.

```
SHOW FUNCTION STATUS WHERE Name = 'UkupnaVrijednostRacuna';
```

Ova naredba prikazuje status i detalje o funkciji UkupnaVrijednostRacuna. Korisna je za provjeru postoji li funkcija u bazi podataka, kao i za dobivanje informacija o njenim karakteristikama kao što su tip povratne vrijednosti i parametri.

```
SELECT UkupnaVrijednostRacuna(1) AS UkupnaVrijednost;
```

Ovaj upit pokreće funkciju UkupnaVrijednostRacuna s parametrom ID-a računa 1, vraćajući ukupnu vrijednost tog specifičnog računa. Rezultat je prikazan pod imenom stupca UkupnaVrijednost, što korisniku omogućava lako čitanje i razumijevanje izlaza.

# 11.6. Patricia Lazić

## 11.6.1. Funkcija - dohvaćanje broja sastojaka vezanih za malu nezgodu

```
DELIMITER //

CREATE FUNCTION broj_sastojaka_male_nezgode (p_mala_nezgoda_id INT)
RETURNS INT
```

```
DETERMINISTIC
BEGIN
    DECLARE broj_sastojaka INT;

    SELECT COUNT(*)
    INTO broj_sastojaka
    FROM mala_nezgoda_sastojak
    WHERE mala_nezgoda_id = p_mala_nezgoda_id;

    RETURN broj_sastojaka;
END//

DELIMITER ;
SELECT broj_sastojaka_male_nezgode(3);
```

Funkciju počinjemo sa standardnom naredbom DELIMITER //, zatim koristimo CREATE FUNCTION kako bismo stvorili funkciju pod nazivom „broj_sastojaka_male_nezgode“. Funkcija prima parametar „p_mala_nezgoda_id“ (ID određene male nezgode) tipa integer i označena je kao DETERMINISTIC što znači da uvijek vraća isti rezultat za iste ulazne parametre. Deklariramo lokalnu varijablu broj_sastojaka tipa integer koja će sadržavati broj sastojaka. Za brojanje sastojaka koristimo agregacijsku funkciju COUNT(*) koja broji sve retke iz tablice mala_nezgoda_sastojak koji zadovoljavaju određeni uvjet koji je u ovom slučaju da ograničava pretragu redova u kojima je ID male nezgode jednak vrijednosti parametra (WHERE mala_nezgoda_id = p_mala_nezgoda_id;). Rezultat tog broja spremamo u varijablu broj_sastojaka pomoću INTO i funkcija vraća vrijednost varijable broj_sastojaka koristeći naredbu RETURN. Funkciju završavamo s naredbom END// DELIMITER;. Pozivamo funkciju koristeći SELECT, ime funkcije i u zagradi proslijedimo parametar.

SELECT broj_sastojaka_male_nezgode(3);. Funkcija vraća broj različitih sastojaka povezanih s određenom malom nezgodom.

# 12. Procedure

## 12.1. Paula Vorih

### 12.1.1. Procesiranje mjesečne plaće

```
DELIMITER //
CREATE PROCEDURE DajPlacu(zaposlenik_id INT, iznos DECIMAL(10, 2), mjesec DATE)
BEGIN
    INSERT INTO zaposlenik_placa (zaposlenik_id, iznos, mjesec)
    VALUES (zaposlenik_id, iznos, mjesec);
END;
//
DELIMITER ;
```

Kreiramo proceduru “DajPlacu” koja prima tri ulazna parametra: zaposlenik_id tipa INT, iznos tipa DECIMAL(10, 2) i mjesec tipa DATE. Parametri se automatski smatraju ulaznima, čak i ako prije svakog od njih ne napišemo IN. Unutar BEGIN i END naredbi definiramo što će se dogoditi kada pozovemo ovu proceduru, a u ovom slučaju će to biti da u tablicu zaposlenik_placa, u stupce zaposlenik_id, iznos i mjesec, unosimo nove vrijednosti za te iste stupce.

**Primjer korištenja ove procedure gdje zaposleniku sa id-em 1 dajemo iznos plaće od 800 jedinica valute za četvrti mjesec 2025. godine:**

CALL DajPlacu(1, 800.00, '2025-04-01');

## 12.1.2. Zapošljavanje novog radnika

```
DELIMITER //

CREATE PROCEDURE ZaposliRadnika(
    IN restoran_id INT,
    IN zaposlenik_tip ENUM('vlasnik', 'kuhar', 'glavni_konobar', 'konobar',
'cistac'),
    IN ime VARCHAR(31),
    IN prezime VARCHAR(31),
    IN email VARCHAR(255),
    IN datum_rodenja DATE,
    IN iznos_place DECIMAL(10, 2),
    IN slika BLOB
)
BEGIN
    INSERT INTO zaposlenik (restoran_id, zaposlenik_tip, ime, prezime, email,
datum_rodenja, iznos_place, slika)
    VALUES (restoran_id, zaposlenik_tip, ime, prezime, email, datum_rodenja,
iznos_place, slika);
END;
//
DELIMITER ;
```

Kreiramo proceduru ZaposliRadnika koja ima osam ulaznih parametra koji se podudaraju sa atributima iz tablice zaposlenik. Unutar BEGIN i END naredbi, kao i u prethodnoj proceduri, definiramo što moramo upisati kada pozovemo proceduru, u ovom slučaju u tablicu zaposlenik unosimo vrijednosti koje će opisivati novoga radnika, a to su restoran_id, zaposlenik_tip, ime, prezime, email, datum_rodenja, iznos_place i slika.

**Primjer korištenja ove procedure gdje zapošljavamo konobara Vlado Zadravec u restoran s id-em 3:**

CALL ZaposliRadnika(3, 'konobar', 'Vlado', 'Zadravec', 'vladek45@hotmail.com', '1986-09-15', 900.00, NULL);

**U slučaju da zapošljavamo zaposlenika gdje ne navedemo postojeću zaposleničku ulogu, rezultat će biti greška:**

CALL ZaposliRadnika(3, 'pomocni_kuhar', 'Maksimilijan', 'Marich', 'maxich@outlook.com', '1999-02-02', 1000.00, NULL);

**Error Code: 1265. Data truncated for column 'zaposlenik_tip' at row 1**

# 12.2. Dinko Nađ

## 12.2.1. Procedura - Dodavanje troška u tablicu trošak

```
DROP PROCEDURE dodaj_trosak;
DELIMITER //

DELIMITER //
CREATE PROCEDURE dodaj_trosak(
    IN p_restoran_id INT,
    IN p_naziv VARCHAR(64),
    IN p_iznos DECIMAL(10, 2),
    IN p_mjesecno BOOLEAN
)
BEGIN
    IF p_restoran_id IS NULL OR p_naziv IS NULL OR p_iznos IS NULL OR p_mjesecno
IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Niste unjeli sve podatke,
molimo vas da popunite sve parametre!';
    ELSE
        INSERT INTO trosak (restoran_id, naziv, iznos, mjesecno)
        VALUES (p_restoran_id, p_naziv, p_iznos, p_mjesecno);
    END IF;
END //

DELIMITER ;

CALL dodaj_trosak(1, 'Trošak za oglašavanje', 522, TRUE);
CALL dodaj_trosak(1, 'Trošak za oglašavanje', null, TRUE);
```

Na samom početku pomoću naredbe DROP PROCEDURE brišemo proceduru ako već postoji. DELIMITER // je standardni naredba s kojom započinjemo pisanje svake procedure. Naredbom CREATE PRODEDURE kreira se procedura pod imenom „dodaj_trosak“. Svi parametri koje procedura prima imaju IN. To znači da rezultati mogu samo ulaziti u poceduru. Parametar „p_restoran_id“ tipa je int. Ovaj parametra predstavlja ID restorana kojem će se pridružiti trošak. Parametar „p_naziv“ tipa je Varchar(64). Ovaj parametar je naziv troška. Parametar „p_iznos“ je decimal(20) i prikazuje iznos troška, te parametar „p_mjesecno“ koji je tipa BOOLEAN i predstavlja jel trošak mjesečan. Tijelo procedure započinje naredbom BEGIN. Na početku se pomoći IF naredbe provjerava jesu li svi parametri popunjeni, ako jedan od svih parametara nije popunjen izbacit ćemo poruku pomoću naredbe „SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Niste uneli sve podatke, molimo vas da popunite sve parametre!';“. Ukoliko su svi parametri pravilno proslijeđeni, procedura unosi podatke u tablicu „trosak“. Prvo se određuje raspored kojim se unose podatci, a nakon toga u VALUES se prosljeđuju podatci iz parametara. Završavamo proceduru sa END // odnosno DELIMITER;. Na samom kraju, pozivamo proceduru sa CALL i punimo ga navedenim podatcima. Ukoliko je neka vrijednost null, izbacit će gore navedenu grešku kao u drugom primjeru.

## 12.2.2. Procedura - Ažuriranje količine sastojaka

```
#DROP PROCEDURE azuriraj_kolicinu_sastojka;
DELIMITER //
CREATE PROCEDURE azuriraj_kolicinu_sastojka(IN p_sastojak_id INT,IN
p_nova_kolicina INT)
BEGIN
    IF NOT EXISTS (SELECT 1 FROM sastojak WHERE id = p_sastojak_id) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Greška: Sastojak sa unetim ID-om ne postoji!';
    END IF;
    IF p_nova_kolicina < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Greška: Količina ne može biti negativna!';
    END IF;
    UPDATE sastojak
        SET trenutna_kolicina = p_nova_kolicina
    WHERE id = p_sastojak_id;

END //

DELIMITER ;
CALL azuriraj_kolicinu_sastojka(6, 10);
```

Na samom početku pomoću naredbe DROP PROCEDURE brišemo proceduru ako već postoji. DELIMITER // je standardni naredba s kojom započinjemo pisanje svake procedure. Naredbom CREATE PRODEDURE kreira se procedura pod imenom „azuriraj_kolicinu_sastojka“. Svi parametri koje procedura prima imaju IN. Procedura se sastoji od dva parametra. Prvi parametar je „p_sastojak_id“ koji predstavlja ID sastojka čiju količinu želimo promijeniti, dok drugi parametar „p_nova_kolicina“ predstavlja novu količinu koju želimo postaviti za odabrani sastojak. Prvo se dolazi u IF naredbu u kojoj provjeravamo postoji li taj id u tablici sastojci. U FROM dijelu se uzima tablica „sastojci“ i uspoređuju se id iz tablice i zadani id. Ako takav red ne postoji, uvjet „NOT EXIST“ će biti zadovoljen. Ako sastojak ne postoji, generira se greška SQLSTATE 45000. Pomoću naredbe SET MESSAGE_TEXT šaljemo specifičnu poruku korisniku, u ovom slučaju je to „'Greška: Sastojak sa unetim ID-om ne postoji!“. U drugom IF provjera se jel nova količina koja je predana kroz parametar veća od 0, odnosno osigurava da se ne unosi negativna veličina. Ako je količina negativna, ponovno se generira SQLSTATE 45000. Pomoću naredbe SET MESSAGE_TEXT šaljemo specifičnu poruku korisniku, u ovom slučaju je to „Greška: Količina ne može biti negativna!“. Ako su oba if-a zadovoljena, prelazi se na UPDATE naredbu koja ažurira tablicu sastojak tako da uz pomoć naredbe SET postavlja „trenutna_kolicina“ na količinu koju uzima iz parametra, odnosno koju smo mi zadali. Na kraju WHERE dio osigurava da se promjena dogodi na točno tom id na kojem mi to želimo. Završavamo proceduru sa END // odnosno DELIMITER;. Na samom kraju, pozivamo proceduru sa CALL i punimo podatcima o novoj količini.

### 12.2.3. Procedura - Ažuriranje stanja skladišta za prethodno unesene podatke

```
UPDATE skladiste AS skladiste_tablica
JOIN (
    SELECT
        skladiste_id,
        SUM(trenutna_kolicina) AS ukupno_trenutna,
        SUM(potrebna_kolicina) AS ukupno_potrebna
    FROM sastojak
    GROUP BY skladiste_id
) AS skladiste_statistika
ON skladiste_tablica.id = skladiste_statistika.skladiste_id
SET skladiste_tablica.stanje =
    CASE
        WHEN skladiste_statistika.ukupno_trenutna = 0 THEN 'prazno'
        WHEN skladiste_statistika.ukupno_trenutna <
(skladiste_statistika.ukupno_potrebna * 0.5) THEN 'kritično'
        WHEN skladiste_statistika.ukupno_trenutna <
skladiste_statistika.ukupno_potrebna THEN 'normalno'
        ELSE 'puno'
    END;

Select * from skladiste;
```

Ovaj SQL upit ažurira stanje skladišta u tablici „skladiste“ na temelju trenutne količine sastojaka u skladištima. Upit koristi UPDATE naredbu koja se povezuje s podupitom, koji izračunava ukupne količine trenutnih i potrebnih sastojaka za svako skladište. U podupitu, koji koristi naredbe SUM(trenutna_kolicina) i SUM(potrebna_kolicina) za svako skladište, izračunavaju se ukupne trenutne i potrebne količine sastojaka, grupirane prema „skladiste_id“. Ovaj podupit se spaja s glavnom tablicom „skladiste“ putem JOIN naredbe, koja povezuje „skladiste.id“ s „skladiste_statistika.skladiste_id“. Na temelju rezultata izračuna, upit koristi SET naredbu i postavlja novo stanje skladišta. Naredbom CASE, stanje skladišta se ažurira prema sljedećim pravilima: ako je ukupna trenutna količina sastojaka (ukupno_trenutna) jednaka 0, stanje skladišta postaje 'prazno'; ako je ukupna trenutna količina manja od 50% potrebne količine, stanje postaje 'kritično'; ako je ukupna trenutna količina manja od potrebne, ali veća ili jednaka 50%, stanje postaje 'normalno'; i na kraju, ako je ukupna trenutna količina veća ili jednaka potrebnoj količini, stanje skladišta postaje 'puno'. Ovaj upit osigurava da stanje svakog skladišta bude ažurirano u skladu s trenutnom popunjenošću i potrebama skladišta za sastojcima. Na samom kraju se odabiru sve stavke iz tablice skladište, i vidi se su se sva stanja ažurirala prema popunjenosti skladišta.

## 12.3. Luka Vilagoš

### 12.3.1. Procedura - Ažuriranje stanja narudžbe

```
DELIMITER //

CREATE PROCEDURE zavrsi_narudzbu(IN narudzba_id INT)
BEGIN
    UPDATE narudzba
```

```
    SET status_narudzbe = 'ZAVRSENO'
    WHERE id = narudzba_id;

    IF ROW_COUNT() = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No rows updated';
    END IF;
END;
//

DELIMITER ;
```

Kreiramo proceduru “zavrsi_narudzbu” koja prima ulazni parametar narudzba_id tipa INT. Unutar BEGIN i END naredbi definiramo što će se dogoditi kada pozovemo ovu proceduru, a u ovom slučaju će to biti da ažuriramo status_narudzbe u tablici narudzba i postavljamo ga na “ZAVRSENO” tamo gdje je id u tablici narudzba jednak narudzba_id-u kojeg ćemo upisati prilikom poziva ove procedure. Kao dodatnu funkcionalnost, dodana je IF klauzula koja provjerava koliko će biti vraćenih redova. Ako je broj vraćenih redova 0, tada signaliziramo SQLSTATE ‘45000’ i korisniku šaljemo povratnu informaciju “No rows updated.”

## 12.3.2. Procedura - Procesiranje transakcije

```
CREATE PROCEDURE process_financial_transaction(
    IN p_restoran_id INT,
    IN p_iznos DECIMAL(10,2),
    IN p_naziv VARCHAR(256)
)
BEGIN
    UPDATE restoran_racun
    SET stanje = stanje + p_iznos
    WHERE restoran_id = p_restoran_id;

    INSERT INTO transakcija_restoran (restoran_racun_id, iznos, naziv)
    SELECT id, p_iznos, p_naziv
    FROM restoran_racun
    WHERE restoran_id = p_restoran_id;
END;
//
```

Kreiramo proceduru “process_financial_transaction” koja prima ulazne parametre p_restoran_id INT, p_iznos DECIMAL(10,2) i p_naziv VARCHAR(256). Ažurira se račun restorana tako da se stanje računa postavi na to trenutno stanje plus iznos nove transakcije.

## 12.3.3. Procedura - Procedura za novi račun

```
CREATE PROCEDURE process_racun_transaction(
    IN p_racun_id INT
)
BEGIN
    DECLARE v_iznos DECIMAL(10,2);
    DECLARE v_restoran_id INT;
```

```
    DECLARE v_broj_racuna VARCHAR(31);

    SELECT iznos + napojnica, restoran_id, COALESCE(broj_racuna, 'N/A')
    INTO v_iznos, v_restoran_id, v_broj_racuna
    FROM racun
    WHERE id = p_racun_id;

    CALL process_financial_transaction(
        v_restoran_id,
        v_iznos,
        CONCAT('Račun: ', v_broj_racuna)
    );
END;
//
```

Kreiramo proceduru process_racun_transaction koja prima ulazni parametar p_racun_id tipa INT. Unutar BEGIN i END naredbi definiramo što će se dogoditi kada pozovemo ovu proceduru, a u ovom slučaju, procedura dohvaća podatke o računu (uključujući iznos s napojnicom, restoran kojem račun pripada i broj računa) i koristi ih za pokretanje druge procedure za financijsku transakciju. Ova procedura dohvaća informacije o računu pomoću ulaznog parametra p_racun_id i koristi ih za iniciranje druge procedure (process_financial_transaction) koja obavlja financijsku transakciju za restoran. Na taj način procedura povezuje podatke o računu i financijske operacije u sustavu.

## 12.3.4. Procedura - Procedura za novi trošak

```
CREATE PROCEDURE process_trosak_transaction(
    IN p_trosak_id INT
)
BEGIN
    DECLARE v_iznos DECIMAL(10,2);
    DECLARE v_restoran_id INT;
    DECLARE v_naziv VARCHAR(64);

    SELECT iznos, restoran_id, naziv
    INTO v_iznos, v_restoran_id, v_naziv
    FROM trosak
    WHERE id = p_trosak_id;

    CALL process_financial_transaction(
        v_restoran_id,
        -v_iznos,
        CONCAT('Trošak: ', v_naziv)
    );
END;
//
```

Kreiramo proceduru process_trosak_transaction koja prima ulazni parametar p_trosak_id tipa INT. Unutar BEGIN i END naredbi definiramo što će se dogoditi kada pozovemo ovu proceduru, a u ovom slučaju dohvaćamo podatke o trošku iz tablice trosak i koristimo ih za iniciranje financijske transakcije pomoću procedure process_financial_transaction.

## 12.3.5. Procedura - Procedura za završenu narudžbu

```
CREATE PROCEDURE process_narudzba_transaction(
    IN p_narudzba_id INT
)
BEGIN
    DECLARE v_total_cost DECIMAL(10,2);
    DECLARE v_restoran_id INT;
    DECLARE v_naziv VARCHAR(31);

    SELECT
        n.naziv,
        SUM(sn.kolicina * s.cijena),
        sk.restoran_id
    INTO v_naziv, v_total_cost, v_restoran_id
    FROM narudzba n
    JOIN sastojak_narudzba sn ON sn.narudzba_id = n.id
    JOIN sastojak s ON s.id = sn.sastojak_id
    JOIN skladiste sk ON sk.id = n.skladiste_id
    WHERE n.id = p_narudzba_id
    GROUP BY n.naziv, sk.restoran_id;

    CALL process_financial_transaction(
        v_restoran_id,
        -v_total_cost,
        CONCAT('Narudžba: ', v_naziv)
    );
END;
//
```

Kreiramo proceduru process_narudzba_transaction koja prima ulazni parametar p_narudzba_id tipa INT. Unutar BEGIN i END naredbi definiramo što će se dogoditi kada pozovemo ovu proceduru. Ova procedura računa ukupnu cijenu narudžbe iz tablice narudzba, identificira restoran kojem pripada narudžba, i prosljeđuje te podatke u proceduru za financijske transakcije. Procedura process_narudzba_transaction automatski bilježi financijski trošak povezanu s narudžbom u restoran. Kada se pozove s ID-om narudžbe (p_narudzba_id), procedura dohvaća naziv narudžbe, izračunava ukupni trošak na temelju količine i cijene sastojaka, te šalje te podatke u proceduru za financijske transakcije. Transakcija se knjiži kao negativan saldo za restoran.

## 12.3.6. Procedura - Procedura za plaću korisnika

```
CREATE PROCEDURE process_zaposlenik_placa_transaction(
    IN p_zaposlenik_placa_id INT
)
BEGIN
    DECLARE v_iznos DECIMAL(10,2);
    DECLARE v_zaposlenik_id INT;
    DECLARE v_restoran_id INT;
    DECLARE v_mjesec DATE;
    DECLARE v_ime_prezime VARCHAR(63);
```

```
    SELECT zp.iznos, zp.zaposlenik_id, z.restoran_id, zp.mjesec,
           CONCAT(z.ime, ' ', z.prezime)
    INTO v_iznos, v_zaposlenik_id, v_restoran_id, v_mjesec, v_ime_prezime
    FROM zaposlenik_placa zp
    JOIN zaposlenik z ON z.id = zp.zaposlenik_id
    WHERE zp.id = p_zaposlenik_placa_id;

    CALL process_financial_transaction(
        v_restoran_id,
        -v_iznos,
        CONCAT('Plaća zaposlenika: ', v_ime_prezime)
    );

    INSERT INTO transakcija_zaposlenik (zaposlenik_id, iznos, naziv)
    VALUES (v_zaposlenik_id, v_iznos, CONCAT('Plaća za: ', DATE_FORMAT(v_mjesec,
'%Y-%m')));
END;
//

DELIMITER ;
```

Kreiramo proceduru process_zaposlenik_placa_transaction koja prima ulazni parametar p_zaposlenik_placa_id tipa INT. Unutar BEGIN i END naredbi definiramo što će se dogoditi kada pozovemo ovu proceduru, a u ovom slučaju obrađujemo transakciju isplate plaće zaposleniku, knjižimo je kao trošak za restoran, i spremamo podatke o transakciji u posebnu tablicu. Procedura process_zaposlenik_placa_transaction koristi ulazni parametar p_zaposlenik_placa_id za dohvaćanje podataka o plaći zaposlenika. Bilježi trošak isplate plaće za restoran pozivom na proceduru za financijske transakcije, a zatim evidentira transakciju u tablici transakcija_zaposlenik. Ovo omogućuje automatsko knjiženje troškova i stvaranje povijesti transakcija plaća za zaposlenike.

# 12.4. Alma Reka

## 12.5.1. Procedura - SastojciZaRecept

Procedura za ispisivanje svih sastojaka potrebnih za određeni recept, uključujući njihove količine i jedinice mjere.

```
DELIMITER $$
CREATE PROCEDURE SastojciZaRecept(receptID INT)
BEGIN
    SELECT
        s.naziv AS Sastojak,
        rs.kolicina AS PotrebnaKolicina,
        rs.kolicina_tip AS KolicinaTip
    FROM
        recept_sastojak rs
    JOIN
        sastojak s ON rs.sastojak_id = s.id
    WHERE
        rs.recept_id = receptID;
```

```
END$$
DELIMITER ;

SHOW PROCEDURE STATUS WHERE Name = 'SastojciZaRecept';
CALL SastojciZaRecept(2);
```

Ova SQL procedura SastojciZaRecept dizajnirana je kako bi olakšala dobivanje detaljnih informacija o sastojcima potrebnim za pripremu specifičnog recepta. Procedura prima receptID kao ulazni parametar, koji se koristi za filtriranje i izvlačenje relevantnih podataka iz tablice recept_sastojak. Procedura pridružuje tablicu sastojak s tablicom recept_sastojak kako bi se dobili nazivi sastojaka, njihove potrebne količine i tip količine (npr. grami, mililitri, komadi). Rezultat ove procedure nudi jasan i precizan pregled svih sastojaka potrebnih za izradu recepta.

```
SHOW PROCEDURE STATUS WHERE Name = 'SastojciZaRecept';
```

Ova naredba se koristi za prikaz statusa i informacija o proceduri SastojciZaRecept. Pomaže u verifikaciji da je procedura ispravno kreirana i dostupna unutar baze podataka, pružajući detalje poput imena, tipa, sigurnosnih prava, i druge važne informacije.

```
CALL SastojciZaRecept(2);
```

Naredba izvršava proceduru SastojciZaRecept s konkretnim primjerom, koristeći ID recepta 2 kao argument. Omogućava korisniku da izvrši proceduru i dobije popis i detalje sastojaka potrebnih za recept s ID-em 2, što je korisno za praktične potrebe u pripremi jela.

# 12.5. Patricia Lazić

## 12.5.1. Procedura - dohvaćanje detalja male nezgode

```
DELIMITER //

CREATE PROCEDURE detalji_male_nezgode (IN p_nezgoda_id INT)
BEGIN
    SELECT restoran_id, zaposlenik_id, ukupno, created_at
    FROM mala_nezgoda
    WHERE id = p_nezgoda_id;

    SELECT s.naziv AS sastojak, mns.kolicina
    FROM mala_nezgoda_sastojak mns
    JOIN sastojak s ON mns.sastojak_id = s.id
    WHERE mns.mala_nezogda_id = p_nezgoda_id;
END//

DELIMITER ;
```

Deklarirali smo proceduru pomoću naredbe CREATE PROCEDURE pod nazivom „detalji_male_nezgode“ koja prima jedan ulazni parametar (p_nezgoda_id) tipa integer što označava ID određene male nezgode. Svi parametri koje procedura prihvaća definirani su kao IN, što znači da se koriste isključivo za unos podataka u proceduru, bez mogućnosti vraćanja rezultata kroz te parametre. Odabrali smo osnovne informacije o maloj nezgodi kao što su: restoran u

kojem se nezgoda dogodila (restoran_id), zaposlenik koji je odgovoran za nezgodu (zaposlenik_id), ukupni trošak male nezgode (ukupno) i datum i vrijeme evidencije nezgode (created_at) s izabranim ID-em (p_nezgoda_id). Zatim smo iz tablice „mala_nezgoda_sastojak“ dohvatili sve sastojke koji su povezani s malom nezgodom i spojili s tablicom „sastojak“ na temelju ID-a kako bi dobili naziv svakog sastojka. Pomoću naredbe „WHERE mns.mala_nezgoda_id = p_nezgoda_id;“ dobivamo rezultate samo za one gdje je mns.mala_nezgoda_id jednak ulaznom parametru p_nezgoda_id  što drugim riječima znači da rezultat prikazuje samo sastojke koji su povezani s malom nezgodom čiji je ID jednak ID-u proslijeđene vrijednosti. Proceduru završavamo sa END // odnosno DELIMITER;. Pozivamo proceduru koristeći naredbu CALL i proslijedimo ulazni parametar: „CALL detalji_male_nezgode(1);“. Rezultat procedure su dvije tablice, u prvoj se prikazuju osnovne informacije male nezgode dok se u drugoj prikazuju sastojci i njihova količina povezana s malom nezgodom. Ova procedura pomaže u daljnjoj analizi troškova i praćenju troškova uzrokovanih nezgodama.

## 12.5.2. Procedura - unos nove male nezgode

```
DELIMITER //

CREATE PROCEDURE unos_male_nezgode (
    IN p_restoran_id INT,
    IN p_zaposlenik_id INT,
    IN p_ukupno DECIMAL(10, 2),
    IN sastojci JSON
)
BEGIN
    DECLARE nezgoda_id INT;
    DECLARE i INT DEFAULT 0;
    DECLARE sastojak_id INT;
    DECLARE kolicina INT;
    DECLARE sastojci_count INT;

    INSERT INTO mala_nezgoda (restoran_id, zaposlenik_id, ukupno)
    VALUES (p_restoran_id, p_zaposlenik_id, p_ukupno);

    SET nezgoda_id = LAST_INSERT_ID();

    SET sastojci_count = JSON_LENGTH(sastojci);

    WHILE i < sastojci_count DO
        SET sastojak_id = CAST(JSON_UNQUOTE(JSON_EXTRACT(sastojci, CONCAT('$[',
i, '].sastojak_id'))) AS UNSIGNED);
        SET kolicina = CAST(JSON_UNQUOTE(JSON_EXTRACT(sastojci, CONCAT('$[', i,
'].kolicina'))) AS UNSIGNED);

        INSERT INTO mala_nezgoda_sastojak (mala_nezgoda_id, sastojak_id,
kolicina)
        VALUES (nezgoda_id, sastojak_id, kolicina);

        SET i = i + 1;
    END WHILE;
END //
```

```
DELIMITER ;


    COMMIT;
END //

DELIMITER ;
```

Deklarirali smo proceduru pomoću naredbe CREATE PROCEDURE pod nazivom unos_male_nezgode koji prima parametre: ID restorana , ID zaposlenika, ukupni trošak male nezgode i sastojci to jest JSON polje koje sadrži sastojke povezane s nezgodom (ID sastojak i količina). Naredba IN omogućuje da se vrijednost tih parametra prosljeđuje proceduri kada se ona poziva ali ih ne može mijenjati izvan svog opsega. Deklariramo varijablu „nezgoda_id“ tipa integer koja pohranjuje ID nove male nezgode nakon umetanja u tablicu mala_nezgoda, zatim varijablu „i“ kojoj je default vrijednost 0 i ona služi kao brojač za iteraciju kroz JSON popis sastojaka. Potom smo deklarirali varijablu „sastojak_id“ tipa integer koja pohranjuje trenutni sastojak_id iz JSON podataka, ujedno smo deklarirali i varijablu „kolicina“ tipa integer koja pohranjuje trenutnu količinu iz JSON podataka i na kraju smo deklarirali varijablu „sastojci_count“ tipa integer koji pohranjuje ukupan broj elemenata u JSON popisu sastojaka. Pomoću naredbe INSERT INTO, dodali smo novu nezgodu u tablicu mala_nezgoda i zatim smo vrijednosti (VALUES) p_restoran_id, p_zaposlenik_id i p_ukupno unijeli u odgovarajući stupac što su restoran_id, zaposlenik_id i ukupno. Dohvaćamo posljednji uneseni ID pomoću funkcije LAST_INSERT_ID() što je u ovom slučaju ID retka koji je umetnut u tablicu mala_nezgoda i pomoću naredbe SET spremamo vrijednost u lokalnu varijablu „nezgoda_id“. JSON_LENGTH(sastojci) je funkcija koja računa broj elemenata unutar JSON niza ili objekta i parametar koji prosljeđujemo funkciji što su sastojci je JSON string. Vrijednost koju funkcija vrati se pomoću naredbe SET dodjeljuje lokalnoj varijabli „sastojci_count“. Pomoću te varijable možemo odrediti koliko se puta treba izvršiti petlja koja obrađuje JSON podatke. „WHILE i < sastojci_count DO“ omogućuje ponavljanje petlje sve dok je uvjet zadovoljen. Petlja iterira kroz svaki element JSON niza i prestaje kada su svi elementi obrađeni to jest prestaje kada je brojač „i“ veći ili jednak broju sastojka (sastojci_count). Unutar petlje dohvaćamo podatke o sastojku što su ID i količina iz JSON-a pomoću funkcije JSON_EXTRACT koja iz JSON niza ili objekta izvlači specifičnu vrijednost to jest vratit će vrijednost sastojak_id za odgovarajući element u polju na primjer ako je prvi element niza s sastojak_id 1, funkcija će vratiti 1. „Sastojci“ je ulazni JSON niz koji sadrži podatke o sastojcima npr. {"sastojak_id": 2, "kolicina": 3} i koristimo CONCAT funkciju koja spaja stringove. JSON_UNQUOTE() funkcija uklanja navodnike ako je riječ o stringu čime osigurava da vrijednost u stringu bude pretvorena u broj i CAST omogućuje pretvorbu jednog tipa podatka u drugi što je u ovom slučaju AS UNSIGNED, samo pozitivni brojevi. Na kraju, sastojak_id će sadržavati vrijednost ID-a sastojka za trenutni element u JSON nizu. Isti princip je i za „kolicina“, znači JSON_EXTRACT dohvaća polje „kolicina“ unutar svakog sastojka, uklanjaju se navodnici i rezultat „kolicina“ će sadržavati količinu sastojka kao cijeli broj. Prilikom svake iteracije naredbom INSERT INTO unosimo podatke o sastojku u tablicu mala_nezgoda_sastojak što znači da će za svaki element u JSON nizu unijeti novi zapis u tablicu. Pomoću naredbe SET povećavamo varijablu „i“ za 1 nakon svake iteracije i time pristupamo sljedećem elementu u JSON nizu. Proceduru pozivamo pomoću naredbe CALL na primjer: CALL unos_male_nezgode(1, 2, 130.75, '[{"sastojak_id": 2, "kolicina": 3}, {"sastojak_id": 4, "kolicina": 5}]');. Ako otvorimo tablicu „mala_nezgoda“ vidjet ćemo podatke koje smo unijeli prilikom poziva procedure. Proceduru završavamo s END // DELIMITER.

### 12.5.3. Procedura - unos nove velike nezgode

```
DROP PROCEDURE IF EXISTS unos_velike_nezgode;

DELIMITER //

CREATE PROCEDURE unos_velike_nezgode (
    IN p_restoran_id INT,
    IN p_zaposlenik_id INT,
    IN stavke JSON
)
BEGIN
    DECLARE nezgoda_id INT;
    DECLARE i INT DEFAULT 0;
    DECLARE stavka_id INT;
    DECLARE kolicina INT;
    DECLARE stavke_count INT;
    DECLARE nova_cijena DECIMAL(10, 2);
    DECLARE ukupna_vrijednost DECIMAL(10, 2) DEFAULT 0;

    INSERT INTO velika_nezgoda (restoran_id, zaposlenik_id)
    VALUES (p_restoran_id, p_zaposlenik_id);

    SET nezgoda_id = LAST_INSERT_ID();

    SET stavke_count = JSON_LENGTH(stavke);

    WHILE i < stavke_count DO
        SET stavka_id = CAST(JSON_UNQUOTE(JSON_EXTRACT(stavke, CONCAT('$[', i,
'].stavka_id'))) AS UNSIGNED);
        SET kolicina = CAST(JSON_UNQUOTE(JSON_EXTRACT(stavke, CONCAT('$[', i,
'].kolicina'))) AS UNSIGNED);

        SELECT cijena INTO nova_cijena FROM stavka WHERE id = stavka_id;

        INSERT INTO velika_nezgoda_stavka (velika_nezgoda_id, stavka_id,
kolicina)
        VALUES (nezgoda_id, stavka_id, kolicina);

        SET ukupna_vrijednost = ukupna_vrijednost + (nova_cijena * CAST(kolicina
AS DECIMAL(10, 2)));

        SET i = i + 1;
    END WHILE;

    UPDATE velika_nezgoda
    SET ukupno = ukupna_vrijednost
    WHERE id = nezgoda_id;

    COMMIT;
END //

DELIMITER ;
```

DROP PROCEDURE osigurava da procedura ne postoji , a ako procedura unos_velike_nezgode već postoji, briše se. Deklarirali smo proceduru pomoću naredbe CREATE PROCEDURE pod nazivom unos_velike_nezgode koja prima parametre p_restoran_id, p_zaposlenik_id i stavke to jest JSON polje koji sadrži popis stavki povezanih s nezgodom ( stavka_id i kolicina). Deklariramo varijablu „nezgoda_id“ tipa integer koja pohranjuje ID novounesene velike nezgode, zatim varijablu „i“ kojoj je default vrijednost 0 i ona služi kao brojač za iteraciju kroz stavke u JSON objektu potom varijabla stavka_id tipa integer koja označuje ID pojedinačne stavke iz JSON-a. Također smo deklarirali i varijablu „kolicina“, varijablu „stavke_count“ koja iznosi ukupan broj stavki u JSON polju, varijablu „nova_cijena“ koja predstavlja cijenu pojedine stavke tipa DECIMAL i „ukupna_vrijednost“ koja predstavlja ukupnu vrijednost svih stavki povezanih s velikom nezgodom. Pomoću naredbe INSERT INTO, dodali smo novu nezgodu u tablicu velika_nezgoda i zatim smo vrijednosti (VALUES) p_restoran_id, p_zaposlenik_id unijeli u odgovarajući stupac što su restoran_id, zaposlenik_id. Dohvaćamo posljednji uneseni ID pomoću funkcije LAST_INSERT_ID() što je u ovom slučaju ID nove nezgode i pomoću naredbe SET spremamo vrijednost u lokalnu varijablu „nezgoda_id“. JSON_LENGTH(stavke) je funkcija koja računa broj elemenata (stavki) unutar JSON objekta stavke i pohranjuje ga u stavke_count. Pomoću te varijable možemo odrediti koliko se puta treba izvršiti petlja koja obrađuje JSON podatke. WHILE i < stavke_count DO petlja iterira kroz sve stavke unutar JSON objekta i izvršava se dok brojač „i“ ne dostigne broj stavki. Dohvaćamo podatke iz JSON-a koristeći JSON_EXTRACT koji dohvaća vrijednost iz JSON objekta za polje stavka_id i kolicina i JSON_UNQUOTE koji uklanja dodatne navodnike iz JSON vrijednosti. Pomoću CAST(... AS UNSIGNED) pretvaramo dohvaćene vrijednosti u numerički tip (stavka_id i kolicina). Dohvaćamo cijenu stavke iz tablice stavka na temelju stavka_id i pohranjujemo u varijablu nova_cijena i unosimo novu stavku povezanu s trenutnom velikom nezgodom u tablicu velika_nezgoda_stavka. Varijabla „nova_cijena“ je cijena pojedine stavke koja je dohvaćena iz tablice stavka i množimo cijenu stavke s njenom količinom kako bismo dobili ukupnu vrijednost te stavke. Pomoću naredbe CAST pretvara „kolicina“ iz cijelog broja u decimalni broj s preciznošću od 2 decimalna mjesta. Na kraju nova_cijena se množi sa količinom i time dobivamo ukupnu vrijednost pojedine stavke i pohranjuje se u varijablu „ukupna_vrijednost“. Pomoću naredbe SET povećavamo varijablu „i“ za 1 nakon svake iteracije i time pristupamo sljedećem elementu u JSON objektu. Nakon što su sve stavke obrađene, ažurira se kolona ukupno u tablici „velika_nezgoda“ postavljajući je na ukupnu vrijednost svih stavki. Završavamo transakciju s COMMIT koja potvrđuje sve promjene napravljene tijekom izvršavanja procedure i zatvaramo proceduru s END // DELIMITER. Ova procedura omogućava dodavanje nove "velike nezgode" zajedno sa svim pripadajućim stavkama, pri čemu automatski računa ukupnu vrijednost stavki.

# 13. Okidači

## 13.1. Paula Vorih

### 13.1.1. Okidač

```
DELIMITER //

CREATE TRIGGER after_narudzba_zavrseno
AFTER UPDATE ON narudzba
FOR EACH ROW
BEGIN
```

```
    IF NEW.status_narudzbe = 'ZAVRSENO' AND OLD.status_narudzbe <> 'ZAVRSENO'
THEN
        UPDATE sastojak s
        JOIN sastojak_narudzba sn ON s.id = sn.sastojak_id
        SET s.trenutna_kolicina = s.trenutna_kolicina + sn.kolicina
        WHERE sn.narudzba_id = NEW.id;
    END IF;
END;
//

DELIMITER ;
```

Ovaj okidač okida se za svaki red nakon svakog ažuriranja u tablici narudzba. Njegova funkcionalnost aktivira se samo kada se status narudžbe promijeni u 'ZAVRSENO'. U ovom slučaju ažurira se količina sastojaka u tablici sastojak. Koristeći UPDATE, ažurira se polje trenutna_kolicina u tablici sastojak tako da se na postojeću količinu pribroji količina iz tablice sastojak_narudzba za sastojke koji pripadaju toj narudžbi. Mehanizam osigurava da se promjene događaju isključivo za narudžbu koja je upravo ažurirana (identificirana preko NEW.id).

# 13.2. Dinko Nađ

## 13.2.1. Okidač - Ažurira stanje skladišta prilikom unosa podataka u tablicu sastojak

```
drop trigger if exists after_sastojak;
DELIMITER //
CREATE TRIGGER after_sastojak
AFTER INSERT ON sastojak
FOR EACH ROW
BEGIN
    DECLARE ukupna_trenutna_kolicina INT;
    DECLARE ukupna_potrebna_kolicina INT;
    DECLARE postotak INT;
    DECLARE novo_stanje ENUM('puno', 'prazno', 'kritično', 'normalno');

    SELECT SUM(trenutna_kolicina), SUM(potrebna_kolicina)
    INTO ukupna_trenutna_kolicina, ukupna_potrebna_kolicina
    FROM sastojak
    WHERE skladiste_id = NEW.skladiste_id;

    IF ukupna_potrebna_kolicina > 0 THEN
        SET postotak = (ukupna_trenutna_kolicina / ukupna_potrebna_kolicina) *
100;

            IF postotak = 0 THEN
                SET novo_stanje = 'prazno';
            ELSEIF postotak <= 50 THEN
                SET novo_stanje = 'kritično';
            ELSEIF postotak < 100 THEN
                SET novo_stanje = 'normalno';
            ELSE
                SET novo_stanje = 'puno';
```

```
            END IF;


        UPDATE skladiste
            SET stanje = novo_stanje
        WHERE id = NEW.skladiste_id;

    ELSE
        UPDATE skladiste
            SET stanje = 'prazno'
        WHERE id = NEW.skladiste_id;
    END IF;
END //
DELIMITER ;


INSERT INTO sastojak (skladiste_id, naziv, cijena, kolicina_tip,
potrebna_kolicina, trenutna_kolicina) VALUES
(18, 'Maslac', 7.00, 'g', 1000, 550);

select * from skladiste;

#AŽURIRA SE NAKON INSERT SASTOJAK
```

Ovaj SQL kod definira trigger koji automatski ažurira stanje skladišta nakon unosa novog sastojka u tablicu "sastojak". Triger se naziva "after_sastojak" i aktivira se nakon što se izvrši naredba INSERT na tablicu "sastojak". Prvo, sa naredbom DROP TRIGGER IF EXISTS "after_sastojak" se briše prethodni triger ako već postoji. Triger se kreira sa naredbom CREATE TRIGGER "after_sastojak" koji će se postaviti da se izvrši AFTER INSERT na tablicu "sastojak", što znači da će se triger aktivirati nakon što se novi red unese u tablicu. Triger koristi FOR EACH ROW, tome se osigurava da će se pokrenuti za svaki red koji bude umetnut. Deklariraju se return varijable: "ukupna_trenutna_kolicina" - za sumu trenutnih količina sastojaka u određenom skladištu, "ukupna_potrebna_kolicina" - za sumu potrebnih količina tih sastojaka, "postotak" - postotak popunjenosti skladišta temeljen na odnosu trenutnih i potrebnih količina i "novo_stanje" koje vraća novo stanje skladišta, koje može biti jedna od sljedećih vrijednosti: 'puno', 'prazno', 'kritično', 'normalno'. Triger koristi SELECT naredbu kako bi izračunao ukupnu trenutnu količinu i potrebnu količinu za skladište povezano s novim unosom u tablici "sastojak", koristeći NEW."skladiste_id" kako bi se odredilo koje skladište je u pitanju. Ako ukupna potrebna količina veća od nule, izračunava se postotak popunjenosti skladišta prema sljedećoj formuli (ukupna_trenutna_kolicina / ukupna_potrebna_kolicina) * 100. Prema dobivenim rezultatima stanje skladišta se postavlja prema sljedećim uvjetima: ako je postotak 0, stanje skladišta postaje 'prazno'; ako je postotak manji ili jednak 50, stanje skladišta postaje 'kritično; ako je postotak manji od 100, ali veći od 50, stanje skladišta postaje 'normalno'; ako je postotak 100 ili veći, stanje skladišta postaje 'puno'. Ako je ukupna potrebna količina jednaka nuli, stanje skladišta se postavlja na 'prazno'. Zatim, pomoću UPDATE naredbe, ažurira se stanje skladišta u tablici "skladiste" temeljem izračunatog stanja, koristeći NEW."skladiste_id" za identifikaciju odgovarajućeg skladišta. Završavamo trigger sa END // odnosno DELIMITER;

INSERT INTO "sastojak" služi za dodavanje novog sastojka u tablicu "sastojak" sa zadanim podacima, a zatim se pomoću SELECT naredbe provjerava jel se ažurirao kod.

### 13.2.2. Okidač - Ažurira se nakon obavljanja update na tablici sastojci

```
Drop trigger if exists after_sastojak;
DELIMITER //
CREATE TRIGGER after_sastojak
AFTER UPDATE ON sastojak
FOR EACH ROW
BEGIN
    DECLARE ukupna_trenutna_kolicina INT;
    DECLARE ukupna_potrebna_kolicina INT;
    DECLARE postotak INT;
    DECLARE novo_stanje ENUM('puno', 'prazno', 'kritično', 'normalno');
    SELECT SUM(trenutna_kolicina), SUM(potrebna_kolicina)
    INTO ukupna_trenutna_kolicina, ukupna_potrebna_kolicina
    FROM sastojak
    WHERE skladiste_id = NEW.skladiste_id;
    IF ukupna_potrebna_kolicina > 0 THEN
        SET postotak = (ukupna_trenutna_kolicina / ukupna_potrebna_kolicina) *
100;
        IF postotak = 0 THEN
            SET novo_stanje = 'prazno';
        ELSEIF postotak <= 50 THEN
            SET novo_stanje = 'kritično';
        ELSEIF postotak < 100 THEN
            SET novo_stanje = 'normalno';
        ELSE
            SET novo_stanje = 'puno';
        END IF;

        UPDATE skladiste
        SET stanje = novo_stanje
        WHERE id = NEW.skladiste_id;

    ELSE
        UPDATE skladiste
        SET stanje = 'prazno'
        WHERE id = NEW.skladiste_id;
    END IF;
END //

DELIMITER ;

CALL azuriraj_kolicinu_sastojka(18,"Riba", 0);
select * from skladiste;
select * from sastojak;
```

Ovaj SQL kod definira trigger koji automatski ažurira stanje skladišta nakon ažuriranja sastojka u tablicu "sastojak". Triger se naziva " after_update_sastojak" i aktivira se nakon što se izvrši naredba UPDATE na tablicu "sastojak". Prvo, sa naredbom DROP TRIGGER IF EXISTS " after_update_sastojak" se briše prethodni triger ako već postoji. Triger se kreira sa naredbom CREATE TRIGGER " after_update_sastojak" koji će se postaviti da se izvrši AFTER UPDATE na tablicu "sastojak", što znači da će se triger aktivirati nakon ažuriraju podatci u tablici. Triger koristi FOR

EACH ROW, tome se osigurava da će se pokrenuti za svaki red koji bude umetnut. Deklariraju se return varijable: "ukupna_trenutna_kolicina" - za sumu trenutnih količina sastojaka u određenom skladištu, "ukupna_potrebna_kolicina" - za sumu potrebnih količina tih sastojaka, "postotak" - postotak popunjenosti skladišta temeljen na odnosu trenutnih i potrebnih količina i "novo_stanje" koje vraća novo stanje skladišta, koje može biti jedna od sljedećih vrijednosti: 'puno', 'prazno', 'kritično', 'normalno'. Triger koristi SELECT naredbu kako bi izračunao ukupnu trenutnu količinu i potrebnu količinu za skladište povezano s novim unosom u tablici "sastojak", koristeći NEW."skladiste_id" kako bi se odredilo koje skladište je u pitanju. Ako ukupna potrebna količina veća od nule, izračunava se postotak popunjenosti skladišta prema sljedećoj formuli (ukupna_trenutna_kolicina / ukupna_potrebna_kolicina) * 100. Prema dobivenim rezultatima stanje skladišta se postavlja prema sljedećim uvjetima: ako je postotak 0, stanje skladišta postaje 'prazno'; ako je postotak manji ili jednak 50, stanje skladišta postaje 'kritično; ako je postotak manji od 100, ali veći od 50, stanje skladišta postaje 'normalno'; ako je postotak 100 ili veći, stanje skladišta postaje 'puno'. Ako je ukupna potrebna količina jednaka nuli, stanje skladišta se postavlja na 'prazno'. Zatim, pomoću UPDATE naredbe, ažurira se stanje skladišta u tablici "skladiste" temeljem izračunatog stanja, koristeći NEW."skladiste_id" za identifikaciju odgovarajućeg skladišta. Završavamo trigger sa END // odnosno DELIMITER;
INSERT INTO "sastojak" služi za dodavanje novog sastojka u tablicu "sastojak" sa zadanim podacima, a zatim se pomoću SELECT naredbe provjerava jel se ažurirao kod.

## 13.2.3. Okidač - Ažurira se nakon obavljanja update na tablici sastojci

```
DROP TRIGGER IF EXISTS update_iznos_velika_nezgoda;

DELIMITER //

CREATE TRIGGER update_iznos_velika_nezgoda
AFTER INSERT ON velika_nezgoda_stavka
FOR EACH ROW
BEGIN
    DECLARE stavka_cijena DECIMAL(10, 2);

    SELECT cijena INTO stavka_cijena
    FROM stavka
    WHERE id = NEW.stavka_id;

    UPDATE velika_nezgoda
    SET ukupno = ukupno + (stavka_cijena * NEW.kolicina)
    WHERE id = NEW.velika_nezgoda_id;
END//

DELIMITER ;
```

DROP TRIGGER IF EXISTS služi za brisanje postojećeg okidača pod nazivom update_iznos_velika_nezgoda ako postoji u bazi podataka  Stvorimo okidač koristeći naredbu CREATE TRIGGER koja se zove update_iznos_velika_nezgoda Okidač se pokreće nakon umetanja (AFTER INSERT) svakog retka u tablicu velika_nezgoda_stavka. Okidač koristi FOR EACH ROW što znači da će se aktivirati za svaki pojedinačni redak umetnut u tablicu. Za pohranu cijene stavke koji je povezan s umetnutim redom deklarirali smo varijablu stavka_cijena kojeg je tipa DECIMAL(10,2)

što znači da broj može sadržavati ukupno 10 znamenki i dvije znamenke su za decimalni dio. Dohvatili smo cijenu stavke iz tablice stavka koja zadovoljava uvjet WHERE id = NEW.stavka_id i pohranjujemo tu vrijednost pomoću INTO u varijablu stavka_cijena. NEW je posebna referenca u okidačima koja predstavlja podatke iz novog retka koji je umetnut u tablicu na kojoj se okidač aktivira što je u ovom slučaju ID stavke= stavka_id iz retka koji je umetnut u tablicu velika_nezgoda_stavka. Ažuriramo podatke u tablici velika_nezgoda koristeći UPDATE velika_nezgoda i SET specifično stupac ukupno. Cijenu pojedine stavke (stavka_cijena) koju smo dohvatili ranije množimo s količinom stavke iz umetnutog retka u tablici velika_nezgoda_stavka (NEW.kolicina) i dodajemo u postojeću vrijednost u stupcu ukupno. WHERE id = NEW.velika_nezgoda_id definira koji redak se treba ažurirati u tablici velika_nezgoda to jest id iz tablice velika_nezgoda uspoređuje s vrijednosti iz umetnutog retka koja specificira kojoj nezgodi pripada stavka koja je umetnuta (NEW.velika_nezgoda_id). Završavamo okidač sa END// DELIMITER. Svrha okidača je da se automatski ažurira ukupan iznos u tablici velika_nezgoda svaki put kada se doda novi redak u tablicu velika_nezgoda_stavka.

## 13.3. Luka Vilagoš

### 13.3.1. Okidač - ažuriranje stanja na restoranskom računu kada se dogodi nova transakcija u restoranu

```
DELIMITER //

CREATE TRIGGER after_narudzba_zavrseno
AFTER UPDATE ON narudzba
FOR EACH ROW
BEGIN
    IF NEW.status_narudzbe = 'ZAVRSENO' AND OLD.status_narudzbe <> 'ZAVRSENO'
THEN
        UPDATE sastojak s
        JOIN sastojak_narudzba sn ON s.id = sn.sastojak_id
        SET s.trenutna_kolicina = s.trenutna_kolicina + sn.kolicina
        WHERE sn.narudzba_id = NEW.id;
    END IF;
END;
//

DELIMITER ;
```

Ovaj okidač okida se za svaki red nakon svakog ažuriranja vrijednosti u tablici “narudzba”. Ako je novi status narudžbe jednak “ZAVRSENO” i stari status narudžbe se pretvara u “ZAVRSENO”, a do tada nije bio, onda je potrebno ažurirati sastojak (s) koji je pridružen sa tablicom sastojak_narudzba (sn) tako da je s.id jednak sn.sastojak_id. Postavljamo trenutnu količinu u tablici sastojak na trenutnu količinu pribrojenu količini iz sastojak_narudzba.kolicina tako da se sn.narudzba_id podudara s ažuriranim id-em narudžbe.

### 13.3.2. Okidač - Ažuriranje količine sastojaka nakon male nezgode

```
DELIMITER //

CREATE TRIGGER after_mala_nezgoda_sastojak_insert
AFTER INSERT ON mala_nezgoda_sastojak
FOR EACH ROW
BEGIN
    UPDATE sastojak s
    JOIN mala_nezgoda_sastojak mns ON s.id = mns.sastojak_id
    SET s.trenutna_kolicina = s.trenutna_kolicina - NEW.kolicina
    WHERE mns.mala_nezgoda_id = NEW.mala_nezgoda_id AND mns.sastojak_id =
NEW.sastojak_id;
END;
//

DELIMITER ;
```

Ovaj okidač okida se za svaki red nakon svakog umetanja vrijednosti u tablicu “mala_nezgoda_sastojak”. Ažuriramo količinu sastojaka tako da od trenutne količine sastojaka oduzmemo novu količinu sastojaka koji su kompromizirani u maloj nezgodi.

### 13.3.3. Okidač - Ažuriranje količine sastojaka nakon velike nezgode

```
DELIMITER //
CREATE TRIGGER after_velika_nezgoda_stavka_insert
AFTER INSERT ON velika_nezgoda_stavka
FOR EACH ROW
BEGIN
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_sastojak_id INT;
    DECLARE v_kolicina DECIMAL(10, 2);
    DECLARE v_recept_id INT;

    DECLARE cur CURSOR FOR
        SELECT
            rs.sastojak_id,
            rs.kolicina * NEW.kolicina
        FROM recept_sastojak rs
        WHERE rs.recept_id = (SELECT recept_id FROM stavka WHERE id =
NEW.stavka_id);

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;

    SELECT recept_id INTO v_recept_id
    FROM stavka
    WHERE id = NEW.stavka_id;

    OPEN cur;
```

```
    read_loop: LOOP
        FETCH cur INTO v_sastojak_id, v_kolicina;

        IF v_done THEN
            LEAVE read_loop;
        END IF;

        UPDATE sastojak
        SET trenutna_kolicina = trenutna_kolicina - CEILING(v_kolicina)
        WHERE id = v_sastojak_id;

    END LOOP;

    CLOSE cur;
END;
//
DELIMITER ;
```

U ovom okidaču se koristi kursor. Kursor je ovdje potreban jer tablica recept_sastojak može sadržavati više sastojaka za jedan recept te svaki sastojak mora biti procesiran sam za sebe kako bi se pravilno ažurirala tablica trenutna_kolicina za svaki sastojak. Imamo lokalne varijable v_sastojak_id i v_kolicina koje drže potreban id i količinu tjekom svake iteracije petlje. v_recept_id pohranjuje id recepta. Varijabla v_done je potrebna za determiniranje trenutka kada su dohvaćeni svi redovi iz kursora. Svako dohvaćanje kursora uzima id sastojka i ukupnu količinu iz tablice recept_sastojak. Ako više nema redova za dohvatiti, v_done se postavlja na TRUE i izlazi se iz petlje. Na kraju se za svaki sastojak (v_sastojak_id) smanjuje trenutna_kolicina za zaokruženu vrijednost v_kolicina na višu vrijednost CEILING(v_kolicina).

## 13.3.4. Okidač - Ažuriranje stanja sastojaka nakon stvaranja računa

```
DELIMITER //
CREATE TRIGGER after_stavka_racun_insert
AFTER INSERT ON stavka_racun
FOR EACH ROW
BEGIN
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_sastojak_id INT;
    DECLARE v_kolicina DECIMAL(10, 2);
    DECLARE v_recept_id INT;

    DECLARE cur CURSOR FOR
        SELECT
            rs.sastojak_id,
            rs.kolicina * NEW.kolicina
        FROM stavka s
        JOIN recept_sastojak rs ON rs.recept_id = s.recept_id
        WHERE s.id = NEW.stavka_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;
```

```
    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO v_sastojak_id, v_kolicina;

        IF v_done THEN
            LEAVE read_loop;
        END IF;

        UPDATE sastojak
        SET trenutna_kolicina = trenutna_kolicina - CEILING(v_kolicina)
        WHERE id = v_sastojak_id;

    END LOOP;

    CLOSE cur;
END;
//
DELIMITER ;
```

I u ovom okidaču se koristi kursor. Kursor je potreban kako bi mogli procesirati svaki red, jer recept za određenu stavku može imati više sastojaka. On nam i služi kako bi mogli kasnije raditi operacije poput zaokruživanja na višu vrijednost. Imamo slične lokalne varijable. v_sastojak_id sprema id sastojka koji se ažurira, v_kolicina je ukupna količina sastojka koja je potrebna za recept, v_recept_id je recept povezan sa stavkom te v_done koja služi za izlazak iz petlje kada se njeno stanje postavi na TRUE.

# 13.4. Sara Maljić

## 13.4.1. Okidač - Validacija dodjele stola i rezervacije

```
DELIMITER //
CREATE TRIGGER prije_rezervacije
BEFORE INSERT ON rezervacija
FOR EACH ROW
BEGIN
    IF NOT provjeri_dostupnost_stola(NEW.stol_id, NEW.vrijeme) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stol nije dostupan za
rezervaciju';
    END IF;
END;
//
DELIMITER ;
```

Okidač prije_rezervacije je SQL trigger koji se pokreće prije nego što se izvrši umetanje nove rezervacije u tablicu rezervacija. Cilj ovog okidača je provjeriti dostupnost stola za određeno vrijeme prije nego što se rezervacija unese u bazu podataka. Konkretno, okidač koristi funkciju provjeri_dostupnost_stola kako bi provjerio je li stol koji se pokušava rezervirati (prema NEW.stol_id) dostupan u odabranom vremenu (prema NEW.vrijeme). Ako funkcija provjeri_dostupnost_stola vrati FALSE, što znači da stol nije dostupan, okidač generira grešku korištenjem naredbe SIGNAL SQLSTATE '45000', koja prekida izvršavanje unosa i prikazuje poruku

'Stol nije dostupan za rezervaciju'. Ovo omogućava sprečavanje unosa nevažeće rezervacije u slučajevima kada stol nije slobodan za odabrano vrijeme. U suprotnom, ako je stol dostupan, rezervacija će biti uspješno dodana u tablicu. Ovaj okidač osigurava integritet podataka i sprječava konfliktne rezervacije za iste stolove u isto vrijeme.

# 13.5. Alma Reka

## 13.5.1. Okidač - ValidacijaRačun

Okidač koji provjerava valjanost iznosa računa prije njegovog unošenja u bazu podataka.

```
DELIMITER $$
CREATE TRIGGER ValidacijaRacun BEFORE INSERT ON racun
FOR EACH ROW
BEGIN
    IF NEW.iznos < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Iznos racuna ne moze biti manji od nule';
    END IF;
END$$
DELIMITER ;
```

Ovaj SQL okidač, nazvan ValidacijaRacun, aktivira se prije svakog pokušaja unošenja novog zapisa u tablicu racun. Njegova osnovna funkcija je osigurati da iznos računa ne bude negativan, što bi bilo nevaljano za financijske transakcije. Ako se pokuša unijeti račun s negativnim iznosom, okidač prekida proces unošenja i vraća grešku s jasnom porukom "Iznos racuna ne moze biti manji od nule". Ovo se postiže korištenjem SQL naredbe SIGNAL koja omogućava izbacivanje grešaka s definiranim SQLSTATE i porukom.

Ovakva upotreba okidača pomaže u održavanju integriteta podataka unutar baze, sprječavajući greške pri unosu koje bi mogle dovesti do pogrešnih financijskih izvještaja ili analiza. Okidač automatski štiti podatke bez potrebe za ručnom provjerom u aplikaciji ili softveru koji koristi bazu podataka.

## 13.5.1. Indeks

# 13.6. Patricia Lazić

## 13.6.1. Okidač - Automatsko ažuriranje ukupnog iznosa u tablici mala_nezgoda kada se dodaje novi sastojak

```
DELIMITER //

CREATE TRIGGER update_iznos
AFTER INSERT ON mala_nezgoda_sastojak
FOR EACH ROW
BEGIN
```

```
    DECLARE sastojak_cijena DECIMAL(10, 2);

    SELECT cijena INTO sastojak_cijena
    FROM sastojak
    WHERE id = NEW.sastojak_id;

    UPDATE mala_nezgoda
    SET ukupno = ukupno + (sastojak_cijena * NEW.kolicina)
    WHERE id = NEW.mala_nezgoda_id;
END//

DELIMITER ;
INSERT INTO mala_nezgoda_sastojak (mala_nezgoda_id, sastojak_id, kolicina)
VALUES (2, 4, 3);
SELECT * FROM mala_nezgoda_sastojak
```

Stvorimo okidač koristeći naredbu CREATE TRIGGER koja se zove update_iznos. Okidač se pokreće nakon umetanja (AFTER INSERT) svakog retka u tablicu mala_nezgoda_sastojak. Okidač koristi FOR EACH ROW što znači da će se aktivirati za svaki pojedinačni redak umetnut u tablicu. Za pohranu cijene sastojka koji je povezan s umetnutim redom deklarirali smo varijablu sastojak_cijena kojeg je tipa DECIMAL(10,2) što znači da broj može sadržavati ukupno 10 znamenki i dvije znamenke su za decimalni dio. Dohvatili smo cijenu sastojka iz tablice sastojak koja zadovoljava uvjet WHERE id = NEW.sastojak_id i pohranjujemo tu vrijednost pomoću INTO u varijablu sastojak_cijena. NEW je posebna referenca u okidačima koja predstavlja podatke iz novog retka koji je umetnut u tablicu na kojoj se okidač aktivira što je u ovom slučaju ID sastojka = sastojak_id iz retka koji je umetnut u tablicu mala_nezgoda_sastojak. Ažuriramo podatke u tablici mala_nezgoda koristeći UPDATE mala_nezgoda i SET specifično stupac ukupno. Cijenu pojedinog sastojka (sastojak_cijena) koju smo dohvatili ranije množimo s količinom sastojka iz umetnutog retka u tablici mala_nezgoda_sastojak (NEW.kolicina) i dodajemo u postojeću vrijednost u stupcu ukupno. WHERE id = NEW.mala_nezogda_id definira koji redak se treba ažurirati u tablici mala_nezgoda to jest id iz tablice mala_nezgoda uspoređuje s vrijednosti iz umetnutog retka koja specificira kojoj nezgodi pripada sastojak koji je umetnut (NEW.mala_nezogda_id). Završavamo okidač sa END// DELIMITER. Pomoću INSERT INTO mala_nezgoda_sastojak dodajemo novi redak u tablicu sa zadanim vrijednostima i koristeći SELECT testiramo je li se kod ažurirao. Svrha okidača je da se automatski ažurira ukupan iznos u tablici mala_nezgoda svaki put kada se doda novi redak (količina i cijena sastojka) u tablicu mala_nezgoda_sastojak.

# 14. Generiranje podataka

Podaci se generiraju kroz aplikaciju primijenjenu za ovaj sustav.

# Dokumentacija Flask Aplikacije

## Struktura Projekta

```
app/
```

```
    __init__.py
    config/
        __init__.py
        config.py
    interfaces/
        __init__.py
    router/
        __init__.py
        sql_routes.py
    routes/
        __init__.py
        recept_routes.py
        narudzba_routes.py
        trosak_routes.py
        zaposlenik_routes.py
        ...
    services/
        __init__.py
        recept_service.py
        narudzba_service.py
        trosak_service.py
        zaposlenik_service.py
        ...
    static/
        css/
        js/
        images/
    templates/
        index.html
        recept_create.html
        narudzba_create.html
        trosak_create.html
        zaposlenik_create.html
        ...
    utils/
        __init__.py
        sql_utils.py
certificates/
main.py
README.md
requirements.txt
sql/
    auth/
    jelovnik/
    mala_nezgoda/
    narudzba/
    racun/
    recept/
    restoran/
    restoran_racun/
    rezervacija/
    sastojak/
    skladiste/
    stavka/
    stol/
    transakcija_restoran/
```

```
    transakcija_zaposlenik/
    trosak/
venv/
```

# Opis Projekta

Ovaj projekt je web aplikacija za upravljanje restoranom koja omogućuje korisnicima upravljanje narudžbama, računima, receptima, zaposlenicima, skladištima i troškovima. Aplikacija je razvijena koristeći Flask framework i MySQL bazu podataka.

# Instalacija

1. Klonirajte repozitorij:

```
git clone <URL_REPOZITORIJA>
cd <NAZIV_REPOZITORIJA>
```

2. Kreirajte virtualno okruženje i aktivirajte ga:

```
python -m venv venv
source venv/bin/activate  # Na Windowsima: venv\Scripts\activate
```

3. Instalirajte potrebne pakete:

```
pip install -r requirements.txt
```

4. Pokrenite aplikaciju:

```
flask run
```

# Konfiguracija

Konfiguracijske datoteke se nalaze u direktoriju `app/config`. Ovdje možete postaviti različite parametre aplikacije kao što su baza podataka, API ključevi i drugi parametri.

# Struktura Koda

## `app/__init__.py`

Ova datoteka inicijalizira Flask aplikaciju i registrira sve potrebne blueprintove i rute.

```
from flask import Flask
from app.routes.recept_routes import recept_routes
from app.routes.narudzba_routes import narudzba_routes
from app.routes.trosak_routes import trosak_routes
from app.routes.zaposlenik_routes import zaposlenik_routes
```

```
# Import drugih ruta...

def create_app():
    app = Flask(__name__)

    # Registracija blueprintova
    app.register_blueprint(recept_routes, url_prefix='/recept')
    app.register_blueprint(narudzba_routes, url_prefix='/narudzba')
    app.register_blueprint(trosak_routes, url_prefix='/trosak')
    app.register_blueprint(zaposlenik_routes, url_prefix='/zaposlenik')
    # Registracija drugih blueprintova...

    return app
```

## `app/routes`

Ovaj direktorij sadrži sve rute aplikacije. Svaka ruta je definirana u zasebnoj datoteci.

### Primjer: `app/routes/recept_routes.py`

```
from flask import Blueprint, request, jsonify
from app.services.recept_service import ReceptService

recept_routes = Blueprint('recept_routes', __name__)
recept_service = ReceptService()

@recept_routes.route('/create', methods=['POST'])
def create_recept():
    data = request.get_json()
    recept_service.create_recept(data)
    return jsonify({'message': 'Recept created successfully'}), 201

@recept_routes.route('/<int:id>', methods=['GET'])
def get_recept(id):
    recept = recept_service.get_recept(id)
    return jsonify(recept)
```

## `app/services`

Ovaj direktorij sadrži sve servisne klase koje obavljaju poslovnu logiku aplikacije.

### Primjer: `app/services/recept_service.py`

```
class ReceptService:
    def __init__(self):
        self.cursor = mysql.connection.cursor()

    def create_recept(self, data):
        sql_script = "INSERT INTO recept (naziv, upute) VALUES (%s, %s)"
        self.cursor.execute(sql_script, (data['naziv'], data['upute']))
        mysql.connection.commit()

    def get_recept(self, id):
        sql_script = "SELECT * FROM recept WHERE id = %s"
```

```
        self.cursor.execute(sql_script, (id,))
        recept = self.cursor.fetchone()
        return recept
```

## `app/templates`

Ovaj direktorij sadrži sve HTML predloške koji se koriste za prikazivanje stranica.

### Primjer: `app/templates/recept_create.html`

```
{% extends "index.html" %}
{% block content %}
<h2>Dodaj Recept</h2>
<form action="{{ url_for('recept_routes.create_recept') }}" method="POST">
    <div class="form-group">
        <label for="naziv">Naziv:</label>
        <input type="text" name="naziv" id="naziv" required>
    </div>
    <div class="form-group">
        <label for="upute">Upute:</label>
        <textarea name="upute" id="upute" required></textarea>
    </div>
    <button type="submit">Dodaj Recept</button>
</form>
{% endblock %}
```

## `sql`

Ovaj direktorij sadrži sve SQL skripte koje se koriste za kreiranje i upravljanje bazom podataka.

### Primjer: `sql/recept/select-all.sql`

```
SELECT  rec.id id,
        rec.created_at created_at,
        rec.updated_at updated_at,
        rec.deleted_at deleted_at,
        rec.disabled disabled,
        res.naziv naziv_restoran,
        rec.naziv naziv,
        rec.upute upute
FROM recept rec
JOIN restoran res ON rec.restoran_id = res.id
WHERE rec.disabled = false;
```

# 15. Zaključak

Ovaj projekt predstavlja sveobuhvatnu web aplikaciju za upravljanje restoranom, razvijenu koristeći Flask framework i MySQL bazu podataka. Aplikacija omogućuje korisnicima upravljanje različitim aspektima poslovanja restorana, uključujući narudžbe, račune, recepte, zaposlenike, skladišta i troškove.

## Ključne Značajke:

1. **Upravljanje Narudžbama**: Omogućuje kreiranje, ažuriranje i pregled narudžbi, uključujući izračun ukupne cijene narudžbe.
2. **Upravljanje Računima**: Omogućuje kreiranje i pregled računa, uključujući izračun ukupne vrijednosti računa s napojnicom.
3. **Upravljanje Receptima**: Omogućuje kreiranje, ažuriranje i pregled recepata, uključujući sastojke potrebne za svaki recept.
4. **Upravljanje Zaposlenicima**: Omogućuje upravljanje zaposlenicima, uključujući dodjelu plaća i praćenje transakcija zaposlenika.
5. **Upravljanje Skladištima**: Omogućuje praćenje stanja skladišta, uključujući trenutne i potrebne količine sastojaka.
6. **Upravljanje Troškovima**: Omogućuje praćenje mjesečnih i jednokratnih troškova restorana.
7. **Upravljanje Rezervacijama**: Omogućuje kreiranje, ažuriranje i pregled rezervacija, uključujući provjeru dostupnosti stolova.

## Tehnički Aspekti:

- **Flask Framework**: Koristi se za razvoj backend dijela aplikacije, uključujući definiranje ruta, servisa i kontrolera.
- **MySQL Baza Podataka**: Koristi se za pohranu podataka o restoranima, narudžbama, računima, receptima, zaposlenicima, skladištima i troškovima.
- **Stored Procedures i Triggers**: Koriste se za implementaciju poslovne logike na razini baze podataka, uključujući procesiranje transakcija, ažuriranje stanja skladišta i validaciju podataka.
- **HTML Templates**: Koriste se za prikazivanje podataka korisnicima, uključujući forme za unos podataka i tablice za pregled podataka.

## Zaključak:

Ovaj projekt pruža sveobuhvatno rješenje za upravljanje restoranom, omogućujući korisnicima učinkovito upravljanje svim aspektima poslovanja. Korištenje Flask frameworka i MySQL baze podataka omogućuje skalabilnost i fleksibilnost aplikacije, dok implementacija stored procedures i triggers osigurava integritet podataka i efikasnost poslovne logike. Aplikacija je dizajnirana s fokusom na korisničko iskustvo, pružajući intuitivno sučelje za upravljanje restoranom.