Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Dokumentacija uz projektni zadatak

"Sustav za upravljanje vinarijom"

TIM 9

Laura Đurinec

Marko Valečić

Marta Kralj

Davor Škiljić

Vid Ernečić

Danijel Milašinović

Smjer : Informatika

Kolegij : Baze podataka II

Mentor : doc. Dr. sc. Goran Oreški

Asistent : mag. Inf. Romeo Šajina

Pula, 2024/2025

Sadržaj:

1. OPIS POSLOVNOGA PROCESA:

Poslovni proces koji se bavi ovom bazom podataka odnosi se na upravljanje proizvodnjom, distribucijom i prodajom vina, uz praćenje potrebnih resursa, narudžbi i logistike.

Poslovni proces obuhvaća cjelokupni tijek aktivnosti vezanih uz proizvodnju, skladištenje, prodaju i distribuciju vina, s posebnim naglaskom na učinkovito upravljanje resursima i praćenje ključnih podataka. Proces započinje definiranim osnovama proizvodnje, gdje se evidentiraju vrste vina, sorte grožđa i karakteristike svake berbe, uključujući godinu i postotak alkohola.

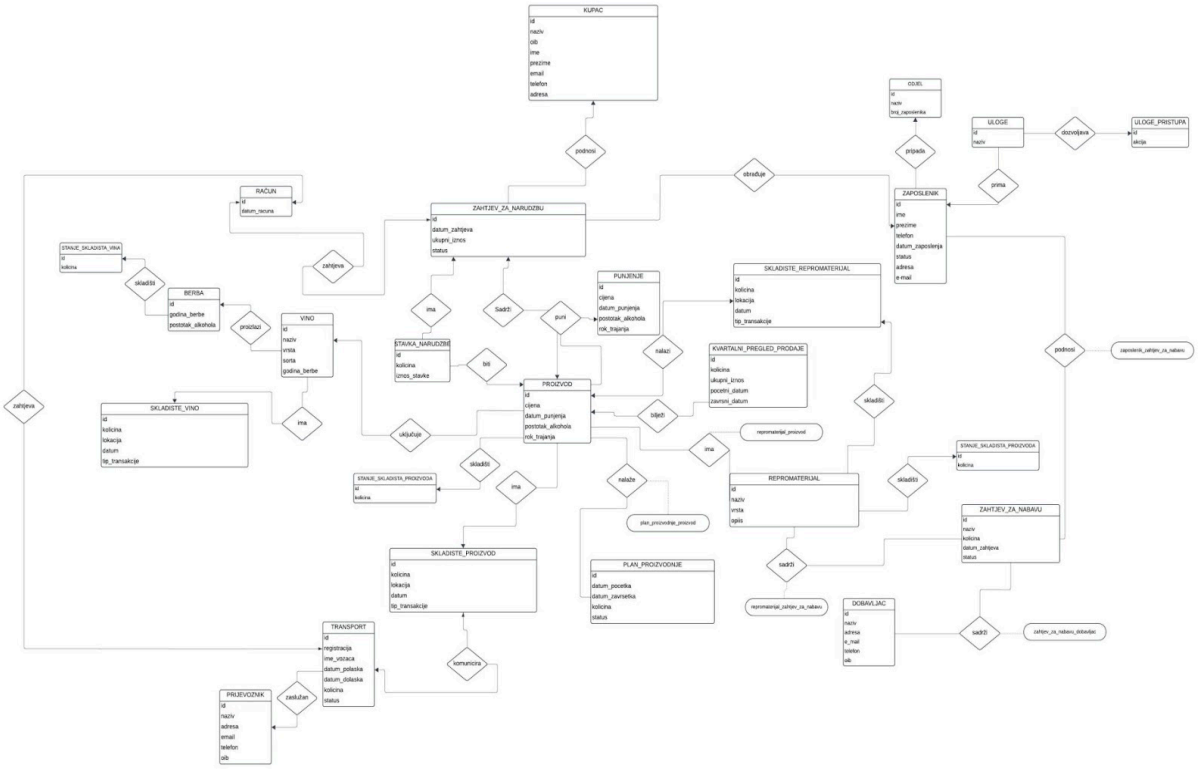
Planiranje proizvodnje ključni je korak koji omogućuje definiranje potrebnih količina i vremenskih okvira za proizvodnju vina. Proizvedeno vino pohranjuje se u skladištima, gdje se bilježe sve transakcije – ulaz i izlaz proizvoda – uz praćenje lokacije i zaliha.

Repromaterijali, poput boca, etiketa i drugih potrebnih sirovina, nabavljaju se od dobavljača putem sustava zahtjeva za nabavu. Svaki zahtjev povezan je s odgovornim zaposlenikom i praćenjem statusa, čime se osigurava pravovremena opskrba proizvodnog procesa.

Kupci naručuju proizvode putem zahtjeva za narudžbu, u kojem se bilježe detalji o naručenim proizvodima, količinama i ukupnom iznosu. Nakon obrade narudžbe izdaje se račun, čime se završava prodajni proces. Dostava proizvoda kupcima organizira se putem prijevoznika, a sustav omogućuje praćenje vozača, registracije vozila, količine robe i statusa dostave.

Ovaj poslovni proces osmišljen je kako bi integrirao sve ključne korake u lancu vrijednosti, od nabave i proizvodnje do prodaje i distribucije, osiguravajući pritom transparentnost, učinkovitost i točnost u svakom segmentu poslovanja.

2. ER-DIJAGRAM



- ER dijagram prikazuje strukturu baze podataka koja modelira poslovne procese vinarije. Svaki pravokutnik predstavlja entitet (tablicu) u bazi podataka, dok linije između pravokutnika predstavljaju veze između entiteta. Ključni entiteti i njihovi atributi su:

1. Kupac: Sadrži informacije o kupcima vina. Atributi uključuju jedinstveni ID kupca, ime, prezime, adresu, telefon i e-mail. Kupac je ključni entitet za praćenje prodaje i narudžbi.
2. Odjel: Predstavlja različite odjele unutar vinarije, poput proizvodnje, skladišta ili administracije. Atributi uključuju ID odjela, naziv i opis.
3. Zaposlenik: Sadrži podatke o zaposlenicima, uključujući ID, ime, prezime, datum zaposlenja, adresu, e-mail i broj telefona. Svaki zaposlenik pripada određenom odjelu.
4. Vino: Predstavlja proizvode vinarije, s atributima kao što su ID vina, naziv, sorta, godina berbe i postotak alkohola. Vino je povezano s procesom proizvodnje i skladištenjem.
5. Berba: Sadrži informacije o berbi grožđa, uključujući ID berbe, godinu berbe, količinu u kilogramima i postotak alkohola. Povezana je s proizvodnjom vina.

6. Proizvod: Predstavlja gotove proizvode, poput vina u bocama. Atributi uključuju ID proizvoda, naziv, vrstu, cijenu i postotak alkohola. Proizvod je povezan s punjenjem vina i skladištenjem.
7. Punjenje: Sadrži podatke o procesu punjenja vina u boce. Atributi uključuju ID punjenja, datum, količinu i tip vina. Veza između vina i proizvoda omogućuje praćenje ovog procesa.
8. Dobavljač: Predstavlja dobavljače repromaterijala potrebnog za proizvodnju. Atributi uključuju ID dobavljača, naziv, adresu, e-mail, telefon i OIB.
9. Repromaterijal: Sadrži informacije o materijalima poput boca, etiketa ili čepova. Atributi uključuju ID repromaterijala, naziv, vrstu i cijenu.
10. Prijevoznik: Predstavlja prijevoznike koji transportiraju proizvode. Atributi uključuju ID prijevoznika, naziv, adresu, e-mail, telefon i OIB.
11. Transport: Sadrži podatke o transportu proizvoda, uključujući ID transporta, ID prijevoznika, registraciju vozila, ime vozača, datume polaska i dolaska, količinu i status transporta.
12. Repromaterijal_Proizvod: Povezuje repromaterijal s proizvodima. Sadrži ID repromaterijala i ID proizvoda kako bi se pratilo koji materijali se koriste za određeni proizvod.
13. Zahtjev za narudžbu: Predstavlja zahtjeve za narudžbu vina od strane kupaca. Atributi uključuju ID zahtjeva, ID kupca, datum zahtjeva i status narudžbe.
14. Stavka narudžbe: Sadrži detalje narudžbe, poput ID-a stavke, ID zahtjeva, ID proizvoda, količine i iznosa stavke.
15. Račun: Predstavlja račune izdane za narudžbe. Atributi uključuju ID računa, ID zaposlenika, ID zahtjeva za narudžbu i datum računa.
16. Plan proizvodnje: Sadrži podatke o planiranoj proizvodnji vina. Atributi uključuju ID plana, ID proizvoda, datum početka i količinu.
17. Skladište vino: Prati transakcije vina u skladištu. Atributi uključuju ID, ID berbe, datum, tip transakcije (ulaz ili izlaz), količinu i lokaciju.
18. Skladište repromaterijal: Sadrži podatke o skladištenju repromaterijala. Atributi uključuju ID, ID repromaterijala, datum, tip transakcije, količinu i lokaciju.
19. Skladište proizvod: Prati skladištenje gotovih proizvoda. Atributi uključuju ID, ID proizvoda, datum, tip transakcije, količinu i lokaciju.

Veze između entiteta:

1. Kupac ↔ Zahtjev za narudžbu
   Veza: Jedan kupac može napraviti više zahtjeva za narudžbu, ali svaki zahtjev pripada samo jednom kupcu.
   Tip veze: 1:N
   Ključ: id_kupac u tablici *zahtjev_za_narudzbu* referencira id u tablici *kupac*.

2. Odjel ↔ Zaposlenik
   Veza: Jedan odjel može imati više zaposlenika, ali svaki zaposlenik pripada samo jednom odjelu.
   Tip veze: 1:N
   Ključ: id_odjel u tablici *zaposlenik* referencira id u tablici *odjel*.

3. Zaposlenik ↔ Zahtjev za narudžbu
   Veza: Jedan zaposlenik može obraditi više zahtjeva za narudžbu, ali svaki zahtjev obrađuje samo jedan zaposlenik.
   Tip veze: 1:N
   Ključ: id_zaposlenik u tablici *zahtjev_za_narudzbu* referencira id u tablici *zaposlenik*.

4. Vino ↔ Berba
   Veza: Jedno vino može imati više berbi (godina proizvodnje), ali svaka berba pripada samo jednom vinu.
   Tip veze: 1:N
   Ključ: id_vino u tablici *berba* referencira id u tablici *vino*.

5. Berba ↔ Proizvod
   Veza: Jedna berba može biti korištena za proizvodnju više proizvoda, ali svaki proizvod dolazi iz jedne berbe.
   Tip veze: 1:N
   Ključ: id_berba u tablici *proizvod* referencira id u tablici *berba*.

6. Proizvod ↔ Punjenje
   Veza: Jedan proizvod može imati više serija punjenja, ali svaka serija punjenja odnosi se na jedan proizvod.
   Tip veze: 1:N
   Ključ: id_proizvod u tablici *punjenje* referencira id u tablici *proizvod*.

7. Dobavljač ↔ Repromaterijal
   Veza: Jedan dobavljač može isporučivati više vrsta repromaterijala, ali svaki repromaterijal dolazi od jednog dobavljača.
   Tip veze: 1:N
   Ključ: id_dobavljac u tablici *repromaterijal* referencira id u tablici *dobavljac*.

8. Repromaterijal ↔ Proizvod (kroz repromaterijal_proizvod)
   Veza: Jedan repromaterijal može biti korišten u više proizvoda, a jedan proizvod može koristiti više vrsta repromaterijala.

Tip veze: M:N
Ključ: *repromaterijal_proizvod* povezuje id_repromaterijal s id_proizvod.

9. Prijevoznik ↔ Transport
   Veza: Jedan prijevoznik može biti odgovoran za više transporta, ali svaki transport pripada samo jednom prijevozniku.
   Tip veze: 1:N
   Ključ: id_prijevoznik u tablici *transport* referencira id u tablici *prijevoznik*.

10. Zahtjev za narudžbu ↔ Stavka narudžbe
    Veza: Jedan zahtjev za narudžbu može sadržavati više stavki, ali svaka stavka pripada samo jednom zahtjevu.
    Tip veze: 1:N
    Ključ: id_zahtjev_za_narudzbu u tablici *stavka_narudzbe* referencira id u tablici *zahtjev_za_narudzbu*.

11. Proizvod ↔ Stavka narudžbe
    Veza: Jedan proizvod može biti dio više stavki narudžbi, ali svaka stavka narudžbe odnosi se na jedan proizvod.
    Tip veze: 1:N
    Ključ: id_proizvod u tablici *stavka_narudzbe* referencira id u tablici *proizvod*.

12. Zahtjev za narudžbu ↔ Račun
    Veza: Jedan zahtjev za narudžbu ima jedan pripadajući račun, dok svaki račun pokriva samo jedan zahtjev.
    Tip veze: 1:1
    Ključ: id_zahtjev_za_narudzbu u tablici *racun* referencira id u tablici *zahtjev_za_narudzbu*.

13. Proizvod ↔ Plan proizvodnje
    Veza: Jedan proizvod može biti uključen u više planova proizvodnje, ali svaki plan proizvodnje odnosi se na jedan proizvod.
    Tip veze: 1:N
    Ključ: id_proizvod u tablici *plan_proizvodnje* referencira id u tablici *proizvod*.

14. Berba ↔ Skladište vino
    Veza: Jedna berba može imati više unosa u skladištu vina, ali svaki unos pripada samo jednoj berbi.
    Tip veze: 1:N
    Ključ: id_berba u tablici *skladiste_vino* referencira id u tablici *berba*.

15. Repromaterijal ↔ Skladište repromaterijal
    Veza: Jedan repromaterijal može imati više unosa u skladištu, ali svaki unos pripada samo jednom repromaterijalu.
    Tip veze: 1:N
    Ključ: id_repromaterijal u tablici *skladiste_repromaterijal* referencira id u tablici *repromaterijal*.

16. Proizvod ↔ Skladište proizvod
    Veza: Jedan proizvod može imati više unosa u skladištu, ali svaki unos pripada samo jednom proizvodu.
    Tip veze: 1:N
    Ključ: id_proizvod u tablici *skladiste_proizvod* referencira id u tablici *proizvod*.

17. Zaposlenik ↔ Zahtjev za nabavu
    Veza: Jedan zaposlenik može podnijeti više zahtjeva za nabavu, ali svaki zahtjev pripada samo jednom zaposleniku.
    Tip veze: 1:N
    Ključ: id_zaposlenik u tablici *zahtjev_za_nabavu* referencira id u tablici *zaposlenik*.

18. Repromaterijal ↔ Zahtjev za nabavu
    Veza: Jedan repromaterijal može biti dio više zahtjeva za nabavu, ali svaki zahtjev za nabavu odnosi se na jedan repromaterijal.
    Tip veze: 1:N
    Ključ: id_repromaterijal u tablici *zahtjev_za_nabavu* referencira id u tablici *repromaterijal*.

3. TABLICE

1. Tablica kupac

```
CREATE TABLE kupac (
    id INT AUTO_INCREMENT PRIMARY KEY,
    naziv VARCHAR(100) NOT NULL,
    oib CHAR(11) UNIQUE NOT NULL,
    ime VARCHAR(50) NOT NULL,
    prezime VARCHAR(50) NOT NULL,
    adresa VARCHAR(100),
    email VARCHAR(100),
    telefon VARCHAR(20)
);
```

Svrha:
Tablica kupac pohranjuje podatke o kupcima koji kupuju proizvode ili usluge od organizacije. Kupci mogu biti pravne osobe (tvrtke) ili fizičke osobe. Informacije iz ove tablice koriste se za vođenje evidencije o klijentima, olakšavanje komunikacije i upravljanje narudžbama.

Atributi:

- id (INT, AUTO_INCREMENT, PRIMARY KEY):
  Jedinstveni identifikator svakog kupca. Automatski se generira i koristi kao primarni ključ za razlikovanje kupaca.

- naziv (VARCHAR(100), NOT NULL):
  Naziv tvrtke ako je kupac pravna osoba. Obavezno je polje jer omogućuje prepoznavanje kupca.
- oib (CHAR(11), UNIQUE, NOT NULL):
  Osobni identifikacijski broj (OIB) kupca. Ovo je jedinstveno polje koje osigurava da svaki kupac ima svoj jedinstveni identifikator prema zakonodavstvu.
- ime (VARCHAR(50), NOT NULL):
  Ime kupca, ako je fizička osoba. Ovo je obavezno za identifikaciju fizičkih osoba.
- prezime (VARCHAR(50), NOT NULL):
  Prezime kupca, koristi se zajedno s imenom za identifikaciju fizičkih osoba.
- adresa (VARCHAR(100), NULL):
  Adresa kupca, opcionalno polje koje se koristi za dostavu ili kontakt.
- email (VARCHAR(100), NULL):
  Email adresa kupca, opcionalno polje za elektroničku komunikaciju.
- telefon (VARCHAR(20), NULL):
  Kontakt broj kupca, opcionalno polje za bržu komunikaciju

2. Tablica odjel

```
CREATE TABLE odjel (
    id INT AUTO_INCREMENT PRIMARY KEY,
    naziv VARCHAR(100) NOT NULL,
    broj_zaposlenika INT CHECK (broj_zaposlenika >= 0) DEFAULT 0
);
```

Svrha:
Tablica odjel koristi se za evidenciju različitih odjela unutar organizacije. Odjeli predstavljaju funkcionalne jedinice, poput prodaje, proizvodnje ili marketinga, i omogućuju organizaciji da prati broj zaposlenika i upravlja njihovim resursima.

Atributi:

- id (INT, AUTO_INCREMENT, PRIMARY KEY):
  Jedinstveni identifikator svakog odjela. Automatski se generira i koristi za razlikovanje odjela.
- naziv (VARCHAR(100), NOT NULL):
  Naziv odjela, obavezno polje jer omogućuje identifikaciju odjela (npr. "Prodaja", "Proizvodnja").
- broj_zaposlenika (INT, CHECK (broj_zaposlenika >= 0), DEFAULT 0):
  Broj zaposlenika unutar odjela. Ovo polje ima zadanu vrijednost 0 i mora biti nenegativno. Koristi se za praćenje ljudskih resursa u odjelu.

3. Tablica zaposlenik

```sql
CREATE TABLE zaposlenik (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_odjel INT NOT NULL,
    ime VARCHAR(50) NOT NULL,
    prezime VARCHAR(50) NOT NULL,
    adresa VARCHAR(100),
    email VARCHAR(100),
    telefon VARCHAR(20),
    datum_zaposlenja DATE NOT NULL,
    status_zaposlenika ENUM('aktivan', 'neaktivan') NOT NULL,
    FOREIGN KEY (id_odjel) REFERENCES Odjel(id)
);
```

Svrha:
Tablica zaposlenik pohranjuje podatke o zaposlenicima organizacije. Svaki zaposlenik je povezan s određenim odjelom, a podaci poput datuma zaposlenja i statusa omogućuju praćenje radne snage.

Atributi:

- id (INT, AUTO_INCREMENT, PRIMARY KEY):
  Jedinstveni identifikator zaposlenika, automatski generiran.
- id_odjel (INT, NOT NULL, FOREIGN KEY):
  Identifikator odjela kojem zaposlenik pripada. Ovo je strani ključ koji se povezuje s tablicom odjel.
- ime (VARCHAR(50), NOT NULL):
  Ime zaposlenika, obavezno polje za identifikaciju.
- prezime (VARCHAR(50), NOT NULL):
  Prezime zaposlenika, koristi se zajedno s imenom za identifikaciju.
- adresa (VARCHAR(100), NULL):
  Adresa zaposlenika, opcionalno polje.
- email (VARCHAR(100), NULL):
  Email adresa zaposlenika za komunikaciju unutar organizacije.
- telefon (VARCHAR(20), NULL):
  Kontakt broj zaposlenika.
- datum_zaposlenja (DATE, NOT NULL):
  Datum kada je zaposlenik počeo raditi u organizaciji. Ovo je obavezno polje za praćenje radnog staža.
- status_zaposlenika (ENUM('aktivan', 'neaktivan'), NOT NULL):
  Status zaposlenika koji pokazuje je li zaposlenik trenutno aktivan ili neaktivan.

---

4. Tablica vino

```
CREATE TABLE vino (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    naziv VARCHAR(255) NOT NULL,
    vrsta ENUM('bijelo', 'crno', 'rose', 'pjenušavo') NOT NULL,
    sorta VARCHAR(100) NOT NULL
);
```

Svrha:
Tablica vino pohranjuje informacije o vrstama vina koje organizacija proizvodi. Svako vino ima svoj naziv, vrstu, i sortu grožđa.

Atributi:

- id (INT, AUTO_INCREMENT, PRIMARY KEY):
  Jedinstveni identifikator vina.
- naziv (VARCHAR(255), NOT NULL):
  Naziv vina, obavezno polje jer omogućuje identifikaciju proizvoda.
- vrsta (ENUM('bijelo', 'crno', 'rose', 'pjenušavo'), NOT NULL):
  Vrsta vina, obavezno polje koje definira kategoriju vina.
- sorta (VARCHAR(100), NOT NULL):
  Sorta grožđa od koje je vino proizvedeno, obavezno polje za preciznu identifikaciju

5. Tablica berba

```
CREATE TABLE berba (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_vino INTEGER NOT NULL,
    godina_berbe INTEGER NOT NULL,
    postotak_alkohola DECIMAL(5, 2) NOT NULL,
    FOREIGN KEY (id_vino) REFERENCES vino(id),
    CONSTRAINT berba_postotak_alkohola_ck CHECK (postotak_alkohola BETWEEN 5 AND 25),
    CONSTRAINT berba_godina_berbe_ck CHECK (godina_berbe > 2000)
);
```

Svrha:
Tablica berba prati podatke o berbama grožđa za proizvodnju vina, uključujući godinu berbe i postotak alkohola.

Atributi:

- id (INT, AUTO_INCREMENT, PRIMARY KEY):
  Jedinstveni identifikator berbe.
- id_vino (INT, NOT NULL, FOREIGN KEY):
  Identifikator vina kojem berba pripada. Ovo je strani ključ povezan s tablicom vino.

- godina_berbe (INT, NOT NULL, CHECK (godina_berbe > 2000):
  Godina kada je grožđe ubrano, ograničeno na godine kasnije od 2000.
- postotak_alkohola (DECIMAL(5,2), NOT NULL, CHECK (postotak_alkohola BETWEEN 5 AND 25)):
  Postotak alkohola u vinu, ograničen na raspon od 5% do 25%.

---

6. Tablica proizvod

```
CREATE TABLE proizvod (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_berba INTEGER NOT NULL,
    volumen DECIMAL(4,2) NOT NULL,
    cijena DECIMAL(10, 2) NOT NULL CHECK (cijena > 0),
    CONSTRAINT proizvod__berba_fk FOREIGN KEY (id_berba) REFERENCES berba(id),
    INDEX (id_berba)
);
```

Tablica **proizvod** pohranjuje informacije o konkretnim proizvodima koji su proizvedeni iz određene berbe.

- Atributi:
  - id *(INTEGER AUTO_INCREMENT PRIMARY KEY)*: Jedinstveni identifikator proizvoda.
  - id_berba *(INTEGER NOT NULL)*: Strani ključ koji povezuje proizvod s određenom berbom.
  - volumen *(DECIMAL(4,2) NOT NULL)*: Volumen proizvoda (npr. veličina boce u litrama).
  - cijena *(DECIMAL(10,2) NOT NULL)*: Cijena proizvoda. Mora biti veća od 0 (CHECK ograničenje).
- Svrha:
  Omogućuje praćenje proizvoda po serijama, uključujući informacije o volumenu i cijeni.

---

7. Tablica punjenje

```
CREATE TABLE punjenje (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_proizvod INTEGER NOT NULL,
    oznaka_serije VARCHAR(20) NOT NULL UNIQUE,
    pocetak_punjenja DATE NOT NULL,
    zavrsetak_punjenja DATE NOT NULL,
    kolicina INTEGER NOT NULL,
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id),
    CONSTRAINT punjenje_kolicina_ck CHECK (kolicina > 0)
);
```

Tablica **punjenje** bilježi podatke o punjenju proizvoda u serijama.

- **Atributi:**
  - id *(INTEGER AUTO_INCREMENT PRIMARY KEY)*: Jedinstveni identifikator serije punjenja.
  - id_proizvod *(INTEGER NOT NULL UNIQUE)*: Strani ključ koji povezuje seriju punjenja s odgovarajućim proizvodom.
  - oznaka_serije *(VARCHAR(20) NOT NULL)*: Jedinstvena oznaka za seriju punjenja.
  - pocetak_punjenja *(DATE NOT NULL)*: Datum početka punjenja.
  - zavrsetak_punjenja *(DATE NOT NULL)*: Datum završetka punjenja.
  - kolicina *(INTEGER NOT NULL)*: Broj jedinica napunjenih u toj seriji. Mora biti veći od 0 (CHECK ograničenje).
- **Svrha:**
  Omogućuje praćenje proizvodnih serija i planiranje punjenja.

8. Tablica dobavljac

```
CREATE TABLE dobavljac (
    id INT AUTO_INCREMENT PRIMARY KEY,
    naziv VARCHAR(50),
    adresa VARCHAR(50),
    email VARCHAR(50),
    telefon VARCHAR(20),
    oib CHAR(11) UNIQUE
);
```

Tablica **dobavljac** pohranjuje informacije o dobavljačima repromaterijala.

- **Atributi:**
  - id *(INTEGER AUTO_INCREMENT PRIMARY KEY)*: Jedinstveni identifikator dobavljača.
  - naziv *(VARCHAR(50))*: Naziv dobavljača.
  - adresa *(VARCHAR(50))*: Adresa dobavljača.
  - email *(VARCHAR(50))*: Kontakt e-mail adresa dobavljača.
  - telefon *(VARCHAR(20))*: Kontakt broj dobavljača.
  - oib *(CHAR(11) UNIQUE)*: Jedinstveni identifikator (OIB) dobavljača.
- **Svrha:**
  Omogućuje praćenje dobavljača koji isporučuju potrebni materijal za proizvodnju.

---

9. Tablica repromaterijal

```
CREATE TABLE repromaterijal (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_dobavljac INTEGER NOT NULL,
    vrsta VARCHAR(100),
    opis TEXT,
    jedinicna_cijena DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_dobavljac) REFERENCES dobavljac(id),
    CONSTRAINT repromaterijal_cijena_ck CHECK (jedinicna_cijena > 0),
    INDEX (id_dobavljac)
);
```

Tablica **repromaterijal** pohranjuje informacije o materijalima koji se koriste u proizvodnji.

- **Atributi:**
    - id *(INTEGER AUTO_INCREMENT PRIMARY KEY)*: Jedinstveni identifikator repromaterijala.
    - id_dobavljac *(INTEGER NOT NULL)*: Strani ključ koji povezuje repromaterijal s odgovarajućim dobavljačem.
    - vrsta *(VARCHAR(100))*: Vrsta repromaterijala (npr. boce, čepovi).
    - opis *(TEXT)*: Opis repromaterijala.
    - jedinicna_cijena *(DECIMAL(10,2) NOT NULL)*: Cijena po jedinici repromaterijala. Mora biti veća od 0 (CHECK ograničenje).
- **Svrha:**
  Omogućuje praćenje troškova i vrsta repromaterijala koji se koriste u proizvodnji.

---

10. Tablica prijevoznik

```
CREATE TABLE prijevoznik (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    naziv VARCHAR(50),
    adresa VARCHAR(50),
    email VARCHAR(50),
    telefon VARCHAR(50),
    oib CHAR(11) UNIQUE
);
```

Tablica **prijevoznik** pohranjuje informacije o prijevoznicima koji su odgovorni za transport robe.

Atributi:

- **id** (INTEGER AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator prijevoznika. Automatski se generira za svaki novi zapis.

- naziv (VARCHAR(50)):
  Naziv prijevoznika ili prijevozničke tvrtke.
- adresa (VARCHAR(50)):
  Adresa sjedišta prijevoznika.
- email (VARCHAR(50)):
  Kontakt e-mail adresa prijevoznika.
- telefon (VARCHAR(50)):
  Kontakt broj telefona prijevoznika.
- oib (CHAR(11) UNIQUE):
  Jedinstveni identifikator (OIB) prijevoznika.

Svrha:

Omogućuje praćenje prijevoznika koji su zaduženi za transport proizvoda ili repromaterijala, čime se osigurava evidencija i olakšava komunikacija.

---

11. Tablica transport

```
CREATE TABLE transport (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_prijevoznik INTEGER NOT NULL,
    registracija VARCHAR(50) NOT NULL,
    ime_vozaca VARCHAR(50) NOT NULL,
    datum_polaska DATE NOT NULL,
    datum_dolaska DATE,
    kolicina INTEGER,
    status_transporta ENUM('Obavljen', 'U tijeku', 'Otkazan') NOT NULL,
    FOREIGN KEY (id_prijevoznik) REFERENCES prijevoznik(id)
);
```

Tablica transport pohranjuje informacije o pojedinačnim transportima koje obavljaju prijevoznici.

Atributi:

- id (INTEGER AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator transporta. Automatski se generira za svaki novi zapis.
- id_prijevoznik (INTEGER NOT NULL):
  Strani ključ koji povezuje transport s prijevoznikom. Referencira stupac id iz tablice prijevoznik.
- registracija (VARCHAR(50) NOT NULL):
  Registracijski broj vozila koje je korišteno za transport.
- ime_vozaca (VARCHAR(50) NOT NULL):
  Ime vozača koji je upravljao vozilom tijekom transporta.
- datum_polaska (DATE NOT NULL):
  Datum kada je transport započeo.

- **datum_dolaska** (DATE):
  Datum kada je transport završen. Može biti prazan ako transport još nije završen.
- **kolicina** (INTEGER):
  Količina robe koja se transportira.
- **status_transporta** (ENUM('Obavljen', 'U tijeku', 'Otkazan') NOT NULL):
  Trenutni status transporta:
  - **Obavljen:** Transport je završen.
  - **U tijeku:** Transport je u procesu.
  - **Otkazan:** Transport je otkazan prije završetka.

Svrha:

Omogućuje praćenje pojedinačnih transporta, uključujući vozače, vozila, datume, količine robe i status transporta, čime se osigurava točna evidencija i bolja kontrola logistike.

---

12. Tablica repromaterijal_proizvod

```
CREATE TABLE repromaterijal_proizvod (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_proizvod INTEGER NOT NULL,
    id_repromaterijal INTEGER NOT NULL,
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id),
    FOREIGN KEY (id_repromaterijal) REFERENCES repromaterijal(id),
    CONSTRAINT repromaterijal_proizvod_uk UNIQUE (id_proizvod, id_repromaterijal)
);
```

Tablica **repromaterijal_proizvod** predstavlja M:N vezu između repromaterijala i proizvoda.

- **Atributi:**
  - id *(INTEGER AUTO_INCREMENT PRIMARY KEY)*: Jedinstveni identifikator zapisa.
  - id_proizvod *(INTEGER NOT NULL)*: Strani ključ koji povezuje zapis s tablicom proizvod.
  - id_repromaterijal *(INTEGER NOT NULL)*: Strani ključ koji povezuje zapis s tablicom repromaterijal.
- **Svrha:**
  Omogućuje praćenje koji repromaterijali su korišteni za određene proizvode.

---

13. Tablica zahtjev_za_narudzbu

```
CREATE TABLE zahtjev_za_narudzbu (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_kupac INTEGER NOT NULL,
    id_zaposlenik INTEGER NOT NULL,
    id_transport INTEGER,
    datum_zahtjeva DATE NOT NULL,
    ukupni_iznos DECIMAL(8, 2),
    status_narudzbe ENUM('Primljena', 'U obradi', 'Na čekanju', 'Spremna za isporuku', 'Poslana', 'Završena', 'Otkazana') NOT NULL DEFAULT 'Na čekanju',
    FOREIGN KEY (id_kupac) REFERENCES kupac(id),
    FOREIGN KEY (id_zaposlenik) REFERENCES zaposlenik(id),
    FOREIGN KEY (id_transport) REFERENCES transport(id)
);
```

Tablica **zahtjev_za_narudzbu** pohranjuje informacije o narudžbama kupaca.

- **Atributi:**
    - id *(INTEGER AUTO_INCREMENT PRIMARY KEY)*: Jedinstveni identifikator narudžbe.
    - id_kupac *(INTEGER NOT NULL)*: Strani ključ koji povezuje narudžbu s kupcem.
    - id_zaposlenik *(INTEGER NOT NULL)*: Strani ključ koji povezuje narudžbu sa zaposlenikom.
    - id_transport *(INTEGER)*: Strani ključ koji povezuje narudžbu s transportom.
    - datum_zahtjeva *(DATE NOT NULL)*: Datum kreiranja narudžbe.
    - ukupni_iznos *(DECIMAL(8,2))*: Ukupan iznos narudžbe.
    - status_narudzbe *(ENUM)*: Status narudžbe (npr. "Primljena", "U obradi").
- **Svrha:**
  Omogućuje praćenje narudžbi kupaca i njihovog statusa.

---

14. Tablica stavka_narudzbe

```
CREATE TABLE stavka_narudzbe (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_zahtjev_za_narudzbu INTEGER NOT NULL,
    id_proizvod INTEGER NOT NULL,
    kolicina INTEGER NOT NULL,
    iznos_stavke DECIMAL(8, 2) NOT NULL,
    FOREIGN KEY (id_zahtjev_za_narudzbu) REFERENCES zahtjev_za_narudzbu(id),
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id),
    CONSTRAINT stavka_narudzbe_uk UNIQUE (id_zahtjev_za_narudzbu, id_proizvod),
    CONSTRAINT stavka_narudzbe_kolicina_ck CHECK (kolicina > 0)
);
```

Tablica **stavka_narudzbe** pohranjuje informacije o pojedinačnim stavkama unutar narudžbe. Svaka stavka predstavlja određeni proizvod i količinu unutar jednog zahtjeva za narudžbu.

Atributi:

- id (INTEGER AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator stavke narudžbe. Automatski se generira za svaku novu stavku.
- id_zahtjev_za_narudzbu (INTEGER NOT NULL):
  Strani ključ koji povezuje stavku s određenim zahtjevom za narudžbu. Referencira stupac id iz tablice zahtjev_za_narudzbu.
- id_proizvod (INTEGER NOT NULL):
  Strani ključ koji povezuje stavku s određenim proizvodom. Referencira stupac id iz tablice proizvod.
- kolicina (INTEGER NOT NULL):
  Količina proizvoda naručena u ovoj stavci. Ograničenje CHECK (kolicina > 0) osigurava da količina mora biti veća od nule.
- iznos_stavke (DECIMAL(8, 2) NOT NULL):
  Ukupni iznos za ovu stavku narudžbe, izračunat na temelju cijene proizvoda i naručene količine. Format DECIMAL(8, 2) omogućuje pohranu vrijednosti s do 8 znamenki ukupno i 2 decimalna mjesta za preciznost.

Ograničenja:

1. stavka_narudzbe_uk (UNIQUE (id_zahtjev_za_narudzbu, id_proizvod)):
   Osigurava da unutar jednog zahtjeva za narudžbu ne može postojati više od jedne stavke za isti proizvod.
2. stavka_narudzbe_kolicina_ck (CHECK (kolicina > 0)):
   Osigurava da naručena količina mora biti pozitivna, čime se sprječava unos nevažećih podataka

- Svrha:

Tablica stavka_narudzbe omogućuje detaljno praćenje sadržaja svakog zahtjeva za narudžbu, uključujući proizvode, količine i iznose. Povezivanjem sa zahtjevima za narudžbu i proizvodima osigurava se točna evidencija narudžbi, a jedinstveno ograničenje sprječava dupliciranje stavki unutar iste narudžbe.

---

15. Tablica racun

```
CREATE TABLE racun (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    id_zaposlenik INTEGER NOT NULL,
    id_zahtjev_za_narudzbu INTEGER NOT NULL UNIQUE,
    datum_racuna DATE NOT NULL,
    FOREIGN KEY (id_zaposlenik) REFERENCES zaposlenik(id),
    FOREIGN KEY (id_zahtjev_za_narudzbu) REFERENCES zahtjev_za_narudzbu(id)
);
```

Tablica **racun** pohranjuje informacije o izdanim računima za narudžbe.

Atributi:

- **id** (INTEGER AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator računa. Automatski se generira za svaki novi račun.
- **id_zaposlenik** (INTEGER NOT NULL):
  Strani ključ koji povezuje račun sa zaposlenikom koji ga je izdao. Referencira stupac id iz tablice zaposlenik.
- **id_zahtjev_za_narudzbu** (INTEGER NOT NULL UNIQUE):
  Strani ključ koji povezuje račun s narudžbom za koju je izdan. Referencira stupac id iz tablice zahtjev_za_narudzbu. Ograničenje UNIQUE osigurava da svaki račun može biti povezan s jednom narudžbom.
- **datum_racuna** (DATE NOT NULL):
  Datum kada je račun izdan.

Svrha:

Tablica omogućuje praćenje izdanih računa, uključujući povezivanje s odgovornim zaposlenikom i narudžbom, čime se osigurava točna evidencija financijskih transakcija.

---

16. Tablica plan_proizvodnje

```
CREATE TABLE plan_proizvodnje (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_proizvod INT NOT NULL,
    datum_pocetka DATE NOT NULL,
    kolicina INT NOT NULL,
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id)
);
```

Tablica **plan_proizvodnje** pohranjuje planirane proizvodne aktivnosti.

Atributi:

- **id** (INT AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator plana proizvodnje. Automatski se generira za svaki novi plan.
- **id_proizvod** (INT NOT NULL):
  Strani ključ koji povezuje plan s određenim proizvodom. Referencira stupac id iz tablice proizvod.
- **datum_pocetka** (DATE NOT NULL):
  Datum kada proizvodnja prema planu treba započeti.
- **kolicina** (INT NOT NULL):
  Količina proizvoda koja se planira proizvesti.

Svrha:

Tablica omogućuje praćenje proizvodnih planova, uključujući proizvode koji se proizvode, količine i datume početka proizvodnje.

---

17. Tablica skladiste_vino

```
CREATE TABLE skladiste_vino (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_berba INT NOT NULL,
    datum DATE NOT NULL,
    tip_transakcije ENUM('ulaz', 'izlaz') NOT NULL,
    kolicina INT NOT NULL,
    lokacija VARCHAR(100),
    FOREIGN KEY (id_berba) REFERENCES berba(id)
);
```

Tablica **skladiste_vino** pohranjuje informacije o ulazima i izlazima vina u skladištu.

Atributi:

- **id** (INT AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator transakcije u skladištu. Automatski se generira za svaki novi zapis.

- **id_berba** (INT NOT NULL):
  Strani ključ koji povezuje transakciju s određenom berbom vina. Referencira stupac id iz tablice berba.
- **datum** (DATE NOT NULL):
  Datum kada je transakcija obavljena.
- **tip_transakcije** (ENUM('ulaz', 'izlaz') NOT NULL):
  Vrsta transakcije:
  - **ulaz:** Dodavanje vina u skladište.
  - **izlaz:** Uklanjanje vina iz skladišta.
- **kolicina** (INT NOT NULL):
  Količina vina uključena u transakciju.
- **lokacija** (VARCHAR(100)):
  Lokacija unutar skladišta gdje se vino pohranjuje.

Svrha:

Tablica omogućuje praćenje kretanja vina u skladištu, uključujući količine, datume i lokacije.

---

18. Tablica skladiste_repromaterijal

```
CREATE TABLE skladiste_repromaterijal (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_repromaterijal INT NOT NULL,
    datum DATE NOT NULL,
    tip_transakcije ENUM('ulaz', 'izlaz') NOT NULL,
    kolicina INT NOT NULL,
    lokacija VARCHAR(100),
    FOREIGN KEY (id_repromaterijal) REFERENCES repromaterijal(id)
);
```

Tablica **skladiste_repromaterijal** pohranjuje informacije o ulazima i izlazima repromaterijala u skladištu.

Atributi:

- **id** (INT AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator transakcije u skladištu. Automatski se generira za svaki novi zapis.
- **id_repromaterijal** (INT NOT NULL):
  Strani ključ koji povezuje transakciju s određenim repromaterijalom. Referencira stupac id iz tablice repromaterijal.
- **datum** (DATE NOT NULL):
  Datum kada je transakcija obavljena.

- **tip_transakcije** (ENUM('ulaz', 'izlaz') NOT NULL):
  Vrsta transakcije:
    - **ulaz:** Dodavanje repromaterijala u skladište.
    - **izlaz:** Uklanjanje repromaterijala iz skladišta.
- **kolicina** (INT NOT NULL):
  Količina repromaterijala uključena u transakciju.
- **lokacija** (VARCHAR(100)):
  Lokacija unutar skladišta gdje se repromaterijal pohranjuje.

Svrha:

Tablica omogućuje praćenje kretanja repromaterijala u skladištu, uključujući količine, datume i lokacije.

---

19. Tablica skladiste_proizvod

```sql
CREATE TABLE skladiste_proizvod (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_proizvod INT,
    datum DATE,
    tip_transakcije ENUM('ulaz', 'izlaz') NOT NULL,
    kolicina INT NOT NULL,
    lokacija VARCHAR(50),
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id)
);
```

Tablica **skladiste_proizvod** pohranjuje informacije o ulazima i izlazima gotovih proizvoda u skladištu.

Atributi:

- **id** (INT AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator transakcije u skladištu. Automatski se generira za svaki novi zapis.
- **id_proizvod** (INT):
  Strani ključ koji povezuje transakciju s određenim proizvodom. Referencira stupac id iz tablice proizvod.
- **datum** (DATE):
  Datum kada je transakcija obavljena.
- **tip_transakcije** (ENUM('ulaz', 'izlaz') NOT NULL):
  Vrsta transakcije:
    - **ulaz:** Dodavanje proizvoda u skladište.
    - **izlaz:** Uklanjanje proizvoda iz skladišta.
- **kolicina** (INT NOT NULL):
  Količina proizvoda uključena u transakciju.

- **lokacija** (VARCHAR(50)):
  Lokacija unutar skladišta gdje se proizvod pohranjuje.

Svrha:

Tablica omogućuje praćenje kretanja gotovih proizvoda u skladištu, uključujući količine, datume i lokacije.

---

20. Tablica zahtjev_za_nabavu

```
CREATE TABLE zahtjev_za_nabavu (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_repromaterijal INT NOT NULL,
    kolicina INT NOT NULL,
    datum_zahtjeva DATE NOT NULL,
    status_nabave ENUM('u obradi', 'odobreno', 'odbijeno', 'dostavljeno') NOT NULL,
    id_zaposlenik INT NOT NULL,
    FOREIGN KEY (id_repromaterijal) REFERENCES repromaterijal(id),
    FOREIGN KEY (id_zaposlenik) REFERENCES zaposlenik(id)
);
```

Tablica **zahtjev_za_nabavu** pohranjuje informacije o zahtjevima za nabavu repromaterijala.

Atributi:

- **id** (INT AUTO_INCREMENT PRIMARY KEY):
  Jedinstveni identifikator zahtjeva za nabavu. Automatski se generira za svaki novi zapis.
- **id_repromaterijal** (INT NOT NULL):
  Strani ključ koji povezuje zahtjev s određenim repromaterijalom. Referencira stupac id iz tablice repromaterijal.
- **kolicina** (INT NOT NULL):
  Količina materijala koja se traži u zahtjevu.
- **datum_zahtjeva** (DATE NOT NULL):
  Datum kada je zahtjev kreiran.
- **status_nabave** (ENUM('u obradi', 'odobreno', 'odbijeno', 'dostavljeno') NOT NULL):
  Status zahtjeva:
  - **u obradi:** Zahtjev se još razmatra.
  - **odobreno:** Zahtjev je odobren.
  - **odbijeno:** Zahtjev je odbijen.
  - **dostavljeno:** Traženi materijal je dostavljen.
- **id_zaposlenik** (INT NOT NULL):
  Strani ključ koji povezuje zahtjev sa zaposlenikom koji ga je kreirao. Referencira stupac id iz tablice zaposlenik.

Svrha: Tablica omogućuje praćenje zahtjeva za nabavu repromaterijala, uključujući količine, statuse i odgovorne zaposlenike.

21. Tablica kvartalni_pregled_prodaje

```
CREATE TABLE kvartalni_pregled_prodaje (
    id_proizvod INTEGER,
    kolicina INTEGER NOT NULL,
    ukupni_iznos DECIMAL(10,2) NOT NULL,
    kvartal VARCHAR(20) NOT NULL,
    PRIMARY KEY (id_proizvod, kvartal),
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id)
);
```

Tablica kvartalni_pregled_prodaje pohranjuje informacije o prodaji proizvoda po kvartalima. Tablica je implementacija materijaliziranog pogleda i popunjava se pomoću procedure.

Atributi:

id_proizvod (INTEGER): Strani ključ koji povezuje zapis s određenim proizvodom. Referencira stupac id iz tablice proizvod.

kolicina (INTEGER NOT NULL): Količina proizvoda prodana u određenom kvartalu.

ukupni_iznos (DECIMAL(10,2) NOT NULL): Ukupni iznos prodaje za određeni proizvod u kvartalu.

kvartal (VARCHAR(20) NOT NULL): Kvartal u kojem je prodaja izvršena.

Primarni ključ se sastoji od id_proizvod i kvartal, a strani ključ id_proizvod referencira tablicu proizvod.

22. Tablica stanje_skladista_vina

```
CREATE TABLE stanje_skladista_vina (
    id_berba INTEGER PRIMARY KEY,
    kolicina DECIMAL(8,2) NOT NULL,
    FOREIGN KEY (id_berba) REFERENCES berba(id)
);
```

Tablica stanje_skladista_vina pohranjuje informacije o stanju vina u skladištu prema berbi. Tablica je implementacija materijaliziranog pogleda u MySQL-u i popunjava se pomoću okidača koji pozivaju proceduru.

Atributi:

id_berba (INTEGER PRIMARY KEY): Jedinstveni identifikator berbe, koji također služi kao primarni ključ. Referencira stupac id iz tablice berba.

kolicina (DECIMAL(8,2) NOT NULL): Količina vina koja se nalazi u skladištu za određenu berbu.

Strani ključ id_berba povezuje ovu tablicu s tablicom berba.

23. Tablica stanje_skladista_proizvoda

```
CREATE TABLE stanje_skladista_proizvoda (
    id_proizvod INTEGER PRIMARY KEY,
    kolicina INTEGER NOT NULL,
    FOREIGN KEY (id_proizvod) REFERENCES proizvod(id)
);
```

Tablica **stanje_skladista_proizvoda** pohranjuje informacije o stanju gotovih proizvoda u skladištu. Tablica je implementacija materijaliziranog pogleda u MySQL-u i popunjava se pomoću okidača koji pozivaju proceduru.

Atributi:

id_proizvod (INTEGER PRIMARY KEY): Jedinstveni identifikator proizvoda, koji također služi kao primarni ključ. Referencira stupac id iz tablice proizvod.

kolicina (INTEGER NOT NULL): Količina proizvoda koja se nalazi u skladištu.

Strani ključ **id_proizvod** povezuje ovu tablicu s tablicom proizvod.

24. Tablica stanje_skladista_repromaterijala

```
CREATE TABLE stanje_skladista_repromaterijala (
    id_repromaterijal INTEGER PRIMARY KEY,
    kolicina INTEGER NOT NULL,
    FOREIGN KEY (id_repromaterijal) REFERENCES repromaterijal(id)
);
```

Tablica **stanje_skladista_repromaterijala** pohranjuje informacije o stanju repromaterijala u skladištu. Tablica je implementacija materijaliziranog pogleda u MySQL-u i popunjava se pomoću okidača koji pozivaju proceduru.

Atributi:

id_repromaterijal (INTEGER PRIMARY KEY): Jedinstveni identifikator repromaterijala, koji također služi kao primarni ključ. Referencira stupac id iz tablice repromaterijal.

kolicina (INTEGER NOT NULL): Količina repromaterijala koja se nalazi u skladištu.

Strani ključ **id_repromaterijal** povezuje ovu tablicu s tablicom repromaterijal.

4. POGLEDI

POGLED

```
-- DROP VIEW IF EXISTS pogled_administratori;
-- pogled koji pokazuje sve zaposlenike koji imaju prava admina
CREATE VIEW pogled_administratori AS
SELECT zaposlenik.id, zaposlenik.ime, zaposlenik.prezime, uloge.naziv AS uloga
FROM zaposlenik
JOIN uloge ON zaposlenik.uloga_id = uloge.uloga_id
WHERE uloge.naziv = 'admin';

SELECT * FROM pogled_administratori;
```

Ovaj pogled nazvan **pogled_administratori** kreiran je za prikaz organiziranih informacija o zaposlenicima koji imaju prava administratora. On objedinjuje podatke iz tablica **zaposlenik** i **uloge** na sljedeći način:
Prvo se selektiraju osnovni podaci o zaposlenicima, uključujući njihove identifikacijske brojeve (**id**), imena i prezimena, te naziv uloge koju imaju. Ove informacije su povezane putem vanjskog ključa **uloga_id**, koji omogućava povezivanje tablice **zaposlenik** s tablicom **uloge**.

Uvjet u WHERE klauzuli osigurava da se u pogledu prikazuju isključivo zaposlenici čija je uloga označena kao **admin**. Time se omogućuje brz i efikasan pristup podacima o administratorima unutar baze podataka.

Pogled **pogled_administratori** pruža jasan i fokusiran prikaz korisnika s administrativnim pravima, što olakšava upravljanje i nadzor nad tim specifičnim skupom zaposlenika. Korisnici ovog pogleda mogu brzo pretraživati i analizirati podatke o administratorima bez potrebe za ručnim filtriranjem podataka iz glavnih tablica.

POGLED

```
-- pogleda koji su sve zaposlenici neaktivni
CREATE VIEW pogled_neaktivni_korisnici AS
SELECT zaposlenik.id, zaposlenik.ime, zaposlenik.prezime, zaposlenik.status_zaposlenika
FROM zaposlenik
WHERE zaposlenik.status_zaposlenika = 'neaktivan';
SELECT * FROM pogled_neaktivni_korisnici;
```

Ovaj pogled nazvan **pogled_neaktivni_korisnici** kreiran je za prikaz organiziranih informacija o zaposlenicima koji su trenutno neaktivni. Njegova funkcionalnost temelji se na izdvajanja podataka iz tablice **zaposlenik** prema specifičnom uvjetu, čime omogućuje fokusiran prikaz ove grupe zaposlenika.

Prvo se iz tablice **zaposlenik** selektiraju osnovni podaci kao što su identifikacijski broj zaposlenika (**id**), njihova imena i prezimena, te trenutni status zaposlenika (**status_zaposlenika**). Ključni uvjet postavljen u **WHERE** klauzuli osigurava da se u pogledu prikazuju isključivo zaposlenici čiji je status označen kao **neaktivan**.

Ovaj pogled omogućava jasan uvid u popis svih zaposlenika koji nisu aktivni, pružajući korisnicima baze podataka efikasno sredstvo za upravljanje i analizu statusa zaposlenika. Korištenjem ovog pogleda izbjegava se potreba za ručnim filtriranjem podataka u tablici **zaposlenik**, čime se štedi vrijeme i osigurava tačnost u analizi.

Podaci prikazani u pogledu mogu se koristiti za generiranje izvještaja, praćenje radnog statusa ili donošenje odluka vezanih uz ponovnu aktivaciju zaposlenika.

POGLED

```
-- 4. Pogled za administratore
CREATE VIEW pogled_administratori AS
SELECT zaposlenik.id, zaposlenik.ime, zaposlenik.prezime, uloge.naziv AS uloga
FROM zaposlenik
JOIN uloge ON zaposlenik.uloga_id = uloge.uloga_id
WHERE uloge.naziv = 'admin';

SELECT * FROM pogled_administratori;
```

Ovaj pogled nazvan **pogled_administratori** kreiran je za prikaz organiziranih informacija o zaposlenicima koji imaju administrativna prava. Njegova funkcionalnost temelji se na povezivanju podataka iz tablica **zaposlenik** i **uloge**, omogućujući izdvajanje specifičnog skupa korisnika prema njihovoj ulozi.

Prvo se iz tablice **zaposlenik** dohvaćaju osnovni podaci o zaposlenicima, uključujući njihov identifikacijski broj (**id**), ime i prezime. Ovi podaci se povezuju s nazivom uloge (**uloga**) iz tablice **uloge** pomoću vanjskog ključa (**uloga_id**).

Postavljeni uvjet u **WHERE** klauzuli osigurava da se u pogledu prikazuju samo zaposlenici čija je uloga označena kao **admin**. Time se omogućava precizan uvid u listu korisnika s administrativnim pravima unutar sustava.

Ovaj pogled pruža strukturiran i lako dostupan prikaz administratora, što olakšava upravljanje ovim specifičnim skupom korisnika. Korisnicima baze podataka omogućuje brzu identifikaciju administratora, analizu njihovih podataka ili daljnje filtriranje prema drugim kriterijima. Korištenjem ovog pogleda eliminira se potreba za ručnim pretraživanjem podataka u više tablica, čime se povećava efikasnost i tačnost u radu.

```
-- 5. Pogled za neaktivne korisnike
CREATE VIEW pogled_neaktivni_korisnici AS
SELECT zaposlenik.id, zaposlenik.ime, zaposlenik.prezime, zaposlenik.status_zaposlenika
FROM zaposlenik
WHERE zaposlenik.status_zaposlenika = 'neaktivan';

SELECT * FROM pogled_neaktivni_korisnici;
```

Ovaj pogled nazvan **pogled_neaktivni_korisnici** kreiran je za prikaz organiziranih informacija o zaposlenicima čiji je trenutni status označen kao **neaktivan**. Njegova funkcionalnost temelji se na izdvajanju podataka iz tablice **zaposlenik**, omogućujući fokusiran pregled ovog specifičnog skupa korisnika.

Prvo se iz tablice **zaposlenik** selektiraju ključni podaci, uključujući identifikacijski broj zaposlenika (**id**), ime, prezime i status zaposlenika (**status_zaposlenika**). Uvjet u **WHERE** klauzuli osigurava da se u pogledu prikazuju isključivo zaposlenici čiji je status označen kao **neaktivan**.

Ovaj pogled omogućuje brz i jasan pregled svih neaktivnih zaposlenika u sustavu. Korisnicima baze podataka pruža jednostavan alat za praćenje i analizu statusa zaposlenika, bez potrebe za složenim pretraživanjem u glavnoj tablici.

Podaci iz ovog pogleda mogu se koristiti za generiranje izvještaja, donošenje odluka o eventualnoj reaktivaciji korisnika ili analizu razloga neaktivnosti. Na taj način ovaj pogled doprinosi boljoj organizaciji i upravljanju podacima u sustavu.

POGLED

```
CREATE VIEW repromaterijal_po_proizvodu AS
SELECT CONCAT(v.naziv, ' ', b.godina_berbe, ' ', p.volumen, ' L') AS proizvod, r.opis AS repromaterijal
    FROM vino v
    JOIN berba b ON v.id = b.id_vino
    JOIN proizvod p ON p.id_berba = b.id
    JOIN repromaterijal_proizvod rp ON rp.id_proizvod = p.id
    JOIN repromaterijal r ON rp.id_repromaterijal = r.id;
```

Ovaj pogled nazvan **repromaterijal_po_proizvodu** kreiran je za prikaz detaljnih informacija o repromaterijalima koji se koriste za proizvodnju određenih vina. Njegova funkcionalnost temelji se na povezivanju podataka iz više tablica kako bi se generirao pregled koji povezuje proizvode s odgovarajućim repromaterijalima.

Prvo se iz tablica **vino**, **berba** i **proizvod** kombiniraju podaci o vinima, godinama berbe i volumenu proizvoda. Ovi podaci se zatim formatiraju u čitljiv oblik pomoću funkcije **CONCAT**, koja generira opis proizvoda u formatu: naziv vina, godina berbe i volumen (u litrama).

Nakon toga, pomoću tablica **repromaterijal_proizvod** i **repromaterijal**, dohvaćaju se podaci o repromaterijalima koji su povezani s određenim proizvodima. U konačnom rezultatu, za svaki proizvod prikazuje se odgovarajući opis repromaterijala.

Ovaj pogled omogućava sveobuhvatan pregled veze između proizvoda i repromaterijala, čime se olakšava analiza proizvodnog procesa. Koristan je za praćenje utroška repromaterijala po proizvodu, planiranje zaliha i optimizaciju proizvodnih troškova. Podaci iz ovog pogleda mogu se koristiti za generiranje izvještaja ili za donošenje odluka vezanih uz nabavu i korištenje resursa.

POGLED

```
CREATE VIEW vino_skladiste AS
SELECT CONCAT(v.naziv,' ', b.godina_berbe) AS vino, ssv.kolicina
    FROM vino v
    JOIN berba b ON v.id = b.id_vino
    JOIN stanje_skladista_vina ssv ON ssv.id_berba = b.id;
SELECT * FROM stanje_skladista_vina;
```

Ovaj pogled nazvan **vino_skladiste** kreiran je za prikaz organiziranih informacija o stanju vina u skladištu, uključujući podatke o nazivu vina, godini berbe i količini koja je trenutno dostupna. Njegova funkcionalnost temelji se na povezivanju podataka iz više tablica kako bi se dobio pregled trenutnih zaliha.

Prvo se iz tablica **vino** i **berba** kombiniraju podaci o nazivu vina i godini berbe. Ovi podaci se formatiraju u čitljiv oblik pomoću funkcije **CONCAT**, koja generira naziv vina zajedno s godinom berbe u jedinstvenom formatu.

Zatim se, pomoću tablice **stanje_skladista_vina**, dohvaćaju informacije o količinama vina koje se nalaze na skladištu. Ove količine su povezane s određenim berbama preko atributa **id_berba**, čime se omogućava precizno praćenje dostupnih zaliha za svaku godinu berbe.

Ovaj pogled pruža jasan i pregledan prikaz trenutnog stanja vina u skladištu, što je korisno za potrebe praćenja zaliha, planiranja prodaje i optimizacije upravljanja skladištem. Podaci iz ovog pogleda mogu se koristiti za izradu izvještaja o skladišnim zalihama, analizu prodaje ili donošenje odluka o dodatnoj proizvodnji ili nabavi.

POGLED

```
CREATE VIEW proizvod_skladiste AS
SELECT CONCAT(v.naziv, ' ', b.godina_berbe, ' ', p.volumen, ' L') AS proizvod, ssp.kolicina
    FROM vino v
    JOIN berba b ON v.id = b.id_vino
    JOIN proizvod p ON p.id_berba = b.id
    JOIN stanje_skladista_proizvoda ssp ON p.id = ssp.id_proizvod;
```

Ovaj pogled nazvan **proizvod_skladiste** kreiran je za prikaz organiziranih informacija o proizvodima i njihovom trenutnom stanju u skladištu. Njegova funkcionalnost temelji se na povezivanju podataka iz više tablica kako bi se generirao pregled proizvoda s detaljima o dostupnim količinama.

Prvo se iz tablica **vino**, **berba** i **proizvod** kombiniraju podaci o nazivu vina, godini berbe i volumenu proizvoda. Ovi podaci se formatiraju u čitljiv oblik pomoću funkcije **CONCAT**, koja generira jedinstveni opis proizvoda u formatu: naziv vina, godina berbe i volumen (izražen u litrima).

Nakon toga, pomoću tablice **stanje_skladista_proizvoda**, dohvaćaju se informacije o količinama proizvoda koje se nalaze na skladištu. Ove količine su povezane s određenim proizvodima preko atributa **id_proizvod**, što omogućuje precizno praćenje zaliha za svaki pojedini proizvod.

Ovaj pogled pruža detaljan i pregledan uvid u stanje proizvoda u skladištu, uključujući njihove specifikacije i dostupne količine. Koristan je za potrebe praćenja skladišnih zaliha, planiranja prodaje i

proizvodnje te optimizacije upravljanja resursima. Podaci iz ovog pogleda mogu se koristiti za generiranje izvještaja o skladištu, analizu potražnje ili donošenje odluka vezanih uz nabavu i distribuciju proizvoda.

POGLED

```
CREATE VIEW repromaterijal_skladiste AS
SELECT r.opis AS repromaterijal, ssr.kolicina
    FROM repromaterijal r
    JOIN stanje_skladista_repromaterijala ssr ON r.id = ssr.id_repromaterijal;
```

Ovaj pogled nazvan **repromaterijal_skladiste** kreiran je za prikaz organiziranih informacija o repromaterijalima i njihovom trenutnom stanju u skladištu. Njegova funkcionalnost temelji se na povezivanju podataka iz tablica **repromaterijal** i **stanje_skladista_repromaterijala** kako bi se osigurao pregled repromaterijala s pripadajućim zalihama.

Prvo se iz tablice **repromaterijal** dohvaćaju podaci o opisima repromaterijala. Ovaj opis pruža informacije o vrsti materijala koji se koristi u proizvodnim procesima.

Zatim se pomoću tablice **stanje_skladista_repromaterijala** dohvaćaju informacije o količinama repromaterijala koje su trenutno dostupne na skladištu. Povezivanje se vrši putem atributa **id_repromaterijal**, čime se omogućava praćenje zaliha za svaki pojedini repromaterijal.

Ovaj pogled pruža jasan i pregledan uvid u stanje repromaterijala u skladištu, što je ključno za upravljanje zalihama i planiranje nabave. Koristan je za izradu izvještaja o zalihama repromaterijala, analizu potreba u proizvodnim procesima te donošenje odluka vezanih uz nabavu i optimizaciju skladištenja. Na taj način olakšava se praćenje i upravljanje repromaterijalima u sustavu.

POGLED

```
CREATE VIEW kvartalna_prodaja AS
SELECT CONCAT(v.naziv, ' ', b.godina_berbe, ' ', p.volumen, ' L') AS proizvod, kpp.kolicina, kpp.ukupni_iznos, kpp.kvartal
    FROM vino v
    JOIN berba b ON v.id = b.id_vino
    JOIN proizvod p ON p.id_berba = b.id
    JOIN kvartalni_pregled_prodaje kpp ON p.id = kpp.id_proizvod
    ORDER BY kpp.ukupni_iznos DESC;
```

Pogled **kvartalna_prodaja** predstavlja kvartalni izvještaj o prodaji vina, kombinirajući informacije o vinu, berbi, proizvodu i kvartalnim podacima o prodaji. Za svaki proizvod, pogled prikazuje naziv vina, godinu berbe, volumen proizvoda u litrama, prodanu količinu, ukupni iznos prodaje i kvartal. Podaci su sortirani prema ukupnom iznosu prodaje u opadajućem redoslijedu.

POGLED

```
CREATE VIEW punjenje_pogled AS
SELECT CONCAT(v.naziv, ' ', b.godina_berbe, ' ', p.volumen, ' L') AS proizvod, pu.oznaka_serije, pu.pocetak_punjenja, pu.zavrsetak_punjenja, pu.kolicina
    FROM vino v
    JOIN berba b ON v.id = b.id_vino
    JOIN proizvod p ON p.id_berba = b.id
    JOIN punjenje pu ON p.id = pu.id_proizvod
    ORDER BY pu.pocetak_punjenja ASC, pu.oznaka_serije ASC;
```

Ovaj pogled nazvan **punjenje_pogled** kreiran je za prikaz organiziranih informacija o procesima punjenja proizvoda. Njegova funkcionalnost temelji se na povezivanju podataka iz više tablica kako bi se osigurao detaljan pregled serija punjenja vina, uključujući informacije o proizvodima, količinama i vremenskim okvirima.

Prvo se iz tablica **vino**, **berba** i **proizvod** kombiniraju podaci o nazivu vina, godini berbe i volumenu proizvoda. Ovi podaci se formatiraju u čitljiv oblik pomoću funkcije **CONCAT**, koja generira opis proizvoda u formatu: naziv vina, godina berbe i volumen (izražen u litrima).

Nakon toga, pomoću tablice **punjenje**, dohvaćaju se dodatni podaci o procesu punjenja, uključujući oznaku serije (**oznaka_serije**), početak punjenja (**pocetak_punjenja**), završetak punjenja (**zavrsetak_punjenja**) i količinu proizvoda koja je napunjena (**kolicina**). Povezivanje se vrši preko atributa **id_proizvod**, čime se omogućava precizno praćenje svih punjenja vezanih uz određeni proizvod.

Rezultati u ovom pogledu sortirani su prema vremenu početka punjenja (**pocetak_punjenja**) i oznaci serije (**oznaka_serije**) kako bi se osigurao pregled u kronološkom i logičnom redoslijedu.

Ovaj pogled omogućava detaljan uvid u operacije punjenja proizvoda, što je korisno za praćenje proizvodnih procesa, analizu efikasnosti i planiranje budućih aktivnosti. Podaci iz ovog pogleda mogu se koristiti za izradu izvještaja o proizvodnji, praćenje serija proizvoda i osiguranje kvalitete proizvodnih operacija. Na taj način, ovaj pogled doprinosi boljem upravljanju proizvodnim procesima i osigurava transparentnost u radu.

POGLED

```
-- Pogled, svi prijevoznici s adresom u Zagrebu
CREATE VIEW prijevoznici_adresa_zagreb AS
SELECT
    id, naziv, adresa, email, telefon, oib
FROM prijevoznik
WHERE adresa LIKE '%Zagreb%'
  AND adresa NOT LIKE 'Zagrebačka cesta%';
SELECT * FROM prijevoznici_adresa_zagreb;
```

Ovaj pogled, nazvan *prijevoznici_adresa_zagreb*, kreiran je za prikaz organiziranih informacija o prijevoznicima čije je sjedište smješteno u Zagrebu. Funkcionalnost ovog pogleda temelji se na filtriranju podataka iz tablice *prijevoznik*, kako bi se prikazali samo oni prijevoznici čija adresa uključuje naziv "Zagreb", uz iznimku prijevoznika čije adrese počinju s "Zagrebačka cesta".

Podaci uključuju osnovne informacije o prijevoznicima: ID, naziv, adresu, email, telefon i OIB. Ovaj pogled omogućava brzi uvid u sve relevantne prijevoznike sa sjedištem u Zagrebu, čime se olakšava njihovo praćenje i kontaktiranje.

Rezultati u ovom pogledu nisu sortirani jer je primarni cilj pružiti popis prijevoznika prema adresama povezanim s gradom Zagrebom, bez daljnjih organizacija ili filtriranja podataka.

Ovaj pogled je koristan za analizu prijevoznika smještenih u Zagrebu, praćenje njihovih osnovnih podataka te eventualnu upotrebu u izvještavanju i logističkom planiranju.

POGLED

```
-- Pogled, proizvodi s cijenom većom od prosječne

CREATE VIEW proizvodi_iznad_prosjeka AS
SELECT *
FROM proizvod
WHERE cijena > (SELECT AVG(cijena) FROM proizvod);

SELECT * FROM proizvodi_iznad_prosjeka;
```

Ovaj pogled, nazvan *proizvodi_iznad_prosjeka*, kreiran je za prikaz organiziranih informacija o proizvodima čija je cijena veća od prosječne cijene svih proizvoda. Funkcionalnost ovog pogleda temelji se na filtriranju podataka iz tablice *proizvod*, pri čemu se prikazuju samo oni proizvodi čija cijena premašuje prosječnu cijenu svih proizvoda u sustavu.

Podaci uključuju sve atribute iz tablice *proizvod*, čime se osigurava potpuni pregled proizvoda koji ispunjavaju ovaj uvjet, bez dodatnih obradbi ili transformacija. Prosječna cijena svih proizvoda izračunava se pomoću ugrađene funkcije *AVG(cijena)*, a rezultati se filtriraju tako da uključuju samo proizvode čija cijena premašuje ovaj iznos.

Ovaj pogled omogućava detaljan uvid u proizvode koji imaju višu cijenu od prosjeka, što je korisno za analize cijena, identifikaciju premium proizvoda, kao i za donošenje poslovnih odluka u vezi s cijenama i asortimanom proizvoda. Podaci iz ovog pogleda mogu se koristiti za izradu izvještaja o cijenama, strategijama prodaje ili za usporedbu različitih proizvoda u asortimanu.

POGLED

```
-- Zaposlenici koji rade u prodaji

CREATE VIEW View_Zaposlenici_Prodaja AS
SELECT z.id, z.ime, z.prezime, z.adresa, z.email, z.telefon
FROM zaposlenik z
JOIN odjel o ON z.id_odjel = o.id
WHERE o.naziv = 'Prodaja';
```

Ovaj pogled, nazvan *View_Zaposlenici_Prodaja*, kreiran je za prikaz organiziranih informacija o zaposlenicima koji rade u odjelu prodaje. Funkcionalnost ovog pogleda temelji se na povezivanju podataka iz tablica *zaposlenik* i *odjel*, kako bi se osigurali podaci o zaposlenicima koji pripadaju odjelu s nazivom "Prodaja".

Podaci uključuju osnovne informacije o zaposlenicima: ID, ime, prezime, adresu, email i telefon. Povezivanje se vrši preko atributa *id_odjel*, čime se omogućava precizno filtriranje zaposlenika prema njihovom pripadništvu određenom odjelu.

Rezultati u ovom pogledu filtrirani su tako da uključuju samo zaposlenike koji rade u odjelu prodaje, a sortiranje nije specifično definirano jer je cilj pružiti popis zaposlenika u tom odjelu bez dodatnih organizacija podataka.

Ovaj pogled omogućava uvid u zaposlenike koji rade u prodaji, što je korisno za analizu ljudskih resursa, planiranje prodajnih aktivnosti ili kontaktiranje relevantnih članova tima. Podaci iz ovog pogleda mogu se koristiti za izradu izvještaja o zaposlenicima, optimizaciju timova ili za operativno upravljanje prodajnim odjelom.

POGLED

```sql
-- Kupci koji imaju e-mail domenu @vina.hr

CREATE VIEW View_Kupci_Vina_HR AS
SELECT naziv, oib, ime, prezime, adresa, email, telefon
FROM kupac
WHERE email LIKE '%@vina.hr';
```

Ovaj pogled, nazvan *View_Kupci_Vina_HR*, kreiran je za prikaz organiziranih informacija o kupcima koji imaju e-mail adresu s domenom *@vina.hr*. Funkcionalnost ovog pogleda temelji se na filtriranju podataka iz tablice *kupac*, pri čemu se prikazuju samo oni kupci čije e-mail adrese sadrže ovu specifičnu domenu.

Podaci uključuju osnovne informacije o kupcima: naziv, OIB, ime, prezime, adresu, email i telefon. Filtriranje se vrši pomoću uvjeta *email LIKE '%@vina.hr'*, čime se osigurava da se prikazuju samo kupci koji koriste ovu domenu za svoje e-mail adrese.

Rezultati u ovom pogledu nisu sortirani, jer je primarni cilj pružiti popis kupaca koji imaju e-mail adresu s domenom *@vina.hr*, bez dodatnih obradbi ili organizacija podataka.

Ovaj pogled omogućava uvid u kupce koji pripadaju specifičnoj grupi prema njihovoj e-mail domeni, što je korisno za marketinške aktivnosti, analize kupaca ili ciljanje specifične skupine u poslovnim kampanjama. Podaci iz ovog pogleda mogu se koristiti za izradu ciljanih promotivnih akcija, poboljšanje korisničkog iskustva ili za kontaktiranje relevantnih kupaca.

POGLED

```sql
-- 1. Pogled: Prikaz svih transporta s nazivima prijevoznika
CREATE VIEW transport_prikaz AS
SELECT t.id AS id_transport, t.datum_polaska, t.datum_dolaska, t.kolicina, t.status_transporta, p.naziv AS prijevoznik
FROM transport t
JOIN prijevoznik p ON t.id_prijevoznik = p.id;
```

Ovaj pogled, nazvan *transport_prikaz*, kreiran je za prikaz organiziranih informacija o transportima zajedno s podacima o prijevozniku. Funkcionalnost ovog pogleda temelji se na povezivanju podataka iz tablica *transport* i *prijevoznik*, kako bi se osigurao detaljan pregled informacija o transportnim

operacijama, uključujući datum polaska, datum dolaska, količinu, status transporta i naziv prijevoznika koji obavlja transport.

Podaci uključuju osnovne informacije o transportima: ID transporta, datum polaska, datum dolaska, količinu, status transporta, te naziv prijevoznika. Povezivanje se vrši putem atributa *id_prijevoznik*, čime se omogućava detaljan uvid u svakodnevne transportne aktivnosti zajedno s informacijama o prijevoznicima koji sudjeluju u tim procesima.

Rezultati u ovom pogledu nisu sortirani, jer je cilj pružiti jasan pregled svih transportnih aktivnosti s pripadajućim podacima o prijevoznicima, bez dodatnog organiziranja podataka.

Ovaj pogled omogućava uvid u transportne operacije, praćenje statusa i količina koje se prevoze, te identifikaciju prijevoznika odgovornog za svaku operaciju. Podaci iz ovog pogleda mogu se koristiti za analize učinkovitosti transporta, praćenje logističkih aktivnosti, optimizaciju rute ili za izvještavanje o izvršenju transportnih zadataka.

POGLED

```
-- 2. Pogled: Prikaz prijevoznika i njihovih ukupnih količina transporta
CREATE VIEW prijevoznik_kolicina AS
SELECT p.naziv AS prijevoznik, SUM(t.kolicina) AS ukupna_kolicina
FROM prijevoznik p
LEFT JOIN transport t ON p.id = t.id_prijevoznik
GROUP BY p.naziv;
```

Ovaj pogled, nazvan *prijevoznik_kolicina*, kreiran je za prikaz organiziranih informacija o ukupnim količinama koje su prevezene po svakom prijevozniku. Funkcionalnost ovog pogleda temelji se na povezivanju podataka iz tablica *prijevoznik* i *transport*, kako bi se izračunale ukupne količine robe koju je svaki prijevoznik prevezao.

Podaci uključuju naziv prijevoznika i ukupnu količinu prevezenu od strane svakog prijevoznika. Povezivanje se vrši pomoću *LEFT JOIN* između tablica *prijevoznik* i *transport* na osnovu *id* prijevoznika i *id_prijevoznik*, čime se osigurava da se prikažu svi prijevoznici, uključujući i one koji nisu imali nikakve transportne operacije. Ukupna količina prevezena od svakog prijevoznika izračunava se pomoću funkcije *SUM*.

Rezultati u ovom pogledu grupirani su prema nazivu prijevoznika kako bi se prikazale ukupne količine po svakom prijevozniku, bez dodatnog sortiranja podataka.

Ovaj pogled omogućava uvid u ukupne količine prevezene od strane svakog prijevoznika, što je korisno za analize performansi prijevoznika, praćenje učinkovitosti transportnih usluga, kao i za planiranje daljnjih transportnih aktivnosti i optimizaciju logističkih procesa. Podaci iz ovog pogleda mogu se koristiti za izvještavanje o obimu transporta, analizu tržišnog udjela prijevoznika ili za donošenje poslovnih odluka temeljenih na količinama prevezenih dobara.

POGLED

```
-- Podaci o zaposlenicima s odjelom i statusom

CREATE VIEW View_Zaposlenici_S_ODjelom AS
SELECT z.id, z.ime, z.prezime, z.adresa, z.email, z.telefon, z.datum_zaposlenja, z.status_zaposlenika, o.naziv AS odjel
FROM zaposlenik z
JOIN odjel o ON z.id_odjel = o.id;
```

Ovaj pogled, nazvan *View_Zaposlenici_S_ODjelom*, kreiran je za prikaz organiziranih informacija o zaposlenicima zajedno s podacima o njihovom odjelu i statusu zaposlenja. Funkcionalnost ovog pogleda temelji se na povezivanju podataka iz tablica *zaposlenik* i *odjel*, kako bi se osigurali detalji o zaposlenicima, uključujući njihove osobne podatke, datum zaposlenja, status te naziv odjela u kojem rade.

Podaci uključuju osnovne informacije o zaposlenicima: ID, ime, prezime, adresu, email, telefon, datum zaposlenja, status zaposlenika, te naziv odjela u kojem je zaposlenik smješten. Povezivanje s tablicom *odjel* vrši se preko atributa *id_odjel*, čime se omogućava detaljan prikaz zaposlenika zajedno s informacijama o njihovim odjelima.

Rezultati u ovom pogledu nisu sortirani, jer je primarni cilj prikazati kompletne informacije o zaposlenicima uz povezanost s odjelima i statusima, bez dodatnog organiziranja podataka.

Ovaj pogled omogućava detaljan uvid u zaposlenike, njihov status zaposlenja i pripadnost određenim odjelima, što je korisno za analizu ljudskih resursa, praćenje organizacijske strukture i podršku upravljanju timovima. Podaci iz ovog pogleda mogu se koristiti za izvještaje o zaposlenicima, optimizaciju raspodjele resursa ili za strateško planiranje ljudskih resursa.

5. OKIDAČI

OKIDAČ

```
DELIMITER //
CREATE TRIGGER prije_transport_update
BEFORE UPDATE ON transport
FOR EACH ROW
BEGIN
    IF NEW.status_transporta = 'Obavljen' AND (NEW.datum_dolaska IS NULL OR NEW.datum_dolaska = '') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Unesi datum_dolaska.';
    END IF;
END;//
DELIMITER ;
```

Ovaj okidač, nazvan *prije_transport_update*, kreiran je kako bi osigurao da se prije nego što se ažurira zapis u tablici *transport*, provjeri ispunjenost polja *datum_dolaska* za transport koji ima status *Obavljen*. Funkcionalnost ovog okidača temelji se na uvjetima koji se moraju zadovoljiti prije nego što se izvrši ažuriranje zapisa, čime se sprječava ažuriranje bez unosa potrebnog datuma dolaska.

Ako je status transporta postavljen na *Obavljen*, a polje *datum_dolaska* je prazno ili nije uneseno (NULL), okidač aktivira signal greške pomoću *SIGNAL SQLSTATE '45000'*, te korisniku prikazuje poruku *Unesi datum_dolaska*. Ovaj uvjet osigurava da se datum dolaska ne može ostaviti praznim prilikom označavanja transporta kao obavljenog.

Ovaj okidač omogućava provođenje poslovnih pravila koja sprječavaju nesređene podatke i osiguravaju točnost informacija o transportima. Podaci iz ovog okidača pomažu u održavanju integriteta baze podataka, čime se smanjuje mogućnost pogrešaka i osigurava pravilno praćenje transportnih operacija.

OKIDAČ

```
-- 2. Triger: Automatsko ažuriranje datuma dolaska prilikom promjene statusa transporta na "Otkazan"
DELIMITER //
CREATE TRIGGER za_otkazani_transport
BEFORE UPDATE ON transport
FOR EACH ROW
BEGIN
    IF NEW.status_transporta = 'Otkazan' AND OLD.status_transporta <> 'Otkazan' THEN
        SET NEW.datum_dolaska = NULL;
    END IF;
END;//
DELIMITER ;
```

Ovaj okidač, nazvan *za_otkazani_transport*, kreiran je kako bi osigurao da se prilikom ažuriranja zapisa u tablici *transport*, polje *datum_dolaska* postavi na *NULL* ako se status transporta promijeni u *Otkazan*. Funkcionalnost ovog okidača temelji se na uvjetima koji provjeravaju da li se status transporta promijenio u *Otkazan*, te ako je to slučaj, automatski poništava uneseni datum dolaska.

Okidač aktivira se prije ažuriranja zapisa, te provodi provjeru da li je novi status *Otkazan*, a prethodni status nije bio *Otkazan*. Ako su ovi uvjeti zadovoljeni, polje *datum_dolaska* postavlja se na *NULL*, čime se osigurava da za otkazani transport nije naveden datum dolaska, što je u skladu s poslovnim pravilima.

Ovaj okidač pomaže u održavanju konzistentnosti podataka i točnosti u vezi sa statusima transporta, sprečavajući da transporti koji su otkazani imaju datum dolaska. Podaci iz ovog okidača omogućavaju bolju kontrolu nad transportnim informacijama i osiguravaju pravilno praćenje statusa transporta u sustavu.

OKIDAČ

```
-- Trigger za automatsko postavljanje statusa na 'aktivan' prilikom unosa novog zaposlenika
DELIMITER //
CREATE TRIGGER postavi_status_na_aktivan
BEFORE INSERT ON zaposlenik
FOR EACH ROW
BEGIN
    IF NEW.status_zaposlenika IS NULL THEN
        SET NEW.status_zaposlenika = 'aktivan';
    END IF;
END;
//
DELIMITER ;
```

Ovaj okidač, nazvan *postavi_status_na_aktivan*, kreiran je kako bi osigurao da se prilikom unosa novog zapisa u tablicu *zaposlenik*, polje *status_zaposlenika* automatski postavi na *aktivan* ako nije specificirano nikakvo drugo stanje. Funkcionalnost ovog okidača temelji se na provjeri da li je novo polje *status_zaposlenika* prazno ili NULL, te ako je to slučaj, postavlja ga na zadanu vrijednost *aktivan*.

Okidač se aktivira prije unosa novog zapisa, te provodi provjeru da li je vrijednost *status_zaposlenika* NULL. Ako je, automatski se postavlja na *aktivan*, čime se osigurava da svaki novi zaposlenik, ako nije drugačije specificirano, bude označen kao aktivan u sustavu.

Ovaj okidač pomaže u održavanju dosljednosti podataka u tablici *zaposlenik* i sprečava unos zaposlenika bez odgovarajućeg statusa, čime se smanjuje mogućnost pogrešaka u vođenju podataka o zaposlenicima. Podaci iz ovog okidača omogućuju automatizaciju procesa unosa i olakšavaju upravljanje statusima zaposlenika.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ai_zaposlenik
    AFTER INSERT ON zaposlenik
    FOR EACH ROW
BEGIN
    CALL azuriraj_broj_zaposlenika(new.id_odjel);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ai_zaposlenik*, kreiran je kako bi nakon unosa novog zapisa u tablicu *zaposlenik* automatski pokrenuo proceduru koja ažurira broj zaposlenika u odjelu kojem novi zaposlenik pripada. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_broj_zaposlenika* koja se izvršava nakon što je novi zaposlenik unesen, koristeći njegov *id_odjel* kao argument.

Okidač se aktivira nakon unosa zapisa u tablicu *zaposlenik*, te poziva proceduru *azuriraj_broj_zaposlenika* s *new.id_odjel* kao ulaznim parametrom. Na taj način, broj zaposlenika u odgovarajućem odjelu se automatski ažurira svaki put kada novi zaposlenik bude unesen u sustav, čime se održava točnost podataka o broju zaposlenika po odjelima.

Ovaj okidač omogućava automatizaciju procesa praćenja broja zaposlenika u odjelima, čime se olakšava upravljanje ljudskim resursima i osigurava da podaci o zaposlenicima ostanu ažurni i precizni. Podaci iz ovog okidača pomažu u optimizaciji administrativnih procesa i mogu se koristiti za daljnje analize ili izvještavanje o strukturi zaposlenika unutar organizacije.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ad_zaposlenik
    AFTER DELETE ON zaposlenik
    FOR EACH ROW
BEGIN
    CALL azuriraj_broj_zaposlenika(old.id_odjel);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ad_zaposlenik*, kreiran je kako bi nakon brisanja zapisa iz tablice *zaposlenik* automatski pokrenuo proceduru koja ažurira broj zaposlenika u odjelu kojem je pripadao obrisani zaposlenik. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_broj_zaposlenika* koja se izvršava nakon što je zaposlenik izbrisan, koristeći *old.id_odjel* kao argument.

Okidač se aktivira nakon brisanja zapisa iz tablice *zaposlenik*, te poziva proceduru *azuriraj_broj_zaposlenika* s *old.id_odjel* kao ulaznim parametrom. Na taj način, broj zaposlenika u odgovarajućem odjelu se automatski ažurira svaki put kada zaposlenik bude obrisan iz sustava, čime se održava točnost podataka o broju zaposlenika po odjelima.

Ovaj okidač omogućava automatizaciju procesa praćenja broja zaposlenika u odjelima, čime se olakšava upravljanje ljudskim resursima i osigurava da podaci o zaposlenicima ostanu ažurni i precizni.

Podaci iz ovog okidača pomažu u optimizaciji administrativnih procesa i mogu se koristiti za daljnje analize ili izvještavanje o strukturi zaposlenika unutar organizacije.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER au_zaposlenik
    AFTER UPDATE ON zaposlenik
    FOR EACH ROW
BEGIN
    -- ako je zaposlenik promijenio odjel
    IF new.id_odjel != old.id_odjel THEN
        CALL azuriraj_broj_zaposlenika(old.id_odjel);
        CALL azuriraj_broj_zaposlenika(new.id_odjel);
    ELSE  -- ako se promijenio status zaposlenika
        IF new.status_zaposlenika != old.status_zaposlenika THEN
            CALL azuriraj_broj_zaposlenika(new.id_odjel);
        END IF;
    END IF;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *au_zaposlenik*, kreiran je kako bi nakon ažuriranja zapisa u tablici *zaposlenik* automatski pokrenuo proceduru koja ažurira broj zaposlenika u odjelu kojem pripada zaposlenik, ovisno o promjenama u njegovom odjelu ili statusu. Funkcionalnost ovog okidača temelji se na provjerama koje analiziraju promjene u odjelu i statusu zaposlenika te, u skladu s tim, pokreću odgovarajuće procedure za ažuriranje broja zaposlenika u odjelima.

Okidač se aktivira nakon ažuriranja zapisa u tablici *zaposlenik*, a logika okidača provodi dvije ključne provjere:

1. Ako je zaposlenik promijenio odjel (tj. ako *new.id_odjel* nije isto kao *old.id_odjel*), pokreću se dvije procedure *azuriraj_broj_zaposlenika* – jedna za stari odjel (*old.id_odjel*) i jedna za novi odjel (*new.id_odjel*), čime se ažurira broj zaposlenika u oba odjela.
2. Ako nije došlo do promjene odjela, ali se promijenio status zaposlenika (tj. ako *new.status_zaposlenika* nije isto kao *old.status_zaposlenika*), tada se pokreće samo jedna procedura *azuriraj_broj_zaposlenika* za odjel u kojem zaposlenik trenutno radi (*new.id_odjel*).

Ovaj okidač omogućava automatsku i preciznu prilagodbu broja zaposlenika u odjelima prilikom bilo kakvih promjena, bilo da se radi o premještaju zaposlenika u drugi odjel ili promjeni njegovog statusa. Podaci iz ovog okidača pomažu u održavanju točnih i ažurnih podataka o strukturi zaposlenika u organizaciji te omogućuju bolje praćenje ljudskih resursa i optimizaciju administrativnih procesa.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER bi_stavka_narudzbe
    BEFORE INSERT ON stavka_narudzbe
    FOR EACH ROW
BEGIN
    DECLARE cijena_proizvoda DECIMAL(10,2);

    SELECT cijena INTO cijena_proizvoda
        FROM proizvod
        WHERE id = new.id_proizvod;

    SET new.iznos_stavke = new.kolicina * cijena_proizvoda;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *bi_stavka_narudzbe*, kreiran je kako bi prije unosa novog zapisa u tablicu *stavka_narudzbe* automatski izračunao iznos stavke na temelju količine i cijene proizvoda. Funkcionalnost ovog okidača temelji se na dohvaćanju cijene proizvoda iz tablice *proizvod* i množenju te cijene s količinom unesenom za stavku narudžbe.

Okidač se aktivira prije unosa zapisa u tablicu *stavka_narudzbe* i koristi deklariranu varijablu *cijena_proizvoda* kako bi pohranio cijenu proizvoda koji se nalazi u novoj stavci narudžbe. Cijena proizvoda dohvaća se iz tablice *proizvod* prema *new.id_proizvod*. Nakon što se cijena proizvoda dohvaćena, izračunava se *iznos_stavke* tako da se cijena pomnoži s količinom proizvoda u stavci narudžbe.

Ovaj okidač omogućava automatski izračun iznosa stavke prilikom unosa novih narudžbi, čime se smanjuje potreba za ručnim unosom podataka i osigurava točnost iznosa u sustavu. Podaci iz ovog okidača pomažu u ubrzavanju procesa obrade narudžbi i smanjuju mogućnost pogrešaka u unosu podataka, što doprinosi boljim poslovnim procesima i točnim izvještajima.

OKIDAČ

```
-- trigger za izracun ukupni_iznos u zahtjev_za_narudzbu

DELIMITER //
CREATE TRIGGER ai_stavka_narudzbe
    AFTER INSERT ON stavka_narudzbe
    FOR EACH ROW
BEGIN
    UPDATE zahtjev_za_narudzbu
        SET ukupni_iznos = (
            SELECT SUM(iznos_stavke)
                FROM stavka_narudzbe
                WHERE id_zahtjev_za_narudzbu = new.id_zahtjev_za_narudzbu
        )
        WHERE id = new.id_zahtjev_za_narudzbu;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ai_stavka_narudzbe*, kreiran je kako bi nakon unosa novog zapisa u tablicu *stavka_narudzbe* automatski ažurirao ukupni iznos narudžbe u tablici *zahtjev_za_narudzbu*. Funkcionalnost ovog okidača temelji se na izračunu ukupnog iznosa svih stavki narudžbe i njegovom ažuriranju u odgovarajućem zahtjevu za narudžbu.

Okidač se aktivira nakon unosa novog zapisa u tablicu *stavka_narudzbe*, te pokreće ažuriranje polja *ukupni_iznos* u tablici *zahtjev_za_narudzbu*. Izračun ukupnog iznosa vrši se korištenjem *SUM* funkcije, koja zbraja *iznos_stavke* za sve stavke narudžbe povezane s odgovarajućim *id_zahtjev_za_narudzbu* (identifikatorom zahtjeva za narudžbu). Ažurirani ukupni iznos tada se zapisuje u polje *ukupni_iznos* za odgovarajući zapis u tablici *zahtjev_za_narudzbu*.

Ovaj okidač omogućava automatsko ažuriranje ukupnog iznosa narudžbe svaki put kada se unese nova stavka, čime se osigurava da *ukupni_iznos* uvijek bude točan i ažuran. Podaci iz ovog okidača pomažu u optimizaciji procesa obrade narudžbi, smanjujući potrebu za ručnim izračunima i osiguravajući točnost podataka, što doprinosi efikasnijem praćenju narudžbi i financijskim izvještajima.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ai_skladiste_vino_akv
    AFTER INSERT ON skladiste_vino
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_vina(new.id_berba, new.tip_transakcije, new.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ai_skladiste_vino_akv*, kreiran je kako bi nakon unosa novog zapisa u tablicu *skladiste_vino* automatski pokrenuo proceduru koja ažurira količinu vina u skladištu na temelju unosa. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_vina*, koja ažurira količinu vina u skladištu ovisno o vrsti transakcije (prijem, isporuka i sl.) i količini vina.

Okidač se aktivira nakon unosa novog zapisa u tablicu *skladiste_vino*, te poziva proceduru *azuriraj_kolicinu_vina* s parametrom *new.id_berba* (koji označava godinu berbe vina), *new.tip_transakcije* (koji označava vrstu transakcije, poput prijema ili isporuke vina) i *new.kolicina* (koja označava količinu vina u transakciji).

Ovaj okidač omogućava automatsko ažuriranje količine vina u skladištu svaki put kada se unese nova transakcija, čime se osigurava da podaci o količinama u skladištu uvijek budu točni i ažurni. Podaci iz ovog okidača pomažu u praćenju skladišnih zaliha, optimizaciji logističkih procesa i preciznom izvještavanju o stanju vina u skladištima, čime se olakšava upravljanje skladištem i logističkim operacijama.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ad_skladiste_vino_akv
    AFTER DELETE ON skladiste_vino
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_vina(old.id_berba, old.tip_transakcije, -old.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ad_skladiste_vino_akv*, kreiran je kako bi nakon brisanja zapisa iz tablice *skladiste_vino* automatski pokrenuo proceduru koja ažurira količinu vina u skladištu, smanjujući je u skladu s količinom koja je obrisana. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_vina*, koja smanjuje količinu vina u skladištu prema vrsti transakcije i količini vina koja je obrisana.

Okidač se aktivira nakon brisanja zapisa iz tablice *skladiste_vino*, te poziva proceduru *azuriraj_kolicinu_vina* s parametrima *old.id_berba* (koji označava godinu berbe vina), *old.tip_transakcije* (koji označava vrstu transakcije) i *-old.kolicina* (koja označava količinu koja je obrisana, uz negativni predznak kako bi količina bila smanjena).

Ovaj okidač omogućava automatsko smanjenje količine vina u skladištu svaki put kada se obriše transakcija, čime se održava točnost podataka o količinama vina u skladištu. Podaci iz ovog okidača pomažu u praćenju promjena u skladištima, osiguravajući da se količine uvijek usklade s stvarnim stanjem, te omogućuju precizno izvještavanje i optimizaciju logističkih i skladišnih procesa.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER au_skladiste_vino_akv
    AFTER UPDATE ON skladiste_vino
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_vina(old.id_berba, old.tip_transakcije, -old.kolicina);
    CALL azuriraj_kolicinu_vina(new.id_berba, new.tip_transakcije, new.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *au_skladiste_vino_akv*, kreiran je kako bi nakon ažuriranja zapisa u tablici *skladiste_vino* automatski pokrenuo proceduru koja ažurira količinu vina u skladištu, uzimajući u obzir kako staru tako i novu količinu vina. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_vina*, koja ažurira količinu vina u skladištu tako što prvo smanjuje količinu prema starom zapisu, a zatim povećava prema novom zapisu.

Okidač se aktivira nakon ažuriranja zapisa u tablici *skladiste_vino*, te pokreće dvije procedure:

1. *azuriraj_kolicinu_vina* se poziva s parametrima *old.id_berba* (koji označava godinu berbe vina), *old.tip_transakcije* (koji označava vrstu transakcije) i *-old.kolicina* (koja označava količinu koja se smanjuje prema starom zapisu, uz negativni predznak).
2. *azuriraj_kolicinu_vina* se ponovo poziva, ali s novim parametrima *new.id_berba*, *new.tip_transakcije* i *new.kolicina* (koji označavaju novu količinu vina koja treba biti ažurirana u skladištu).

Ovaj okidač omogućava točno praćenje promjena u količinama vina u skladištu nakon ažuriranja podataka, osiguravajući da se količine uvijek usklade sa stvarnim stanjem nakon izmjena. Podaci iz ovog okidača pomažu u održavanju točnosti skladišnih zaliha, optimizaciji skladišnih operacija i osiguravanju preciznog izvještavanja o stanju vina u skladištima.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ai_skladiste_proizvod
    AFTER INSERT ON skladiste_proizvod
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_proizvoda(new.id_proizvod, new.tip_transakcije, new.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ai_skladiste_proizvod*, kreiran je kako bi nakon unosa novog zapisa u tablicu *skladiste_proizvod* automatski pokrenuo proceduru koja ažurira količinu proizvoda u skladištu na temelju unosa. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_proizvoda*, koja ažurira količinu proizvoda u skladištu, ovisno o vrsti transakcije (prijem, isporuka i sl.) i količini proizvoda.

Okidač se aktivira nakon unosa novog zapisa u tablicu *skladiste_proizvod*, te poziva proceduru *azuriraj_kolicinu_proizvoda* s parametrima *new.id_proizvod* (koji označava identifikator proizvoda), *new.tip_transakcije* (koji označava vrstu transakcije, poput prijema ili isporuke proizvoda) i *new.kolicina* (koja označava količinu proizvoda u transakciji).

Ovaj okidač omogućava automatsko ažuriranje količine proizvoda u skladištu svaki put kada se unese nova transakcija, čime se osigurava da podaci o količinama proizvoda uvijek budu točni i ažurni. Podaci iz ovog okidača pomažu u praćenju skladišnih zaliha, optimizaciji logističkih procesa i preciznom izvještavanju o stanju proizvoda u skladištima, čime se olakšava upravljanje skladištem i logističkim operacijama.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ad_skladiste_proizvod
    AFTER DELETE ON skladiste_proizvod
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_proizvoda(old.id_proizvod, old.tip_transakcije, -old.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ad_skladiste_proizvod*, kreiran je kako bi nakon brisanja zapisa iz tablice *skladiste_proizvod* automatski pokrenuo proceduru koja ažurira količinu proizvoda u skladištu smanjujući je u skladu s količinom koja je obrisana. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_proizvoda*, koja smanjuje količinu proizvoda u skladištu prema vrsti transakcije i količini proizvoda koja je obrisana.

Okidač se aktivira nakon brisanja zapisa iz tablice *skladiste_proizvod*, te poziva proceduru *azuriraj_kolicinu_proizvoda* s parametrima *old.id_proizvod* (koji označava identifikator proizvoda), *old.tip_transakcije* (koji označava vrstu transakcije) i *-old.kolicina* (koja označava količinu koja je obrisana, uz negativni predznak kako bi količina bila smanjena).

Ovaj okidač omogućava automatsko smanjenje količine proizvoda u skladištu svaki put kada se obriše transakcija, čime se održava točnost podataka o količinama proizvoda u skladištu. Podaci iz ovog okidača pomažu u praćenju promjena u skladištima, osiguravajući da se količine uvijek usklade s stvarnim stanjem, te omogućuju precizno izvještavanje i optimizaciju logističkih i skladišnih procesa.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER au_skladiste_proizvod
    AFTER UPDATE ON skladiste_proizvod
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_proizvoda(old.id_proizvod, old.tip_transakcije, -old.kolicina);
    CALL azuriraj_kolicinu_proizvoda(new.id_proizvod, new.tip_transakcije, new.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *au_skladiste_proizvod*, kreiran je kako bi nakon ažuriranja zapisa u tablici *skladiste_proizvod* automatski pokrenuo proceduru koja ažurira količinu proizvoda u skladištu, uzimajući u obzir kako staru tako i novu količinu proizvoda. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_proizvoda*, koja ažurira količinu proizvoda u skladištu smanjujući količinu prema starom zapisu, a zatim povećava količinu prema novom zapisu.

Okidač se aktivira nakon ažuriranja zapisa u tablici *skladiste_proizvod*, te pokreće dvije procedure:

1. *azuriraj_kolicinu_proizvoda* se poziva s parametrima *old.id_proizvod* (koji označava identifikator proizvoda), *old.tip_transakcije* (koji označava vrstu transakcije) i *-old.kolicina* (koja označava količinu koja se smanjuje prema starom zapisu, uz negativni predznak).
2. *azuriraj_kolicinu_proizvoda* se ponovo poziva, ali s novim parametrima *new.id_proizvod*, *new.tip_transakcije* i *new.kolicina* (koji označavaju novu količinu proizvoda koja treba biti ažurirana u skladištu).

Ovaj okidač omogućava točno praćenje promjena u količinama proizvoda u skladištu nakon ažuriranja podataka, osiguravajući da se količine uvijek usklade sa stvarnim stanjem nakon izmjena. Podaci iz ovog okidača pomažu u održavanju točnosti skladišnih zaliha, optimizaciji skladišnih operacija i osiguravanju preciznog izvještavanja o stanju proizvoda u skladištima.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ai_skladiste_repromaterijal
    AFTER INSERT ON skladiste_repromaterijal
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_repromaterijala(new.id_repromaterijal, new.tip_transakcije, new.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ai_skladiste_repromaterijal*, kreiran je kako bi nakon unosa novog zapisa u tablicu *skladiste_repromaterijal* automatski pokrenuo proceduru koja ažurira količinu repromaterijala u skladištu, na temelju unesenih podataka o transakciji. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_repromaterijala*, koja ažurira količinu repromaterijala u skladištu, uzimajući u obzir vrstu transakcije i količinu repromaterijala.

Okidač se aktivira nakon unosa novog zapisa u tablicu *skladiste_repromaterijal*, te poziva proceduru *azuriraj_kolicinu_repromaterijala* s parametrima *new.id_repromaterijal* (koji označava identifikator

repromaterijala), *new.tip_transakcije* (koji označava vrstu transakcije, poput prijema ili isporuke repromaterijala) i *new.kolicina* (koja označava količinu repromaterijala koja je uključena u transakciju).

Ovaj okidač omogućava automatsko ažuriranje količine repromaterijala u skladištu svaki put kada se unese nova transakcija, čime se osigurava da podaci o količinama repromaterijala uvijek budu točni i ažurni. Podaci iz ovog okidača pomažu u praćenju skladišnih zaliha repromaterijala, optimizaciji logističkih procesa i preciznom izvještavanju o stanju repromaterijala u skladištima, čime se olakšava upravljanje skladištem i logističkim operacijama.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER ad_skladiste_repromaterijal
    AFTER DELETE ON skladiste_repromaterijal
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_repromaterijala(old.id_repromaterijal, old.tip_transakcije, -old.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *ad_skladiste_repromaterijal*, kreiran je kako bi nakon brisanja zapisa iz tablice *skladiste_repromaterijal* automatski pokrenuo proceduru koja ažurira količinu repromaterijala u skladištu smanjujući je u skladu s količinom koja je obrisana. Funkcionalnost ovog okidača temelji se na pozivu procedure *azuriraj_kolicinu_repromaterijala*, koja smanjuje količinu repromaterijala u skladištu prema vrsti transakcije i količini repromaterijala koja je obrisana.

Okidač se aktivira nakon brisanja zapisa iz tablice *skladiste_repromaterijal*, te poziva proceduru *azuriraj_kolicinu_repromaterijala* s parametrima *old.id_repromaterijal* (koji označava identifikator repromaterijala), *old.tip_transakcije* (koji označava vrstu transakcije) i *-old.kolicina* (koja označava količinu koja je obrisana, uz negativni predznak kako bi količina bila smanjena).

Ovaj okidač omogućava automatsko smanjenje količine repromaterijala u skladištu svaki put kada se obriše transakcija, čime se održava točnost podataka o količinama repromaterijala u skladištu. Podaci iz ovog okidača pomažu u praćenju promjena u skladištima repromaterijala, osiguravajući da se količine uvijek usklade s stvarnim stanjem, te omogućuju precizno izvještavanje i optimizaciju logističkih i skladišnih procesa.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER au_skladiste_repromaterijal
    AFTER UPDATE ON skladiste_repromaterijal
    FOR EACH ROW
BEGIN
    CALL azuriraj_kolicinu_repromaterijala(old.id_repromaterijal, old.tip_transakcije, -old.kolicina);
    CALL azuriraj_kolicinu_repromaterijala(new.id_repromaterijal, new.tip_transakcije, new.kolicina);
END //
DELIMITER ;
```

Ovaj okidač, nazvan *au_skladiste_repromaterijal*, kreiran je kako bi nakon ažuriranja zapisa u tablici *skladiste_repromaterijal* automatski pokrenuo proceduru koja ažurira količinu repromaterijala u skladištu, uzimajući u obzir kako staru tako i novu količinu repromaterijala. Funkcionalnost ovog

okidača temelji se na pozivu procedure *azuriraj_kolicinu_repromaterijala*, koja smanjuje količinu repromaterijala prema starom zapisu i zatim povećava količinu prema novom zapisu.

Okidač se aktivira nakon ažuriranja zapisa u tablici *skladiste_repromaterijal*, te pokreće dvije procedure:

1. *azuriraj_kolicinu_repromaterijala* se poziva s parametrima *old.id_repromaterijal* (koji označava identifikator repromaterijala), *old.tip_transakcije* (koji označava vrstu transakcije) i *-old.kolicina* (koja označava količinu koja se smanjuje prema starom zapisu, uz negativni predznak).
2. *azuriraj_kolicinu_repromaterijala* se ponovo poziva, ali s novim parametrima *new.id_repromaterijal*, *new.tip_transakcije* i *new.kolicina* (koji označavaju novu količinu repromaterijala koja treba biti ažurirana u skladištu).

Ovaj okidač omogućava točno praćenje promjena u količinama repromaterijala u skladištu nakon ažuriranja podataka, osiguravajući da se količine uvijek usklade sa stvarnim stanjem nakon izmjena. Podaci iz ovog okidača pomažu u održavanju točnosti skladišnih zaliha repromaterijala, optimizaciji skladišnih operacija i osiguravanju preciznog izvještavanja o stanju repromaterijala u skladištima.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER bi_kvartalni_pregled_prodaje
    BEFORE INSERT ON kvartalni_pregled_prodaje
    FOR EACH ROW
BEGIN
    IF DATE_FORMAT(new.pocetni_datum, '%d.%m.') NOT IN ('01.01.', '01.04.', '01.07.', '01.10.') THEN
        SIGNAL SQLSTATE '45008' SET MESSAGE_TEXT = 'Neispravan unos, početni datum mora biti početak kvartala!';
    END IF;

    IF DATE_FORMAT(new.zavrsni_datum, '%d.%m.') NOT IN ('31.03.', '30.6.', '30.09.', '31.12.') THEN
        SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'Neispravan unos, završni datum mora biti završetak kvartala!';
    END IF;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *bi_kvartalni_pregled_prodaje*, kreiran je kako bi prije unosa novog zapisa u tablicu *kvartalni_pregled_prodaje* osigurao da su unosi početnog i završnog datuma ispravni, odnosno da odgovaraju početku i završetku kvartala. Funkcionalnost ovog okidača temelji se na provjeri unesenih datuma i na sprječavanju unosa ako datumi ne zadovoljavaju uvjete koji definiraju početke i završetke kvartala.

Okidač se aktivira prije nego što novi zapis bude umetnut u tablicu *kvartalni_pregled_prodaje* i obavlja dvije provjere:

1. Ako *new.pocetni_datum* nije jedan od dana koji označavaju početak kvartala (01.01., 01.04., 01.07., ili 01.10.), okidač generira grešku s porukom 'Neispravan unos, početni datum mora biti početak kvartala!' koristeći SQLSTATE '45008'.
2. Ako *new.zavrsni_datum* nije jedan od dana koji označavaju završetak kvartala (31.03., 30.06., 30.09., ili 31.12.), okidač generira grešku s porukom 'Neispravan unos, završni datum mora biti završetak kvartala!' koristeći SQLSTATE '45009'.

Ovaj okidač pomaže osigurati točnost podataka u vezi s kvartalnim pregledima prodaje, sprječavajući unose koji ne slijede pravilne vremenske okvire kvartala. Na taj način, korisnici baze podataka su upozoreni na pogrešne unose datuma, čime se održava integritet podataka i omogućava precizno izvještavanje o prodaji na temelju kvartalnih razdoblja.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER au_zahtjev_za_narudzbu_otkazana
    AFTER UPDATE ON zahtjev_za_narudzbu
    FOR EACH ROW
BEGIN
    IF new.status_narudzbe = 'Otkazana' THEN
        DELETE FROM stavka_narudzbe
        WHERE id_zahtjev_za_narudzbu = new.id;
    END IF;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *au_zahtjev_za_narudzbu_otkazana*, kreiran je kako bi nakon ažuriranja zapisa u tablici *zahtjev_za_narudzbu* automatski obrisao sve stavke narudžbe povezane s narudžbom koja je označena kao "Otkazana". Funkcionalnost ovog okidača temelji se na provjeri statusa narudžbe i na automatskom brisanju odgovarajućih stavki narudžbe kada status narudžbe postane "Otkazana".

Okidač se aktivira nakon što se ažurira zapis u tablici *zahtjev_za_narudzbu*. Ako je novi status narudžbe (novi *new.status_narudzbe*) postavljen na "Otkazana", okidač pokreće SQL naredbu koja briše sve stavke iz tablice *stavka_narudzbe* koje su povezane s tim zahtjevom za narudžbu (identificirane preko *new.id*).

Ovaj okidač omogućava automatsko održavanje dosljednosti podataka u slučaju kada se narudžba otkaže, osiguravajući da sve stavke povezane s tom narudžbom budu uklonjene iz sustava. Na taj način, sprječava se nepotrebno zadržavanje podataka koji više nisu relevantni, čime se održava točnost i integritet podataka o narudžbama i stavkama narudžbi.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER bu_transport_datum_dolaska
    BEFORE UPDATE ON transport
    FOR EACH ROW
BEGIN
    IF NEW.datum_dolaska < NEW.datum_polaska THEN
        SIGNAL SQLSTATE '45012' SET MESSAGE_TEXT = 'Datum dolaska ne može biti prije datuma polaska!';
    END IF;
    IF NEW.datum_dolaska IS NOT NULL THEN
        SET NEW.status_transporta = 'Obavljen';
    END IF;
END //
DELIMITER ;
```

Trigger *bu_transport_datum_dolaska* je BEFORE UPDATE trigger na tablici transport koji se aktivira prije nego što se ažurira bilo koji zapis u toj tablici.

Prvo, trigger provjerava je li novi datum dolaska (NEW.datum_dolaska) manji od datuma polaska (NEW.datum_polaska). Ako je to slučaj, signalizira se greška s porukom 'Datum dolaska ne može biti prije datuma polaska!' pomoću naredbe SIGNAL SQLSTATE '45012'.

Zatim, ako je novi datum dolaska različit od NULL, status transporta (NEW.status_transporta) postavlja se na 'Obavljen'.

Ovaj trigger osigurava da se ne može postaviti datum dolaska prije datuma polaska i automatski ažurira status transporta na 'Obavljen' kada je datum dolaska postavljen.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER au_transport_datum_dolaska
    AFTER UPDATE ON transport
    FOR EACH ROW
BEGIN
    IF new.datum_dolaska IS NOT NULL THEN
        UPDATE zahtjev_za_narudzbu
            SET status_narudzbe = 'Završena'
            WHERE id_transport = new.id;
    END IF;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *au_transport_datum_dolaska*, kreiran je kako bi nakon ažuriranja zapisa u tablici *transport* automatski ažurirao status narudžbe na "Završena" kada je uneseni datum dolaska različit od NULL. Funkcionalnost ovog okidača temelji se na provjeri unosa *new.datum_dolaska* i ažuriranju statusa povezane narudžbe ako je datum dolaska prisutan.

Okidač se aktivira nakon što se ažurira zapis u tablici *transport*. Ako je novi *new.datum_dolaska* unesen (nije NULL), okidač izvršava SQL naredbu koja ažurira status narudžbe u tablici *zahtjev_za_narudzbu* na "Završena" za onu narudžbu koja je povezana s tim transportom, putem *new.id*.

Ovaj okidač omogućava automatsko praćenje statusa narudžbe na temelju unosa datuma dolaska transporta, čime se osigurava da se status narudžbe ažurira odmah nakon što transport stigne, poboljšavajući točnost podataka i učinkovitost sustava. Na taj način, svaka narudžba koja je povezana s transportom koji je dobio datum dolaska, automatski prelazi u status "Završena", što omogućava lakše praćenje i izvještavanje o završenim narudžbama.

OKIDAČ

```
DELIMITER //
CREATE TRIGGER postavi_status_na_cekanju
    AFTER INSERT ON stavka_narudzbe
    FOR EACH ROW
BEGIN
    UPDATE zahtjev_za_narudzbu
    SET status_narudzbe = 'Primljena'
    WHERE id = NEW.id_zahtjev_za_narudzbu;
END //
DELIMITER ;
```

Ovaj okidač, nazvan *postavi_status_na_cekanju*, kreiran je kako bi nakon unosa nove stavke u tablicu *stavka_narudzbe* automatski postavio status narudžbe na "Primljena". Funkcionalnost ovog okidača temelji se na automatskom ažuriranju statusa narudžbe kada se unese nova stavka, čime se označava da je narudžba započela i da čeka daljnje obrade.

Okidač se aktivira nakon što se unese nova stavka u tablicu *stavka_narudzbe*. Kada je nova stavka unesena, okidač automatski ažurira status povezane narudžbe u tablici *zahtjev_za_narudzbu* na "Primljena" koristeći *NEW.id_zahtjev_za_narudzbu*, koji predstavlja identifikator povezane narudžbe.

Ovaj okidač omogućava automatsko praćenje statusa narudžbe, čime se sprječava potreba za ručnim unosom statusa i osigurava točnost podataka o narudžbama. Na taj način, svaka narudžba koja dobije prvu stavku automatski prelazi u status "Primljena", što omogućava brže i efikasnije praćenje procesa narudžbe i omogućava korisnicima i sustavu jasniji uvid u stanje narudžbe.

6. FUNKCIJE

FUNKCIJA

```
-- Funkcije
-- funkcija za vraćanje statusa narudžbe po ID-u
DELIMITER //
CREATE FUNCTION vrati_status_narudzbe(order_id INT)
RETURNS ENUM('Primljena', 'U obradi', 'Na čekanju', 'Spremna za isporuku', 'Poslana', 'Završena', 'Otkazana')
DETERMINISTIC
BEGIN
    DECLARE status_nar ENUM('Primljena', 'U obradi', 'Na čekanju', 'Spremna za isporuku', 'Poslana', 'Završena', 'Otkazana');


    SELECT status_narudzbe INTO status_nar
    FROM zahtjev_za_narudzbu
    WHERE id = order_id;


    RETURN status_nar;
END //
DELIMITER ;
```

Ova funkcija, nazvana *vrati_status_narudzbe*, kreirana je kako bi omogućila dohvaćanje trenutnog statusa narudžbe na temelju identifikatora narudžbe (order_id). Funkcionalnost ove funkcije temelji se na upitu koji pretražuje tablicu *zahtjev_za_narudzbu* i vraća status narudžbe koji je povezan s danim identifikatorom.

Funkcija prima jedan ulazni parametar, *order_id*, koji predstavlja jedinstveni identifikator narudžbe. Korištenjem ovog identifikatora, funkcija pretražuje tablicu *zahtjev_za_narudzbu* i dohvaća vrijednost statusa narudžbe. Status narudžbe je pohranjen u varijabli *status_nar*, koja se zatim vraća kao rezultat funkcije.

Ova funkcija omogućava brzi pristup statusu narudžbe na temelju njenog identifikatora, što je korisno za izvještavanje, praćenje stanja narudžbi ili donošenje odluka unutar poslovnog sustava. Na taj način, funkcija pojednostavljuje pristup podacima o statusima narudžbi i omogućava veću automatizaciju i preciznost u poslovnim procesima.

FUNKCIJA

```
-- Funkcija koja vraća ukupnu vrijednost svih narudžba

DELIMITER //
CREATE FUNCTION ukupno_narudzbe()
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE z INT;

    SELECT SUM(ukupni_iznos) INTO z
    FROM zahtjev_za_narudzbu;

    RETURN z;
END //
DELIMITER ;
```

Ova funkcija, nazvana *ukupno_narudzbe*, kreirana je kako bi izračunala ukupni iznos svih narudžbi u sustavu. Funkcionalnost ove funkcije temelji se na upitu koji sumira vrijednosti iznosâ svih narudžbi pohranjenih u tablici *zahtjev_za_narudzbu*.

Funkcija ne prima ulazne parametre, a njezina svrha je dohvatiti ukupni iznos svih narudžbi u sustavu. Korištenjem SQL upita, funkcija zbraja sve vrijednosti u stupcu *ukupni_iznos* i pohranjuje rezultat u varijabli *z*. Taj iznos se zatim vraća kao rezultat funkcije.

Ova funkcija omogućava jednostavno izračunavanje ukupnog iznosa svih narudžbi, što je korisno za generiranje izvještaja o ukupnoj vrijednosti narudžbi, analize poslovnih rezultata ili donošenje financijskih odluka unutar organizacije. Na taj način, funkcija doprinosi boljoj analizi poslovanja i upravljanju financijama.

FUNKCIJA

```
DELIMITER //
CREATE FUNCTION broj_narudzbi_kupca(p_id_kupac INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE broj INT;
    SELECT COUNT(*)
    INTO broj
    FROM zahtjev_za_narudzbu
    WHERE id_kupac = p_id_kupac AND status_narudzbe = 'Završena';

    RETURN broj;
END//
DELIMITER ;
```

Ova funkcija, nazvana *broj_narudzbi_kupca*, kreirana je za izračunavanje broja narudžbi koje je određeni kupac završio. Funkcionalnost ove funkcije temelji se na upitu koji broji sve narudžbe povezanog kupca s statusom "Završena" u tablici *zahtjev_za_narudzbu*.

Funkcija prima jedan ulazni parametar, *p_id_kupac*, koji predstavlja identifikator kupca za kojeg želimo izračunati broj završnih narudžbi. Korištenjem SQL upita, funkcija broji sve narudžbe tog kupca koje imaju status "Završena" i pohranjuje rezultat u varijabli *broj*. Taj broj se zatim vraća kao rezultat funkcije.

Ova funkcija omogućava jednostavno praćenje broja završenih narudžbi za određenog kupca, što je korisno za analizu kupčevih navika, praćenje zadovoljstva kupaca, izvještavanje o poslovnim rezultatima ili donošenje odluka vezanih uz marketing i prodaju. Na taj način, funkcija doprinosi boljoj analizi kupaca i optimizaciji poslovanja.

FUNKCIJA

```
-- 1. Funkcija: Izračunaj ukupnu količinu robe za određenog prijevoznika
DELIMITER //
CREATE FUNCTION ukupna_kolicina_prijevoznika(id_prijevoznik INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE ukupno DECIMAL(10, 2);
    SELECT SUM(kolicina) INTO ukupno
    FROM transport
    WHERE id_prijevoznik = id_prijevoznik;
    RETURN ukupno;
END//
DELIMITER ;
```

Ova funkcija, nazvana *ukupna_kolicina_prijevoznika*, kreirana je za izračunavanje ukupne količine prevezene robe od strane određenog prijevoznika. Funkcionalnost ove funkcije temelji se na upitu koji zbraja količine prevezene u svim transportima povezanima s određenim prijevoznikom u tablici *transport*.

Funkcija prima jedan ulazni parametar, *id_prijevoznik*, koji predstavlja identifikator prijevoznika za kojeg želimo izračunati ukupnu količinu prevezene robe. Korištenjem SQL upita, funkcija sumira količine svih transporta za tog prijevoznika i pohranjuje rezultat u varijabli *ukupno*. Taj rezultat, koji predstavlja ukupnu količinu prevezene robe, zatim se vraća kao rezultat funkcije.

Ova funkcija omogućava praćenje učinkovitosti prijevoznika u pogledu količine prevezene robe, što je korisno za analize performansi, optimizaciju logističkih procesa i donošenje poslovnih odluka u vezi s odabirima prijevoznika. Na taj način, funkcija doprinosi boljoj analizi transportnih aktivnosti i optimizaciji poslovanja.

FUNKCIJA

```
-- 2. Funkcija: Dohvati broj transporta za određenog prijevoznika
DELIMITER //
CREATE FUNCTION broj_transporta_prijevoznika(id_prijevoznik INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE broj_transporta INT;
    SELECT COUNT(*) INTO broj_transporta
    FROM transport
    WHERE id_prijevoznik = id_prijevoznik;
    RETURN broj_transporta;
END//
DELIMITER ;
```

Ova funkcija, nazvana *broj_transporta_prijevoznika*, kreirana je za izračunavanje broja transportnih operacija izvršenih od strane određenog prijevoznika. Funkcionalnost ove funkcije temelji se na upitu koji broji sve zapise u tablici *transport* koji su povezani s određenim prijevoznikom.

Funkcija prima jedan ulazni parametar, *id_prijevoznik*, koji predstavlja identifikator prijevoznika za kojeg želimo izračunati broj transporta. Korištenjem SQL upita, funkcija broji sve transportne zapise koji odgovaraju tom identifikatoru i pohranjuje rezultat u varijabli *broj_transporta*. Taj broj, koji predstavlja ukupni broj transportnih operacija izvršenih od strane prijevoznika, zatim se vraća kao rezultat funkcije.

Ova funkcija omogućava praćenje aktivnosti prijevoznika u pogledu broja izvršenih transporta, što je korisno za analizu učinka prijevoznika, optimizaciju raspodjele zadataka i donošenje poslovnih odluka temeljenih na brojnosti transportnih operacija. Na taj način, funkcija doprinosi boljoj analizi i optimizaciji transportnih procesa i odnosa s prijevoznicima.

FUNKCIJA

```
DELIMITER //
CREATE FUNCTION zaposleni_odjel(p_id_odjel INTEGER) RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE var_broj_zaposlenih INTEGER;

    SELECT COUNT(*) INTO var_broj_zaposlenih
        FROM zaposlenik
        WHERE id_odjel = p_id_odjel
            AND datum_zaposlenja > CURDATE() - INTERVAL 1 YEAR;

    RETURN var_broj_zaposlenih;
END //
DELIMITER ;
```

Funkcija broj_mjesta u MySQL-u vraća ukupan broj kupaca i zaposlenika koji žive u određenom mjestu. Pri tome, koristi se operator LIKE za pretragu adresa koje sadrže naziv mjesta, što omogućuje pronalaženje svih adresa koje uključuju navedeni naziv mjesta bilo gdje unutar adrese.

Funkcija zaposleni_odjel vraća broj zaposlenika koji rade u određenom odjelu i koji su zaposleni unutar posljednje godine. Koristi se funkcija COUNT za brojanje zaposlenika koji zadovoljavaju uvjete, a CURDATE() za dobivanje trenutnog datuma.

FUNKCIJA

```
DELIMITER //
CREATE FUNCTION broj_mjesta(p_mjesto VARCHAR(20)) RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE var_broj INTEGER;

    SELECT COUNT(*) INTO var_broj
        FROM (
            SELECT adresa FROM zaposlenik
            UNION ALL
            SELECT adresa FROM kupac
        ) AS zk
        WHERE zk.adresa LIKE CONCAT('%', p_mjesto);

    RETURN var_broj;
END //
DELIMITER ;
```

Funkcija broj_mjesta u SQL-u vraća ukupan broj kupaca i zaposlenika koji žive u određenom mjestu. Ona spaja adrese iz tablica zaposlenik i kupac pomoću operatora UNION ALL, a zatim broji koliko tih adresa sadrži naziv mjesta koji je predan kao argument funkciji. Rezultat je broj osoba čije adrese sadrže navedeno mjesto.

FUNKCIJA

```
DELIMITER //
CREATE FUNCTION provjera_prava_korisnika(id_zaposlenik INT)
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN
    DECLARE pravo VARCHAR(50);

    SELECT uloge.naziv
    INTO pravo
    FROM zaposlenik
    JOIN uloge ON zaposlenik.uloga_id = uloge.uloga_id
    WHERE zaposlenik.id = id_zaposlenik
    LIMIT 1;

    -- Ako zaposlenik ne postoji, vraća 'korisnik_ne_postoji'
    RETURN IFNULL(pravo, 'korisnik_ne_postoji');
END;
//
DELIMITER ;
```

Ova funkcija, nazvana *provjera_prava_korisnika*, kreirana je za provjeru prava korisnika na temelju njegovih korisničkih uloga u sustavu. Funkcionalnost ove funkcije temelji se na povezivanju tablica *zaposlenik* i *uloge* kako bi se dohvatila uloga koju ima korisnik, te vraćanju te informacije kao naziv uloge.

Funkcija prima jedan ulazni parametar, *id_zaposlenik*, koji predstavlja identifikator zaposlenika za kojeg želimo provjeriti prava. Funkcija koristi SQL upit za povezivanje tablica *zaposlenik* i *uloge*, te dohvaća naziv uloge koja je dodijeljena tom zaposleniku. Ako zaposlenik postoji u sustavu i ima dodijeljenu ulogu, funkcija vraća naziv te uloge.

U slučaju da zaposlenik ne postoji ili nema dodijeljenu ulogu, funkcija će vratiti vrijednost 'korisnik_ne_postoji'. Ova funkcionalnost omogućava provjeru prava korisnika prema njihovoj ulozi u sustavu, čime se olakšava upravljanje pristupom i administrativnim zadacima temeljenim na korisničkim pravima.

Ova funkcija doprinosi sigurnosti sustava jer omogućava validaciju prava pristupa na temelju uloga, te osigurava da se korisnicima dodjeljuju odgovarajuće privilegije za obavljanje određenih zadataka.

FUNKCIJA

```
-- -funkcija s kojom dohvatimo koliko ima admina, koji imaju pristup svim podacima
DELIMITER //
CREATE FUNCTION broj_admin_korisnika()
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE broj_admina INT;

    SELECT COUNT(*) INTO broj_admina
    FROM zaposlenik
    WHERE zaposlenik.uloga_id = 1; -- id 1 je id za admina

    RETURN broj_admina;
END;
//
DELIMITER ;
```

Ova funkcija, nazvana *broj_admin_korisnika*, kreirana je za brojanje broja zaposlenika koji imaju administrativnu ulogu u sustavu. Funkcionalnost ove funkcije temelji se na upitu koji broji zaposlenike čija je uloga postavljena na administrativnu, identificiranu s ID-om 1.

Funkcija ne prima nikakve ulazne parametre. Umjesto toga, koristi hardkodirani ID uloge (1), koji predstavlja administrativnu ulogu u sustavu, te broji sve zaposlenike koji imaju ovu ulogu u tablici *zaposlenik*.

Rezultat ove funkcije je broj zaposlenika s administrativnim pravima, koji se vraća kao cijeli broj. Ova funkcija omogućava praćenje broja administrativnih korisnika u sustavu, što može biti korisno za administraciju korisničkih prava, planiranje resursa i praćenje pristupa sustavu.

Funkcija je korisna za izvještavanje i administrativne svrhe, jer omogućava praćenje broja korisnika s posebnim privilegijama i može se koristiti za analize i optimizaciju sigurnosti sustava.

FUNKCIJA

```
-- 2. Funkcija za broj admina
DELIMITER //
CREATE FUNCTION broj_admin_korisnika()
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE broj_admina INT;

    -- Dohvat broja zaposlenika koji imaju ulogu 'admin'
    SELECT COUNT(*) INTO broj_admina
    FROM zaposlenik
    WHERE zaposlenik.uloga_id = 1;  -- 1 je ID za admina

    RETURN broj_admina;
END;
//
DELIMITER ;
```

Ova funkcija, nazvana *broj_admin_korisnika*, kreirana je za brojanje broja zaposlenika koji imaju administrativnu ulogu u sustavu. Funkcionalnost ove funkcije temelji se na upitu koji broji zaposlenike čija je uloga postavljena na administrativnu, identificiranu s ID-om 1.

Funkcija ne zahtijeva ulazne parametre, a umjesto toga koristi hardkodirani ID uloge (1), koji označava administrativnu ulogu u sustavu. Zatim, funkcija broji sve zaposlenike u tablici *zaposlenik* koji imaju ovu ulogu.

Rezultat funkcije je broj zaposlenika s administrativnim pravima, koji se vraća kao cijeli broj. Ova funkcija je korisna za praćenje broja administrativnih korisnika u sustavu, što je važno za upravljanje korisničkim pravima, resursima i pristupom sustavu.

Funkcija može biti korisna za izvještavanje, administrativne svrhe i optimizaciju sigurnosti sustava, jer omogućava jasnu evidenciju o broju korisnika s posebnim privilegijama.

FUNKCIJA

```
DELIMITER //
CREATE FUNCTION ukupan_iznos_zaposlenik(p_id_zaposlenik INTEGER) RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE var_iznos DECIMAL(10, 2);

    SELECT SUM(ukupni_iznos) INTO var_iznos
        FROM zahtjev_za_narudzbu
        WHERE status_narudzbe = 'Završena'
            AND id_zaposlenik = p_id_zaposlenik;

    RETURN var_iznos;
END //
DELIMITER ;
```

Funkcija **ukupan_iznos_zaposlenik** koristi se za izračunavanje ukupnog iznosa svih narudžbi koje je obradio određeni zaposlenik, gdje su narudžbe statusa 'Završena'. Funkcija prima jedan ulazni parametar, `p_id_zaposlenik`, koji označava identifikator zaposlenika za kojeg se želi izračunati ukupan iznos. Unutar funkcije, koristi se SQL upit koji zbraja `ukupni_iznos` iz tablice `zahtjev_za_narudzbu` za sve narudžbe koje su u statusu `'Završena'` i povezane s navedenim zaposlenikom (`id_zaposlenik`). Rezultat se pohranjuje u varijablu `var_iznos`, koja se zatim vraća kao rezultat funkcije u obliku decimalnog broja s dvije decimale. Ova funkcija omogućava praćenje učinkovitosti zaposlenika u smislu ostvarenih prihoda, što je korisno za izvještaje o prodaji ili uspjehu zaposlenika u radu s narudžbama.

FUNKCIJA

```
DELIMITER //
CREATE FUNCTION kolicina_vina_godina(p_godina INTEGER) RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE var_kolicina_vina VARCHAR(20);

    SELECT CONCAT(COALESCE(SUM(sv.kolicina),0), ' L') INTO var_kolicina_vina
        FROM skladiste_vino sv
        JOIN berba b ON b.id = sv.id_berba
        WHERE tip_transakcije = 'ulaz'
            AND b.godina_berbe = p_godina;

    RETURN var_kolicina_vina;
END //
DELIMITER ;
```

Funkcija **kolicina_vina_godina** koristi se za izračunavanje ukupne količine vina koje je pristiglo u skladište za određenu godinu berbe. Funkcija prima jedan ulazni parametar, `p_godina`, koji označava godinu berbe za koju želimo saznati količinu vina. Unutar funkcije, SQL upit zbraja količine vina (`kolicina`) iz tablice `skladiste_vino` za sve transakcije tipa 'ulaz', povezane s godinom berbe koja odgovara `p_godina`. Ukupna količina vina se zatim formatira u obliku stringa, koji uključuje brojčanu vrijednost i oznaku 'L' za litre. Ako nema podataka za traženu godinu, koristi se funkcija `COALESCE` koja osigurava da se u rezultatu prikaže '0 L' umjesto NULL vrijednosti. Ova funkcija omogućava praćenje količine vina u skladištu prema godinama berbe, što je korisno za upravljanje zalihama i analize proizvodnje.

FUNKCIJA

```
-- funkcija koja vraća broj zaposlenika u određenom odjelu

DELIMITER //
CREATE FUNCTION broj_zaposlenika_u_odjelu(p_id_odjel INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE broj INT;

    SELECT COUNT(*)
    INTO broj
    FROM zaposlenik
    WHERE id_odjel = p_id_odjel AND status_zaposlenika = 'aktivan';

    RETURN broj;
END//
DELIMITER ;
```

Ova funkcija, nazvana *broj_zaposlenika_u_odjelu*, kreirana je za brojanje broja aktivnih zaposlenika unutar određenog odjela. Funkcionalnost ove funkcije temelji se na upitu koji broji zaposlenike u specifičnom odjelu čiji je status postavljen na "aktivan".

Funkcija prima jedan ulazni parametar, *p_id_odjel*, koji predstavlja ID odjela za koji se želi izračunati broj aktivnih zaposlenika. Na temelju tog ID-a, funkcija broji zaposlenike u tablici *zaposlenik* koji pripadaju tom odjelu i čiji je status zaposlenika "aktivan".

Rezultat funkcije je broj aktivnih zaposlenika u navedenom odjelu, koji se vraća kao cijeli broj. Ova funkcija je korisna za praćenje broja zaposlenika koji trenutno rade u određenom odjelu, što može biti važno za administraciju resursa, planiranje, izvještavanje i analizu performansi.

Funkcija omogućava učinkovitije praćenje ljudskih resursa unutar različitih odjela, olakšavajući donošenje odluka i optimizaciju poslovnih procesa na temelju broja aktivnih zaposlenika u svakom odjelu.

7. TRANSAKCIJE

TRANSAKCIJA

```
START TRANSACTION;

INSERT INTO transport (id_prijevoznik, registracija, ime_vozaca, datum_polaska, status_transporta)
VALUES (2, 'ZG4748GG', 'Juraj Jurić', CURDATE(), 'U tijeku');

SET @novi_transport = LAST_INSERT_ID();

UPDATE zahtjev_za_narudzbu
    SET id_transport = @novi_transport, status_narudzbe = 'Poslana'
    WHERE id IN (28, 33);

CALL izracunaj_kolicinu_transporta(@novi_transport);

COMMIT;
```

Ova transakcija, koja se sastoji od više SQL operacija, koristi se za unos novih podataka o transportu, ažuriranje statusa narudžbi te izračunavanje količine transporta. Svi koraci transakcije su obuhvaćeni unutar transakcijskog bloka kako bi se osigurala atomskost operacija, odnosno da će sve promjene biti primijenjene zajedno ili nijedna u slučaju greške.

Transakcija započinje s unosom novog transporta u tablicu *transport*, gdje se dodaju podaci o prijevozniku, registraciji vozila, imenu vozača, datumu polaska i statusu transporta. Ovaj unos je označen kao početak novog transporta, pri čemu se koristi funkcija *CURDATE()* za dohvat trenutnog datuma kao datuma polaska.

Nakon unosa transporta, pomoću funkcije *LAST_INSERT_ID()*, dohvaća se ID nedavno unesenog transporta, koji se pohranjuje u varijablu *@novi_transport*. Ovaj ID se kasnije koristi za povezivanje transporta s određenim narudžbama.

Zatim slijedi ažuriranje statusa narudžbi u tablici *zahtjev_za_narudzbu*. Za narudžbe s ID-ovima 28 i 33, povezani su novi transport i ažuriran je status narudžbi na "Poslana", što označava da je roba poslana na transport.

Nakon ažuriranja statusa narudžbi, poziva se procedura *izracunaj_kolicinu_transporta*, koja izračunava količinu prevoženih dobara za povezani transport.

Na kraju, transakcija se zaključuje s *COMMIT* naredbom, čime se trajno pohranjuju sve izvršene promjene u bazu podataka. Ovaj proces osigurava da su svi podaci dosljedni i da je transport pravilno povezan s odgovarajućim narudžbama te da je količina transporta izračunata.

TRANSAKCIJA

```sql
START TRANSACTION;

UPDATE zahtjev_za_narudzbu
SET status_narudzbe = 'Otkazana'
WHERE id = 36;

DELETE FROM racun
WHERE id_zahtjev_za_narudzbu = 33;

COMMIT;


INSERT INTO zahtjev_za_narudzbu (id, id_kupac, id_zaposlenik, datum_zahtjeva, ukupni_iznos, status_narudzbe)
VALUES (36, 1, 1, '2025-01-01', 100.00, 'Primljena');

INSERT INTO racun (id, id_zaposlenik, id_zahtjev_za_narudzbu, datum_racuna)
VALUES (33, 1, 31, '2025-01-02');

SELECT * FROM zahtjev_za_narudzbu WHERE id = 36;
SELECT * FROM racun WHERE id_zahtjev_za_narudzbu = 1;
```

Ova transakcija se koristi za otkazivanje narudžbe, ažuriranje statusa narudžbe na "Otkazana" i brisanje povezanog računa iz tablice *racun*. Svi ovi koraci su unutar transakcijskog bloka kako bi se osigurala dosljednost podataka — odnosno da će se svi podaci ažurirati ili ukloniti zajedno, ili nijedno od njih u slučaju greške.

Transakcija započinje ažuriranjem statusa narudžbe u tablici *zahtjev_za_narudzbu*. Za narudžbu s ID-om 36, status se mijenja na "Otkazana", što označava da je narudžba prekinuta ili povučena. Ovaj korak signalizira da narudžba više neće biti obrađena.

Nakon toga, u drugom koraku transakcije, briše se račun povezan s narudžbom. Račun s ID-om 33, koji je povezan s narudžbom, uklanja se iz tablice *racun*, čime se poništava bilo koji račun koji je možda bio izdan za otkazanu narudžbu.

Na kraju, transakcija se zaključuje s *COMMIT* naredbom, koja trajno pohranjuje sve promjene u bazu podataka, čime se osigurava da su svi podaci dosljedni i da su obrisani svi podaci povezani s otkazivanjem narudžbe, uključujući račun.

Ova transakcija omogućava efikasno i sigurno otkazivanje narudžbe i uklanjanje povezanih podataka, uz osiguranje da su svi koraci pravilno izvršeni ili nijedan, čime se održava integritet podataka u bazi.

TRANSAKCIJA

```
START TRANSACTION;

DELETE FROM zaposlenik WHERE id = 8;

UPDATE odjel SET broj_zaposlenika = broj_zaposlenika - 1 WHERE id = 2;

COMMIT;
```

Ova transakcija se koristi za uklanjanje zaposlenika iz sustava i ažuriranje broja zaposlenika u odjelu nakon toga. Svi koraci su uključeni unutar transakcijskog bloka kako bi se osigurala dosljednost podataka — zaposlenik je uklonjen, a broj zaposlenika u odjelu ažuriran, ili nijedno od njih u slučaju greške.

Transakcija započinje s *DELETE* naredbom koja uklanja zaposlenika s ID-om 8 iz tablice *zaposlenik*. Ovaj korak uklanja sve podatke povezane s tim zaposlenikom iz baze podataka.

Nakon toga, u drugom koraku transakcije, ažurira se broj zaposlenika u odjelu. Za odjel s ID-om 2, broj zaposlenika se smanjuje za 1 kako bi se odražavala promjena koja se dogodila zbog uklanjanja zaposlenika. Ovaj korak osigurava da broj zaposlenika u odjelu bude točan i u skladu s novim stanjem.

Na kraju, transakcija se završava *COMMIT* naredbom, koja trajno pohranjuje sve promjene u bazu podataka, čime se osigurava da su podaci dosljedni. Time se potvrđuju obrisani podaci zaposlenika i ažurirani broj zaposlenika u odjelu.

Ova transakcija omogućava sigurno uklanjanje zaposlenika i ažuriranje povezanih podataka u odjelu, čime se održava integritet baze podataka, te je korisna za učinkovito upravljanje zaposlenicima i odjelima.

TRANSAKCIJA

```
-- Ažuriranje podataka o kupcu i zaposleniku

START TRANSACTION;

UPDATE kupac SET telefon = '+385919876543' WHERE oib = '12345678912';

UPDATE zaposlenik SET telefon = '+385919876543' WHERE id = 15;

COMMIT;
```

Ova transakcija je dizajnirana za ažuriranje broja telefona kako u tablici *kupac*, tako i u tablici *zaposlenik* za specifične entitete, kako bi se održali ažurirani podaci u sustavu. Svi koraci su unutar transakcijskog bloka, što osigurava da se oba ažuriranja izvrše zajedno ili nijedno ako dođe do greške.

Transakcija počinje s *UPDATE* naredbom koja ažurira broj telefona kupca s OIB-om '12345678912'. Novi broj telefona koji se postavlja je '+385919876543'. Ovaj korak osigurava da podaci o kupcu budu ažurirani u skladu s novim brojem telefona.

Nakon toga, u drugom koraku transakcije, ažurira se broj telefona zaposlenika s ID-om 15, također postavljanjem broja telefona na '+385919876543'. Time se osigurava da podaci o zaposleniku budu točni i ažurirani.

Na kraju, transakcija završava s *COMMIT* naredbom, koja trajno pohranjuje obje promjene u bazu podataka, osiguravajući dosljednost podataka. Time se potvrđuju promjene u oba zapisa — kupca i zaposlenika.

Ova transakcija omogućava sigurno ažuriranje podataka o telefonu kako za kupca, tako i za zaposlenika u jednom atomskom procesu, čime se osigurava da podaci u bazi budu uvijek usklađeni i ažurirani.

TRANSAKCIJA

```
-- 1. Transakcija: Dodavanje novog transporta
START TRANSACTION;
INSERT INTO transport (id_prijevoznik, registracija, ime_vozaca, datum_polaska, datum_dolaska, kolicina, status_transporta)
VALUES (1, 'ZG6423JK', 'Goran Lukić', '2025-01-12', '2025-01-15', 500, 'u tijeku');
COMMIT;
```

Ova transakcija unosi novi zapis u tablicu *transport* za specifični transportni događaj. Cilj ove transakcije je zabilježiti informacije o novom transportu u sustavu, uključujući podatke o vozilu, vozaču, datumima polaska i dolaska, količini tereta i trenutnom statusu transporta.

Transakcija počinje s *INSERT INTO* naredbom koja dodaje novi zapis u tablicu *transport*. Sljedeći podaci se unose:

- id_prijevoznik: 1 (identifikator prijevoznika),
- registracija: 'ZG6423JK' (registracija vozila),
- ime_vozaca: 'Goran Lukić' (ime vozača),
- datum_polaska: '2025-01-12' (datum kada transport počinje),
- datum_dolaska: '2025-01-15' (datum kada se transport završava),
- kolicina: 500 (količina tereta koja se prevozi),
- status_transporta: 'u tijeku' (trenutni status transporta).

Nakon što je novi transportni zapis unesen, transakcija se završava s *COMMIT* naredbom, čime se trajno pohranjuje novi zapis u bazu podataka.

Ova transakcija omogućava unos svih potrebnih podataka o novom transportu i osigurava da su ti podaci odmah pohranjeni, čime se olakšava praćenje transporta i osigurava točnost podataka u sustavu.

TRANSAKCIJA

```
-- 2. Transakcija: Brisanje transporta
START TRANSACTION;
DELETE FROM transport WHERE id = 1;
COMMIT;
```

Ova transakcija briše zapis o transportu iz tablice *transport* na temelju specifičnog identifikatora (ID-a) transporta.

Transakcija započinje naredbom *DELETE FROM*, koja briše zapis iz tablice *transport* gdje je id jednak 1. Ovaj zapis predstavlja određeni transport koji se želi ukloniti iz baze podataka. Brisanje se temelji na jedinstvenom identifikatoru, što znači da će biti uklonjen samo onaj transport s ID-jem 1.

Nakon što je zapis o transportu uspješno uklonjen, transakcija se završava naredbom *COMMIT*. Ovo osigurava da je promjena (brisanje zapisa) trajno pohranjena u bazi podataka.

Ova transakcija omogućava uklanjanje nepotrebnih ili neispravnih podataka o transportu, čime se održava točnost i ažurnost podataka u sustavu.

8. PROCEDURE

PROCEDURA

```
-- 1. Procedura: Dodavanje novog transporta
DELIMITER //
CREATE PROCEDURE dodaj_transport (
    IN p_id_prijevoznik INT,
    IN p_tip_robe VARCHAR(50),
    IN p_datum DATE,
    IN p_kolicina DECIMAL(10, 2),
    IN p_status ENUM('u tijeku', 'završeno', 'otkazano')
)
BEGIN
    INSERT INTO transport (id_prijevoznik, tip_robe, datum_transporta, kolicina, status)
    VALUES (p_id_prijevoznik, p_tip_robe, p_datum, p_kolicina, p_status);
END//
DELIMITER ;
```

Ova procedura, nazvana *dodaj_transport*, omogućava unos novih podataka o transportu u tablicu *transport*. Parametri ove procedure omogućuju unos specifičnih podataka o transportu kao što su: identifikator prijevoznika, tip robe, datum transporta, količina i status transporta.

Kada se procedura pozove, ona izvršava *INSERT* operaciju u tablicu *transport*, koristeći vrijednosti koje su proslijeđene kao parametri:

- p_id_prijevoznik: identifikator prijevoznika koji će biti povezan s novim transportom.

- p_tip_robe: opis tipa robe koja se transportira.
- p_datum: datum kada je transport zakazan.
- p_kolicina: količina robe koja se transportira, izražena u decimalnom formatu.
- p_status: status transporta, koji može biti jedan od tri moguća: 'u tijeku', 'završeno' ili 'otkazano'.

Procedura omogućava jednostavan način za dodavanje novih transporta u sustav, što poboljšava učinkovitost upravljanja podacima. Sve informacije o transportu, uključujući status i količinu robe, automatski se pohranjuju u tablicu *transport*, bez potrebe za ručnim unosom podataka.

Nakon što je unos izvršen, novi transport je odmah pohranjen u bazu podataka, a svi povezani podaci o transportu postaju dostupni za daljnje praćenje i analize.

PROCEDURA

```
-- 2. Procedura: Dodavanje novog prijevoznika
DELIMITER //
CREATE PROCEDURE dodaj_prijevoznika (
    IN p_naziv VARCHAR(100),
    IN p_oib CHAR(11),
    IN p_kontakt VARCHAR(100)
)
BEGIN
    INSERT INTO prijevoznik (naziv, oib, kontakt)
    VALUES (p_naziv, p_oib, p_kontakt);
END//
DELIMITER ;
```

Ova procedura, nazvana *dodaj_prijevoznika*, omogućava unos novih podataka o prijevozniku u tablicu *prijevoznik*. Parametri ove procedure omogućuju unos ključnih informacija o prijevozniku, uključujući naziv, OIB (osobni identifikacijski broj) i kontakt podatke.

Kada se procedura pozove, ona izvršava *INSERT* operaciju u tablicu *prijevoznik*, koristeći vrijednosti koje su proslijeđene kao parametri:

- p_naziv: naziv prijevoznika koji se dodaje u sustav.
- p_oib: OIB prijevoznika koji je jedinstveni identifikator za pravnu osobu ili fizičku osobu.
- p_kontakt: kontakt podaci prijevoznika, poput broja telefona ili email adrese.

Procedura omogućava jednostavan način za dodavanje novih prijevoznika u sustav, što olakšava upravljanje podacima o prijevoznicima i njihovim informacijama. Nakon što je unos izvršen, svi podaci o novom prijevozniku automatski se pohranjuju u bazu podataka, čime postaju dostupni za daljnje operacije i analize.

Ovaj proces automatizira unos podataka, čime se smanjuje mogućnost grešaka i povećava brzina unosa novih prijevoznika u sustav.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE analiziraj_narudzbe_po_mjesecu(IN p_mjesec INTEGER, IN p_godina INTEGER, OUT p_info_mjesec VARCHAR(200))
BEGIN
    DECLARE var_broj_narudzbi INTEGER;
    DECLARE var_prosjecni_iznos, var_najmanji_iznos, var_najveci_iznos DECIMAL(10,2);
    SELECT COUNT(id), ROUND(AVG(ukupni_iznos), 2), MIN(ukupni_iznos), MAX(ukupni_iznos) INTO var_broj_narudzbi, var_prosjecni_iznos, var_najmanji_iznos, var_najveci_iznos
        FROM zahtjev_za_narudzbu
        WHERE MONTH(datum_zahtjeva) = p_mjesec
            AND YEAR(datum_zahtjeva) = p_godina
        GROUP BY YEAR(datum_zahtjeva), MONTH(datum_zahtjeva);

    SET p_info_mjesec = CONCAT('U ', p_mjesec, '. mjesecu ', p_godina, '. godine je bilo ', var_broj_narudzbi, ' narudžbi. Najmanji iznos narudžbe je bio €', var_najmanji_iznos,
    ' a najveći €', var_najveci_iznos, '. Prosječan iznos narudžbe je iznosio €', var_prosjecni_iznos, '.');
END //
DELIMITER ;
```

Procedura analiziraj_narudzbe_po_mjesecu analizira narudžbe u određenom mjesecu i godini, pružajući informacije o broju narudžbi, najmanjem, najvećem i prosječnom iznosu narudžbi.

Parametri:

- p_mjesec (INTEGER): Mjesec za analizu (1-12).
- p_godina (INTEGER): Godina za analizu.
- p_info_mjesec (VARCHAR(200)): Izlazni parametar koji sadrži sažetak analize.

Funkcionalnost:

1. **Deklaracija varijabli:**
    - var_broj_narudzbi: Broj narudžbi u zadanom mjesecu i godini.
    - var_prosjecni_iznos: Prosječan iznos narudžbi.
    - var_najmanji_iznos: Najmanji iznos narudžbe.
    - var_najveci_iznos: Najveći iznos narudžbe.
2. **Izbor podataka:**
    - Iz tablice zahtjev_za_narudzbu odabiru se broj narudžbi (COUNT(id)), prosječan iznos (AVG(ukupni_iznos)), najmanji iznos (MIN(ukupni_iznos)) i najveći iznos (MAX(ukupni_iznos)) za zadani mjesec i godinu.
    - Rezultati se pohranjuju u prethodno deklarirane varijable.
3. **Kreiranje izlazne poruke:**
    - Kombiniranjem varijabli u tekstualnu poruku koja sadrži broj narudžbi, najmanji, najveći i prosječni iznos narudžbi.
    - Poruka se pohranjuje u izlazni parametar p_info_mjesec.

Ova procedura omogućuje brzo generiranje sažetka o narudžbama za određeni mjesec i godinu, što je korisno za analizu poslovanja.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE prikazi_narudzbe_za_proizvod(IN p_id_proizvod INTEGER)
BEGIN
    DECLARE var_id, var_kolicina INTEGER;
    DECLARE var_datum DATE;
    DECLARE handler_broj INTEGER DEFAULT 0;

    DECLARE cur CURSOR FOR
        SELECT zzn.id, zzn.datum_zahtjeva, sn.kolicina
            FROM zahtjev_za_narudzbu zzn
            JOIN stavka_narudzbe sn ON zzn.id = sn.id_zahtjev_za_narudzbu
            WHERE sn.id_proizvod = p_id_proizvod;

    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET handler_broj = 1;

    CREATE TEMPORARY TABLE narudzbe_za_proizvod (
        id_narudzba INTEGER,
        datum_narudzbe DATE,
        kolicina_proizvoda INTEGER
    );

    OPEN cur;

    petlja: LOOP
        FETCH cur INTO var_id, var_datum, var_kolicina;

        IF handler_broj = 1 THEN
            LEAVE petlja;
        END IF;

        INSERT INTO narudzbe_za_proizvod VALUES (var_id, var_datum, var_kolicina);

    END LOOP petlja;

    CLOSE cur;

    SELECT * FROM narudzbe_za_proizvod;
    DROP TEMPORARY TABLE narudzbe_za_proizvod;
END //
DELIMITER ;
```

Procedura prikazi_narudzbe_za_proizvod koristi kursor za dohvat informacija o narudžbama za specifični proizvod. Za zadani id_proizvod, procedura:

1. **Kreira privremenu tablicu** narudzbe_za_proizvod koja pohranjuje podatke o narudžbama (ID narudžbe, datum narudžbe i količinu proizvoda).

2. **Kroz kursor** iterira kroz narudžbe povezane s dotičnim proizvodom, dohvaća ID narudžbe, datum i količinu te ih pohranjuje u privremenu tablicu.
3. **Ispisuje** sve narudžbe za proizvod iz privremene tablice.
4. Na kraju **briše** privremenu tablicu.

Ova procedura omogućuje pregled svih narudžbi za određeni proizvod u bazi podataka.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE dodaj_novu_nabavu(IN p_id_repromaterijal INTEGER, p_kolicina INTEGER, IN p_id_zaposlenik INTEGER)
BEGIN
    DECLARE var_trenutna_kolicina INTEGER;

    SELECT kolicina INTO var_trenutna_kolicina
        FROM stanje_skladista_repromaterijala
        WHERE id_repromaterijal = p_id_repromaterijal;

    IF var_trenutna_kolicina IS NULL THEN
        SIGNAL SQLSTATE '45014' SET MESSAGE_TEXT = 'Repromaterijal ne postoji u skladištu!';
    ELSEIF var_trenutna_kolicina >= 300 THEN
        INSERT INTO zahtjev_za_nabavu (id_repromaterijal, kolicina, datum_zahtjeva, status_nabave, id_zaposlenik) VALUES (p_id_repromaterijal, p_kolicina, CURDATE(), 'odbijeno', p_id_zaposlenik);
        SIGNAL SQLSTATE '45015' SET MESSAGE_TEXT = 'Stanje repromaterijala na skladištu je dostatno, zahtjev za nabavu je odbijen!';
    ELSE
        INSERT INTO zahtjev_za_nabavu (id_repromaterijal, kolicina, datum_zahtjeva, status_nabave, id_zaposlenik) VALUES (p_id_repromaterijal, p_kolicina, CURDATE(), 'odobreno', p_id_zaposlenik);
    END IF;
END //
DELIMITER ;
```

Procedura dodaj_novu_nabavu provodi sljedeće:

1. Provjerava trenutnu količinu repromaterijala u skladištu za zadani p_id_repromaterijal.
2. Ako repromaterijal ne postoji (količina je NULL), generira se greška.
3. Ako je količina u skladištu veća ili jednaka 300, zahtjev za nabavu se označava kao odbijen.
4. Ako količina nije dostatna, zahtjev za nabavu se označava kao odobren.

Ova procedura omogućuje upravljanje zahtjevima za nabavu.

PROCEDURA

```
--  Ažuriranje uloge zaposlenika
DELIMITER //
CREATE PROCEDURE azuriraj_ulogu_zaposlenika (
    IN p_id INT,
    IN p_uloga_id INT
)
BEGIN
    IF EXISTS (SELECT 1 FROM zaposlenik WHERE id = p_id)
    THEN
        UPDATE zaposlenik
        SET uloga_id = p_uloga_id
        WHERE id = p_id;
    ELSE
        SELECT 'Zaposlenik s ovim ID-om ne postoji!' AS poruka;
    END IF;
END;
//
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_ulogu_zaposlenika*, omogućava ažuriranje uloge zaposlenika u sustavu na temelju njegovog jedinstvenog identifikatora (p_id). Procedura prima dva parametra:

- p_id: ID zaposlenika čija se uloga treba ažurirati.
- p_uloga_id: Novi ID uloge koji će biti postavljen zaposleniku.

Prvo se provodi provjera postojanja zaposlenika s danim ID-em (p_id). Ako zaposlenik postoji u tablici *zaposlenik*, tada se izvršava SQL upit koji ažurira njegovu ulogu u tablici, postavljajući uloga_id na vrijednost koja je proslijeđena kao p_uloga_id.

Ako zaposlenik s danim ID-em ne postoji, procedura vraća poruku 'Zaposlenik s ovim ID-om ne postoji!' kako bi obavijestila korisnika o nepostojećem zapisu.

Ova procedura omogućava administrativno ažuriranje uloga zaposlenika unutar sustava, s provjerom valjanosti unosa, čime se smanjuje mogućnost pogrešnog ažuriranja ili dodavanja nepostojećih zaposlenika.

PROCEDURA

```
-- azuriranje statusa zaposlenika aktivan i neaktivan
DELIMITER //
CREATE PROCEDURE azuriraj_status_zaposlenika (
    IN p_id INT,
    IN p_status ENUM('aktivan', 'neaktivan')
)
BEGIN
    IF EXISTS (SELECT 1 FROM zaposlenik WHERE id = p_id) THEN
        UPDATE zaposlenik
        SET status_zaposlenika = p_status
        WHERE id = p_id;
    ELSE
        SELECT 'Zaposlenik s ovim ID-om ne postoji!' AS poruka;
    END IF;
END;
//
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_status_zaposlenika*, omogućava ažuriranje statusa zaposlenika u sustavu na temelju njegovog jedinstvenog identifikatora (p_id). Procedura prima dva parametra:

- p_id: ID zaposlenika čiji status treba biti ažuriran.
- p_status: Novi status zaposlenika koji se postavlja. Status može biti jedan od dva moguća: 'aktivan' ili 'neaktivan'.

Prvo se provodi provjera postoji li zaposlenik s danim ID-em (p_id) u tablici *zaposlenik*. Ako zaposlenik postoji, izvršava se SQL upit koji ažurira njegov status u tablici, postavljajući status_zaposlenika na vrijednost proslijeđenu kao p_status.

Ako zaposlenik s danim ID-em ne postoji, procedura vraća poruku 'Zaposlenik s ovim ID-om ne postoji!' kako bi obavijestila korisnika o nepostojećem zapisu.

Ova procedura omogućava administrativno ažuriranje statusa zaposlenika, što omogućava dinamičko praćenje njihovog trenutnog statusa (aktivnosti) u sustavu, dok se osigurava integritet podataka provjerom postojanja zaposlenika prije nego što se izvrši ažuriranje.

PROCEDURA

```
-- 3. Ažuriranje uloge zaposlenika
DELIMITER //
CREATE PROCEDURE azuriraj_ulogu_zaposlenika (
    IN p_id INT,
    IN p_uloga_id INT  -- Uloga se sada ažurira putem uloga_id (ID uloge)
)
BEGIN
    IF EXISTS (SELECT 1 FROM zaposlenik WHERE id = p_id)
    THEN
        UPDATE zaposlenik
        SET uloga_id = p_uloga_id
        WHERE id = p_id;
    ELSE
        SELECT 'Zaposlenik s ovim ID-om ne postoji!' AS poruka;
    END IF;
END;
//
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_ulogu_zaposlenika*, omogućava ažuriranje uloge zaposlenika u sustavu na temelju njegovog jedinstvenog identifikatora (p_id). Procedura prima dva parametra:

- p_id: ID zaposlenika čija uloga treba biti ažurirana.
- p_uloga_id: ID nove uloge koja će biti postavljena zaposleniku. Ovaj parametar predstavlja jedinstveni identifikator uloge iz tablice *uloge*.

Prvo se provodi provjera postoji li zaposlenik s danim ID-em (p_id) u tablici *zaposlenik*. Ako zaposlenik postoji, izvršava se SQL upit koji ažurira njegovu ulogu u tablici, postavljajući uloga_id na vrijednost proslijeđenu kao p_uloga_id.

Ako zaposlenik s danim ID-em ne postoji, procedura vraća poruku 'Zaposlenik s ovim ID-om ne postoji!' kako bi obavijestila korisnika o nepostojećem zapisu.

Ova procedura omogućava administrativno ažuriranje uloge zaposlenika u sustavu, što omogućava dinamičko praćenje i upravljanje ulogama zaposlenika, osiguravajući ispravnost podataka kroz provjeru postojanja zaposlenika prije nego što se izvrši ažuriranje.

PROCEDURA

```
-- 6. Ažuriranje statusa zaposlenika (aktivan / neaktivan)
DELIMITER //
CREATE PROCEDURE azuriraj_status_zaposlenika (
    IN p_id INT,
    IN p_status ENUM('aktivan', 'neaktivan')
)
BEGIN
    IF EXISTS (SELECT 1 FROM zaposlenik WHERE id = p_id) THEN
        UPDATE zaposlenik
        SET status_zaposlenika = p_status
        WHERE id = p_id;
    ELSE
        SELECT 'Zaposlenik s ovim ID-om ne postoji!' AS poruka;
    END IF;
END;
//
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_status_zaposlenika*, omogućava ažuriranje statusa zaposlenika u sustavu na temelju njegovog jedinstvenog identifikatora (p_id). Procedura prima dva parametra:

- p_id: ID zaposlenika čiji status treba biti ažuriran.
- p_status: Novi status zaposlenika koji može biti jedan od dva moguća stanja, 'aktivan' ili 'neaktivan'.

Prvo se provodi provjera postoji li zaposlenik s danim ID-em (p_id) u tablici *zaposlenik*. Ako zaposlenik postoji, izvršava se SQL upit koji ažurira njegov status u tablici, postavljajući status_zaposlenika na vrijednost proslijeđenu kao p_status.

Ako zaposlenik s danim ID-em ne postoji, procedura vraća poruku 'Zaposlenik s ovim ID-om ne postoji!' kako bi obavijestila korisnika o nepostojećem zapisu.

Ova procedura omogućava administrativno upravljanje statusima zaposlenika, omogućujući jednostavno ažuriranje statusa zaposlenika u skladu sa promjenama u njihovom zaposlenju ili drugim uvjetima. Provođenje provjere postojanja zaposlenika prije same izmjene osigurava da se izmjene vrše samo na postojećim zaposlenicima, čime se sprječavaju pogreške.

PROCEDURA

```
-- Procedura koja vraca popis dopustenja koje ima zaposlenik po id
DELIMITER //

CREATE PROCEDURE prikazi_prava_korisnika(
    IN p_id_zaposlenik INT
)
BEGIN
    DECLARE uloga_naziv VARCHAR(50);
    SELECT uloge.naziv
    INTO uloga_naziv
    FROM zaposlenik
    JOIN uloge ON zaposlenik.uloga_id = uloge.uloga_id
    WHERE zaposlenik.id = p_id_zaposlenik
    LIMIT 1;

        SELECT uloge.naziv AS uloga, uloge_pristupa.akcija
        FROM uloge
        JOIN uloge_pristupa ON uloge.uloga_id = uloge_pristupa.uloga_id
        WHERE uloge.naziv = uloga_naziv;

END;
//
DELIMITER ;
```

Ova procedura, nazvana *prikazi_prava_korisnika*, omogućava prikaz prava (akcija) korisnika na temelju njegove uloge. Procedura prima jedan ulazni parametar:

- p_id_zaposlenik: ID zaposlenika čija prava (akcije) se žele pregledati.

Postupak je sljedeći:

1. Dohvat uloge zaposlenika: Prvo se provodi SQL upit koji dohvaća naziv uloge zaposlenika temeljem njegovog ID-a (p_id_zaposlenik). To se radi pomoću *JOIN* operacije između tablica *zaposlenik* i *uloge*, pri čemu se identificira naziv uloge zaposlenika.
2. Prikaz prava za ulogu: Nakon što je naziv uloge zaposlenika dohvaćen, slijedi drugi SQL upit koji prikazuje sve akcije (prava) povezane s tom ulogom. Koristi se *JOIN* između tablica *uloge* i *uloge_pristupa* kako bi se dohvatili svi pristupi (akcije) koji su povezani s tom ulogom.

Na kraju, ova procedura vraća popis svih akcija (prava) koje zaposlenik ima, na temelju njegove uloge, što omogućava administratorima sustava uvid u specifična prava korisnika.

Ova funkcionalnost je korisna za provjeru pristupa zaposlenika određenim resursima i omogućava efikasno upravljanje pristupima u sustavu.

PROCEDURA

```
-- procedura za ažuriranje broja zaposlenika u tablici odjel
DELIMITER //
CREATE PROCEDURE azuriraj_broj_zaposlenika(IN p_id_odjel INTEGER)
BEGIN
    UPDATE odjel
        SET broj_zaposlenika = broj_zaposlenika_u_odjelu(p_id_odjel)
        WHERE id = p_id_odjel;
END //
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_broj_zaposlenika*, ažurira broj zaposlenika unutar određenog odjela na temelju funkcije koja izračunava broj aktivnih zaposlenika u tom odjelu. Procedura prima jedan ulazni parametar:

- p_id_odjel: ID odjela za koji se želi ažurirati broj zaposlenika.

Postupak je sljedeći:

1. **Ažuriranje broja zaposlenika**: Prvo se poziva funkcija broj_zaposlenika_u_odjelu(p_id_odjel), koja računa broj aktivnih zaposlenika u određenom odjelu na temelju ID-a odjela (p_id_odjel). Funkcija vraća broj zaposlenika koji su aktivni u tom odjelu.
2. **Postavljanje novog broja zaposlenika**: Na temelju rezultata funkcije, u tablici *odjel* ažurira se vrijednost polja broj_zaposlenika za odjel sa specifičnim ID-om (p_id_odjel).

Ova procedura omogućava automatsko ažuriranje broja zaposlenika u određenom odjelu svaki put kada se broj zaposlenika promijeni, čime se osigurava da je broj zaposlenika u sustavu uvijek točan i u skladu s podacima u drugim dijelovima sustava.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE azuriraj_kolicinu_vina (IN p_id_berba INTEGER, IN p_tip_transakcije ENUM('ulaz', 'izlaz'), IN p_kolicina DECIMAL(8,2))
BEGIN
    DECLARE nova_kolicina DECIMAL(8,2);

    IF NOT EXISTS (SELECT 1 FROM stanje_skladista_vina WHERE id_berba = p_id_berba) THEN
        IF p_tip_transakcije = 'ulaz' THEN
            INSERT INTO stanje_skladista_vina VALUES (p_id_berba, p_kolicina);
        ELSE
            SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = 'Nije moguće dodati izlaznu transakciju za berbu koja nema ulaznih transakcija!';
        END IF;
    ELSE
        IF p_tip_transakcije = 'ulaz' THEN
            UPDATE stanje_skladista_vina
                SET kolicina = kolicina + p_kolicina
                WHERE id_berba = p_id_berba;
        ELSE
            UPDATE stanje_skladista_vina
                SET kolicina = kolicina - p_kolicina
                WHERE id_berba = p_id_berba;
        END IF;
    END IF;

    SELECT kolicina INTO nova_kolicina
        FROM stanje_skladista_vina
        WHERE id_berba = p_id_berba;

    IF nova_kolicina < 0 THEN
        SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Količina vina ne može biti negativna!';
    END IF;
END //
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_kolicinu_vina*, koristi se za ažuriranje količine vina u skladištu na temelju tipa transakcije ("ulaz" ili "izlaz"). Procedura prima tri ulazna parametra:

- p_id_berba: ID berbe na koju se odnosi transakcija.
- p_tip_transakcije: Tip transakcije koji može biti "ulaz" (dodavanje vina u skladište) ili "izlaz" (oduzimanje vina iz skladišta).
- p_kolicina: Količina vina koja se dodaje ili oduzima.

Postupak je sljedeći:

1. Provjera postojanja berbe u skladištu: Prvo se provjerava postoji li unos za specifičnu berbu u tablici *stanje_skladista_vina*. Ako zapis za tu berbu ne postoji, u slučaju da je tip transakcije "ulaz", u tablicu se dodaje novi zapis sa količinom vina koja je unesena. Ako je tip transakcije "izlaz" i berba ne postoji, izbacuje se greška s porukom 'Nije moguće dodati izlaznu transakciju za berbu koja nema ulaznih transakcija!'.
2. Ažuriranje količine vina: Ako zapis za berbu postoji, procedura ažurira količinu vina u skladištu. Ako je tip transakcije "ulaz", količina se povećava za unesenu količinu, dok se za "izlaz" količina smanjuje za unesenu količinu.
3. Provjera negativne količine: Nakon što je količina ažurirana, procedura provodi dodatnu provjeru da količina vina u skladištu ne postane negativna. Ako nova količina postane negativna, izbacuje se greška s porukom 'Količina vina ne može biti negativna!'.

Ova procedura omogućava precizno upravljanje količinom vina u skladištu, osiguravajući da se količina uvijek održava ispravno, a transakcije su kontrolirane na način da se spriječi unos negativnih količina.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE azuriraj_kolicinu_proizvoda (IN p_id_proizvod INTEGER, IN p_tip_transakcije ENUM('ulaz', 'izlaz'), IN p_kolicina INTEGER)
BEGIN
    DECLARE nova_kolicina INTEGER;

    IF NOT EXISTS (SELECT 1 FROM stanje_skladista_proizvoda WHERE id_proizvod = p_id_proizvod) THEN
        IF p_tip_transakcije = 'ulaz' THEN
            INSERT INTO stanje_skladista_proizvoda VALUES (p_id_proizvod, p_kolicina);
        ELSE
            SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Nije moguće dodati izlaznu transakciju za proizvod koji nema ulaznih transakcija!';
        END IF;
    ELSE
        IF p_tip_transakcije = 'ulaz' THEN
            UPDATE stanje_skladista_proizvoda
                SET kolicina = kolicina + p_kolicina
                WHERE id_proizvod = p_id_proizvod;
        ELSE
            UPDATE stanje_skladista_proizvoda
                SET kolicina = kolicina - p_kolicina
                WHERE id_proizvod = p_id_proizvod;
        END IF;
    END IF;

    SELECT kolicina INTO nova_kolicina
        FROM stanje_skladista_proizvoda
        WHERE id_proizvod = p_id_proizvod;

    IF nova_kolicina < 0 THEN
        SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'Količina proizvoda ne može biti negativna!';
    END IF;
END //
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_kolicinu_proizvoda*, koristi se za ažuriranje količine proizvoda u skladištu na temelju tipa transakcije ("ulaz" ili "izlaz"). Procedura prima tri ulazna parametra:

- p_id_proizvod: ID proizvoda koji se ažurira.
- p_tip_transakcije: Tip transakcije koji može biti "ulaz" (dodavanje proizvoda u skladište) ili "izlaz" (oduzimanje proizvoda iz skladišta).
- p_kolicina: Količina proizvoda koja se dodaje ili oduzima.

Postupak je sljedeći:

1. Provjera postojanja proizvoda u skladištu: Prvo se provjerava postoji li zapis za specifični proizvod u tablici *stanje_skladista_proizvoda*. Ako zapis za taj proizvod ne postoji, a tip transakcije je "ulaz", u tablicu se dodaje novi zapis sa količinom proizvoda koja je unesena. Ako je tip transakcije "izlaz" i proizvod ne postoji, izbacuje se greška s porukom 'Nije moguće dodati izlaznu transakciju za proizvod koji nema ulaznih transakcija!'.
2. Ažuriranje količine proizvoda: Ako zapis za proizvod već postoji, procedura ažurira količinu proizvoda u skladištu. Ako je tip transakcije "ulaz", količina proizvoda u skladištu se povećava za unesenu količinu, dok se za "izlaz" količina smanjuje za unesenu količinu.
3. Provjera negativne količine: Nakon ažuriranja količine, procedura provodi dodatnu provjeru kako bi osigurala da količina proizvoda ne postane negativna. Ako nova količina postane negativna, izbacuje se greška s porukom 'Količina proizvoda ne može biti negativna!'.

Ova procedura omogućava precizno praćenje i upravljanje količinama proizvoda u skladištu, osiguravajući da se transakcije pravilno obrađuju i sprječava unos negativnih količina, što bi moglo dovesti do pogrešaka u skladištima i poslovnim procesima.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE azuriraj_kolicinu_repromaterijala (IN p_id_repromaterijal INTEGER, IN p_tip_transakcije ENUM('ulaz', 'izlaz'), p_kolicina INTEGER)
BEGIN
    DECLARE nova_kolicina INTEGER;

    IF NOT EXISTS (SELECT 1 FROM stanje_skladista_repromaterijala WHERE id_repromaterijal = p_id_repromaterijal) THEN
        IF p_tip_transakcije = 'ulaz' THEN
            INSERT INTO stanje_skladista_repromaterijala VALUES (p_id_repromaterijal, p_kolicina);
        ELSE
            SIGNAL SQLSTATE '45006' SET MESSAGE_TEXT = 'Nije moguće dodati izlaznu transakciju za repromaterijal koji nema ulaznih transakcija!';
        END IF;
    ELSE
        IF p_tip_transakcije = 'ulaz' THEN
            UPDATE stanje_skladista_repromaterijala
                SET kolicina = kolicina + p_kolicina
                WHERE id_repromaterijal = p_id_repromaterijal;
        ELSE
            UPDATE stanje_skladista_repromaterijala
                SET kolicina = kolicina - p_kolicina
                WHERE id_repromaterijal = p_id_repromaterijal;
        END IF;
    END IF;

    SELECT kolicina INTO nova_kolicina
        FROM stanje_skladista_repromaterijala
        WHERE id_repromaterijal = p_id_repromaterijal;

    IF nova_kolicina < 0 THEN
        SIGNAL SQLSTATE '45007' SET MESSAGE_TEXT = 'Količina repromaterijala ne može biti negativna!';
    END IF;

END //
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_kolicinu_repromaterijala*, koristi se za ažuriranje količine repromaterijala u skladištu na temelju tipa transakcije ("ulaz" ili "izlaz"). Procedura prima tri ulazna parametra:

- p_id_repromaterijal: ID repromaterijala koji se ažurira.
- p_tip_transakcije: Tip transakcije koji može biti "ulaz" (dodavanje repromaterijala u skladište) ili "izlaz" (oduzimanje repromaterijala iz skladišta).
- p_kolicina: Količina repromaterijala koja se dodaje ili oduzima.

Postupak je sljedeći:

1. Provjera postojanja repromaterijala u skladištu: Prvo se provjerava postoji li zapis za specifični repromaterijal u tablici *stanje_skladista_repromaterijala*. Ako zapis za repromaterijal ne postoji, a tip transakcije je "ulaz", u tablicu se dodaje novi zapis sa količinom repromaterijala koja je unesena. Ako je tip transakcije "izlaz" i repromaterijal ne postoji, izbacuje se greška s porukom 'Nije moguće dodati izlaznu transakciju za repromaterijal koji nema ulaznih transakcija!'.
2. Ažuriranje količine repromaterijala: Ako zapis za repromaterijal već postoji, procedura ažurira količinu repromaterijala u skladištu. Ako je tip transakcije "ulaz", količina repromaterijala u skladištu se povećava za unesenu količinu, dok se za "izlaz" količina smanjuje za unesenu količinu.
3. Provjera negativne količine: Nakon ažuriranja količine, procedura provodi dodatnu provjeru kako bi osigurala da količina repromaterijala ne postane negativna. Ako nova količina postane negativna, izbacuje se greška s porukom 'Količina repromaterijala ne može biti negativna!'.

Ova procedura omogućava precizno praćenje i upravljanje količinama repromaterijala u skladištu, osiguravajući da se transakcije pravilno obrađuju i sprječava unos negativnih količina, što bi moglo dovesti do pogrešaka u skladištima i poslovnim procesima vezanim uz repromaterijal.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE azuriraj_prodane_proizvode(IN p_id_proizvod INTEGER, IN p_kolicina INTEGER, IN p_ukupni_iznos DECIMAL(10,2), p_kvartal VARCHAR(20))
BEGIN
    IF NOT EXISTS (SELECT 1 FROM kvartalni_pregled_prodaje WHERE id_proizvod = p_id_proizvod AND kvartal = p_kvartal) THEN
        INSERT INTO kvartalni_pregled_prodaje VALUES (p_id_proizvod, p_kolicina, p_ukupni_iznos, p_kvartal);
    ELSE
        UPDATE kvartalni_pregled_prodaje
            SET kolicina = kolicina + p_kolicina, ukupni_iznos = ukupni_iznos + p_ukupni_iznos
        WHERE id_proizvod = p_id_proizvod
            AND kvartal = p_kvartal;
    END IF;
END //
DELIMITER ;
```

Procedura azuriraj_prodane_proizvode prima atribute tablice kvartalni_pregled_prodaje kao parametre i provjerava postoji li određena kombinacija id_proizvoda i kvartala već u tablici. Ako takva kombinacija ne postoji, izvršava se INSERT operacija kako bi se unijele vrijednosti parametara u tablicu. Ako kombinacija već postoji, tada se izvršava UPDATE operacija koja ažurira količinu i ukupni iznos za tu kombinaciju id_proizvoda i kvartala, povećavajući postojeće vrijednosti za količinu i ukupni iznos prema novim parametrima.

Ova procedura omogućava praćenje prodaje proizvoda po kvartalima, s automatskim ažuriranjem podataka o količini i iznosu prodaje ako se isti proizvod već nalazi u tablici za određeni kvartal.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE azuriraj_prodaju(IN p_pocetni_datum DATE, IN p_zavrsni_datum DATE)
BEGIN
    DECLARE var_id_proizvod, var_kolicina INTEGER;
    DECLARE var_ukupni_iznos DECIMAL(10,2);
    DECLARE var_kvartal VARCHAR(20);
    DECLARE handler_broj INTEGER DEFAULT 0;

    DECLARE cur CURSOR FOR
        SELECT id_proizvod, kolicina, iznos_stavke
            FROM stavka_narudzbe
            WHERE id_zahtjev_za_narudzbu IN (
                SELECT id_zahtjev_za_narudzbu
                    FROM racun
                    WHERE datum_racuna BETWEEN p_pocetni_datum AND p_zavrsni_datum
                );
    DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
        SET handler_broj = 1;
    END;
    IF DATE_FORMAT(p_pocetni_datum, '%d.%m.') NOT IN ('01.01.', '01.04.', '01.07.', '01.10.') THEN
        SIGNAL SQLSTATE '45008' SET MESSAGE_TEXT = 'Neispravan unos, početni datum mora biti početak kvartala!';
    END IF;

    IF DATE_FORMAT(p_zavrsni_datum, '%d.%m.') NOT IN ('31.03.', '30.6.', '30.09.', '31.12.') THEN
        SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'Neispravan unos, završni datum mora biti završetak kvartala!';
    END IF;

    IF p_pocetni_datum > p_zavrsni_datum THEN
        SIGNAL SQLSTATE '45010' SET MESSAGE_TEXT = 'Početni datum ne može biti nakon završnog datuma!';
    END IF;



   SELECT
       CASE
           WHEN p_zavrsni_datum > p_pocetni_datum + INTERVAL 3 MONTH THEN 'Krivi unos!'
           WHEN MONTH(p_pocetni_datum) = 1 AND MONTH(p_zavrsni_datum) = 3 THEN CONCAT('Q1 ', YEAR(p_pocetni_datum))
           WHEN MONTH(p_pocetni_datum) = 4 AND MONTH(p_zavrsni_datum) = 6 THEN CONCAT('Q2 ', YEAR(p_pocetni_datum))
           WHEN MONTH(p_pocetni_datum) = 7 AND MONTH(p_zavrsni_datum) = 9 THEN CONCAT('Q3 ', YEAR(p_pocetni_datum))
           WHEN MONTH(p_pocetni_datum) = 10 AND MONTH(p_zavrsni_datum) = 12 THEN CONCAT('Q4 ', YEAR(p_pocetni_datum))
           ELSE 'Krivi unos!'
       END
   INTO var_kvartal;

   IF var_kvartal = 'Krivi unos!' THEN
       SIGNAL SQLSTATE '45013' SET MESSAGE_TEXT = 'Uneseni datumi ne odgovaraju niti jednom kvartalu!';
   END IF;

   OPEN cur;

   petlja: LOOP
       FETCH cur INTO var_id_proizvod, var_kolicina, var_ukupni_iznos;

       IF handler_broj = 1 THEN
           LEAVE petlja;
       END IF;

       CALL azuriraj_prodane_proizvode(var_id_proizvod, var_kolicina, var_ukupni_iznos, var_kvartal);

   END LOOP petlja;

   CLOSE cur;
END //
DELIMITER ;
```

Procedura **azuriraj_prodaju** koristi kursor i LOOP petlju za iteraciju kroz id_proizvod, kolicina i iznos_stavke u tablici stavka_narudzbe za zahtjeve za narudžbu, gdje je račun izdan između početnog i završnog datuma koji se predaju kao parametri proceduri. Prije početka iteracije, procedura provodi validaciju predanih datuma i signalizira odgovarajuće greške ako neki uvjet nije zadovoljen. Ako su datumi ispravno uneseni, na temelju tih datuma odredjuje se kvartal (npr. Q4 2024) koji se sprema u varijablu var_kvartal.

Nakon što kursor dohvati svaki redak koji nije prazan, poziva se procedura **azuriraj_prodane_proizvode**, kojoj se predaju varijable var_id_proizvod, var_kolicina, var_ukupni_iznos i var_kvartal. Kada kursor dođe do praznog retka, CONTINUE HANDLER postavlja handler_broj na 1, čime se završava petlja.

Ova procedura omogućava ažuriranje podataka o prodanim proizvodima prema kvartalima na temelju izdanih računa unutar zadatog vremenskog razdoblja.

Tablica **kvartalni_pregled_prodaje**, koja se popunjava pomoću ove procedure, zamišljena je kao kvartalni izvještaj o prodaji, koji sadrži podatke o prodanoj količini i zarađenom iznosu za svaki proizvod. Izvještaj će se generirati putem eventa kvartalni_izvjestaj, koji poziva proceduru **azuriraj_prodaju** s argumentima CURDATE()- INTERVAL 3 MONTH i CURDATE()- INTERVAL 1 DAY. Argumenti su postavljeni na ovaj način jer event počinje 01.04. i ponavlja se svaka 3 mjeseca. Tako se dobivaju datumi koji točno odgovaraju proteklom kvartalu (npr. kad se event prvi put aktivira 01.04., proceduri će biti predani datumi 01.01. i 31.03., što odgovara početnom i završnom datumu za Q1 2025).

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE izracunaj_kolicinu_transporta(IN p_id_transport INTEGER)
BEGIN
    DECLARE kolicina_transporta INTEGER;

    SELECT SUM(sn.kolicina) INTO kolicina_transporta
        FROM stavka_narudzbe sn
        JOIN zahtjev_za_narudzbu zzn ON zzn.id = sn.id_zahtjev_za_narudzbu
        JOIN transport t ON t.id = zzn.id_transport
        WHERE zzn.id_transport = p_id_transport;

    IF kolicina_transporta IS NULL THEN
        SIGNAL SQLSTATE '45010' SET MESSAGE_TEXT = 'Transport mora biti dodijeljen barem jednoj narudžbi!';
    ELSE
        UPDATE transport
            SET kolicina = kolicina_transporta
            WHERE id = p_id_transport;
    END IF;
END //
DELIMITER ;
```

Ova procedura, nazvana *izracunaj_kolicinu_transporta*, koristi se za izračunavanje ukupne količine robe koja je povezana s određenim transportom, temeljem stavki narudžbi koje su povezane s tim transportom. Procedura prima jedan ulazni parametar:

- p_id_transport: ID transporta za koji se izračunava ukupna količina.

Postupak se sastoji od nekoliko ključnih koraka:

1. **Deklaracija varijable**:
    - kolicina_transporta: Varijabla koja će pohraniti izračunatu ukupnu količinu robe povezanu s transportom.
2. **Dohvat podataka**:

- Procedura koristi SQL upit za dohvat ukupne količine robe povezane s transportom. Upit koristi **JOIN** između tablica stavka_narudzbe, zahtjev_za_narudzbu i transport, filtrirajući zapise prema ID-u transporta (p_id_transport).
- Količina robe (kolicina) koja je povezana s transportom se sumira i pohranjuje u varijablu kolicina_transporta.

3. **Provjera valjanosti podataka**:
   - Ako je izračunata količina transporta (kolicina_transporta) NULL (što znači da nije pronađena nijedna stavka narudžbe povezane s transportom), izvršava se signalna greška s odgovarajućom porukom: 'Transport mora biti dodijeljen barem jednoj narudžbi!'.
4. **Ažuriranje podataka**:
   - Ako je količina validna (nije NULL), procedura ažurira tablicu transport, postavljajući izračunatu količinu na odgovarajući zapis transporta u polje kolicina.

Cilj ove procedure je automatski izračunati ukupnu količinu robe koja je povezana s određenim transportom i ažurirati podatke o količini u sustavu. Ako transport nije povezan s nijednom narudžbom, procedura generira grešku, upozoravajući korisnika na potrebu za povezivanjem transporta s barem jednim zahtjevom za narudžbu.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE dodaj_novu_berbu (IN p_id_vino INTEGER, IN p_godina_berbe INTEGER, IN p_postotak_alkohola DECIMAL(5, 2))
BEGIN
    IF p_godina_berbe > YEAH(CURDATE()) THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Godina berbe ne može biti u budućnosti';
    END IF;
    INSERT INTO berba (id_vino, godina_berbe, postotak_alkohola)
    VALUES (p_id_vino, p_godina_berbe, p_postotak_alkohola);
END //
DELIMITER ;
```

Ova procedura, nazvana *dodaj_novu_berbu*, omogućava unos nove berbe u sustav. Procedura prima tri ulazna parametra:

- p_id_vino: ID vina na koje se odnosi berba.
- p_godina_berbe: Godina berbe (koja se koristi za označavanje kada je berba izvršena).
- p_postotak_alkohola: Postotak alkohola u vinu koje je proizvedeno iz te berbe.

Postupak:

1. **Provjera godine berbe**:
   - Procedura prvo provjerava je li godina berbe (p_godina_berbe) veća od trenutne godine. Ako je godina berbe u budućnosti, generira se greška pomoću komande SIGNAL SQLSTATE '45000' s porukom: 'Godina berbe ne može biti u budućnosti'. Ovo sprječava unos podataka koji nisu realni.
2. **Unos nove berbe**:
   - Ako godina berbe nije u budućnosti, izvodi se SQL upit za unos nove berbe u tablicu berba. Novi zapis uključuje:

- p_id_vino (ID vina),
- p_godina_berbe (godina berbe),
- p_postotak_alkohola (postotak alkohola u vinu).

Cilj procedure:

Procedura *dodaj_novu_berbu* omogućava unos nove berbe u bazu podataka, ali s dodatnim uvjetom da godina berbe mora biti u prošlosti ili tekućoj godini. Ako je unesena godina u budućnosti, procedura neće dozvoliti unos i korisnik će biti obaviješten o grešci.

PROCEDURA

```
DELIMITER //
CREATE PROCEDURE azuriraj_stanje(IN p_tip_transakcije ENUM('ulaz', 'izlaz'), IN p_lokacija VARCHAR(100), IN p_kolicina INT)

BEGIN
    DECLARE l_postoji INT;
    SELECT COUNT(*) INTO l_postoji
        FROM mp_stanje_skladista_vina
        WHERE lokacija = p_lokacija;
    SET SQL_SAFE_UPDATES = 0;
    IF p_tip_transakcije = 'ulaz' THEN
        IF l_postoji = 0 THEN
            INSERT INTO mp_stanje_skladista_vina VALUES(p_lokacija, p_kolicina);
        ELSE
            UPDATE mp_stanje_skladista_vina
            SET kolicina = kolicina + p_kolicina WHERE lokacija = p_lokacija;
        END IF;
    END IF;
    IF p_tip_transakcije = 'izlaz' THEN
        IF l_postoji = 0 THEN
                INSERT INTO mp_stanje_skladista_vina VALUES(p_lokacija, p_kolicina);
            ELSE
                UPDATE mp_stanje_skladista_vina
                SET kolicina = kolicina - p_kolicina WHERE lokacija = p_lokacija;
            END IF;

    END IF;
    SET SQL_SAFE_UPDATES = 1;
END //
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_stanje*, služi za ažuriranje stanja skladišta vina u skladu s ulaznim ili izlaznim transakcijama. Procedura prima tri ulazna parametra:

- p_tip_transakcije: Tip transakcije, može biti 'ulaz' (dodavanje vina u skladište) ili 'izlaz' (iznošenje vina iz skladišta).
- p_lokacija: Lokacija skladišta vina na kojoj se stanje treba ažurirati.
- p_kolicina: Količina vina koja se dodaje ili oduzima iz skladišta.

Postupak:

1. **Provjera postoji li zapis za lokaciju**:
    - Procedura prvo provodi upit kako bi provjerila postoji li već zapis za unesenu lokaciju (p_lokacija) u tablici mp_stanje_skladista_vina. Rezultat upita pohranjuje se u varijablu l_postoji. Ako zapis za ovu lokaciju ne postoji, l_postoji će biti 0.
2. **Ažuriranje stanja za ulaznu transakciju**:
    - Ako je tip transakcije 'ulaz' (dodavanje vina), procedura provodi provjeru:

- Ako zapis za lokaciju ne postoji (l_postoji = 0), umetne se novi zapis u tablicu mp_stanje_skladista_vina s početnom količinom p_kolicina.
- Ako zapis za lokaciju već postoji, količina vina na toj lokaciji se uvećava za p_kolicina.

3. Ažuriranje stanja za izlaznu transakciju:
    - Ako je tip transakcije 'izlaz' (izlazak vina), procedura također provodi provjeru:
        - Ako zapis za lokaciju ne postoji (l_postoji = 0), umetne se novi zapis u tablicu mp_stanje_skladista_vina s početnom količinom p_kolicina (količina se ne oduzima jer u tom slučaju još nije bilo vina na lokaciji).
        - Ako zapis za lokaciju već postoji, količina vina na toj lokaciji se smanjuje za p_kolicina.
4. Sigurnosna postavka za ažuriranje:
    - U proceduri je postavljeno SET SQL_SAFE_UPDATES = 0, što omogućava izvršavanje ažuriranja čak i kad postoje uvjeti koji obično ograničavaju promjene (npr. korištenje WHERE klauzule s ograničenim uvjetima). Nakon što su ažuriranja završena, postavka se vraća na SET SQL_SAFE_UPDATES = 1 kako bi se omogućila normalna pravila za kasnija ažuriranja.

Cilj procedure:

Procedura *azuriraj_stanje* omogućava ažuriranje skladišta vina s obzirom na tip transakcije. U slučaju ulazne transakcije, količina vina na određenoj lokaciji se povećava, dok u slučaju izlazne transakcije količina vina na toj lokaciji opada. Ako zapis za određenu lokaciju ne postoji, on se kreira s početnom količinom vina.

PROCEDURA

```
-- Procedura za dodavanje novog proizvoda:

DELIMITER //
CREATE PROCEDURE dodaj_novo_vino(
    IN p_naziv VARCHAR(255),
    IN p_vrsta ENUM('bijelo', 'crno', 'rose', 'pjenušavo'),
    IN p_sorta VARCHAR(100)
)
BEGIN
    INSERT INTO vino (naziv, vrsta, sorta)
    VALUES (p_naziv, p_vrsta, p_sorta);
END //
DELIMITER ;
```

Ova procedura, nazvana *dodaj_novo_vino*, omogućava unos novog vina u bazu podataka. Procedura prima tri ulazna parametra:

- p_naziv: Naziv vina (npr. "Cabernet Sauvignon").
- p_vrsta: Vrsta vina koja može biti jedan od četiri moguća izbora: 'bijelo', 'crno', 'rose', ili 'pjenušavo'.
- p_sorta: Sorta grožđa koja je korištena za proizvodnju vina (npr. "Merlot").

Postupak:

1. Unos novog vina:
    - Procedura izvršava SQL upit koji unosi novi zapis u tablicu vino, koristeći vrijednosti koje su poslane kao argumenti:
        - naziv se postavlja na vrijednost p_naziv,
        - vrsta se postavlja na vrijednost p_vrsta,
        - sorta se postavlja na vrijednost p_sorta.

Cilj procedure:

Procedura *dodaj_novo_vino* omogućava unos novih vina u bazu podataka, sa specifičnim informacijama o nazivu, vrsti i sorti grožđa.

PROCEDURA

```
-- procedura koja omogućuje ažuriranje statusa narudžbe u tablici zahtjev_za_narudzbu

DELIMITER //
CREATE PROCEDURE azuriraj_status_narudzbe (
    IN p_id_narudzbe INT,
    IN p_novi_status ENUM('Primljena', 'U obradi', 'Na čekanju', 'Spremna za isporuku', 'Poslana', 'Završena', 'Otkazana')
)
BEGIN
    UPDATE zahtjev_za_narudzbu
    SET status_narudzbe = p_novi_status
    WHERE id = p_id_narudzbe;
END//
DELIMITER ;
```

Ova procedura, nazvana *azuriraj_status_narudzbe*, omogućava ažuriranje statusa narudžbe u sustavu. Procedura prima dva ulazna parametra:

- p_id_narudzbe: ID narudžbe čiji status treba biti ažuriran.
- p_novi_status: Novi status narudžbe koji se postavlja, a može biti jedan od sljedećih: 'Primljena', 'U obradi', 'Na čekanju', 'Spremna za isporuku', 'Poslana', 'Završena', ili 'Otkazana'.

Postupak:

1. **Ažuriranje statusa narudžbe**:
    - SQL upit u proceduri ažurira tablicu zahtjev_za_narudzbu tako da postavlja novi status (status_narudzbe) za narudžbu s navedenim ID-em (p_id_narudzbe).

Cilj procedure:

Procedura omogućava jednostavno ažuriranje statusa narudžbe, čime se prati napredak narudžbe kroz različite faze obrade, od primanja do isporuke ili otkazivanja.

PROCEDURA

```
-- procedura koja generira račun za određeni zahtjev za narudžbu i automatski ga dodaje u tablicu racun

DELIMITER //
CREATE PROCEDURE generiraj_racun (
    IN p_id_zaposlenik INT,
    IN p_id_narudzba INT
)
BEGIN
    INSERT INTO racun (id_zaposlenik, id_zahtjev_za_narudzbu, datum_racuna)
    VALUES (p_id_zaposlenik, p_id_narudzba, CURDATE());
END//
DELIMITER ;
```

Ova procedura, nazvana *generiraj_racun*, koristi se za generiranje računa za određenu narudžbu. Procedura prima dva ulazna parametra:

- p_id_zaposlenik: ID zaposlenika koji generira račun.
- p_id_narudzba: ID narudžbe na temelju koje se generira račun.

Postupak:

1. **Unos novog računa**:
    - SQL upit u proceduri umetne novi zapis u tablicu racun, postavljajući:
        - id_zaposlenik na ID zaposlenika koji generira račun (p_id_zaposlenik),
        - id_zahtjev_za_narudzbu na ID narudžbe (p_id_narudzba),
        - datum_racuna na trenutni datum (dobiven funkcijom CURDATE()).

Cilj procedure:

Procedura automatski generira račun za narudžbu u trenutku kada je zaposlenik odgovoran za izdavanje računa, pomoću zadnjih podataka vezanih uz narudžbu i zaposlenika.

PROCEDURA

```sql
-- Kreiranje procedure za ažuriranje podataka o zaposleniku
DELIMITER $$

CREATE PROCEDURE update_zaposlenik(
    IN p_id INT,
    IN p_ime VARCHAR(255),
    IN p_prezime VARCHAR(255),
    IN p_adresa VARCHAR(255),
    IN p_email VARCHAR(255),
    IN p_telefon VARCHAR(20),
    IN p_status_zaposlenika VARCHAR(50)
)
BEGIN
    UPDATE zaposlenik
    SET ime = p_ime, prezime = p_prezime, adresa = p_adresa, email = p_email, telefon = p_telefon, status_zaposlenika = p_status_zaposlenika
    WHERE id = p_id;
END $$

DELIMITER ;
```

Ova procedura, nazvana *update_zaposlenik*, koristi se za ažuriranje podataka zaposlenika u tablici zaposlenik. Procedura prima sedam ulaznih parametara:

- p_id: ID zaposlenika koji se ažurira.
- p_ime: Novo ime zaposlenika.
- p_prezime: Novo prezime zaposlenika.
- p_adresa: Nova adresa zaposlenika.
- p_email: Novi email zaposlenika.
- p_telefon: Novi telefonski broj zaposlenika.
- p_status_zaposlenika: Novi status zaposlenika (npr. "aktivan", "neaktivan").

Postupak:

1. **Ažuriranje podataka zaposlenika**:
    - SQL upit unutar procedure koristi UPDATE naredbu za ažuriranje svih informacija o zaposleniku u tablici zaposlenik, postavljajući nove vrijednosti za ime, prezime, adresu, email, telefon i status zaposlenika.
    - WHERE id = p_id uvjet osigurava da se ažuriranje odnosi samo na zaposlenika s određenim ID-em (p_id).

Cilj procedure:

Procedura omogućava ažuriranje podataka o zaposleniku na temelju unesenih novih podataka.

PROCEDURA

```sql
-- Kreiranje procedure za unos novog zaposlenika
DELIMITER $$

CREATE PROCEDURE insert_zaposlenik(
    IN p_id_odjel INT,
    IN p_ime VARCHAR(255),
    IN p_prezime VARCHAR(255),
    IN p_adresa VARCHAR(255),
    IN p_email VARCHAR(255),
    IN p_telefon VARCHAR(20),
    IN p_datum_zaposlenja DATE,
    IN p_status_zaposlenika VARCHAR(50)
)
BEGIN
    INSERT INTO zaposlenik (id_odjel, ime, prezime, adresa, email, telefon, datum_zaposlenja, status_zaposlenika)
    VALUES (p_id_odjel, p_ime, p_prezime, p_adresa, p_email, p_telefon, p_datum_zaposlenja, p_status_zaposlenika);
END $$

DELIMITER ;
```

Ova procedura, nazvana *insert_zaposlenik*, koristi se za unos novih podataka o zaposleniku u tablicu zaposlenik. Procedura prima osam ulaznih parametara:

- p_id_odjel: ID odjela u kojem zaposlenik radi.
- p_ime: Ime zaposlenika.
- p_prezime: Prezime zaposlenika.
- p_adresa: Adresa zaposlenika.
- p_email: Email zaposlenika.
- p_telefon: Telefonski broj zaposlenika.
- p_datum_zaposlenja: Datum kada je zaposlenik započeo s radom.
- p_status_zaposlenika: Status zaposlenika (npr. "aktivan", "neaktivan").

Postupak:

1. **Unos novog zaposlenika**:
    - SQL upit unutar procedure koristi INSERT INTO naredbu za dodavanje novog zaposlenika u tablicu zaposlenik. Svi ulazni podaci (ime, prezime, adresa, email, telefon, datum zaposlenja, status zaposlenika, te ID odjela) unose se u odgovarajuće kolone tablice.

Cilj procedure:

Procedura omogućava unos novog zaposlenika u sustav s pratećim podacima, kao što su osobni podaci, datum zaposlenja, status zaposlenika, te pripadnost određenom odjelu.

9. INICIJALIZACIJA I KONFIGURACIJA APLIKACIJE

app = Flask(name): Kreira instancu Flask aplikacije, gdje __name__ označava naziv trenutnog modula, što omogućava Flasku da prepozna gdje se nalazi aplikacija.

app.config['JSON_AS_ASCII'] = False: Postavlja konfiguraciju koja omogućava aplikaciji da koristi ne-ASCII znakove u JSON odgovorima. Ovo je korisno ako aplikacija treba raditi s podacima koji sadrže znakove iz drugih jezika osim engleskog.

app.config['MYSQL_HOST'] = 'localhost': Definira naziv hosta za povezivanje s MySQL bazom podataka. U ovom slučaju, postavljeno je na localhost, što znači da se baza podataka nalazi na istoj mašini kao i aplikacija.

app.config['MYSQL_DB'] = 'vinarija': Postavlja naziv baze podataka koja će biti korištena.

mysql = MySQL(app): Inicijalizira MySQL objekt za aplikaciju, omogućujući aplikaciji interakciju s MySQL bazom podataka. Ova instanca omogućava korištenje metoda za izvršavanje SQL upita, dohvat podataka i druge operacije na bazi.

app.config['SECRET_KEY'] = '984/%$fs8': Postavlja tajni ključ za aplikaciju. Tajni ključ se koristi za enkripciju podataka, kao što su sesije i kolačići, što je važno za sigurnost aplikacije.

10. RUTE I FUNKCIONALNOSTI

PRIMJER I OBJAŠNJENJE:

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    message = ''
    if request.method == 'POST':
        db_username = request.form.get('db_username')
        db_password = request.form.get('db_password')

        if db_username and db_password:
            try:
                session['db_username'] = db_username
                session['db_password'] = db_password

                # Konfiguracija za pristup MySQL
                app.config['MYSQL_USER'] = db_username
                app.config['MYSQL_PASSWORD'] = db_password
                mysql.init_app(app)

                cur = mysql.connection.cursor()
                cur.execute('SELECT 1')
                result = cur.fetchone()

                if result:
                    session['loggedin'] = True
                else:
                    message = 'Failed to connect to the database'
                    print("Failed to connect to the database")

            except Exception as e:
                message = f"Error: {e}"

        else:
            message = 'Please provide database username and password'
            print('Please provide database username and password')

        return render_template('index.html', message=message)
```

@app.route('/login', methods=['GET', 'POST']): Ova ruta omogućava korisnicima pristup stranici za prijavu putem URL-a /login i podržava HTTP metode GET i POST. Ruta koristi GET za prikazivanje forme za prijavu i POST za obradu podataka koje korisnik unese u formu.

def login(): Funkcija login() obrađuje prijavu korisnika. Kada korisnik posjeti stranicu, funkcija prvo provjerava koji HTTP zahtjev je poslan.

message = '': Inicijalizacija varijable message koja se koristi za pohranu poruka koje će biti prikazane korisniku, kao što su greške u unosu podataka ili uspješna prijava.

if request.method == 'POST': Provjera da li je HTTP zahtjev tipa POST. To znači da je korisnik poslao podatke putem forme za prijavu. Ako je metoda POST, funkcija nastavlja s obradom podataka.

db_username = request.form.get('db_username') i **db_password = request.form.get('db_password')**: Dohvaćanje unesenih podataka (korisničkog imena i lozinke) iz forme. Podaci se pohranjuju u varijable db_username i db_password.

if db_username and db_password: Provjera da li su korisničko ime i lozinka uneseni. Ako su oba podatka prisutna, nastavak s obradom.

session['db_username'] = db_username i session['db_password'] = db_password: Pohranjivanje unesenih podataka u sesiju kako bi se mogli koristiti u kasnijim zahtjevima. Ovo omogućava praćenje prijavljenog korisnika.

app.config['MYSQL_USER'] = db_username i app.config['MYSQL_PASSWORD'] = db_password: Konfiguracija za pristup MySQL bazi podataka s korisničkim imenom i lozinkom koje je korisnik unio. Ove vrijednosti će se koristiti za povezivanje s bazom podataka.

mysql.init_app(app): Inicijalizacija MySQL aplikacije pomoću konfiguracije koju smo postavili (korisničkog imena i lozinke).

cur = mysql.connection.cursor(): Kreiranje kursora za izvršavanje SQL upita prema bazi podataka.

cur.execute('SELECT 1'): Izvršavanje jednostavnog SQL upita koji samo provjerava može li aplikacija uspostaviti vezu s bazom podataka.

result = cur.fetchone(): Dohvaćanje rezultata upita. Ako je veza uspješno uspostavljena, rezultat će biti prisutan.

if result: Provjera da li je rezultat prisutan, što znači da je veza s bazom podataka uspješno uspostavljena. Ako je uspješno povezivanje, postavlja se sesijska varijabla loggedin = True, čime se označava da je korisnik uspješno prijavljen.

else: Ako se ne uspije povezati s bazom podataka, postavlja se poruka o grešci u varijablu message: 'Failed to connect to the database'. Također se ispisuje poruka u konzolu.

except Exception as e: Ako tijekom obrade podataka dođe do bilo kakvih grešaka, ta greška se hvata i ispisuje poruka o grešci u varijablu message.

else: Ako korisnik nije unio korisničko ime ili lozinku, postavlja se poruka: 'Please provide database username and password', koja obavještava korisnika da su potrebni podaci za prijavu.

return render_template('index.html', message=message): Na kraju, funkcija vraća HTML stranicu (template) index.html i prosljeđuje poruku (ako postoji) koja će biti prikazana korisniku na stranici, npr. greške u prijavi ili obavijest o uspješnom prijavljivanju.

POPIS RUTA

Rute za početnu stranicu:

1. **/ (GET)**
   Prikazuje početnu stranicu (login).
2. **/home (GET)**
   Prikazuje početnu stranicu za korisnika.

Rute za zaposlenike:

1. **/zaposlenik (GET)**
   Dohvaća sve podatke o zaposlenicima i prikazuje ih na stranici zaposlenik.html.

2. **/dodaj_zaposlenika_forma (GET)**
   Prikazuje formu za dodavanje novog zaposlenika.

3. **/dodaj_zaposlenika (POST)**
   Dodaje novog zaposlenika u bazu i preusmjerava na /zaposlenik.

4. **/edit_zaposlenika (GET)**
   Prikazuje popis zaposlenika za uređivanje.

5. **/obrisi_zaposlenika/int:id (POST)**
   Briše zaposlenika s danim id i preusmjerava na /edit_zaposlenika.

6. **/edit_zaposlenika_forma/int:id (GET)**
   Prikazuje formu za uređivanje podataka o odabranom zaposleniku.

7. **/update_zaposlenika/int:id (POST)**
   Ažurira podatke o zaposleniku i preusmjerava na /zaposlenik.

Rute za dobavljače:

1. **/dodaj_dobavljaca_forma (GET)**
   Prikazuje formu za dodavanje novog dobavljača.

2. **/dodaj_dobavljaca (POST)**
   Dodaje novog dobavljača u bazu i preusmjerava na /dobavljac.

3. **/edit_dobavljaca (GET)**
   Prikazuje popis dobavljača za uređivanje.

4. **/obrisi_dobavljaca/int:id (POST)**
   Briše dobavljača s danim id i preusmjerava na /edit_dobavljaca.

5. **/edit_dobavljaca_forma/int:id (GET)**
   Prikazuje formu za uređivanje podataka o odabranom dobavljaču.

6. **/update_dobavljaca/int:id (POST)**
   Ažurira podatke o dobavljaču i preusmjerava na /dobavljac.

7. **/dobavljac (GET)**
   Dohvaća sve podatke o dobavljačima i prikazuje ih na stranici dobavljac.html.

Rute za repromaterijal:

1. **/repromaterijal (GET)**
   Dohvaća sve podatke o repromaterijalu i prikazuje ih na stranici repromaterijal.html.

2. **/dodaj_repromaterijal_forma (GET)**
   Prikazuje formu za dodavanje novog repromaterijala.

3. **/dodaj_repromaterijal (POST)**
   Dodaje novi repromaterijal u bazu i preusmjerava na /repromaterijal.

4. **/izbrisi_repromaterijal_forma (GET)**
   Prikazuje formu za brisanje repromaterijala.

5. **/izbrisi_repromaterijal (POST)**
   Briše repromaterijal s danim id i preusmjerava na /repromaterijal.

6. **/repromaterijal_proizvod (GET)**
   Prikazuje popis svih repromaterijala po proizvodu.

Rute za vino:

1. **/vino (GET)**
   Dohvaća sve podatke o vinima i prikazuje ih na stranici vino.html.
2. **/dodaj_vino_forma (GET)**
   Prikazuje formu za dodavanje novog vina.
3. **/dodaj_vino (POST)**
   Dodaje novo vino u bazu i preusmjerava na /vino.
4. **/obrisi_vino_forma (GET)**
   Prikazuje formu za brisanje vina.
5. **/izbrisi_vino (POST)**
   Briše vino s danim id i preusmjerava na /vino.

Rute za prijevoznike:

1. **/prijevoznik (GET)**
   Dohvaća sve podatke o prijevoznicima i prikazuje ih na stranici prijevoznik.html.
2. **/dodaj_prijevoznika_forma (GET)**
   Prikazuje formu za dodavanje novog prijevoznika.
3. **/dodaj_prijevoznika (POST)**
   Dodaje novog prijevoznika u bazu i preusmjerava na /prijevoznik.
4. **/izbrisi_prijevoznika_forma (GET)**
   Prikazuje formu za brisanje prijevoznika.
5. **/izbrisi_prijevoznika (POST)**
   Briše prijevoznika s danim id i preusmjerava na /izbrisi_prijevoznika_forma.

Rute za proizvode:

1. **/proizvod (GET)**
   Dohvaća sve podatke o proizvodima i prikazuje ih na stranici proizvod.html.
2. **/dodaj_proizvod_forma (GET)**
   Prikazuje formu za dodavanje novog proizvoda.
3. **/dodaj_proizvod (POST)**
   Dodaje novi proizvod u bazu i preusmjerava na /proizvod.
4. **/izbrisi_proizvod_forma (GET)**
   Prikazuje formu za brisanje proizvoda.
5. **/izbrisi_proizvod (POST)**
   Briše proizvod s danim id i preusmjerava na /proizvod.
6. **/update_proizvod_forma (GET)**
   Prikazuje formu za ažuriranje proizvoda.

7. **/azuriraj_proizvod (POST)**
   Ažurira podatke o proizvodu i preusmjerava na /proizvod.

Rute za zahtjeve i stavke narudžbe:

1. **/zahtjev_za_nabavu (GET)**
   Dohvaća sve podatke o zahtjevima za nabavu i prikazuje ih na stranici zahtjev_za_nabavu.html.
2. **/zahtjev_za_narudzbu (GET)**
   Prikazuje popis svih narudžbi.
3. **/stavka_narudzbe (GET)**
   Prikazuje popis stavki u narudžbama.

Rute za berbu:

1. **/berba (GET)**
   Dohvaća podatke o berbi i prikazuje ih na stranici berba.html.
2. **/dodaj_berbu_forma (GET)**
   Prikazuje formu za dodavanje nove berbe.
3. **/dodaj_berbu (POST)**
   Dodaje novu berbu u bazu i preusmjerava na /berba.
4. **/uredi_berbu_forma/int:id (GET)**
   Prikazuje formu za uređivanje odabrane berbe.
5. **/uredi_berbu/int:id (POST)**
   Ažurira podatke o berbi i preusmjerava na /berba.
6. **/uredi_berbu_str (GET)**
   Prikazuje sve berbe za uređivanje na stranici uredi_berbu.html.
7. **/obrisi_berbu/int:id (POST)**
   Briše berbu s danim id i preusmjerava na /uredi_berbu_str.

Rute za skladišta:

1. **/stanje_skladista_vina (GET)**
   Prikazuje stanje skladišta vina.
2. **/stanje_skladista_proizvoda (GET)**
   Prikazuje stanje skladišta proizvoda.
3. **/skladiste_proizvod (GET)**
   Prikazuje ulazne i izlazne transakcije skladišta proizvoda
4. **/stanje_skladista_repromaterijala (GET)**
   Prikazuje stanje skladišta repromaterijala.

Ostale rute:

1. **/kvartalni_pregled_prodaje (GET)**
   Prikazuje kvartalni pregled prodaje.

2. **/transport (GET)**
   Prikazuje popis svih transporta.
3. **/racun (GET)**
   Prikazuje popis svih računa.
4. **/plan_proizvodnje (GET)**
   Prikazuje plan proizvodnje.
5. **/punjenje (GET)**
   Prikazuje popis svih punjenja proizvoda u bazi.
6. **/kupac (GET)**
   Dohvaća sve podatke o kupcima i prikazuje ih na stranici kupac.html.

INTERAKCIJA S BAZOM PODATAKA

cursor: Koristi se za izvođenje SQL naredbi i upita.

fetchall(): Dohvaća sve rezultate SQL upita.

commit(): Sprema promjene u bazu podataka.

close(): Zatvara veze s bazom nakon upotrebe.

PORUKE I REDIREKCIJA

redirect(url_for('route_name')): Preusmjerava korisnika na drugu rutu nakon izvršene akcije.

RENDERIRANJE HTML PREDLOŽAKA

render_template: Korišteno za prikaz HTML stranica, s mogućnošću slanja podataka iz Pythona u predložak.

POKRETANJE APLIKACIJE

Python app.py