

Primer 71 – Kodiranje od dna ka vrhu

*Kako biste kreirali SQL upit koji implementira pristup kodiranja od dna ka vrhu za optimizaciju skladištenih procedura u bazi podataka **SkladišteDB**, pretpostavljajući da baza podataka sadrži tabelu **Zalihe** sa informacijama o artiklima u skladištu, uključujući šifru artikla, naziv artikla, trenutnu količinu na zalihama, i minimalnu potrebnu količinu, i kako biste osigurali da se identifikuju artikli čija je trenutna količina ispod minimalne potrebne, uzimajući u obzir potrebu za efikasnim izvršenjem upita i minimiziranjem opterećenja na bazi podataka?*

SQLgpt: Za kreiranje SQL upita koji implementira kodiranje od dna ka vrhu za optimizaciju skladištenih procedura u bazi podataka **SkladišteDB**, posebno fokusirajući se na identifikaciju artikala čija je trenutna količina ispod minimalne potrebne, može se koristiti sledeći pristup:

```
SELECT
    ID_artikla,
    NazivArtikla,
    TrenutnaKolicina,
    MinimalnaPotrebnaKolicina
FROM
    Zalihe
WHERE
    TrenutnaKolicina < MinimalnaPotrebnaKolicina;
```

Objašnjenje SQL upita:

1. **Fokus na specifičan problem:** Pristup od dna ka vrhu počinje od rešavanja specifičnih, pojedinačnih problema. U ovom slučaju, cilj je identifikovati artikle čija je trenutna količina ispod minimalne potrebne.
2. **Efikasno izvršenje upita:** Upit je dizajniran da bude što jednostavniji, smanjujući složenost i potrebu za resursima, čime se povećava efikasnost izvršenja. Ovo je ključno u pristupu od dna ka vrhu, gde se pažnja posvećuje osiguranju da svaki element baze podataka radi efikasno pre nego što se poveže u veće strukture.

3. **Direktno uslovljavanje: WHERE** klauzula direktno uslovljava selekciju artikala baziranih na količini, što olakšava razumevanje i održavanje upita, kao i optimizaciju performansi.
4. **Minimiziranje opterećenja na bazi podataka:** Ograničavanjem upita na bitne kriterijume i izbegavanjem nepotrebnih operacija (kao što su nepotrebne JOIN operacije ili podupiti), minimizira se opterećenje na sistem baze podataka.

Korišćenjem ovog upita, administratori skladišta ili menadžeri lanca snabdevanja mogu brzo identifikovati i reagovati na situacije gde su zalihe određenih artikala ispod željenih nivoa, što omogućava pravovremeno naručivanje ili preduzimanje drugih koraka za osiguranje dostupnosti artikala. Ovaj pristup je u skladu sa metodologijom od dna ka vrhu, gde se pažnja posvećuje detaljima i pojedinačnim komponentama sistema pre nego što se razmotri njihovo povezivanje u složenije strukture.

Primer 72 – Menjanje strukture tabele

*Kako biste kreirali SQL upit koji menja strukturu tabele **Zaposleni** u bazi podataka **KompanijaDB**, pretpostavljajući da trenutno tabela **Zaposleni** sadrži kolone za šifru zaposlenog, ime, prezime, i platu, a potrebno je dodati novu kolonu **Email** za kontakt informacije, kao i izmeniti tip podatka kolone **Plata** sa **INTEGER** na **DECIMAL** za preciznije predstavljanje plata sa realnim brojevima, i kako biste osigurali da ove promene ne naruše postojeće podatke i integritet baze podataka?*

SQLgpt: Za kreiranje SQL upita koji menja strukturu tabele **Zaposleni** u bazi podataka **KompanijaDB** u skladu sa postavljenim zahtevima, možete koristiti sledeći pristup:

```
BEGIN TRANSACTION;

-- Dodavanje nove kolone 'Email'
ALTER TABLE KompanijaDB.Zaposleni
ADD Email VARCHAR(255);

-- Menjanje tipa podatka za kolonu 'Plata'
ALTER TABLE KompanijaDB.Zaposleni
ALTER COLUMN Plata DECIMAL(10, 2);

COMMIT TRANSACTION;
```

Objašnjenje SQL upita:

1. **Korišćenje transakcija:** **BEGIN TRANSACTION;** i **COMMIT TRANSACTION;** obuhvataju izmene u strukturi tabele, osiguravajući da se sve izmene ili primene zajedno ili da se nijedna ne primeni u slučaju greške. Ovo pomaže u očuvanju integriteta baze podataka.
2. **Dodavanje nove kolone:** **ALTER TABLE ... ADD** naredba dodaje novu kolonu **Email** u tabelu **Zaposleni**. **VARCHAR(255)** je odabran kao tip podatka za kolonu **Email** kako bi se omogućilo skladištenje standardnih imejl adresa.
3. **Izmena tipa podatka postojeće kolone:** **ALTER TABLE ... ALTER COLUMN** naredba menja tip podatka za kolonu **Plata** iz **INTEGER** u **DECIMAL(10, 2)**. Ovaj tip podatka omogućava skladištenje realnih vrednosti sa dve cifre posle decimalnog zareza, što omogućava preciznije predstavljanje plata.
4. Ako dođe do bilo kakve greške ili problema tokom transakcije, može se izvršiti **ROLLBACK** da bi se poništile sve izmene.

Ovaj upit osigurava pažljivo i kontrolisano menjanje strukture tabele **Zaposleni**, uvodeći potrebne izmene bez narušavanja postojećih podataka. Korišćenjem transakcija, promene se mogu bezbedno primeniti, smanjujući rizik od grešaka ili gubitka podataka tokom procesa.

Primer 73 – Dizajniranje korisničkog interfejsa

*Kako biste dizajnirali SQL upit koji omogućava korisnički interfejs aplikacije za upravljanje ljudskim resursima da dinamički prikaže informacije o zaposlenima iz tabele **Zaposleni** u bazi podataka **HRDB**, pretpostavljajući da tabela **Zaposleni** sadrži kolone za šifru zaposlenog, ime, prezime, poziciju, odsek, datum zaposlenja, i platu, i kako biste omogućili korisnicima da filtriraju i sortiraju podatke na osnovu različitih kriterijuma kao što su odsek, pozicija, i datum zaposlenja, kao i da vrše pretragu po imenu i prezimenu?*

SQLgpt: Za dizajniranje SQL upita koji omogućava korisnički interfejs aplikacije za upravljanje ljudskim resursima da dinamički prikaže

informacije o zaposlenima iz tabele **Zaposleni** u bazi podataka **HRDB**, mogli biste koristiti sledeći pristup:

```
SELECT
    ID_zaposlenog,
    Ime,
    Prezime,
    Pozicija,
    Odsek,
    DatumZaposlenja,
    Plata
FROM
    HRDB.Zaposleni
WHERE
    (Ime LIKE '%'+ @PretragaIme + '%'
     OR Prezime LIKE '%'+ @PretragaPrezime + '%')
    AND Odsek = COALESCE(@Odsek, Odsek)
    AND Pozicija = COALESCE(@Pozicija, Pozicija)
ORDER BY
    CASE WHEN @Sortiranje = 'Odsek' THEN Odsek END,
    CASE WHEN @Sortiranje = 'Pozicija' THEN Pozicija END,
    CASE WHEN @Sortiranje = 'DatumZaposlenja' THEN
DatumZaposlenja END;
```

Objašnjenje SQL upita:

1. **Dinamički prikaz informacija:** Osnovni deo ovog upita koristi **SELECT** naredbu za izdvajanje osnovnih informacija o zaposlenima, uključujući šifru zaposlenog, ime, prezime, poziciju, odsek, datum zaposlenja i platu. Ovo omogućava korisničkom interfejsu da dinamički prikaže detalje o zaposlenima.
2. **Filtriranje na osnovu kriterijuma:** **WHERE** klauzula koristi kombinaciju **LIKE** operatora za pretragu po imenu i prezimenu, i **COALESCE** funkciju za omogućavanje filtriranja na osnovu odseka i pozicije. **COALESCE** funkcija omogućava da se, ako nisu navedeni specifični filteri za odsek ili poziciju, koriste sve vrednosti. Na primer, **COALESCE(@Odsek, Odsek)** znači da će se, ako korisnik nije specificirao vrednost za odsek **@Odsek**, koristiti vrednost iz kolone **Odsek**, omogućavajući tako širok spektar filtera..
3. **Sortiranje na osnovu korisničkog izbora:** **ORDER BY** klauzula sa **CASE** izrazima omogućava dinamičko sortiranje rezultata na osnovu korisničkog izbora, bilo da je to po odseku, poziciji ili datumu zaposlenja. Svaki **CASE** izraz predstavlja jedan od ovih

kriterijuma, a korisnički izbor se proverava pomoću **@SortirajPo** promenljive. Ovo omogućava korisnicima da prilagode prikaz podataka prema svojim potrebama.

Ovaj upit podržava razvoj fleksibilnog korisničkog interfejsa za aplikaciju za upravljanje ljudskim resursima, omogućavajući korisnicima da efikasno pretražuju, filtriraju i sortiraju podatke o zaposlenima, čime se poboljšava korisničko iskustvo i omogućava lako pristupanje informacijama.

Primer 74 – XML izveštaj

*Kako biste implementirali SQL upit koji generiše XML izveštaj o trenutnim projektima unutar baze podataka **ProjektiDB**, pretpostavljajući da tabela **Projekti** sadrži kolone za šifru projekta, naziv projekta, datum početka, datum završetka, i status projekta, i kako biste omogućili da se XML izveštaj dinamički prilagođava za prikazivanje projekata u određenim fazama (npr. u toku, završeni), omogućavajući korisnicima da dobiju strukturirani pregled projekata sa relevantnim detaljima?*

SQLgpt: Za implementaciju SQL upita koji generiše XML izveštaj o trenutnim projektima u bazi podataka **ProjektiDB** i omogućava dinamičko prilagođavanje za prikazivanje projekata u određenim fazama, možete koristiti sledeći pristup:

```
SELECT
    ID_projekta AS "@ID",
    NazivProjekta AS "Naziv",
    DatumPočetka AS "Početak",
    DatumZavršetka AS "Završetak",
    StatusProjekta AS "Status"
FROM
    ProjektiDB.Projekti
WHERE
    StatusProjekta = @StatusProjekta
FOR XML PATH('Projekat'), ROOT('Projekti'), ELEMENTS;
```

Objašnjenje SQL upita:

1. **Generisanje XML strukture:** **FOR XML PATH** omogućava definisanje strukture XML dokumenta, gde **PATH('Projekat')** određuje da će svaki red biti predstavljen kao **<Projekat>** element unutar krajnjeg XML dokumenta. **ROOT('Projekti')** dodaje

korenski element **<Projekti>** koji obuhvata sve **<Projekat>** elemente, dajući strukturu celokupnom XML dokumentu.

2. **Dinamičko filtriranje: WHERE StatusProjekta = @StatusProjekta** omogućava korisnicima da dinamički filtriraju projekte na osnovu njihovog statusa, kao što su „u toku” ili „završeni”. Ovo omogućava fleksibilnost u generisanju izveštaja zasnovanih na trenutnim potrebama analize.
3. **Definisanje elemenata XML dokumenta:** Pseudonimi kolona (npr. **AS "@ID"**, **AS "Naziv"**) koriste se za definisanje kako će se kolone mapirati na elemente ili attribute unutar XML dokumenta. Na primer, **@ID** označava atribut, dok ostali pseudonimi predstavljaju podelemente unutar svakog **<Projekat>** elementa.
4. **ELEMENTS opcija: ELEMENTS** ključna reč uz **FOR XML PATH** naredbu specificira da kolone treba da budu predstavljene kao podelementi unutar svakog **<Projekat>** elementa, pružajući jasnu i čitljivu strukturu XML dokumenta.

Ovaj upit omogućava efikasno kreiranje detaljnih XML izveštaja o projektima unutar **ProjektiDB**, pružajući korisnicima strukturirani i lako prilagodljivi način za pregledanje podataka o projektima, što je posebno korisno za integraciju sa drugim sistemima ili za dalju analizu i obradu u alatima koji podržavaju XML.

Primer 74 – Hijerarhijski model maloprodajne baze podataka

*Kako biste kreirali hijerarhijski model prodajne baze podataka za maloprodajni biznis u SQL jeziku, pretpostavljajući da baza podataka **MaloprodajaDB** treba da obuhvati tabele za **Prodavnice**, **Zaposlene**, **Proizvode**, **KategorijeProizvoda**, **Narudžbe** i **StavkeNarudžbe**, i kako biste osigurali da model podržava praćenje prodaja po prodavnicama, upravljanje zalihama, identifikaciju zaposlenih i njihovih uloga, kao i detaljnu evidenciju narudžbi i stavki unutar narudžbi, uz očuvanje integriteta podataka i omogućavanje efikasne analize prodajnih podataka?*

SQLgpt: Za kreiranje hijerarhijskog modela prodajne baze podataka za maloprodajni biznis u SQL jeziku, možete koristiti sledeći pristup:

Definisanje tabela:

- **Prodavnice:** Sadrži informacije o svakoj prodavnici, uključujući šifru prodavnice, naziv, lokaciju i šifru menadžera.

```
CREATE TABLE Prodavnice (  
    ID_prodavnice INT PRIMARY KEY,  
    Naziv VARCHAR(100),  
    Lokacija VARCHAR(255),  
    ID_menadžera INT,  
    FOREIGN KEY (ID_menadžera)  
    REFERENCES Zaposleni(ID_zaposlenog)  
);
```

Zaposleni: Sadrži detalje o zaposlenima, uključujući šifru zaposlenog, ime, prezime, uloga, i šifru prodavnice gde su zaposleni.

```
CREATE TABLE Zaposleni (  
    ID_zaposlenog INT PRIMARY KEY,  
    Ime VARCHAR(50),  
    Prezime VARCHAR(50),  
    Uloga VARCHAR(50),  
    ID_prodavnice INT,  
    FOREIGN KEY (ID_prodavnice)  
    REFERENCES Prodavnice(ID_prodavnice)  
);
```

KategorijeProizvoda: Definiše kategorije proizvoda, uključujući šifru kategorije i naziv kategorije.

```
CREATE TABLE KategorijeProizvoda (  
    ID_kategorije INT PRIMARY KEY,  
    NazivKategorije VARCHAR(100)  
);
```

Proizvodi: Sadrži informacije o proizvodima, uključujući šifru proizvoda, naziv, cenu, količinu na zalihama, i šifru kategorije.

```
CREATE TABLE Proizvodi (  
    ID_proizvoda INT PRIMARY KEY,  
    NazivProizvoda VARCHAR(100),  
    Cena DECIMAL(10, 2),  
    KoličinaNaZalihama INT,  
    ID_kategorije INT,  
    FOREIGN KEY (ID_kategorije)  
    REFERENCES KategorijeProizvoda(ID_kategorije)  
);
```

Narudžbe: Evidencija narudžbina uključujući šifre narudžbine, datum narudžbine, šifru zaposlenog koji je obradio narudžbu, i šifru prodavnice gde je narudžba napravljena.

```
CREATE TABLE Narudzbine (  
    ID_narudzbine INT PRIMARY KEY,  
    DatumNarudzbine DATETIME,  
    ID_zaposlenog INT,  
    ID_prodavnice INT,  
    FOREIGN KEY (ID_zaposlenog)  
    REFERENCES Zaposleni(ID_zaposlenog),  
    FOREIGN KEY (ID_prodavnice)  
    REFERENCES Prodavnice(ID_prodavnice)  
);
```

StavkeNarudžbine: Detalji stavki unutar svake narudžbe, uključujući šifru stavke, šifru narudžbe, šifru proizvoda, količinu i cenu po stavci.

```
CREATE TABLE StavkeNarudzbine (  
    ID_stavke INT PRIMARY KEY,  
    ID_narudzbine INT,  
    ID_proizvoda INT,  
    Kolicina INT,  
    CenaPoStavci DECIMAL(10, 2),  
    FOREIGN KEY (ID_narudzbe)  
    REFERENCES Narudzbine(ID_narudzbine),  
    FOREIGN KEY (ID_proizvoda)  
    REFERENCES Proizvodi(ID_proizvoda)  
);
```

1. **Hijerarhijski odnosi:** Model uspostavlja odnose između prodavnica i zaposlenih (zaposleni rade u prodavnicama), proizvoda i kategorija (proizvodi pripadaju kategorijama), narudžbi, prodavnica i zaposlenih (zaposleni u prodavnicama obrađuju narudžbe), kao i odnos između narudžbi i stavki narudžbi (narudžbe sadrže stavke).
2. **Očuvanje integriteta podataka:** Upotrebom stranih ključeva (**FOREIGN KEY**) osigurava se referencijalni integritet, tako da sve reference između tabela moraju pokazivati na postojeće zapise.

Ovaj hijerarhijski model baze podataka omogućava maloprodajnom biznisu da efikasno upravlja zalihama, zaposlenima, i narudžbama, dok istovremeno pruža snažnu osnovu za analizu prodajnih podataka, omogućavajući dublje uvide u operacije i performanse.

Primer 75 – Mrežni model transakcija maloprodajnog objekta

*Kako biste kreirali mrežni model transakcija za online prodavnicu u SQL jeziku, pretpostavljajući da baza podataka **OnlineProdavnicaDB** treba da podržava kompleksne odnose između **Korisnika**, **Proizvoda**, **Narudžbi**, **StavkiNarudžbe** i **RecenzijaProizvoda**, i kako biste osigurali da model omogućava praćenje korisničkih narudžbi, detaljnu evidenciju stavki unutar narudžbi, veze između korisnika i njihovih recenzija proizvoda, kao i povezivanje proizvoda sa narudžbama i recenzijama, uzimajući u obzir potrebu za efikasnom obradom i analizom podataka?*

1. Definisane tabele:

Korisnici: Sadrži informacije o registrovanim korisnicima, uključujući šifru korisnika, korisničko ime, imejl adresu i lozinku.

```
CREATE TABLE Korisnici (  
    ID_korisnika INT PRIMARY KEY,  
    KorisničkoIme VARCHAR(50),  
    Email VARCHAR(100),  
    Lozinka VARCHAR(50)  
);
```

Proizvodi: Sadrži detalje o proizvodima, uključujući šifru proizvoda, naziv, opis, i cenu.

```
CREATE TABLE Proizvodi (  
    ID_proizvoda INT PRIMARY KEY,  
    Naziv VARCHAR(100),  
    Opis TEXT,  
    Cena DECIMAL(10, 2)  
);
```

Narudžbe: Evidencija narudžbi uključujući šifru narudžbine, šifru korisnika koji je napravio narudžbu, datum narudžbe, I ukupan iznos.

```
CREATE TABLE Narudžbine (  
    ID_narudžbine INT PRIMARY KEY,  
    ID_korisnika INT,  
    DatumNarudžbine DATETIME,  
    UkupanIznos DECIMAL(10, 2),  
    FOREIGN KEY (ID_korisnika)  
    REFERENCES Korisnici(ID_korisnika)  
);
```

SQLgpt složenije pitanje: Kako biste kreirali mrežni model transakcija za online prodavnicu u SQL jeziku, pretpostavljajući da baza podataka **OnlineProdavnicaDB** treba da podržava kompleksne odnose između **Korisnika**, **Proizvoda**, **Narudžbi**, **StavkiNarudžbe** i **RecenzijaProizvoda**, i kako biste osigurali da model omogućava praćenje korisničkih narudžbi, detaljnu evidenciju stavki unutar narudžbi, veze između korisnika i njihovih recenzija proizvoda, kao i povezivanje proizvoda sa narudžbama i recenzijama, uzimajući u obzir potrebu za efikasnom obradom i analizom podataka?

1. Definisane tabele:

Korisnici: Sadrži informacije o registrovanim korisnicima, uključujući šifru korisnika, korisničko ime, imejl adresu i lozinku.

```
CREATE TABLE Korisnici (  
    ID_korisnika INT PRIMARY KEY,  
    KorisničkoIme VARCHAR(50),  
    Email VARCHAR(100),  
    Lozinka VARCHAR(50)  
);
```

Proizvodi: Sadrži detalje o proizvodima, uključujući šifru proizvoda, naziv, opis, i cenu.

```
CREATE TABLE Proizvodi (  
    ID_proizvoda INT PRIMARY KEY,  
    Naziv VARCHAR(100),  
    Opis TEXT,  
    Cena DECIMAL(10, 2)  
);
```

Narudžbine: Evidencija narudžbina uključujući šifru narudžbine, šifru korisnika koji je napravio narudžbinu, datum narudžbine, i ukupan iznos.

```
CREATE TABLE Narudžbine (  
    ID_narudžbine INT PRIMARY KEY,  
    ID_korisnika INT,  
    DatumNarudžbine DATETIME,  
    UkupanIznos DECIMAL(10, 2),  
    FOREIGN KEY (ID_korisnika)  
    REFERENCES Korisnici(ID_korisnika)  
);
```

StavkeNarudžbine: Detalji stavki unutar svake narudžbe, uključujući šifru stavke, šifru narudžbe, šifru proizvoda, količinu, i cenu po stavci.

```
CREATE TABLE StavkeNarudžbine (  
    ID_stavke INT PRIMARY KEY,
```

```

    ID_narudzbine INT,
    ID_proizvoda INT,
    Količina INT,
    CenaPoStavci DECIMAL(10, 2),
    FOREIGN KEY (ID_narudzbine)
    REFERENCES Narudžbe(ID_narudzbine),
    FOREIGN KEY (ID_proizvoda)
    REFERENCES Proizvodi(ID_proizvoda)
);

```

RecenzijeProizvoda: Sadrži recenzije koje korisnici ostavljaju za proizvode, uključujući šifru recenzije, šifru proizvoda, šifru korisnika koji je ostavio recenziju, tekst recenzije, i ocenu.

```

CREATE TABLE RecenzijeProizvoda (
    ID_recenzije INT PRIMARY KEY,
    ID_proizvoda INT,
    ID_korisnika INT,
    TekstRecenzije TEXT,
    Ocjena INT,
    FOREIGN KEY (ID_proizvoda)
    REFERENCES Proizvodi(ID_proizvoda),
    FOREIGN KEY (ID_korisnika)
    REFERENCES Korisnici(ID_korisnika)
);

```

Mrežni odnosi:

- **Korisnici - Narudžbe:** Jedan korisnik može imati više narudžbi, što stvara „jedan prema više” odnos između tabela **Korisnici** i **Narudžbe**.
- **Narudžbe - StavkeNarudžbe:** Jedna narudžba može sadržati više stavki, čime se formira „jedan prema više” odnos između **Narudžbe** i **StavkeNarudžbe**.
- **Proizvodi - StavkeNarudžbe:** Proizvodi se povezuju sa narudžbama preko stavki narudžbe, što omogućava praćenje koji su proizvodi naručeni.
- **Korisnici - RecenzijeProizvoda:** Korisnici mogu ostaviti više recenzija, što stvara „jedan prema više” odnos između tabela **Korisnici** i **RecenzijeProizvoda**.
- **Proizvodi - RecenzijeProizvoda:** Proizvodi mogu imati više recenzija, čime se formira „jedan prema više” odnos između tabela **Proizvodi** i **RecenzijeProizvoda**.

Ovaj mrežni model omogućava online prodavnici da efikasno upravlja korisnicima, narudžbama, proizvodima i recenzijama, pružajući bogatu osnovu za analizu kupovnih navika, popularnosti proizvoda, i zadovoljstva korisnika. Kompleksni odnosi između entiteta omogućavaju detaljnu analizu i izveštavanje, što doprinosi boljem razumevanju dinamike prodaje i poboljšanju korisničkog iskustva.

Primer 76 – Prikaz podataka na različite načine

*Razmotrite tvrdnju „Podaci se mogu predstaviti samo na jedan način, kao vrednosti kolona unutar redova tabele” u kontekstu relacionih baza podataka. Kako biste konstruisali SQL upit koji ilustruje mogućnost predstavljanja podataka na više načina unutar iste relacione baze podataka **VišestrukiPrikaziDB**, pretpostavljajući da baza podataka sadrži tabelu **Zaposleni** sa standardnim informacijama (šifra zaposlenog, ime, prezime, odeljenje, plata), i kako biste demonstrirali alternativne pristupe prikazu podataka, kao što su agregacija, pivotiranje i korišćenje podupita, omogućavajući korisnicima da dobiju različite uvide i perspektive iz istog skupa podataka?*

SQLgpt: Evo kako bismo konstruisali SQL upit koji ilustruje ove mogućnosti unutar baze podataka **VišestrukiPrikaziDB**:

Agregacija

Agregacija podataka može se koristiti za sumiranje, računanje proseka, ili pronalaženje minimalnih i maksimalnih vrednosti unutar grupe podataka. Na primer, da bismo dobili prosečnu platu po odeljenju unutar tabele **Zaposleni** možemo da koristimo sledeći upit:

-- Agregacija plata po odeljenjima

```
SELECT
    Odeljenje,
    AVG(Plata) AS ProsečnaPlata,
    MAX(Plata) AS NajvećaPlata,
    MIN(Plata) AS NajmanjaPlata
FROM
    VišestrukiPrikaziDB.Zaposleni
GROUP BY
    Odeljenje;
```

Ovaj upit demonstrira kako se podaci mogu agregirati da bi se dobile prosečne, maksimalne i minimalne plate po odeljenjima, pružajući uvid u raspodelu plata unutar kompanije.

Pivotiranje podataka:

Zamislite da želite da prikazete podatke tako da redovi postanu kolone. U SQL-u, to možete uraditi kroz pivot operaciju, ali SQL standardno ne podržava direktnu PIVOT sintaksu kao neka proširenja jezika, kao što je T-SQL jezik. Međutim, možete koristiti CASE izraze i agregatne funkcije da biste postigli sličan rezultat.

-- Pivotiranje podataka za prikaz broja zaposlenih po odeljenjima

```
SELECT
    Odeljenje,
    SUM(CASE WHEN Plata < 50000 THEN 1 ELSE 0 END) AS
'Ispod_50000',
    SUM(CASE WHEN Plata BETWEEN 50000 AND 100000 THEN 1 ELSE
0 END) AS 'Između_50000_i_100000',
    SUM(CASE WHEN Plata > 100000 THEN 1 ELSE 0 END) AS
'Iznad_100000'
FROM
    VišestrukiPrikaziDB.Zaposleni
GROUP BY
    Odeljenje;
```

Ovaj upit transformiše podatke tako da prikazuje broj zaposlenih u različitim opsezima plata unutar svakog odeljenja, što omogućava laku analizu strukture plata po odeljenjima.

Korišćenje podupita:

-- Prikazivanje zaposlenih koji zarađuju više od prosečne plate u svom odeljenju

```
SELECT
    e.Ime,
    e.Prezime,
    e.Odeljenje,
    e.Plata
FROM
    VišestrukiPrikaziDB.Zaposleni e
WHERE
    e.Plata > (
        SELECT AVG(Plata)
        FROM VišestrukiPrikaziDB.Zaposleni
        WHERE Odeljenje = e.Odeljenje
```

);

Ovaj upit koristi podupit da identifikuje zaposlene koji zarađuju više od prosečne plate unutar svog odeljenja, pružajući perspektivu o platama zaposlenih u odnosu na njihove kolege.

Svaki od ovih upita pokazuje različite načine reprezentacije i analize podataka unutar istog skupa podataka u tabeli **Zaposleni**, opovrgavajući tvrdnju da se podaci mogu predstaviti samo na jedan način. Ovim pristupima se omogućava fleksibilnost u analizi podataka i izvlačenju različitih vrsta informacija i uvida iz istih podataka.

Primer 76 – Odnos jedan prema više

*Kako biste kreirali SQL upit koji ilustruje „jedan prema više” odnos između tabele **Osoba** i tabele **Karta** u kontekstu baze podataka **BioskopDB**, pretpostavljajući da tabela **Osoba** sadrži informacije o pojedinačnim osobama (šifru osobe, ime, prezime) i da tabela **Karta** sadrži detalje o kartama koje su kupili posetioci (šifra karte, šifra osobe, naslov filma, datum projekcije), i kako biste omogućili da se za svaku osobu prikaže lista kupljenih karata, uzimajući u obzir potrebu za efikasnim povezivanjem ovih entiteta i pružanjem pregleda kupovina karata za svaku osobu?*

SQLgpt:

Definisanje tabela:

Osoba: Sadrži informacije o posetiocima bioskopa.

```
CREATE TABLE Osoba (
    ID_osobe INT PRIMARY KEY,
    Ime VARCHAR(50),
    Prezime VARCHAR(50)
);
```

Karta: Sadrži detalje o kartama koje su kupili posetioci.

```
CREATE TABLE Karta (
    ID_karte INT PRIMARY KEY,
    ID_osobe INT,
    NaslovFilma VARCHAR(100),
    DatumProjekcije DATETIME,
    FOREIGN KEY (ID_osobe) REFERENCES Osoba(ID_osobe)
);
```

Odnos „Jedan prema mnogo”

Odnos između tabela **Osoba** i **Karta** je „jedan prema više”, gde jedna osoba može kupiti više karata, ali svaka karta pripada samo jednoj osobi.

SQL upit za prikazivanje liste karata po osobi:

```
SELECT
    O.Ime,
    O.Prezime,
    K.NaslovFilma,
    K.DatumProjekcije
FROM
    Osoba O
JOIN
    Karta K ON O.ID_osobe = K.ID_osobe
ORDER BY
    O.Prezime, O.Ime, K.DatumProjekcije;
```

Ovaj upit koristi **JOIN** operaciju da poveže tabele **Osoba** i **Karta** na osnovu kolone **ID_osobe**, što omogućava da se za svaku osobu izlistaju sve kupljene karte. Sortiranje po prezimenu i imenu osobe, kao i po datumu projekcije, omogućava pregledan i organizovan prikaz podataka.

Ovaj pristup omogućava efikasno povezivanje entiteta **Osoba** i **Karta** unutar baze podataka **BioskopDB**, pružajući jasan i detaljan pregled kupovina karata za svaku osobu, što može biti korisno za analizu kupovnih navika posetilaca ili za personalizovane marketinške kampanje.

Primer 77 – Odnos jedan prema jedan

*Kako biste kreirali SQL upit koji ilustruje „jedan prema jedan” odnos između tabele **Osoba** i tabele **Dozvola** u kontekstu baze podataka **RegistarVozača**, pretpostavljajući da tabela **Osoba** sadrži osnovne informacije o pojedincima (šifra osobe, ime, prezime) i da tabela **Dozvola** sadrži informacije o vozačkim dozvolama (šifra dozvole, šifra osobe, broj dozvole, datum izdavanja, datum isteka), i kako biste osigurali da svaka osoba može imati samo jednu vozačku dozvolu, dok svaka vozačka dozvola pripada tačno jednoj osobi, omogućavajući jasan pregled vozačkih dozvola i njihovih vlasnika?*

Definisanje tabela:

Osoba: Sadrži informacije o pojedincima.

```
CREATE TABLE Osoba (
    ID_osobe INT PRIMARY KEY,
```

```
    Ime VARCHAR(50),
    Prezime VARCHAR(50)
);
```

Dozvola: Sadrži detalje o vozačkim dozvolama pojedinaca.

```
CREATE TABLE Dozvola (
    ID_dozvole INT PRIMARY KEY,
    ID_osobe INT UNIQUE,
    BrojDozvole VARCHAR(50),
    DatumIzdavanja DATE,
    DatumIsteka DATE,
    FOREIGN KEY (ID_osobe) REFERENCES Osoba (ID_osobe)
);
```

Odnos „jedan prema jedan”

Odnos između tabela **Osoba** i **Dozvola** je „jedan prema jedan”, gde svaka osoba može imati najviše jednu vozačku dozvolu i svaka vozačka dozvola je jedinstveno povezana sa jednom osobom. Ovo se osigurava korišćenjem **UNIQUE** ograničenja na **ID_osobe** u tabeli **Dozvola**.

SQL upit za prikazivanje osoba sa njihovim vozačkim dozvolama:

```
SELECT
    O.Ime,
    O.Prezime,
    D.BrojDozvole,
    D.DatumIzdavanja,
    D.DatumIsteka
FROM
    Osoba O
JOIN
    Dozvola D ON O.ID_osobe = D.ID_osobe;
```

Ovaj upit koristi **JOIN** operaciju da poveže tabele **Osoba** i **Dozvola** na osnovu **ID_osobe**, što omogućava da se za svaku osobu prikaže njena vozačka dozvola. Prikazivanje imena i prezimena osobe zajedno sa detaljima o njenoj vozačkoj dozvoli pruža jasan pregled odnosa između pojedinaca i njihovih dozvola unutar baze podataka **RegistarVozača**.

Ovaj pristup omogućava efikasno upravljanje i pregled informacija o vozačkim dozvolama unutar **RegistarVozača** baze podataka, osiguravajući da svaka osoba ima jedinstvenu vozačku dozvolu i da svaka dozvola pripada samo jednoj osobi, što je ključno za održavanje integriteta podataka i preciznost evidencije.

Primer 78 – Odnos više prema više

*Kako biste kreirali SQL upit koji ilustruje „više prema više” odnos između tabele **Student** i tabele **Kurs** u kontekstu baze podataka **AkadskaBaza**, pretpostavljajući da tabela **Student** sadrži podatke o studentima (šifra studenta, ime, prezime) i da tabela **Kurs** sadrži informacije o dostupnim kursevima (šifra kursa, naziv kursa, opis), te da postoji pomoćna tabela **Upis** koja povezuje studente sa kursevima (šifra upisa, šifra studenta, šifra kursa), i kako biste omogućili pregled koji studenti su upisani na koje kurseve, kao i koji kursevi imaju kojeg studenta, čime se efikasno upravlja evidencijom upisa i promoviše transparentnost u akademskoj zajednici?*

SQLgpt:

Definisanje tabela:

Student: Sadrži osnovne informacije o studentima.

```
CREATE TABLE Student (  
    ID_studenta INT PRIMARY KEY,  
    Ime VARCHAR(50),  
    Prezime VARCHAR(50)  
);
```

Kurs: Sadrži detalje o kursevima koji se nude.

```
CREATE TABLE Kurs (  
    ID_kursa INT PRIMARY KEY,  
    NazivKursa VARCHAR(100),  
    Opis TEXT  
);
```

Upis: Pomoćna tabela koja povezuje tabele **Student** i **Kurs**, omogućavajući „više prema više” odnos.

```
CREATE TABLE Upis (  
    ID_upisa INT PRIMARY KEY,  
    ID_studenta INT,  
    ID_kursa INT,  
    FOREIGN KEY (ID_studenta)  
    REFERENCES Student(ID_studenta),  
    FOREIGN KEY (ID_kursa) REFERENCES Kurs(ID_kursa)  
);
```

Odnos „više prema više”

Odnos između tabela **Student** i **Kurs** je „više prema više”, gde jedan student može biti upisan na više kurseva, a jedan kurs može imati više upisanih studenata. Ovo se postiže kroz pomoćnu tabelu **Upis**.

SQL upit za prikazivanje studenata po kursu:

```
SELECT
    K.NazivKursa,
    S.Ime,
    S.Prezime
FROM
    Kurs K
JOIN
    Upis U ON K.ID_kursa = U.ID_kursa
JOIN
    Student S ON U.ID_studenta = S.ID_studenta
ORDER BY
    K.NazivKursa, S.Prezime, S.Ime;
```

Ovaj upit koristi dve **JOIN** operacije da poveže tabele **Kurs**, **Upis**, i **Student**, omogućavajući prikaz koji studenti su upisani na svaki kurs. Sortiranje po nazivu kursa, kao i po prezimenu i imenu studenta, pruža jasnu i organizovanu listu upisa.

SQL upit za prikazivanje kurseva po studentu:

```
SELECT
    S.Ime,
    S.Prezime,
    K.NazivKursa
FROM
    Student S
JOIN
    Upis U ON S.ID_studenta = U.ID_studenta
JOIN
    Kurs K ON U.ID_kursa = K.ID_kursa
ORDER BY
    S.Prezime, S.Ime, K.NazivKursa;
```

Ovaj upit radi na sličan način kao prethodni, ali fokusira se na prikaz kurseva na koje je svaki student upisan, sortiran po studentima i zatim po kursu.

Ovi upiti omogućavaju efikasno upravljanje i pregled upisa u bazi podataka **AkademskiBaza**, pružajući ključne informacije za studente, nastavnike i administraciju, čime se promovira transparentnost i olakšava planiranje i praćenje akademskog napretka.

Primer 79 – Normalizacija podataka

*Kako biste primenili proces normalizacije podataka na postojeću bazu podataka **FirmaDB** koja sadrži tabelu **Zaposleni** sa kolonama za ime, prezime, odeljenje, poziciju, direktan broj telefona, imejl adresu, i adresu stanovanja, s ciljem poboljšanja strukture baze, eliminisanja redundancije, i osiguranja integriteta podataka, te kako biste definisali korake normalizacije od prve do treće normalne forme (1NF, 2NF, 3NF), i kreirali SQL upite za restrukturiranje baze podataka **FirmaDB** baze podataka u skladu sa principima normalizacije?*

SQLgpt:

Prva normalna forma (1NF):

Cilj: Osigurati da kolone tabele imaju samo atomične vrednosti i da ne postoje viševrednosni atributi.

Koraci:

Kolona **Adresa** stanovanja je složeni atribut, te je potrebno je razbiti na više kolona: **Ulica, Broj, Grad, i Poštanski broj**.

Uvesti **ID_zaposlenog** kao primarni ključ.

SQL upit za modifikaciju tabele:

```
ALTER TABLE Zaposleni
ADD ID_zaposlenog INT PRIMARY KEY,
ADD Ulica VARCHAR(100),
ADD Broj VARCHAR(10),
ADD Grad VARCHAR(50),
ADD PoštanskiBroj VARCHAR(10);
```

```
UPDATE Zaposleni
SET Ulica = [ekstrakcija iz 'Adresa stanovanja'],
    Broj = [ekstrakcija iz 'Adresa stanovanja'],
    Grad = [ekstrakcija iz 'Adresa stanovanja'],
    PoštanskiBroj = [ekstrakcija iz 'Adresa stanovanja'];
```

```
ALTER TABLE Zaposleni
DROP COLUMN AdresaStanovanja;
```

Druga normalna forma (2NF):

Cilj: Osigurati da sve neključne informacije zavise od celog primarnog ključa, a ne od nekog njegovog dela ili da uopšte ne zavise.

Koraci:

Izdvojiti **Odeljenje** i **Poziciju** u zasebne tabele jer njihove vrednosti ne zavise direktno od **ID_zaposlenog**.

SQL upit za kreiranje novih tabela:

```
CREATE TABLE Odeljenje (  
    ID_odeljenja INT PRIMARY KEY,  
    NazivOdeljenja VARCHAR(50)  
);  
  
CREATE TABLE Pozicija (  
    ID_pozicije INT PRIMARY KEY,  
    NazivPozicije VARCHAR(50)  
);  
  
-- Ažuriranje tabele 'Zaposleni' da uključi ključeve  
-- novih tabela  
ALTER TABLE Zaposleni  
ADD ID_odeljenja INT,  
ADD ID_pozicije INT;  
  
UPDATE Zaposleni  
SET ID_odeljenja = [odgovarajući ID iz tabele 'Odeljenje'],  
    ID_pozicije = [odgovarajući ID iz tabele 'Pozicija'];  
  
ALTER TABLE Zaposleni  
DROP COLUMN Odeljenje,  
DROP COLUMN Pozicija;
```

Treća normalna forma (3NF):

Cilj: Osigurati da se sve neključne kolone direktno odnose na primarni ključ, a ne na druge neključne kolone. Tačnije, osigurati da ne postoje tranzitivne zavisnost između kolona.

Koraci:

Razmotriti odnose među neključnim kolonama i, ako je potrebno, dodatno izdvojiti tabele.

Primer:

Ako postoje kolone koje zavise od **ID_odeljenja** a ne direktno od **ID_zaposlenog** (npr. lokacija odeljenja), treba kreirati zasebnu tabelu za te informacije.

```
CREATE TABLE LokacijaOdeljenja (  
    ID_odeljenja INT,  
    Lokacija VARCHAR(100),  
    FOREIGN KEY (ID_odeljenja)  
    REFERENCES Odeljenje(ID_odeljenja)  
);
```

Primena ovih koraka normalizacije na bazu podataka **FirmaDB** smanjuje redundanciju podataka, poboljšava integritet i efikasnost baze podataka, i olakšava upravljanje podacima. Ovo omogućava jednostavnije ažuriranje, brže pretrage i bolju skalabilnost sistema.

Primer 80 - Denormalizacija podataka

*Kako biste primenili proces denormalizacije na visoko normalizovanu bazu podataka **WebShopDB** koja sadrži više povezanih tabela poput **Korisnici**, **Proizvodi**, **Narudžbine**, **StavkeNarudžbine**, i **RecenzijeProizvoda**, s ciljem poboljšanja performansi čitanja i upita u scenarijima visokog prometa, te kako biste definisali korake denormalizacije koji bi mogli uključivati reintegraciju određenih tabela, uvođenje redundantnih podataka i promenu šema relacija, i kreirali SQL upite za rekonfiguraciju baze podataka **WebShopDB** baze podataka tako da se optimizuje za brze operacije čitanja, dok se istovremeno osigurava da se održi prihvatljiv nivo integriteta podataka?*

Reintegracija tabela:

Cilj: Smanjiti broj **JOIN** operacija potrebnih za uobičajene upite kombinovanjem tabela koje se često koriste zajedno.

Koraci:

Spajanje table **StavkeNarudžbine** sa tabelom **Narudžbine**, pretpostavljajući da većina upita traži informacije o narudžbinama zajedno sa njihovim stavkama.

SQL upit za reintegraciju:

```
ALTER TABLE Narudžbine  
ADD Stavka VARCHAR(100),
```

```
ADD Količina INT,  
ADD CenaPoJedinici DECIMAL(10,2);  
  
INSERT INTO Narudžbine (Stavka, Količina, CenaPoJedinici)  
SELECT S.NazivProizvoda, S.Količina, S.CenaPoJedinici  
FROM StavkeNarudžbine S  
JOIN Narudžbine N ON S.ID_narudžbine = N.ID;
```

```
DROP TABLE StavkeNarudžbine;
```

Uvođenje redundantnih podataka:

Cilj: Povećati brzinu upita dodavanjem izračunatih ili agregiranih podataka direktno u tabele.

Koraci:

Dodavanje kolone **UkupnaCena** u tabelu **Narudžbine** za skladištenje ukupne cene svake narudžbine.

SQL upit za dodavanje redundantnih podataka:

```
ALTER TABLE Narudžbine  
ADD UkupnaCena DECIMAL(10,2);  
  
UPDATE Narudžbine  
SET UkupnaCena = Količina * CenaPoJedinici;
```

Promena šeme relacija:

Cilj: Umanjiti složenost šeme smanjivanjem broja relacija.

Koraci:

Uklanjanje nekih stranih ključeva i normalizovanih struktura koje nisu često korišćene ili koje uzrokuju složene upite.

Primer:

Ako se recenzije proizvoda retko koriste zajedno s detaljima proizvoda, razmotriti uklanjanje stranog ključa iz **RecenzijeProizvoda** koji ukazuje na **Proizvodi**.

```
ALTER TABLE RecenzijeProizvoda  
DROP FOREIGN KEY FK_RecenzijeProizvoda_Proizvodi;
```

Primena ovih koraka denormalizacije na bazi podataka **WebShopDB** može značajno poboljšati performanse čitanja, posebno u scenarijima visokog prometa, kao što su online prodavnice tokom perioda velikih

rasprodaja. Međutim, važno je pažljivo balansirati između poboljšanja performansi i očuvanja integriteta podataka, redovno procenjujući uticaj denormalizacije na ukupno održavanje i ažuriranje baze podataka.

Primer 81 – Entiteti i veze

*U kontekstu projektovanja baze podataka **BibliotekaDB** za upravljanje bibliotekom, kako biste definisali entitete i veze koji su neophodni za efikasno upravljanje knjigama, članovima, izdavanjem i vraćanjem knjiga, kao i evidencijom autora i izdavača, te kako biste konstruisali SQL upite za kreiranje tabela koje odražavaju ove entitete i veze, osiguravajući da šema baze podataka **BibliotekaDB** omogućava jednostavno dodavanje, ažuriranje i pretragu podataka, dok se istovremeno održava visok nivo organizacije i integriteta podataka?*

SQLgpt:

Definisanje entiteta:

- **Knjiga:** Sadrži informacije o svakoj knjizi u biblioteci.
- **Član:** Sadrži detalje o svakom članu biblioteke.
- **Autor:** Sadrži informacije o autorima knjiga.
- **Izdavač:** Sadrži detalje o izdavačima knjiga.
- **IzdavanjeKnjiga:** Povezuje knjige i članove za evidenciju izdavanja knjiga.

Definisanje veza:

- Knjiga-Autor: „Više prema više” veza, jer jedna knjiga može imati više autora, a jedan autor može imati više knjiga.
- Knjiga-Izdavač: „Jedan prema više” veza, jer jedan izdavač može izdati više knjiga.
- Član-IzdavanjeKnjiga-Knjiga: „Više prema više” veza preko entiteta **IzdavanjeKnjiga**, jer jedan član može iznajmiti više knjiga, a jedna knjiga može biti iznajmljena više puta.

SQL upiti za kreiranje tabela:

Tabela Knjiga:

```
CREATE TABLE Knjiga (  
    ID_knjige INT PRIMARY KEY,  
    Naslov VARCHAR(255),  
    GodinaIzdavanja INT,  
    ID_izdavača INT,  
    FOREIGN KEY (ID_izdavača)  
    REFERENCES Izdavač(ID_izdavača)  
);
```

Tabel Član:

```
CREATE TABLE Član (  
    ID_člana INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    DatumUčlanjenja DATE  
);
```

Tabela Autor:

```
CREATE TABLE Autor (  
    ID_atora INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100)  
);
```

Tabela Izdavač:

```
CREATE TABLE Izdavač (  
    ID_izdavača INT PRIMARY KEY,  
    Naziv VARCHAR(100),  
    Grad VARCHAR(100)  
);
```

Tabela KnjigaAutor (za „više prema više” vezu između **Knjiga** i **Autor**):

```
CREATE TABLE KnjigaAutor (  
    ID_knjige INT,  
    ID_atora INT,  
    PRIMARY KEY (ID_knjige, ID_atora),  
    FOREIGN KEY (ID_knjige) REFERENCES Knjiga(ID_knjige),  
    FOREIGN KEY (ID_atora) REFERENCES Autor(ID_atora)  
);
```

Tabela IzdavanjeKnjiga:

```
CREATE TABLE IzdavanjeKnjiga (  
    ID_izdavanja INT PRIMARY KEY,  
    ID_člana INT,  
    ID_knjige INT,  
    DatumIzdavanja DATE,  
    DatumPovratka DATE,  
    FOREIGN KEY (ID_člana) REFERENCES Član(ID_člana),
```



```
FOREIGN KEY (ID_knjige) REFERENCES Knjiga(ID_knjige)
);
```

Ovim pristupom se osigurava da baza podataka **BibliotekaDB** baza podataka ima jasno definisane entitete i veze koje odražavaju realne odnose unutar biblioteke, omogućavajući efikasno upravljanje i pružajući fleksibilnost za različite vrste upita i izveštaja.

Primer 82 – Informacioni sistem za online kurseve

U procesu razvoja informacionog sistema „EdukacijaOnline” koji podržava online kurseve, kako biste pristupili izradi i dizajnu tabela koje će sadržati informacije o kursovima, predavačima, studentima, lekcijama, testovima i ocenama, te kako biste osigurali da struktura baze podataka podržava evidenciju kurseva sa više lekcija, dodeljivanje predavača kursovima, upis studenata na kurseve, praćenje napretka studenta kroz lekcije i testove, kao i ocenjivanje, i kako biste konstruisali SQL upite za kreiranje ovih tabela, omogućavajući jednostavno ažuriranje, pretragu i analizu podataka unutar sistema „EdukacijaOnline”?

SQLgpt:

Definisanje entiteta:

- **Kursevi:** Sadrži osnovne informacije o dostupnim online kursovima.
- **Predavači:** Sadrži detalje o predavačima koji vode kurseve.
- **Studenti:** Sadrži informacije o studentima koji se upisuju na kurseve.
- **Lekcije:** Definiše lekcije koje čine svaki kurs.
- **Testovi:** Obuhvata testove koji su deo kurseva za procenu znanja studenata.
- **Ocene:** Evidentira ocene koje studenti dobijaju na testovima.

Veze:

- **Kursevi-Predavači:** „Jedan prema više” veza, gde jedan predavač može voditi više kurseva.

- Kursevi-Studenti: „Više prema više” veza, gde studenti mogu biti upisani na više kurseva, a kursevi mogu imati više studenata.
- Kursevi-Lekcije: „Jedan prema više” veza, svaki kurs sadrži više lekcija.
- Lekcije-Testovi: „Jedan prema više” veza, svaka lekcija može imati pridružene testove.
- Studenti-Ocene: „Više prema više” veza, studenti dobijaju ocene na testovima iz različitih kurseva i lekcija.

SQL upiti za kreiranje tabela:

Tabela Kursevi:

```
CREATE TABLE Kursevi (  
    ID_kursa INT PRIMARY KEY,  
    NazivKursa VARCHAR(255),  
    OpisKursa TEXT,  
    ID_predavača INT,  
    FOREIGN KEY (ID_predavača)  
    REFERENCES Predavači (ID_predavača)  
);
```

Tabela Predavači:

```
CREATE TABLE Predavači (  
    ID_predavača INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    Specijalizacija VARCHAR(255)  
);
```

Tabela Studenti:

```
CREATE TABLE Studenti (  
    ID_studenta INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    Email VARCHAR(100)  
);
```

Tabela Lekcije:

```
CREATE TABLE Lekcije (  
    ID_lekcije INT PRIMARY KEY,  
    ID_kursa INT,  
    NaslovLekcije VARCHAR(255),  
    Sadržaj TEXT,  
    FOREIGN KEY (ID_kursa)
```

```
REFERENCES Kursevi(ID_kursa)
);
```

Tabela Testovi:

```
CREATE TABLE Testovi (
  ID_testa INT PRIMARY KEY,
  ID_lekcije INT,
  NazivTesta VARCHAR(255),
  OpisTesta TEXT,
  FOREIGN KEY (ID_lekcije)
  REFERENCES Lekcije(ID_lekcije)
);
```

Tabela Ocene:

```
CREATE TABLE Ocene (
  ID_ocene INT PRIMARY KEY,
  ID_studenta INT,
  ID_testa INT,
  Ocena INT,
  FOREIGN KEY (ID_studenta)
  REFERENCES Studenti(ID_studenta),
  FOREIGN KEY (ID_testa)
  REFERENCES Testovi(ID_testa)
);
```

Pomoćne tabelle za „više prema više” veze (npr. za Kursevi-Studenti):

```
CREATE TABLE UpisNaKurs (
  ID_studenta INT,
  ID_kursa INT,
  DatumUpisa DATE,
  PRIMARY KEY (ID_studenta, ID_kursa),
  FOREIGN KEY (ID_studenta)
  REFERENCES Studenti(ID_studenta),
  FOREIGN KEY (ID_kursa)
  REFERENCES Kursevi(ID_kursa)
);
```

Prateći ove korake, baza podataka „EdukacijaOnline” će biti sposobna da efikasno upravlja svim aspektima online obrazovanja, od upravljanja kursevima i lekcijama do praćenja napretka i ocenjivanja studenata, dok istovremeno omogućava laku navigaciju, pretragu i ažuriranje podataka.

Primer 83 – Primarni i strani ključevi

*Kako biste konstruisali i implementirali sistem primarnih i stranih ključeva u bazi podataka **ZdravstveniSistemDB** koja obuhvata entitete kao što su **Pacijenti**, **Doktori**, **Termini**, **Dijagnoze** i **Tretmani**, sa ciljem da se osigura integritet podataka, omogućiti precizno praćenje medicinskih*

*usluga pruženih pacijentima, i omogućí efikasna razmena informacija između različitih delova sistema, te kako biste definisali SQL upite za kreiranje odgovarajućih tabela sa adekvatno postavljenim primarnim i stranim ključevima, osiguravajući tako da baza podataka **ZdravstveniSistemDB** podržava sve neophodne operacije upravljanja pacijentima i zdravstvenim uslugama, dok se istovremeno održava visok nivo organizacije i ažurnosti podataka?*

SQLgpt:

Definisanje entiteta:

- **Pacijenti:** Sadrži osnovne informacije o pacijentima.
- **Doktori:** Sadrži detalje o doktorima unutar sistema.
- **Termini:** Evidentira zakazane termine između pacijenata i doktora.
- **Dijagnoze:** Obuhvata dijagnoze postavljene pacijentima.
- **Tretmani:** Sadrži informacije o tretmanima koji su propisani pacijentima.

Primarni ključevi:

Svaki entitet će imati svoj jedinstveni primarni ključ, npr. **ID_pacijenta** za tabelu **Pacijenti**, **ID_doktora** za tabelu **Doktori**, itd.

Strani ključevi:

- U tabeli **Termini** koristićemo **ID_pacijenta** i **ID_doktora** kao strane ključeve da povežemo termine sa odgovarajućim pacijentima i doktorima.
- U tabeli **Dijagnoze**, **ID_pacijenta** će biti strani ključ.
- U tabeli **Tretmani**, **ID_dijagnoze** će povezivati tretmane sa odgovarajućim dijagnozama.

SQL upiti za kreiranje tabela:

Tabela Pacijenti:

```
CREATE TABLE Pacijenti (
```

```
    ID_pacijenta INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    JMBG VARCHAR(13),  
    DatumRodjenja DATE  
);
```

Tabela Doktori:

```
CREATE TABLE Doktori (  
    ID_doktora INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    Specijalizacija VARCHAR(100)  
);
```

Tabela Termini:

```
CREATE TABLE Termini (  
    ID_termina INT PRIMARY KEY,  
    ID_pacijenta INT,  
    ID_doktora INT,  
    DatumTermina DATE,  
    VremeTermina TIME,  
    FOREIGN KEY (ID_pacijenta)  
    REFERENCES Pacijenti(ID_pacijenta),  
    FOREIGN KEY (ID_doktora)  
    REFERENCES Doktori(ID_doktora)  
);
```

Tabela Dijagnoze:

```
CREATE TABLE Dijagnoze (  
    ID_dijagnoze INT PRIMARY KEY,  
    ID_pacijenta INT,  
    Dijagnoza TEXT,  
    DatumDijagnoze DATE,  
    FOREIGN KEY (ID_pacijenta)  
    REFERENCES Pacijenti(ID_pacijenta)  
);
```

Tabela Tretmani:

```
CREATE TABLE Tretmani (  
    ID_tretmana INT PRIMARY KEY,  
    ID_dijagnoze INT,  
    OpisTretmana TEXT,  
    DatumPočetka DATE,  
    DatumZavršetka DATE,  
    FOREIGN KEY (ID_dijagnoze)  
    REFERENCES Dijagnoze(ID_dijagnoze)  
);
```

Ovako strukturirana baza podataka **ZdravstveniSistemDB** omogućava efikasno praćenje i upravljanje medicinskim informacijama, osiguravajući

da svaka dijagnoza, tretman ili termin budu adekvatno povezani sa pravim pacijentom ili doktorom. Primena primarnih i stranih ključeva unapređuje integritet i doslednost podataka, što je ključno u osetljivim sistemima kao što su zdravstveni informacioni sistemi.

Primer 84 – Ograničenje integriteta

*Kako biste projektovati i implementirali sistem ograničenja integriteta u bazi podataka **FinansijskiSistemDB** koja upravlja finansijskim transakcijama, računima, korisnicima i plaćanjima, s ciljem da se osigura tačnost, doslednost i pouzdanost podataka unutar sistema, te kako biste definisali SQL upite za kreiranje tabela sa odgovarajućim ograničenjima integriteta kao što su ograničenja **NOT NULL**, **UNIQUE**, **PRIMARY KEY**, **FOREIGN KEY**, i **CHECK**, omogućavajući tako da baza podataka **FinansijskiSistemDB** efikasno upravlja finansijskim operacijama uz minimalan rizik od grešaka u podacima?*

SQLgpt:

Definisanje entiteta:

- **Računi:** Sadrži informacije o računima unutar sistema.
- **Korisnici:** Sadrži podatke o korisnicima koji imaju račune.
- **Transakcije:** Evidentira sve finansijske transakcije izvršene na računima.
- **Plaćanja:** Definiše informacije o različitim plaćanjima koja korisnici vrše.

Ograničenja integriteta:

- **NOT NULL:** Osigurava da određeni ključni atributi ne mogu imati **NULL** vrednosti.
- **UNIQUE:** Garantuje jedinstvenost određenih podataka, poput broja računa ili korisničkog imena.
- **PRIMARY KEY:** Postavlja jedinstveni identifikator za svaki red u tabeli.

- **FOREIGN KEY:** Povezuje tabele i održava referencijalni integritet.
- **CHECK:** Osigurava da vrednosti u koloni zadovoljavaju određeni uslov, npr. pozitivno stanje na računu.

SQL upiti za kreiranje tabela:

Tabela Korisnici:

```
CREATE TABLE Korisnici (  
    ID_korisnika INT PRIMARY KEY,  
    KorisničkoIme VARCHAR(50) NOT NULL UNIQUE,  
    Ime VARCHAR(100) NOT NULL,  
    Prezime VARCHAR(100) NOT NULL,  
    Email VARCHAR(100) NOT NULL UNIQUE  
);
```

Tabela Računi:

```
CREATE TABLE Računi (  
    ID_računa INT PRIMARY KEY,  
    ID_korisnika INT,  
    BrojRačuna VARCHAR(20) NOT NULL UNIQUE,  
    Saldo DECIMAL(15, 2) CHECK (Saldo >= 0),  
    FOREIGN KEY (ID_korisnika)  
    REFERENCES Korisnici(ID_korisnika)  
);
```

Tabela Transakcije:

```
CREATE TABLE Transakcije (  
    ID_transakcije INT PRIMARY KEY,  
    ID_računa INT,  
    Iznos DECIMAL(15, 2),  
    DatumTransakcije DATETIME NOT NULL,  
    TipTransakcije VARCHAR(50),  
    FOREIGN KEY (ID_računa)  
    REFERENCES Računi(ID_računa)  
);
```

Tabela Plaćanja:

```
CREATE TABLE Plaćanja (  
    ID_plaćanja INT PRIMARY KEY,  
    ID_transakcije INT,  
    ID_korisnika INT,  
    IznosPlaćanja DECIMAL(15, 2) CHECK (IznosPlaćanja > 0),  
    DatumPlaćanja DATETIME NOT NULL,  
    FOREIGN KEY (ID_transakcije)  
    REFERENCES Transakcije(ID_transakcije),  
    FOREIGN KEY (ID_korisnika)
```

```
REFERENCES Korisnici(ID_korisnika)
);
```

Implementacijom ovih ograničenja integriteta, baza podataka **FinansijskiSistemDB** osigurava visok nivo tačnosti i doslednosti podataka, što je ključno za pouzdano upravljanje finansijskim operacijama. Ograničenja integriteta pomažu u sprečavanju uobičajenih grešaka u podacima kao što su duplikati, nekompletni podaci ili neispravne transakcije, čime se poboljšava efikasnost i pouzdanost finansijskog sistema.

Primer 85 – Rad sa pogledima

*Kako biste projektovati i koristiti poglede u bazi podataka **ImovinskiPortfeljDB** koja upravlja portfeljom nekretnina, uključujući entitete kao što su **Nekretnine**, **Vlasnici**, **Ugovori** i **Plaćanja**, sa ciljem da se omogući efikasna analiza i izveštavanje o statusu imovine, ugovorima i finansijskim transakcijama, te kako biste definisali **SQL** upite za kreiranje pogleda koji olakšavaju uobičajene upite, omogućavajući korisnicima **ImovinskiPortfeljDB** da lako pristupe agregiranim i filtriranim informacijama, dok se istovremeno osigurava da osnovni podaci ostaju zaštićeni i neizmenjeni?*

SQLgpt:

Definisanje entiteta:

- **Nekretnine:** Sadrži detalje o nekretninama unutar portfelja.
- **Vlasnici:** Evidentira informacije o vlasnicima nekretnina.
- **Ugovori:** Definiše ugovore vezane za kupoprodaju ili izdavanje nekretnina.
- **Plaćanja:** Obuhvata finansijske transakcije povezane sa ugovorima o nekretninama.

Projektovanje pogleda:

Pregled finansijskog statusa nekretnina: Prikazuje ukupne prihode od svih nekretnina, sortirane po vlasnicima.

```
CREATE VIEW PregledFinansijskogStatusa AS
```



```
SELECT V.Ime, V.Prezime, SUM(P.Iznos) AS UkupniPrihod
FROM Vlasnici V
JOIN Nekretnine N ON V.ID_vlasnika = N.ID_vlasnika
JOIN Ugovori U ON N.ID_nekretnine = U.ID_nekretnine
JOIN Plaćanja P ON U.ID_ugovora = P.ID_ugovora
GROUP BY V.Ime, V.Prezime;
```

Pregled aktivnih ugovora: Prikazuje sve ugovore koji su trenutno aktivni, sa detaljima o nekretninama i vlasnicima.

```
CREATE VIEW PregledAktivnihUgovora AS
SELECT N.Adresa, V.Ime, V.Prezime, U.DatumPočetka,
U.DatumZavršetka
FROM Ugovori U
JOIN Nekretnine N ON U.ID_nekretnine = N.ID_nekretnine
JOIN Vlasnici V ON N.ID_vlasnika = V.ID_vlasnika
WHERE U.DatumZavršetka > CURRENT_DATE;
```

Pregled plaćanja po nekretninama: Prikazuje sva plaćanja po nekretninama za određeni period.

```
CREATE VIEW PregledPlaćanjaPoNekretninama AS
SELECT N.Adresa, SUM(P.Iznos) AS UkupnoPlaćeno,
COUNT(P.ID_plaćanja) AS BrojPlaćanja
FROM Plaćanja P
JOIN Ugovori U ON P.ID_ugovora = U.ID_ugovora
JOIN Nekretnine N ON U.ID_nekretnine = N.ID_nekretnine
GROUP BY N.Adresa;
```

Korišćenje pogleda:

Pogledi omogućavaju korisnicima da izvrše složene upite i analize bez potrebe za detaljnim poznavanjem strukture baze podataka ili pisanjem kompleksnih SQL upita. Na primer:

- Da bi se dobio pregled ukupnih prihoda po vlasnicima, može se izvršiti jednostavan upit na pogledu **PregledFinansijskogStatusa**.
- Da bi se proverili svi aktivni ugovori, može se koristiti pogled **PregledAktivnihUgovora**.

Prednosti korišćenja pogleda:

- **Bezbednost:** Pogledi mogu ograničiti pristup određenim kolonama ili redovima, štiteći osetljive podatke.
- **Jednostavnost:** Pogledi pojednostavljuju složene upite, omogućavajući korisnicima lakši pristup informacijama.

- **Očuvanje performansi:** Pogledi se izvršavaju na zahtev, što znači da ne zauzimaju dodatni prostor za skladištenje podataka.

Implementacijom pogleda u bazi podataka **ImovinskiPortfeljDB**, korisnicima se pruža efikasan alat za upravljanje, analizu i izveštavanje o imovini, ugovorima i finansijskim transakcijama, uz očuvanje integriteta i sigurnosti osnovnih podataka.

Primer 86 – Konsultacije doktora sa drugim doktorima

*U kontekstu baze podataka **ZdravstveniCentarDB** koja omogućava upravljanje medicinskim osobljem, pacijentima i tretmanima, kako biste omogućili funkcionalnost gde doktor može zatražiti konsultacije sa jednim ili više drugih doktora, te kako biste projektovati strukturu tabele koja omogućava evidentiranje zahteva za konsultacijama, uključujući odgovarajuće informacije o doktoru koji inicira zahtev, doktorima sa kojima se konsultuje, vremenu i datumu konsultacija, te specifičnim pitanjima ili ciljevima konsultacije, te kako biste kreirali SQL upite za kreiranje ove tabele i upravljanje zapisima o konsultacijama, osiguravajući da baza podataka **ZdravstveniCentarDB** podržava složene scenarije konsultacija među medicinskim osobljem?*

SQLgpt:

Definisanje entiteta i veza:

- Entiteti: **Doktori, Konsultacije**
- Veze: Jedan doktor (inicijator) može zatražiti konsultacije sa više drugih doktora.

Projektovanje tabele **Konsultacije**:

Tabela **Konsultacije** treba da sadrži informacije o samoj konsultaciji, uključujući identifikator konsultacije, identifikator doktora koji inicira zahtev, datum i vreme konsultacije, kao i specifične detalje vezane za konsultaciju.

Projektovanje pomoćne tabele za „više prema više” vezu:

Za evidenciju kojih doktora uključuje svaka konsultacija, potrebna je pomoćna tabela **DoktoriUKonsultacijama**, koja povezuje doktore sa konsultacijama.

SQL upiti:

Kreiranje tabele Doktori (ukoliko već ne postoji):

```
CREATE TABLE Doktori (  
    ID_doktora INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    Specijalizacija VARCHAR(100)  
);
```

Kreiranje tabele Konsultacije:

```
CREATE TABLE Konsultacije (  
    ID_konsultacije INT PRIMARY KEY,  
    ID_inicijatora INT,  
    DatumVreme DATETIME,  
    Detalji TEXT,  
    CiljKonsultacije TEXT,  
    FOREIGN KEY (ID_inicijatora)  
    REFERENCES Doktori(ID_doktora)  
);
```

Kreiranje pomoćne tabele DoktoriUKonsultacijama:

```
CREATE TABLE DoktoriUKonsultacijama (  
    ID_konsultacije INT,  
    ID_doktora INT,  
    PRIMARY KEY (ID_konsultacije, ID_doktora),  
    FOREIGN KEY (ID_konsultacije)  
    REFERENCES Konsultacije(ID_konsultacije),  
    FOREIGN KEY (ID_doktora) REFERENCES Doktori(ID_doktora)  
);
```

Upravljanje zapisima o konsultacijama:

Za dodavanje nove konsultacije:

```
INSERT INTO Konsultacije  
    (ID_inicijatora, DatumVreme, Detalji, CiljKonsultacije)  
VALUES (?, ?, ?, ?);
```

Za povezivanje doktora sa konsultacijom:

```
INSERT INTO DoktoriUKonsultacijama (ID_konsultacije,  
ID_doktora)  
VALUES (?, ?);
```

Implementacijom ovih tabela i upita, baza podataka **ZdravstveniCentarDB** baza podataka biće sposobna da efikasno upravlja i prati konsultacije među doktorima, čime se olakšava saradnja i razmena stručnih mišljenja unutar medicinskog osoblja, što je ključno za pružanje kvalitetne zdravstvene zaštite.

Primer 87 – Hijerarhija u relacionoj bazi podataka

*U kontekstu projektovanja relacione baze podataka **ObrazovnaInstitucijaDB** koja upravlja podacima o studentima, predavačima, predmetima, ocenama i rasporedima predavanja, kako biste strukturirali hijerarhiju relacione baze podataka da podrži složene odnose i interakcije između različitih entiteta, te kako biste osmislili i implementirali strukturu koja omogućava efikasno skladištenje i pretragu podataka, pritom osiguravajući integritet podataka i podržavajući složene upite za generisanje izveštaja o performansama studenata, rasporedu predavanja, i dostupnosti predavača, te kako biste definisali SQL upite za kreiranje ključnih tabela i njihovih međusobnih veza, omogućavajući tako da baza podataka **ObrazovnaInstitucijaDB** efikasno funkcioniše kao centralizovani sistem za upravljanje informacijama unutar obrazovne institucije?*

SQLgpt:

Definisanje entiteta i njihovih odnosa:

- Entiteti: **Studenti, Predavači, Predmeti, Ocene, RasporedPredavanja**
- Odnosi: Studenti upisuju predmete, predavači predaju predmete, ocene se dodeljuju za predmete koje studenti pohađaju, i raspoređeni predavanja se kreiraju na osnovu dostupnosti predavača i sala.

Projektovanje ključnih tabela:

Tabela Studenti:

```
CREATE TABLE Studenti (  
    ID_studenta INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),
```

```
    BrojIndeksa VARCHAR(50) UNIQUE,  
    DatumUpisa DATE  
);
```

Tabela Predavači:

```
CREATE TABLE Predavači (  
    ID_predavača INT PRIMARY KEY,  
    Ime VARCHAR(100),  
    Prezime VARCHAR(100),  
    Zvanje VARCHAR(100)  
);
```

Tabela Predmeti:

```
CREATE TABLE Predmeti (  
    ID_predmeta INT PRIMARY KEY,  
    NazivPredmeta VARCHAR(100),  
    ID_predavača INT,  
    FOREIGN KEY (ID_predavača)  
    REFERENCES Predavači (ID_predavača)  
);
```

Tabela Ocene:

```
CREATE TABLE Ocene (  
    ID_ocene INT PRIMARY KEY,  
    ID_studenta INT,  
    ID_predmeta INT,  
    Ocenjivač INT,  
    Ocena INT CHECK (Ocena BETWEEN 1 AND 10),  
    DatumOcene DATE,  
    FOREIGN KEY (ID_studenta)  
    REFERENCES Studenti (ID_studenta),  
    FOREIGN KEY (ID_predmeta)  
    REFERENCES Predmeti (ID_predmeta),  
    FOREIGN KEY (Ocenjivač)  
    REFERENCES Predavači (ID_predavača)  
);
```

Tabela RasporedPredavanja:

```
CREATE TABLE RasporedPredavanja (  
    ID_rasporeda INT PRIMARY KEY,  
    ID_predmeta INT,  
    ID_predavača INT,  
    VremePočetka TIME,  
    VremeZavršetka TIME,  
    DanUNedelji VARCHAR(50),  
    Učionica VARCHAR(50),  
    FOREIGN KEY (ID_predmeta)  
    REFERENCES Predmeti (ID_predmeta),  
    FOREIGN KEY (ID_predavača)  
    REFERENCES Predavači (ID_predavača)
```

);

Implementacija hijerarhije i relacija:

- Hijerarhija se odražava kroz strukturirane tabele koje sadrže primarne i strane ključeve, omogućavajući povezivanje entiteta i praćenje njihovih odnosa.
- Odnos „više prema više” između studenata i predmeta može se upravljati kroz tabelu **Ocene** koja deluje kao povezujuća tabela, omogućavajući evidenciju ocena za svaku kombinaciju studenta i predmeta.

Podrška za složene upite:

- Omogućavanje upita koji generišu izveštaje o ocenama studenata, rasporedu predavanja za određenog predavača ili dostupnosti učionica.

Implementacijom ove strukture, baza podataka **ObrazovnaInstitucijaDB** postaje robustna baza podataka sposobna za efikasno upravljanje ključnim informacijama unutar obrazovne institucije, uz podršku za raznovrsne analitičke i operativne potrebe.