Projektovanje baze podataka – kreiranje i uklanjanje tabela, indeksa, pogleda, trigeri

Naredbe **za definisanje podataka** omogućuju definisanje resursa relacione baze podataka. Pod resursima baze podataka se najčešće podrazumevaju: **struktura baze podataka, tabele, atributi, tipovi podataka, ograničenja, pomoćni indeksi za direktan pristup,...**

Efikasan sistem za upravljanje bazom podataka SUBP-a treba da omogući izvršenja operacija:
1. kreiranje baze podataka
2. kreiranje tabele baze podataka
3. kreiranje indeksa nad kombinacijom kolona tabele
4. kreiranje virtuelne tabele - "pogleda"
5. izmena definicije tabele
6. izmena pogleda u bazi podataka
7. promena imena tabeli u bazi podataka
8. brisanje tabele iz baze podataka
9. uklanjanje indeksa iz tabele
10. uklanjanje baze podataka

Podsećanje sa prethodnog časa
1. Šta sve sadrži SQL naredba za kreiranje tabela?
2. SQL naredba za definisanje kolone
3. UNIQUE klauzula kao ograničenje kolone označava :
4. Ako se neka kolona označi kao PRIMARY KEY to znači:
5. Šta znači sledeće ograničenje pri kreiranju tabele: ImeKolone REFERENCES ImeTabele
6. Naredba za uklanjanje tabele iz BP je:
7. Kod brisanja (uklanjanja) tabele iz BP neophodno je da:
8. INDEX je :
9. Koji atributi nisu dobri kandidat za INDEX:
10. Kod uklanjanja indeksa (DROP INDEX ImeIndeksa) karakteristično je sledeće:
11. Izmena definicije postojeće tabele vrši se SQL naredbom:
12. Pogled (VIEW) je:
13. Navesti sve prednosti upotrebe pogleda (VIEW):
14. Kada se proveravaju ograničenja?
    - provera se vrši na kraju izvršavanja svake SQL naredbe. SQL:1999 standard dozvoljava da se provera ograničenja odloži i izvrši na kraju transakcije. Za svako ograničenje je moguće specificirati da li je ili ne dozvoljeno odlaganje provere do kraja transakcije, što se zapisuje navođenjem opcije [NOT]DEFERRABLE. Ako je izabrana opcija DEFERRABLE, neophodno je navesti da li je na početku transakcije provera ograničenja odložena ili ne.
15. Vrste ograničenja?

Podržane vrste ograničenja su:
1.Ograničenje domena (CHECK ograničenje) – primenjuje se na sve kolone definisane nad posmatranim domenom
2.Ograničenje tabele – definiše se za jednu baznu tabelu i to UNIQUE, PRIMARY KEY, FOREIGN KEY i CHECK ograničenje
3.Opšta ograničenja – CHECK ograničenje kojim se definiše uslov nad podacima više tabela baze podataka.
UNIQUE ograničenje je zadovoljeno samo ako ne postoje dva reda bazne tabele sa istim NOT NULL vrednostima u kolonama nad kojima je ograničenje specificirano.

PRIMARY KEY ograničenje definiše primarni ključ tabele. Ono je zadovoljeno ako i samo ako ne postoje dva reda bazne tabele sa istim vrednostima u kolonama nad kojima je ograničenje

specificirano.

FOREIGN KEY ograničenje specificira jednu ili više kolona bazne tabele kao referencirajuće kolone i njima odgovarajuće referencirane kolone u nekoj baznoj tabeli. Ovo ograničenje je zadovoljeno ako su, za svaki red referencirajuće tabele, vrednosti referenciranih kolona jednake vrednostima odgovarajućih referenciranih kolona nekog reda referencirane tabele.

CHECK ograničenje definiše uslov koji može da ograničava dozvoljene vrednosti domena ili kolone, odrediti međuzavisnost vrednosti kolona i redova jedne tabele ili definisati opšte ograničenje, koje se prostire na više tabela baze.

17. Da li je korektan sledeći SQL skript kojim se formira tabela Prodavci?

```
CREATE TABLE Prodavci
(prod_id CHAR(4) PRIMARY KEY
  CHECK  (prod_id LIKE '[A-Z][A-Z][A-Z][A-Z]'  OR  prod_id LIKE '[A-Z][A-Z][0-9][0-9]'),
   prod_ime VARCHAR(40),
   prod_adresa1 VARCHAR(40),
   prod_adresa2 VARCHAR(40),
   grad VARCHAR(20),
   drzava CHAR(2)  DEFAULT 'SR',
   Postanski_broj CHAR(5) UNIQUE
     CHECK (Postanski_broj LIKE '[1-9][0-9][0-9][0-9][0-9]'),
  telefon CHAR(12));
```

Ako je SQL skript korektan, uporedite ga sa rezultatom izvršavanja naredbe SHOW CREATE TABLE Prodavci\G.

```
INSERT INTO Prodavci VALUES ('ABCD', 'Pera', 'PeraSt1', 'PeraSt2', 'BG',default,'12012',
'012888888');
```

18. Data je relacija **POSLOVI(sif_radnika, sif_firme, radno_mesto)**. Ako je ovako određen primarni ključ i ako relacija odgovara realnom modelu jedne firme:

a) Da li radnik može u istoj firmi raditi na više radnih mesta?
b) Da li radnik može raditi u više firmi?
c) Da li firma može imati više radnika zaposlenih na istom radnom mestu?
d) Da li se može upisati novi radnik u relaciju ako nije definisano na kom radnom mestu radi (ako se pri tome podrazumeva da svi atributi u relaciji koji ne pripadaju ključu ne moraju imati zadate vrednosti)

19. Postavite default vrednosti za kolonu grad u tabeli Prodavci i proverite rešenje.
20. Izbacite default vrednosti za kolonu grad u tabeli Prodavci i proverite rešenje.
21. Izbacite kolonu telefon iz tabele Prodavci.

Svojstva ograničenja navedena pri SQL naredbi za kreiranje tabela:

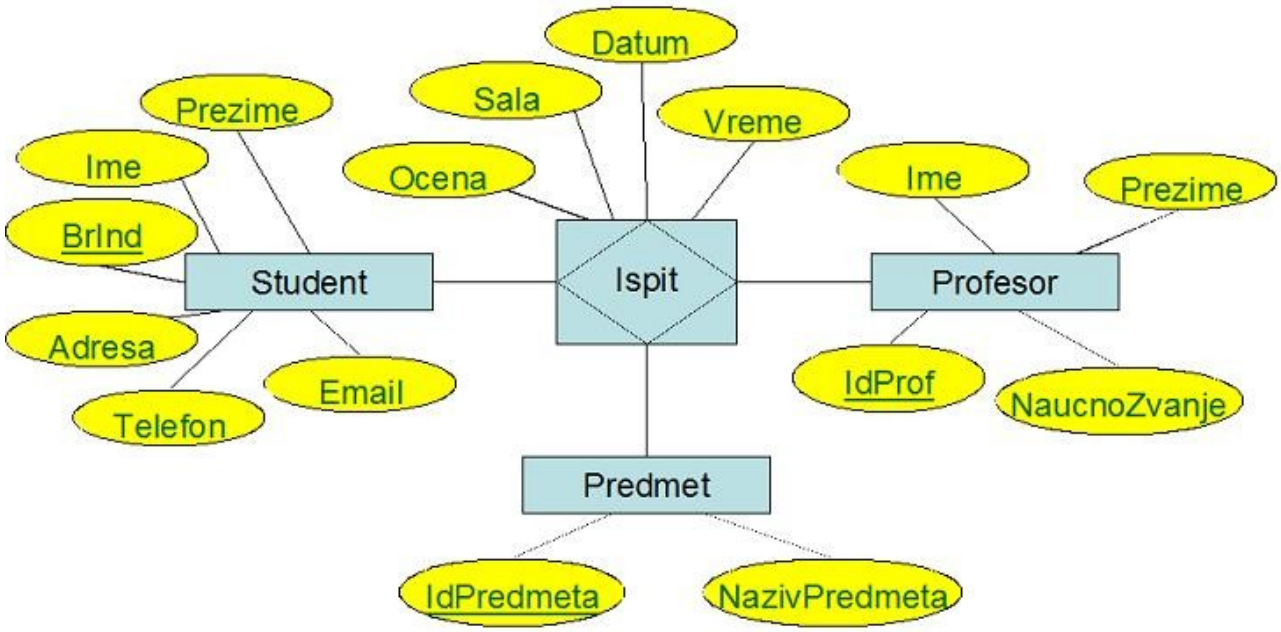
1. Sva ograničenja navedena u CREATE TABLE su aktivna u svakom trenutku postojanja tabele.
2. SUBP će odbiti svaki pokušaj sa tabelom, koji je u suprotnosti sa ograničenjima.
3. Olakšica za projektante i programere BP
4. Provere se ne moraju ugrađivati u aplikativne programe
5. Deklarativna moć naredbe CREATE TABLE je od velikog značaja, naročito kod dinamičke specifikacije referencijalnog integriteta (DELETE, UPDATE,...)

Primer za samostalni rad: Analizirajmo Sakila bazu podataka. Ako želite, forward engineering, tj. kreiranje i smeštanje SQL skripta za kreiranje tabela, indeksa, trigera na DBMS, setimo se opcija:

Database, Forward Engeneer to Database i odaberite opcije koje su Vam potrebne.
Pogledajte triggere.
Kako cete ubaciti CHECK iskaze i ogranicenja?

Primer za samostalni rad:
Studenti(BrInd, Ime, Prezime, Adresa, Telefon, Email)
Profesori(IdProf, Ime Prezime, NaucnoZvanje)
Predmeti(IdPredmet, NazivPredmeta)
Ispit(Brind, IdPredmet, IdProf, Ocena, Sala, Datum, Vreme)

Kakvu sliku očekujemo od MySQL WorkBench za ovu bazu?



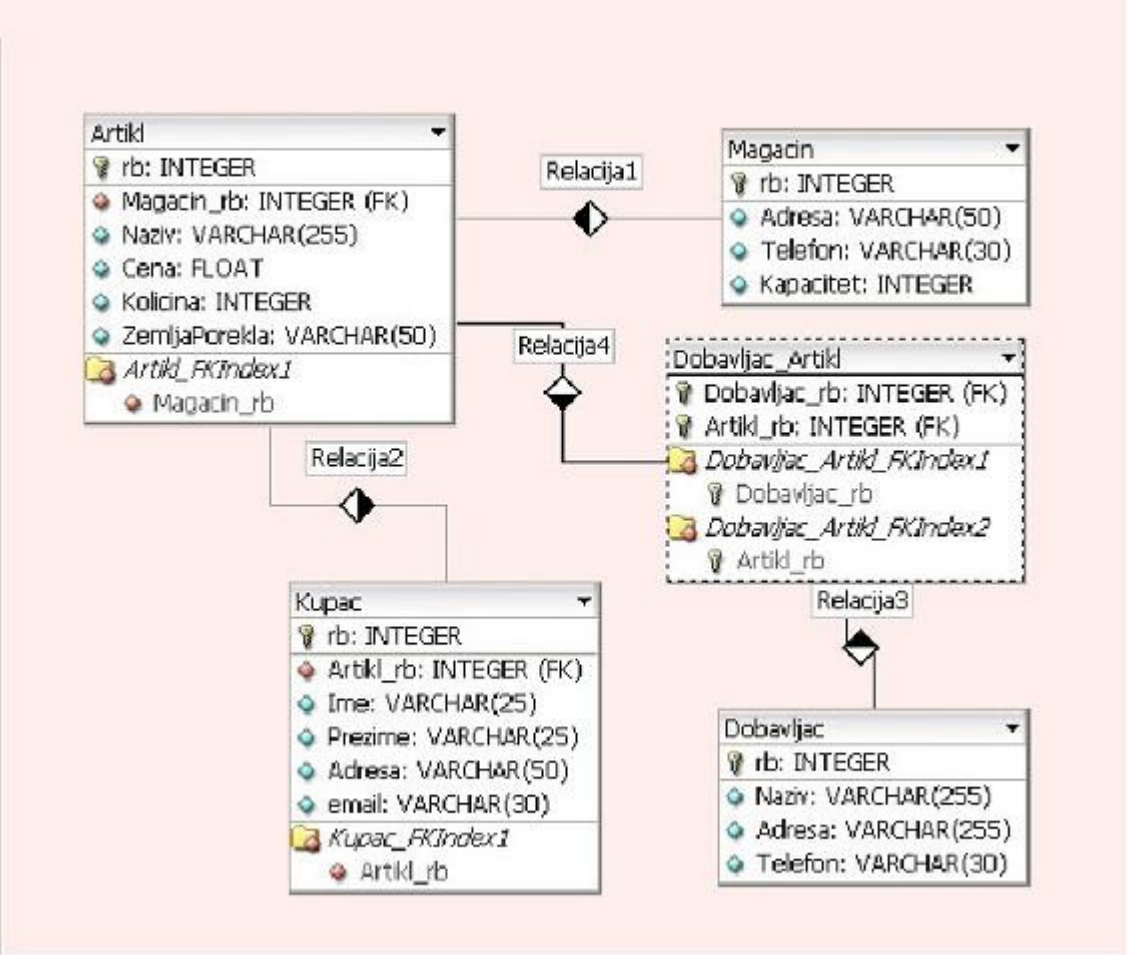
Kreirajte SQL iskaze za kreiranje baze i tabela i uporedite sa izlazom koji je vratio MySQL Workbench. Kako cemo ubaciti ogranicenja? Pogledajte karticu trigger.

```
CREATE TABLE Studenti (
BrInd  INT     PRIMARY KEY CHECK(0<BrInd<=400),
Ime             VARCHAR(20)         NOT NULL,
Prezime         VARCHAR(20)         NOT NULL,
Adresa VARCHAR(50)         NOT NULL,
Telefon         VARCHAR(15),
Email  VARCHAR(30));

CREATE TABLE Ispit (
BrInd  INT     REFERENCES Studenti,
IdPredmet       INT     REFERENCES Predmeti,
IdProf INT     REFERENCES Profesori,
Ocena  INT     NOT NULL CHECK (5<=Ocena<=10),
Sala            CHAR(5) UNIQUE,
Datum DATE,
Vreme TIME
PRIMARY KEY (BrInd,IdPredmeta,IdProf),
```

FOREIGN KEY STD (BrInd) REFERENCES Studenti on DELETE RESTRICT on UPDATE CASCADE,
FOREIGN KEY PRED (IdPredmeta) REFERENCES Predmeti on DELETE RESTRICT on UPDATE CASCADE,
FOREIGN KEY PROF (IdProf) REFERENCES Profesori on DELETE RESTRICT on UPDATE CASCADE)
...

Primer2: Skicirajmo ovakav model baze podataka



Pogledajmo kakav cemo SQL iskaz za kreiranje tabele dobiti u mysql workbench-u

Da li ti iskazi se poklapaju sa očekivanim, na primer:

```
CREATE TABLE Artikl (
  rb INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Magacin_rb INTEGER UNSIGNED NOT NULL,
  Naziv VARCHAR(255) NULL,
  Cena FLOAT NULL,
  Kolicina INTEGER UNSIGNED NOT NULL DEFAULT 0,
  ZemljaPorekla VARCHAR(50) NULL,
  PRIMARY KEY(rb),
  INDEX Artikl_FKIndex1(Magacin_rb)
);

CREATE TABLE Dobavljac (
```

```
  rb INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Naziv VARCHAR(255) NOT NULL,
  Adresa VARCHAR(255) NULL,
  Telefon VARCHAR(30) NULL,
  PRIMARY KEY(rb)
);

CREATE TABLE Dobavljac_Artikl (
  Dobavljac_rb INTEGER UNSIGNED NOT NULL,
  Artikl_rb INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(Dobavljac_rb, Artikl_rb),
  INDEX Dobavljac_has_Artikl_FKIndex1(Dobavljac_rb),
  INDEX Dobavljac_has_Artikl_FKIndex2(Artikl_rb)
);

CREATE TABLE Kupac (
  rb INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Artikl_rb INTEGER UNSIGNED NOT NULL,
  Ime VARCHAR(25) NOT NULL,
  Prezime VARCHAR(25) NOT NULL,
  Adresa VARCHAR(50) NOT NULL,
  Telefon VARCHAR(30) NOT NULL,
  email VARCHAR(30) NULL,
  PRIMARY KEY(rb),
  INDEX Kupac_FKIndex1(Artikl_rb)
);

CREATE TABLE Magacin (
  rb INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Adresa VARCHAR(50) NOT NULL,
  Telefon VARCHAR(30) NOT NULL,
  Kapacitet INTEGER UNSIGNED NOT NULL DEFAULT 0,
  PRIMARY KEY(rb)
);
```

Brisanje tabele iz baze podataka

Izbor dinamičke specifikacije referencijalnih integriteta, kod uklanjanja predstavlja delikatnu operaciju.
Ako se, na primer, nepromišljeno koristi efekat ON DELETE RESTRICT, nameće se krut režim, npr. ne mogu se ukloniti pogrešno uneti podaci iz tabele.

Naredba uklanjanja tabele iz BP

```
DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name] ...
 [RESTRICT | CASCADE]
```

Komandom je moguće izbrisati više tabela ako se zada lista njihovih imena razdvojenih zarezima.

Primer 3: Izbrisati tabelu Artikli iz primera 2

```
DROP TABLE Artikli;
```

Funkcija brisanja tabele iz baze podataka putem SUBP-a se odnosi na uklanjenje neke od tabela koja je prethodno kreirana u bazi podataka tako da ona više fizički ne postoji u bazi podataka.. Ovde je potrebno uočiti bitnu razliku između naredbe DROP TABLE i naredbe DELETE FROM TABLE. Naredbom DROP

TABLE briše se tabela iz baze podataka. Korišćenjem ove naredbe nad tabelom u kojoj se nalaze podaci doći će do gubljenja tih podataka, dok se naredbom DELETE FROM TABLE brišu se svi podaci iz tabele ali se prazna tabela i dalje čuva u bazi podataka tako da se kasnije ponovo može koristiti.

Kreiranje indeksa

Indeks – pomoćna datoteka za ubrzanje pristupa podacima u osnovnoj datoteci

U osnovnoj datoteci zapisi se nalaze u nekom fizičkom redosledu (redosled unošenja)
Indeks – može se preuređivati u rastućoj ili opadajućoj vrednosti indeksnog niza

Funkcija kreiranja indeksa u tabeli u baze podataka putem SUBP-a se odnosi na kreiranje novog indeksa u tabeli u bazi podataka (iako se on uglavnom kreira pri samom kreiranju tabela baze podataka). Pogledajte prethodne primere gde će te primetiti da se u iskazu za kreiranje tabele navodi primarni ključ i ako postoje i ostali indeksi.
Za svaku kolonu koja se deklariše sa opcijom PRIMARY KEY, KEY, UNIQUE ili INDEX, automatski se formira i indeks. Ako se ustanovi da se koristi veći broj upita koji obuhvataju kolonu za koju nije definisan indeks novi indeks se može dodati pomoću komande

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX *index_name* [*index_type*]
ON *tbl_name* (*index_col_name*,...)

index_col_name: *col_name* [(*length*)] [ASC | DESC]
index_type: USING {BTREE | HASH}

Nad istom tabelom se može definisati više indeksa (za različite pristupe podacima)
Indeks se može kreirati odmah (dok je tabela prazna) ili naknadno

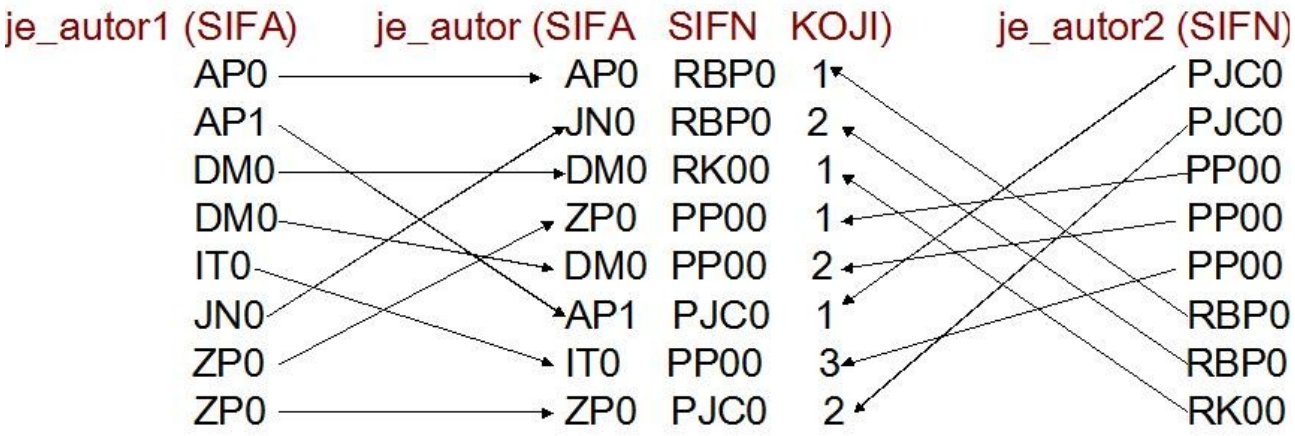
Klauzula UNIQUE u definiciji indeksa ima efekat na klauzulu UNIQUE kod tabele: SUBP odbija svaku izmenu podataka u tabeli koja narušava unikatnost indeksa

SUBP odbija kreiranje unikatnog indeksa za tabelu čiji zatečeni sadržaj narušava tu unikatnost
Indeks se može bilo kada i bez obzira na sadržaj svoje tabele ukloniti naredbom:DROP INDEX *ImeIndeksa*

Primer 4:
Neka je data struktura tabele definicijom
CREATE TABLE Magacin (
rb INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
Adresa VARCHAR(50) NOT NULL,
Telefon VARCHAR(30) NOT NULL,
Kapacitet INTEGER UNSIGNED NOT NULL DEFAULT 0,
PRIMARY KEY(rb)
);

Kreiraćemo indeks u tabeli Magacin nad poljem Telefon sledećom naredbom:
CREATE INDEX K2 ON Magacin(Telefon);

Primer 5: Ako se želi brz pristup podacima u tabeli Je_Autor po dva osnova, po šifri autora i po šifri naslova, kreiraju se dva indeksa:
CREATE INDEX Je_Autor1 ON Je_Autor(SifA);
CREATE INDEX Je_Autor2 ON Je_Autor(SifN);



Napomene:
Svaka indeksna datoteka ima dva dela:

1. ListaKolona, po kojima se vrši pretraživanje i po kojima se vrši uređivanje indeksa.
2. Indeks, koji služi za vezu sa osnovnom datotekom.

Zanimljivo je da se, pre izvršavanja, komanda CREATE INDEX preslikava u komandu ALTER TABLE.

Indeksi definisani nad kolonama tipa CHAR, VARCHAR, BINARY i VARBINARY mogu se ograničiti na prvih nekoliko znakova u polju. To se može uraditi tako što se iza imena indeksirane kolone zada između zagrada broj znakova koji želi da se indeksira:

```
CREATE INDEX K2 ON Magacin(Telefon(6));
```

Pošto indeksi nad kolonama tekstualnog tipa nisu tako efikasni kao indeksi nad numeričkim kolonama, indeksiranje samo nekoliko početnih znakova poboljšava performanse.

Primer 6: Kreiranje indeksa nad jednim atributom koji nije primarni. Zadata je tabela gradjanin (matbr#,prezime,ime,datrodj,adresa)
Atribut Redbr, tj. redni broj zapisa (record number) vodi se u većini programskih paketa
Neka je INDGRAD indeksna datoteka

gradjanin

Redbr	matbr#	prez	ime	datrodj	adresa
1	13248	Antić	Zoran		
2	43286	Jović	Milan		
3	56732	Marić	Goran		
4	56879	Babić	Dragan		
5	42116	Rodić	Petar		
6	89764	Lazić	Ana		
7	13589	Perić	Vera		

indgrad

ind	prez
1	Antić
4	Babić
2	Jović
6	Lazić
3	Marić
7	Perić
5	Rodić

Nisu svi atributi dobri kandidati za indeks. Na primer, bit-map, text ili slika. Po pravilu su strani ključevi kandidati za indeks.

Indeksiranje ima i svojih nedostataka: Prilikom ažuriranja osnovne tabele (brisanje, unošenje), mora se vršiti reindeksiranje, pa se gubi na vremenu.
Tabele sa malim brojem podataka u zapisu se ne indeksiraju, jer se pretraga može efikasno izvršiti

brzim računarima.

Promena imena tabele u bazi podataka

Funkcija promene imena tabele u bazi podataka putem SUBP-a se odnosi na promenu imena neke od tabela koja je prethodno kreirana u bazi podataka. Izmena imena tabele u bazi podataka se u SQL jeziku vrši pomoću naredbe:
RENAME TABLE tbl_name TO new_tbl_name
[, tbl_name2 TO new_tbl_name2] ...

Primer: Promenite ime tabele Kupac u Klijent
```
RENAME TABLE Kupac TO Klijent;
```

Uklanjanje indeksa iz tabele

Funkcija uklanjanja indeksa iz tabele u bazi podataka putem SUBP-a se odnosi na uklanjenje indeksa u nekoj od tabela koji je prethodno kreiran u bazi podataka. Uklanjanje indeksa u nekoj od tabele u bazi podataka se u SQL jeziku vrši pomoću naredbe
DROP INDEX index_name ON tbl_name

Primer: Korišćenjem prethodne naredbe uklonićemo indeks pod nazivom K2,
koji je kreiran nad tabelom Magacin u primeru 4.
```
DROP INDEX K2 ON Magacin;
```

Uklanjanje baze podataka

Funkcija uklanjanja baze podataka putem SUBP-a se odnosi na uklanjenje kompletne baze podataka koja je prethodno kreirana. Ovom naredbom uklanjamo čitavu bazu podataka i sve njene resurse (tabele, atribute, tipove podataka, ograničenja, indekse itd.)).
Uklanjanje baze podataka se u SQL jeziku vrši pomoću naredbe:
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name

DROP DATABASE briše sve tabele u bazi podataka sa datim nazivom i nakon toga briše i samu bazu podataka. Treba biti jako oprezan sa ovim iskazom!
IF EXISTS se koristi da bi se sprečila pojava greške ako na serveru ne postoji baza podataka sa datim nazivom.
DROP DATABASE vraća broj tabela koje su bile obrisane.

Pristup bazi podataka – sloj podataka

Trigeri, okidači (triggers)

Programska procedura u okviru SUBP, koja se aktivira određenim događajem:
- INSERT (upis novog zapisa u tabelu)
- UPDATE (brisanje određenog zapisa)
- DELETE (modifikacija postojećeg zapisa)

Triger predstavlja jedan od mehanizama za proveru uslova integriteta baze podataka
PRIMER: Pre upisa u tabelu *Dobavljac_Artikl* proveriti da li dobavljač dobavlja dati proizvod

Trigeri se automatski pozivaju od strane MySQL-a (ne koristi se Execute naredba). Sami trigeri nemaju ulazne ni izlazne parametre. Triger se poziva prilikom svakog izvršenja određene naredbe odnosno svaki put kada se desi određeni događaj.

Sintaksa za kreiranje trigera je
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt

Trigeri se ne mogu vezivati za privremene tabele. Da bi se pokrenuo triger potrebno je da user ima određene privilegije nad tabelom.Kada je triger aktiviran, Definer klauzula određuje privilegije koje se primenjuju.

Triger_time se odnosi na vreme okidanja, odnosno određuje da li će se triger aktivirati pre (BEFORE) ili posle (AFTER) uslova koja ga aktivira. Dakle, triger_time može biti: BEFORE, AFTER

Trigger_event je jedan od sledećih događaja
• INSERT: triger se aktivira svaki put kada se novi zapis umeće u tabelu (npr. uz naredbe `INSERT`, `LOAD DATA`, `REPLACE`)
• UPDATE: triger se aktivira svaki put kada se zapis ažurira
• DELETE : triger se aktivira svaki put kada se briše zapis iz tabele. Međutim, naredba DROP TABLE ne aktivira triger.

Ne mogu se definisati dva trigera na jednoj tabeli koji se okidaju na pojavu istog događaja i u isto vreme. Na primer, ne možemo imati dva BEFORE UPDATE trigera nad tabelom. Međutim, možemo imati na istoj tabeli BEFORE UPDATE i BEFORE INSERT trigere, ili BEFORE UPDATE i AFTER UPDATE triger.

Trigger_stmt predstavlja komande koje se izvršavaju kada se triger pokrene. Ukoliko želimo da pokrenemo više komandi, koristimo BEGIN ... END konstrukciju. U okviru trigera, možemo da se pozovemo na kolonu tabele nad kojom je triger aktiviran koristeći alijase OLD i NEW.
OLD.col_name se odnosi na kolonu postojećeg zapisa pre njegovog ažuriranja ili brisanja,dok se NEW.col_name odnosi na nov zapis koji će biti insertovan ili na postojeći zapis nakon ažuriranja.

Modifikovanje i brisanje trigera
Ukoliko želimo da promenimo definiciju nekog trigera, onda je to moguće uraditi na dva načina. Ili obrisati triger i ponovo ga kreirati ili koristiti Alter Trigger naredbu.

Za brisanje trigera koristi se:
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name

PRIMER (za samostalni rad): Kreirajmo BEFORE INSERT triger i nazovimo ga ins_sum. U bazi test u tabeli account koja je kreirana iskazom
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Triger treba da sabere vrednosti umetnute u kolonu amount.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

Sad je kreiran triger. Testirajmo njegov rad.

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+----------------------+
| Total amount inserted |
+----------------------+
| 1852.48 |
+----------------------+
```

Dakle, vrednost promenljive @sum (zbog parametara naredbe INSERT) je 14.98 + 1937.50 - 100, to jest 1852.48.
Uklanjanje trigera sa DROP TRIGGER. Mora se navesti ime sheme (sem **ako nije podrazumevana).**

mysql> **DROP TRIGGER test.ins_sum;**

Nakon uklanjanja tabele, uklanjaju se i trigeri.

PRIMER (za samostalni rad): Pogledajmo primer sledećeg trigera napisanog iz komandne linije, koji u sebi ima blok BEGIN...END u okviru koga su naredbe koje se završavaju sa; Stoga se na početku definiše novi delimiter kao |.

```
mysql>CREATE TABLE test1(a1 INT);
mysql>CREATE TABLE test2(a2 INT);
mysql>CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
mysql>CREATE TABLE test4(
a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
b4 INT DEFAULT 0
);

mysql> delimiter |
mysql> CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
INSERT INTO test2 SET a2 = NEW.a1;
DELETE FROM test3 WHERE a3 = NEW.a1;
UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END;
|

mysql>delimiter ;

mysql>INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL), (NULL);

POGLEDAJTE i dobro upamtite SADRZAJ TABELE SA SELECT * FROM test3;


mysql>INSERT INTO test4 (a4) VALUES
(0), (0), (0), (0), (0), (0), (0), (0), (0), (0);

POGLEDAJTE i dobro upamtite SADRZAJ TABELE SA SELECT * FROM test4;
```

Interaktivno pokrenimo ubacivanje vrste u tabelu test1:

```
mysql> INSERT INTO test1 VALUES
-> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8 Duplicates: 0 Warnings: 0
```

Rezultat u sve četiri tabele:

```
mysql> SELECT * FROM test1;
+------+
| a1 |
+------+
| 1 |
| 3 |
| 1 |
| 7 |
| 1 |
| 8 |
| 4 |
| 4 |
+------+
8 rows in set (0.00 sec)
mysql> SELECT * FROM test2;
+------+
| a2 |
+------+
| 1 |
| 3 |
| 1 |
| 7 |
| 1 |
| 8 |
| 4 |
| 4 |
+------+
8 rows in set (0.00 sec)
mysql> SELECT * FROM test3;
+----+
| a3 |
+----+
| 2 |
| 5 |
| 6 |
| 9 |
| 10 |
+----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM test4;
+----+------+
| a4 | b4 |
+----+------+
| 1 | 3 |
| 2 | 0 |
| 3 | 1 |
| 4 | 2 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 1 |
| 9 | 0 |
| 10 | 0 |
+----+------+
10 rows in set (0.00 sec)
```

Posto je u tabeli test1 a1=1 tri puta, onda se b4 za vrstu 1 uveca za 3.
Posto je u tabeli test1 a1=2 nula puta, onda se b4 za vrstu 2 uveca za 0.
…
Posto je u tabeli test1 a1=4 dva puta, onda se b4 za vrstu 4 uveca za 2.
…

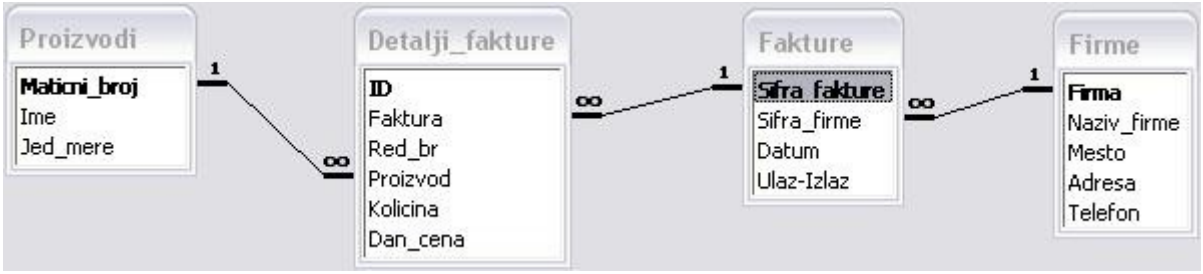
PRIMER: Pogledajmo primer sledećeg trigera napisanog iz komandne linije, koji u sebi ima blok BEGIN...END u okviru koga su naredbe koje se završavaju sa;
Stoga se na početku definiše novi delimiter kao //.
Sam triger treba da pri ažuriranju podataka u tabeli account proveri novu vrednost NEW.amount i modifikuje ih tako da budu u rasponu od 0 do 100.
Ovakav triger mora biti BEFORE triger jer vrednosti moraju da se provere pre nego što se upišu u tabelu:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
-> FOR EACH ROW
-> BEGIN
-> IF NEW.amount < 0 THEN
-> SET NEW.amount = 0;
-> ELSEIF NEW.amount > 100 THEN
-> SET NEW.amount = 100;
-> END IF;
-> END;//
mysql> delimiter ;
```

Ukoliko postoji procedura koja bi trebalo da se ponovi u okviru više različitih trigera, možemo da napravimo uskladištenu proceduru u kojoj su sadržani ti koraci, a da se ova procedura poziva iz različitih trigera na različitim tabelama.

PRIMERI

Sada cemo da kreiramo trigere u našoj bazi poslovanje sa prošlog časa.
Baza podataka **poslovanje** koja služi za praćenje poslovanja jedne firme koja radi sa određenom grupom proizvoda i sarađuje sa određenim brojem drugih firmi.
Baza podataka **poslovanje** ima četiri tabele i to: priozvodi, firme, fakture i detalji_fakture.



Najpre napravite kopiju postojeće baze poslovanje, jer ćemo menjati neke tabele i kolone a takođe ćemo menjati i postojeće podatke. Novu bazu zovimo poslovanje2.

Zašto ćemo napraviti izmene? Zato što nam postojanje stranih ključeva, ukoliko nije postavljeno na opciju CASCADE ne dozvoljava da napravimo trigere koji se odnose na kaskadno ažuriranje i/ili brisanje podataka.

Napravite sledeće izmene u bazi poslovanje 2:

1. U tabeli firme dozvolite da kolona telefon ima Null vrednost, a potom izbrišite sve brojeve telefona u ovoj tabeli (nije obavezno brisanje).

2. U tabeli fakture uklonite strani ključ nad kolonom sifra_firme.
3. U tabeli fakture dozvolite da kolona datum bude Null.

Napravićemo nekoliko trigera.

Primer 1:
Za početak napravimo triger nad tabelom fakture koji pri insertovanju novog podatka u kolonu datum postavlja trenutni datum.U okviru tabele fakture biramo Triggers i desnim tasterom miša otvaramo nov triger

Biramo tip trigera BEFORE, a za događaj Insert.U delu za definisanje izbrišite Begin i End pa ukucajte SET NEW.datum = CURDATE().

Na kartici DDL možete videti kako izgleda definicija celog trigera

CREATE TRIGGER *`fakture_before_ins_tr`* BEFORE INSERT ON *`fakture`*
FOR EACH ROW
SET NEW.*datum* = CURDATE();

Možete i interaktivno uneti trigger.

Iskompajlirajte triger, nakon čega se on pojavljuje kao objekat baze.
Da bi videli kako radi, ubacićemo nov zapis u tabelu firme i u tabelu fakture.

Najpre u tabelu firme unesite firmu sa sifrom 20, ime PROBNA, adresa nepoznata a mesto Beograd.
INSERT INTO *firme (firma, naziv_firme, mesto, adresa)*
VALUES (*20, 'PROBNA', 'Beograd', 'nepoznata'*);

U tabelu fakture unesite podatak o jednoj fakturi koja se odnosi na unetu firmu, na primer
INSERT INTO *fakture (sifra_fakture,sifra_firme,datum,ulaz_izlaz)*
VALUES (*30, 20, ' 2009-11-23', '1'*);

Upišimo podatak a potom pritisnite taster post edit – chekirano(ili refresh) i videćete kako se datum promenio u današnji datum.Sličnu akciju smo mogli proizvesti postavljanjem default vrednosti nad kolonom datum, međutim ima razlike.Ovde je upisan današnji datum bez obzira što smo mi uneli *2009-11-23*. Ovo ima smisla ukoliko postoji pravilo kojim se reguliše da se datum unosa fakture mora poklapati sa datumom na fakturi.

Primer 2:
Napravimo sada triger nad tabelom firme koji će svaki put kada obrišemo neku firmu iz tabele firme izbrisati i sve fakture te firme iz tabele fakture. Ovaj se triger vremenski vezuje za AFTER (posle) događaja brisanja.

CREATE TRIGGER *`firme_after_del_tr`* AFTER DELETE ON *`firme`*
FOR EACH ROW
DELETE FROM *fakture* WHERE *fakture.sifra_firme = old.firma;*

Sada obrišimo firmu sa šifrom 20 (to je firma PROBNA, koji smo maločas uneli i za koju smo uneli i jednu fakturu).
DELETE FROM *firme* WHERE *firma=20*;

U tabeli firme više ne postoji PROBNA

Međutim, ako otvorite tabelu fakture, ni tamo neće postojati faktura vezana za ovu firmu, a koju je izbrisao triger koji se aktivirao prilikom brisanja zapisa iz tabele firme
Naravno,postojanje stranih ključeva omogućiće da se kaskadno brišu podaci upravo onako kako je naš triger uradio.

Primer 3:
Napravimo sada triger koji će omogućiti da se pri promeni šifre firme automatski izvrši odgovarajuća izmena u tabeli fakture

delimiter //
CREATE TRIGGER *`firme_after_upd_tr`* AFTER UPDATE ON *`firme`*
FOR EACH ROW
BEGIN
IF *old.firma != new.firma*
THEN
UPDATE *fakture*
SET *fakture.sifra_firme=new.firma*
WHERE *fakture.sifra_firme=old.firma;*
END IF;
END;//
Probajte izmenu određene šifre firme i pogledajte izmene u tabeli fakture.

Kreiranje virtuelne tabele – "pogleda"

Osnovne tabele – fizički postoje na disku, Pogled – virtuelna (izvedena) tabela i nastaje kao rezultat upita
Pogled je objekt koji se ponaša kao tabela, međutim on sam ne sadrži podatke. To je sačuvani upit koji se svaki put mora da pokrene da bi prikazao određene podatke.Pogledi se koriste kada želimo da se:
1. Izdvoji skup redova
2. Izdvoje određene kolone tabele
3. Spoje povezani redovi iz više tabela
Pogledi, uključujući i poglede koji dozvoljavaju ažuriranje, su implementirani u MySQL Server 5.0.

Sintaksa za kreiranje novog ili zamenu (REPLACE) postojećeg pogleda je:
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { *user* | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW *view_name* [(*column_list*)]
AS *select_statement*
[WITH [CASCADED | LOCAL] CHECK OPTION]

Sam *select_statement* je SELECT iskaz koji omogućava definisanje pogleda. Njime se selektuju podaci iz baznih tabela ili iz drugih pogleda. Da bi se izvršilo kreiranje pogleda potrebno je da korisnik ima privilegiju za kreiranje, dok za korišćenje pogleda korisnik treba da ima privilegiju selektovanja podataka iz kolona koje se pojavljuju u SELECT iskazu.

Sami pogledi su objekti baze koji se u njoj i čuvaju. Pogledi i bazne tabele treba da imaju različite nazive. Sami pogledi, kao i tabele moraju da imaju jedinstvene nazive kolona.

Podrazumevano, nazivi kolona koji se koriste u SELECT iskazu postaju nazivi kolona pogleda, sem ako im se ne dodele drugi nazivi pomoću aliasa. Ukoliko se želi da se eksplicitno definišu nazivi za kolone u pogledu, oni se mogu navesti u *column_list* klauzuli, pri čemu se razdvajaju zarezima. Broj naziva kolona u ovoj listi mora da se poklapa sa brojem kolona koje se izdvajaju SELECT iskazom. Osim kolona baznih tabela, u okviru SELECT iskaza mogu se pojaviti izrazi koji koriste funkcije, operatore, itd. Pogledi imaju sledeća ograničenja:
• SELECT iskaz ne može sadržati podupit u FROM klauzuli
• SELECT iskaz ne može da referencira promenljivu ili parametre procedura
• Nazivi tabela koji se navode u definiciji pogleda moraju unapred da postoje
• Pogledu se ne može dodeliti triger

DEFINER i SQL SECURITY klauzule određuju koji će se koncept sigurnosti sprovoditi pri proveri privilegija za dati pogled. Podrazumevana vrednost za DEFINER je korisnik koji je kreirao pogled. SQL SECURITY određuje koji će se MySQL nalog koristiti kada se proveravaju privilegije koje korisnik ima pri korišćenju pogleda.
Opciona klauzula ALGORITHM može imati jednu od tri vrednosti: MERGE, TEMPTABLE, ili UNDEFINED.
Ovaj algoritam se odnosi na to kako MySQL procesuira poglede. Ukoliko se koriste agregatne funkcije, izbor će biti UNDEFINED.
Neki pogledi mogu ažurirati podatke u baznim tabelama, odnosno mogu se koristiti u odredbama kao što su UPDATE, DELETE ili INSERT, ali samo ako postoji 1-prema-1 veza sa kolonama u baznoj tabeli. Pogledi koji u okviru SELECT iskaza sadrže:
• Agregatne funkcije (SUM(), MIN(), MAX(), COUNT(), itd.)
• DISTINCT
• GROUP BY
• HAVING
• UNION ili UNION ALL
• Podupite
• Spojeve (JOIN)
• Podupit u WHERE klauzuli koji se poziva na tabelu iz FROM klauzule
• ALGORITHM = TEMPTABLE (korišćenje privremenih tabela)
u principu se ne mogu ažurirati.

Generalno, ukoliko se želi da neki pogled omogućava i ažuriranje, onda ga treba generisati nad jednom tabelom. WITH CHECK OPTION klauzula može biti data za poglede koji ažuriraju podatke da bi se izbeglo umetanje ili ažuriranje podataka sem onih za koje je uslov u WHERE klauzuli tačan.

Primer: Kreiraćemo pogled nad već navedenom tabelom Magacin (najjednostavnijim oblikom SELECT naredbe) koji će prikazivati sve podatke koji se nalaze u istoimenoj tabeli sledećom naredbom:
CREATE VIEW SPISAK AS SELECT * FROM Magacin;

Prednosti pogleda

-Jednostavnost korišćenja, uprošćavaju se upiti
-Tajnost, mehanizam za kontrolu pristupa podacima (korisnik vidi samo neke podatke)
-Performanse, definicija pogleda se čuva u kompajliranom, prevedenom obliku
-Nezavisnost podataka, menjaju se definicije pogleda, a ne aplikacije koji koriste podatke iz BP preko pogleda
-Pogled predstavlja jednu vrstu potprograma u SQL-u.
-Jednom kreiran može da se koristi u podupitima, u WHERE i HAVING klauzulama
-Zamenjuje komplikovane upite, jer komplikovani i često korišćeni upiti se mogu formulisati u vidu pogleda (poziv tipa SELECT * FROM ImePogleda;)
-Pogled olakšava kontrolu pristupa BP

Primer 1: Pogled koji ograničava pristup tabeli Naslov na redove sa šifrom ‘PJ’
```
CREATE VIEW NaslovPJ
        AS SELECT *
                FROM Naslov
                WHERE SifO=‘PJ’;
```

Primer 2: Pogled koji ograničava pristup tabeli Naslov na kolone SifN i Naziv
```
CREATE VIEW Naslov1
        AS SELECT SifN, Naziv
                FROM Naslov;
```

Pogledi u već spominjanom primer sa fudbalerima
Data je šema relacione baze podataka fubalskogsaveza za potrebe evidencije utakmica jedne sezone (pretpostavka je da fudbaleri ne mogu da menjaju tim u kome igraju, u toku sezone):
FUDBALER (SifF, Ime, SifT)
TIM (SifT, Naziv, Mesto)
UTAKMICA (SifU, SifTDomaci, SifTGost, Kolo, Ishod, Godina)
IGRAO (SifF, SifU, PozicijaIgraca)
GOL (SifG, SifU, SifF, RedniBrGola, Minut)
KARTON (SifK, SifU, SifF, Tip, Minut)
Napomena: Ishod(1-pobeda domaćih, 2-pobeda gostiju, X-nerešeno)

Primer3. Sastaviti SQL skript koji kao rezultat daje šifre i imena fudbalera kao i prosečan broj golova po utakmici koji su ti fudbaleri dali, ali samo za one fudbalere koji su dali bar jedan gol nakon što su dobili žuti karton, i to u utakmici gde njihov tim bio gostujući.
Primer4. Sastaviti SQL skript koji kao rezultat daje šifre i imena fudbalera, ali samo za one fudbalere koji su dali bar dva gola i to na nekoj od utakmica na kojoj je njihov tim pobedio, i pri tom su tačno dva gola tog fudbalera bili ujedno i poslednji golovi na utakmici.

Rešenje:
```
CREATE VIEW BrUtakmica(SifF, BrUtakmica)
AS SELECT SifF, COUNT(*) FROM IGRAO GROUP BY SifF;
```

```
CREATE VIEW BrGolova(SifF, BrGolova)
AS SELECT SifF, COUNT(*) FROM GOL GROUP BY SifF;

CREATE VIEW DaliGolNakonKartona(SifF)
AS SELECT SifF
FROM FUDBALER F, UTAKMICA U, GOL G, KARTON K
WHERE F.SifF=G.SifF
AND F.SifF=K.SifF
AND G.Minut> K.Minut
AND G.SifU=K.SifU
AND G.SifU=U.SifU
AND F.SifT=U.SifTGost;
--Nije gledan tip kartona, jer se samo nakon žutog može nastaviti i dati gol

SELECT F.SifF, F.Ime, BG.BrGolova/ BU.BrUtakmica
FROM FUDBALER F, BrGolova BG, BrUtakmica BU
WHERE F.SifF=BG.SifF AND F.SifF=BU.SifF
AND F.SifF IN (SELECT SifF FROM DaliGolNakonKartona);
--Poslednji uslov je tu da bi dobili samo one igrače koji su od interesa

4.
CREATE VIEW Uslov(SifF, SifU)
AS SELECT F.SifF,U.SifU
FROM FUDBALER F, UTAKMICA U, GOL G
WHERE F.SifF=G.SifF
AND G.SifU=U.SifU
AND ( (F.SifT=U.SifTDomaci AND Ishod='1')
          OR  (F.SifT=U.SifTGost AND Ishod='2')
       )
AND NOT EXISTS (SELECT SifF
                          FROM GOL L
                          WHERE L.SifF<> F.SifF
                          AND L.SifU=U.SifU
                          AND L.Minut> G.Minut
                          )
GROU BY F.SifF, U.SifU
HAVING COUNT(*)=2;

SELECT DISTINCT F.SifF, F.Ime
FROM FUDBALER F, Uslov U
WHERE F.SifF=U.SifF;
```

Primer 5:
Data je šema relacione baze podataka, za potrebe izračunavanja pomoću matrica:
MATRICA(SifM, Naziv, BrVrsta, BrKolona);
PODACI(SifP, I, J, Vrednost, SifM);
Sastaviti SQL skript koji vraća vrstu, kolonu i vrednost svakog od elemenata matrice nastale množenjem matrica sa nazivima A i B.

Rešenje

```
SELECT T2.I, T4.J, SUM(T2.Vrednost * T4. Vrednost) #T2.I su vrste iz A, T4.J su kolone iz B
FROM Matrica T1, Podaci T2, Matrica T3, Podaci T4
```

```
WHERE T1.SifM= T2.SifM AND T1.Naziv = 'A'    # T1 je matrica A, T2.Vrednost je njen podatak
AND T3.SifM= T4.SifM AND T3.Naziv = 'B' # T3 je matrica B, T4.Vrednost je njen podatak
AND T2.J = T4.I                  # T2.J = T4.I = k, jer se množi a[i][k]*b[k][j]
GROUP BY T2.I, T4.J;
```

Podaci	SifP	I	J	Vrednost	SifM
	1	1	2	3	10
	2	2	1	1	10
	3	1	1	2	10
	4	2	2	2	10
	5	1	3	1	20
	6	1	1	3	20
	7	2	1	2	20
	8	2	2	5	20
	9	2	3	2	20
	10	1	2	4	20
	11	2	1	4	30
	12	1	1	5	30

Matrica	SifM	Naziv	BrVrsta	BrKolona
	10	'A'	2	2
	20	'B'	2	3
	30	'C'	2	1

Dakle, izlazni rezultat je:

1 1 12
2 1 7
1 2 23
2 2 14
1 3 8
2 3 5

Primer 6:
Data je šema relacione deo baze podataka fakulteta:

STUDENT (SifS, Ime, BrIndeksa)
PROFESOR (SifP, Ime, SifO)
ODSEK (SifO, Naziv)
KURS (SifK, Naziv, BrKredita, SifO)
UČIONICA (SifU, BrMesta)
PREDUSLOV (SifK, SifKP)
POHAĐA(SifS, SifR) # sifra studenta, sifra u rasporedu
RASPORED (SifR, SifP, SifK, SifU, Termin, Dan, Br.Prijavljenih)

Sastaviti SQL skript koji proverava kvalitet rasporeda studenata i pored broja indeksa ispisuje i odgovarajuću poruku. Raspored studenta je loš ukoliko u svom rasporedu bar jednog dana ima prekid u terminima u kojima prati predavanje, u suprotnom raspored se smatra dobrim. Za sve studente sa lošim rasporedom treba ispisati broj indeksa i poruku LOS, a pored broja indeksa studenata sa dobrim rasporedom poruku DOBAR.

Rešenje:

```
CREATE VIEW LosRaspored(SifS, Dan)
AS SELECT P.SifS, R.Dan
FROM POHADJA P, RASPORED R
WHERE P.SifR=R.SifR
GROUP BY P.SifS, R.Dan
HAVING COUNT(R.Termin) < (MAX(R.Termin)-MIN(R.Termin)+1);

SELECT S.BrIndeksa, 'LOS'
FROM STUDENT S
WHERE S.SifS IN (SELECT SifS FROM LosRaspored)
UNION
SELECT S.BrIndeksa, 'DOBAR'
FROM STUDENT S
WHERE S.SifS NOT IN (SELECT SifS FROM LosRaspored)
AND S.SifS IN (SELECT SifS FROM POHADJA);
```

Primer 7:
Data je šema relacione baze podataka za potrebe skladišta robe u toku jedne godine:
ROBA(SifR, Naziv, Opis, SifD);
DOBAVLJAC(SifD, Naziv, Adresa);
NABAVKA(SifN, Datum, Kolicina, Cena, SifR);

Sastaviti SQL skript koji vraća cenu pri kojoj bi trebalo prodati 40 jedinica robe sa šifrom 30, ukoliko te robe ima u dovoljnoj količini. Cena se formira tako što se najpre rasprodaju količine koje su najskorije nabavljene (LIFO).

Nabavka	SifN	Datum	Kolicina	Cena	SifR
	10	1	20	1	30
	20	1	10	2	10
	30	2	10	1	20
	40	2	20	2	20
	50	2	30	1	10
	60	5	10	1	30
	70	6	10	2	20
	80	6	20	2	10
	90	6	30	2	20
	100	8	20	1	30
	110	8	10	1	10
	120	9	10	2	30

Rešenje:

Za dati sadržaj, rezultat ukupne cene prodaje robe sa sifrom 30 je : 50 (tj. 10*2+20*1+10*1)

```
CREATE VIEW LIFO (Proizvod, Datum, JedCena, TotalKol, TotalCena)
AS SELECT R1.SifR, R1.Datum, R1.Cena, SUM(R2.Kolicina), SUM(R2.Kolicina*R2.Cena)
FROM Nabavka R1, Nabavka R2
WHERE R2.Datum >= R1.Datum AND R2.SifR= R1.SifR
GROUP BY R1.SifR, R1.Datum, R1.Cena;

SELECT (TotalCena–((TotalKol–40)*JedCena))
FROM LIFO
```

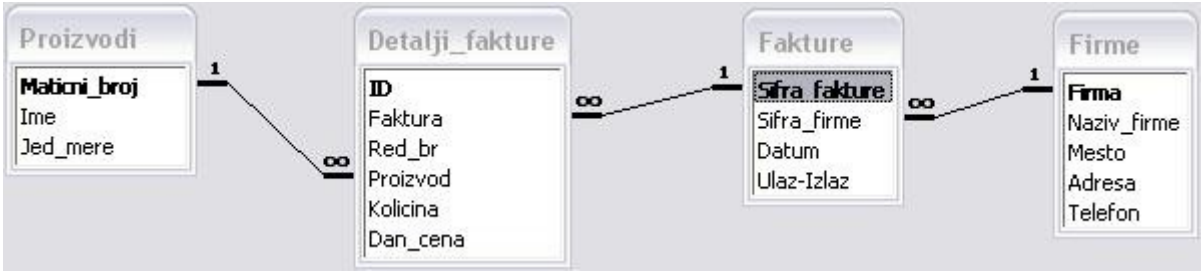
```
WHERE Proizvod = 30
AND Datum = ( SELECT MAX (Datum)
FROM LIFO
WHERE TotalKol>= 40
AND Proizvod =30);
```

Primer 8:
Data je šema relacione baze podataka veleprodajnog lanca prodavnica:
PRODAVNICA(SifP, Adresa, SifM);
MESTO(SifM, Naziv);
KLIJENT(SifK, Naziv, SifM);
RACUN(SifR, SifK, SifP, SifRa, Datum);
PROIZVOD(SifPr, Naziv, Cena);
STAVKA_RACUNA(SifS, SifR, SifPr, RedniBr, Kolicina, Iznos);
RADNIK(SifRa, Ime, SifP);
Sastaviti SQL skript koji za svaki datum, za koji je izdat bar jedan račun, daje ukupan iznos računa izdatih do tog datuma (uključujući i posmatrani datum).
Rešenje:

```
CREATE VIEW IznosPoDatumu(Datum, Total)
AS SELECT R.Datum, SUM(S.Iznos)
FROM RACUN R, STAVKA_RACUNA S
WHERE R.SifR=S.SifR
GROUP BY R.Datum;
SELECT A.Datum, SUM(B.Total)
FROM IznosPoDatumu A, IznosPoDatumu B
WHERE A.Datum>= B.Datum
GROUP BY A.Datum;
```

Pogledi u bazi poslovanje

Sada ćemo da kreiramo trigere u našoj bazi poslovanje sa prošlog časa.
Baza podataka **poslovanje** koja služi za praćenje poslovanja jedne firme koja radi sa određenom grupom proizvoda i sarađuje sa određenim brojem drugih firmi.
Baza podataka **poslovanje** ima četiri tabele i to: priozvodi, firme, fakture i detalji_fakture.



Primer 9: Kreirati pogled koji prikazuje šifru fakture, odgovarajuću firmu, datum i tip fakture (imajući na umu primer baze poslovanje sa prethodnog časa):

```
CREATE VIEW v_fakture AS
SELECT fakture.sifra_fakture AS faktura, firme.naziv_firme AS firma,
fakture.datum AS datum,
IF(fakture.ulaz_izlaz = '1', 'ulazna', 'izlazna') AS tip
FROM fakture, firme
WHERE fakture.sifra_firme = firme.firma;
SELECT * FROM v_fakture;
```

Primer 10: Kreirati pogled koji prikazuje šifru fakture, naziv firme, datum, tip fakture (ulazna ili izlazna), kao i sumu stavki na fakturi.

```
CREATE VIEW suma_na_fakturi AS
SELECT detalji_fakture.faktura AS faktura,
firme.naziv_firme AS naziv_firme,
fakture.datum AS datum,
IF(fakture.ulaz_izlaz = '1', 'ulazna','izlazna') AS tip,
SUM(detalji_fakture.kolicina * detalji_fakture.dan_cena) AS iznos
FROM detalji_fakture, firme, fakture
WHERE firme.firma = fakture.sifra_firme AND fakture.sifra_fakture = detalji_fakture.faktura
GROUP BY detalji_fakture.faktura;
SELECT * FROM suma_na_fakturi;
```

Primer 11: Naći firmu za koju je fakturisana faktura sa najvećim iznosom. U ovom primeru će biti iskorišćen već postojeći pogled suma_na_fakturi:

```
CREATE VIEW najveca_faktura AS
SELECT e1.naziv_firme AS naziv_firme, e1.faktura AS faktura, e1.iznos AS iznos
FROM suma_na_fakturi AS e1
WHERE e1.iznos = (SELECT MAX(suma_na_fakturi.iznos) AS maxiznos
FROM suma_na_fakturi);
SELECT * FROM najveca_faktura;
```

Sintaksa za brisanje pogleda glasi:

```
DROP VIEW [IF EXISTS] view_name [, view_name] ...
 [RESTRICT | CASCADE]
```

Izmena pogleda u bazi podataka

Funkcija izmene pogleda u bazi podataka putem SUBP-a se odnosi na izmenu pogleda koji je prethodno kreiran u bazi podataka.

Sintaksa za izmenu pogleda glasi:

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Primer: Primenićemo gore navedenu naredbu da bi promenili pogled pod nazivom SPISAK tako da prikazuje sve iz tabele Dobavljač, a ne Magacin kako je bilo u prethodnom primeru.

```
ALTER VIEW SPISAK AS SELECT * FROM Dobavljac;
```