*Dr. Šimon Janoš dipl. ing.*

*Visoka Tehnička Škola*
*Strukovnih Studija*
*Subotica*

# *PRIRUČNIK ZA VEŽBE IZ BAZE PODATAKA II*

Subotica - 2014

*"Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it."*

***Steve Jobs***

# SADRŽAJ

# PREDGOVOR

SQL je formalni jezik kojim se pišu programi za pravljenje i menjanje baza podataka, kao i zadavanje upita nad bazama podataka. Naš sistem baze podataka izvršava SQL program, obavlja zadatke koje smo naveli i prikazuje rezultate (ili poruku o grešci). Programski jezici se razlikuju od prirodnih (govornih) jezika po tome što su programski jezici projektovani za određenu namenu, imaju mali rečnik, nefleksibilni su i krajnje nedvosmisleni. Stoga, ako ne dobijemo rezultate koje očekujemo, to je zato što naš program ima grešku, a ne to što je računar pogrešno protumačio instrukcije.

SQL je, kao i svaki formalni jezik, definisan sintaksnim pravilima, koja određuju reči i simbole koje možemo da koristimo i načine na koje ih možemo kombinovati, i semantičkim pravilima, koja određuju pravo značenje sintaksno ispravnog iskaza.

Priručnik je namenjen studentima Visoke tehničke škole strukovnih studija u Subotici na studijama informatike. Omogućuje lakše savlađivanje gradiva iz predmeta baze podataka II. Izloženi materijal obuhvata gradivo prema akreditovanom planu i programu predmeta i bazira se na materijalu sa vežbe. Priručnik se pojavljuje u štampanom obliku, u skriptarnici Visoke tehničke škole strukovnih studija u Subotici.

*Dr. Šimon Janoš dipl. ing.*

# UVOD

SQL predstavlja standardni programski jezik za pravljenje, ažuriranje i uzimanje informacija koje su sačuvane u bazama podataka. Pomoću SQL-a, obična pitanja možemo pretvoriti u iskaze koje naš sistem baze podataka može da razume. SQL takođe možemo da koristimo za dodavanje, izmenu i brisanje podataka i objekata baze podataka. Svi moderni sistemi za upravljanje relacionim bazama podataka (engl. database management systems, DBMS) podržavaju SQL mada sama podrška zavisi od proizvoda. Priručnik je organizovan u 14 poglavlja i sadržaj prati plan i program predmeta baze podataka 2 sa nedeljnim fondom časova 2+2. Poglavlja obuhvataju material sa vežbi prema sledećem:

**Poglavlje 1:** Obuhvata upoznavanje okruženja WAMP, podešavanje radnog okruženja, SQLBuddy, phpMyAdmin, Rad u MySQL konzoli (korisnici / baze podataka), Import / Export podataka iz MySQL konzole.

**Poglavlje 2:** Naredbe za šemu baze podtaka (CREATE, ALTER, DROP) u MySQL okruženju, Naredbe za podatke (SELECT, INSERT, UPDATE,DELETE), Konekcija sa MySQL bazom pomoću PHP okruženja, Import / Export podataka pomoću phpMyAdmin.

**Poglavlje 3:** Rad sa HTML formama, listanje i unos podataka u MySQL bazu pomoću PHP.

**Poglavlje 4:** Uskadištene procedure (engl. STORED PROCEDURES).

**Poglavlje 5:** Pogledi (engl. VIEW).

**Poglavlje 6:** Transakcije (BEGIN TRANSACTION, COMMIT, ROLLBACK).

**Poglavlje 7:** Indeksiranje i pretraga podataka.

**Poglavlje 8:** Složeni podupiti, okidači (engl. TRIGGERS).

**Poglavlje 9:** Tehnike sortiranja podataka, napredne tehnike pretraživanja.

**Poglavlje 10:** MySQL Workbench okruženje, obrnuti inženjering (engl. REVERSE ENGINEERING).

**Poglavlje 11:** XML struktura podataka i upiti.

**Poglavlje 12:** PostgreeSQL okruženje.

SQL ne predstavlja vlasništvo neke firme. To je otvoreni standard koji je definisala radna grupa za međunarodne standarde, koja je pod zajedničkim vodstvom međunarodne organizacije za standardizaciju (International Standardization for Organization, ISO) i međunarodnog konzorcijuma inženjera (International Engineering Consortium, IEC). Američki nacionalni institut za standardizaciju (American National Standardization Institute, ANSI) je učestvovao u radnim grupama i ratifikovao standard.

# 1. POGLAVLJE

## Okruženje WAMP

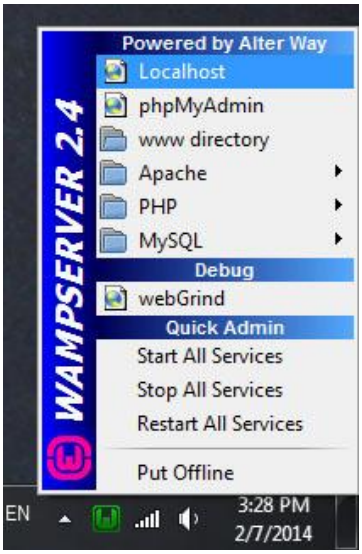
WAMP je besplatni program pod GPL licencom, koji olakšava instalaciju najnovijih verzija većine potrebnih programa za web programiranje ili aplikacije koje za svoj rad trebaju PHP i MySQL. Instalacijom programa WAMP na računalu ćemo moći koristiti sledeće alate:

- Apache - Web server
- PHP5 - programski jezik
- MySQL - baza podataka
- PHPmyadmin - web aplikacija za jednostavno administriranje baze podataka
- SQLitemanager - jednostavna verzija baze podataka, za vrlo malo podataka

Dakle, čemu WAMP služi? Ukratko, WAMP nam omogućava da na svom računaru simuliramo rad servera, i da za naš sajt, prezentaciju ili šta god nameravamo da postavimo na internet proverimo sve funkcionalnosti koje smo u njega implementirali, i uverimo se da sve radi kako treba.

## Koraci instalacije

Instalacija programa može se preuzeti sa web adrese:
http://www.wampserver.com/en/download.php
Nakon pokretanja instalacijske datoteke, sama instalacija se sastoji od nekoliko jednostavnih standardnih koraka. Posle uvodnog prozora koja pokazuje verzije pojedinih komponenti, sledi korak gde se ispisuje licenca i uslovi korišćenja programa. Zatim se nudi mogućnost odabira mape u koju se želi sve instalirati. Obično je C:\wamp. Sama instalacija traje vrlo kratko, a status instalacije moguće je pratiti kroz pomicanje statusne trake. Nakon što je sama instalacija završena, potrebno je podesiti još nekoliko opcija. Potrebno je upisati ime servera (*localhost*) i naziv mail servera putem kojeg će se PHP slati e-mail poruke ako se koristi odgovarajuća funkcija. Pošto smo završili i taj tehnički deo instalacije, pokrenimo WAMP klikom na *Start/All Programs/WampServer/Start WampServer* ili duplim klikom na ikonicu ako smo je postavili na desktop. Sačekajmo par sekundi, da WAMP ikonica koja će se pojaviti u donjem desnom uglu promeni boju iz crvene u zelenu, i WAMP server je tada pokrenut.




*Sl. 1.1 Wamp server 2.4*

Kao što možemo videti, u meniju se nalaze sve komponente WAMP-a koje smo gore opisali, a nas će za sad da interesuje samo *localhost*, koji se nalazi na vrhu menija. Ukoliko je aktivna zelena ikonica, znači da su svi servisi ispravno instalirani i da ne postoji problem u radu WAMP servera Ukoliko je aktivna crvena ikonica, znači da je server trenutno neaktivan i da bi smo ga trebali pokrenuti pomoću ikonice na desktopu koju smo postavili prilikom instalacije Ukoliko je aktivna narandžasta ikonica to znači da jedan od servisa nije podignut. Naime, WAMP radi instalaciju tri glavna servisa, između ostalog i Apache, koji će koristiti port 80. Taj port mogu koristiti i drugi serveri, a najčešće se dogodi da upravo taj port koristi i Skype. Ukoliko imate Skype na svom računaru, vrlo je verovatno da baš on pravi problem.

## Podešavanje okruženja

Podešavanja vezana za sam web server se vrše unutar datoteke *httpd.conf*. Php jezik se podešava u okviru datoteke *php.ini*, a MySQL ima konfiguracioni fajl poda nazivom *my.ini*. Ovde ćemo navesti samo nekoliko bitnih elemenata. Prvo podešavanje je u okviru datoteke *php.ini*.

*max_execution_time = 30*

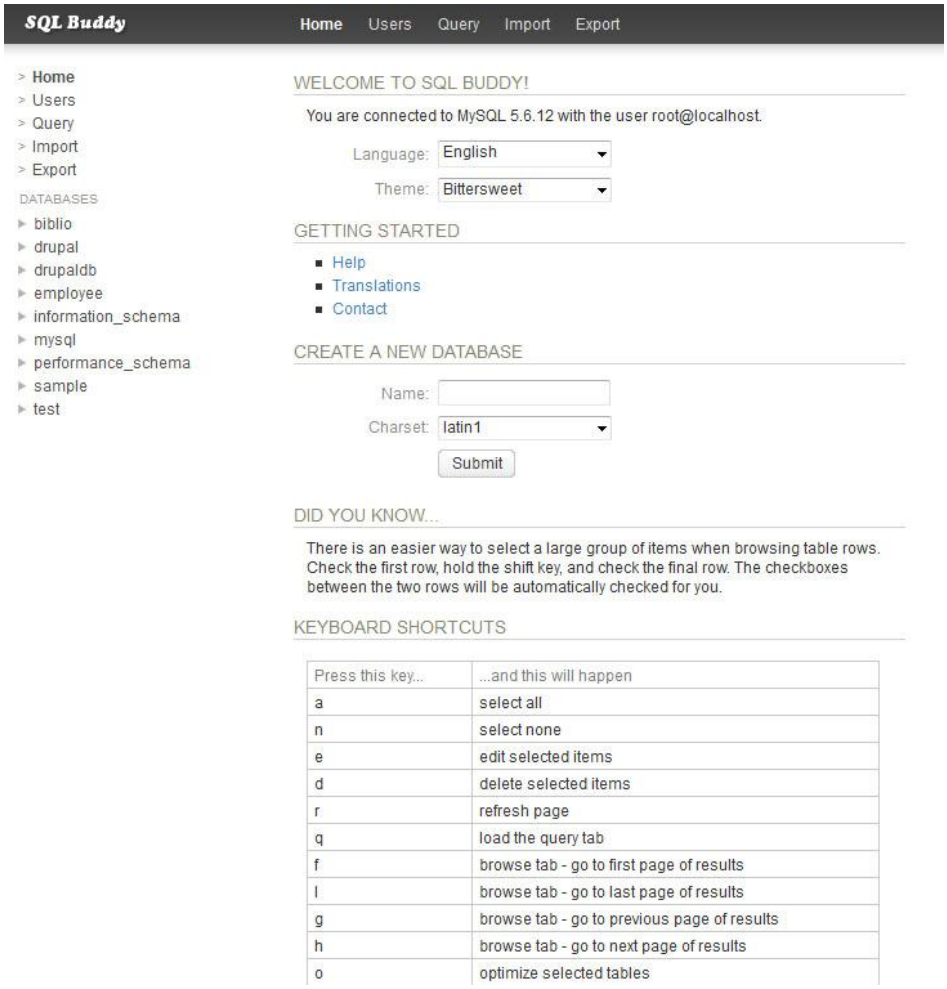
U slučaju da želimo na svom serveru instalirati neku web2.0 aplikaciju kao što je npr. Drupal, vreme izvršavanja ćemo morati da povećamo na 120. U suprotnom će server prekinuti aplikaciju pre nego što se ona završi. Druga bitna promena će biti vezana za određivanje maximalne veličine baze podataka koju možemo koristiti.

*upload_max_filesize = 2M*
*post_max_size = 8M*
*memory_limit = 128M*

Ove vrednosti ćemo menjati u zavisnosti od naših potreba. U slučaju da želimo postaviti bazu veličine 20MB, samim tim i ograničenja moramo podesiti da sistem dozvoli upload datoteke veće od default vrednosti.

## SQLBuddy

SQL Buddy je atraktivan i jednostavan softver koji nam omogućuje da lako menjamo ili administriramo baze podataka iz web pretraživača. Možda neće biti popularniji od široko primenjivanog phpMyAdmin-a, kao de facto standardni editor baze podataka među web programerima , ali je brzo postao omiljeni web - baziran alat ovog tipa.




*Sl. 1.2 SQL Buddy*

Instalacija je krajnje jednostavna i sastoji se od kopiranja sadržaja arhive na server u poseban folder. Mogućnosti koje ima za dodavanje korisnika, dodavanje baze podataka su sasvim dovoljni za početnog korisnika.

## phpMyAdmin

phpMyAdmin je besplatan alat napisan u PHP-u i služi za upravljanje MySQL bazama podataka. PHPMyAdmin možemo preuzeti sa službenih stranica www.phpmyadmin.net, a na istoj adresi naći ćemo i svu potrebnu dokumentaciju. Ukoliko smo instalirali WAMP paket, phpMyAdmin ćemo pronaći na sledećoj adresi:

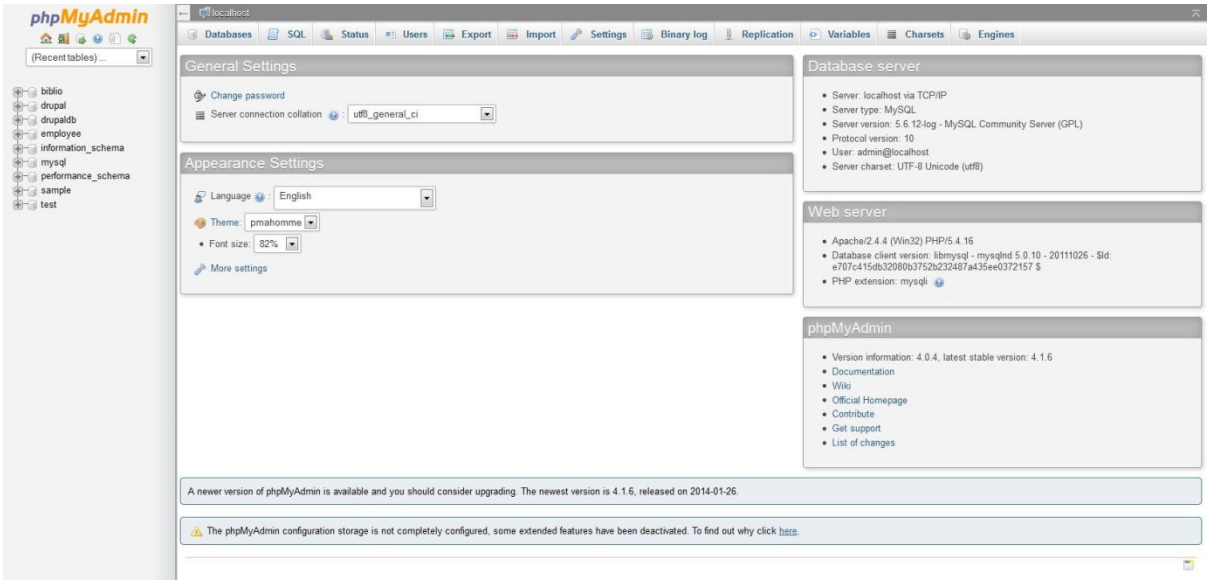
http://localhost/phpmyadmin

Ovaj alat je nezamenjiv kod svakog ozbiljnijeg rada s web aplikacijama koje se oslanjaju na mySQL server. Naime, to je specijalizovani web alat za administraciju mySQL baza. PHPMyAdmin nam omogućuje kompletnu administraciju - izvođenje SQL upita, kreiranje baza, tablica i backup podataka iz koherentnog web interfejsa. Dakle, nije potrebna nikakva instalacija na klijent osim web browsera. Dodatna zanimljivost je i interfejs koje je prevedeno na više jezika. Ovaj alat zaslužio je svaku preporuku jer je kvalitetan rad s PHP/mySQL aplikacijama bez njega naprosto nezamisliv.


*Sl. 1.3 phpMyAdmin login*

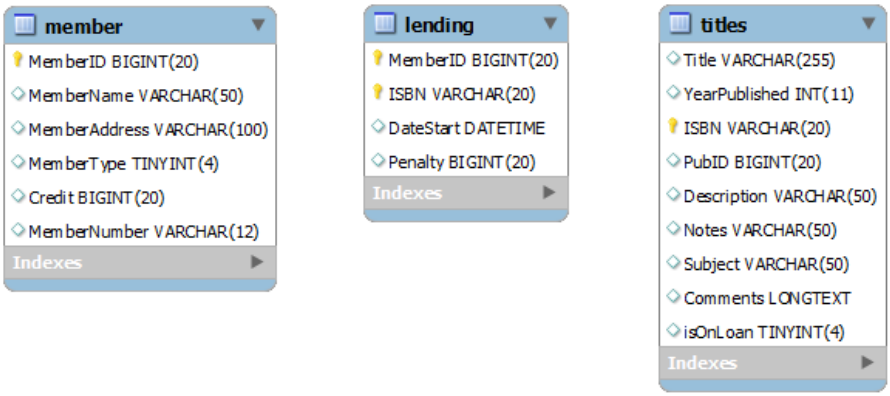
## Kreiranje baze podataka korišćenjem grafičkog korisničkog interfejsa

Pokrenimo phpMyAdmin preko gore prikazane internet adrese. Na početnoj strani u polje "Create new database" unosimo željeno ime baze podataka, npr "sample". Potvrđujemo kreiranje klikom na dugme "Create".




*Sl. 1.4 phpMyAdmin okruženje*

Nakon kreiranja baze podataka, možemo pristupiti i kreiranju tabela i atributa sa odgovarajućim tipom podataka.




*Sl. 1.5 Tabele unutar baze Sample*

Bazu podataka sa odgovarajućim tabelama možemo kreirati pomoću grafičkog interfejsa unutar phpMyAdmin okruženja ili možemo napisati SQL naredbe u priloženom editoru.

```
CREATE TABLE lending (
  MemberID bigint(20) NOT NULL default '0',
  ISBN varchar(20) NOT NULL default '',
  DateStart datetime default NULL,
  Penalty bigint(20) default NULL,
  PRIMARY KEY  (MemberID,ISBN)
);

CREATE TABLE member (
  MemberID bigint(20) default NULL,
  MemberName varchar(50) default NULL,
  MemberAddress varchar(100) default NULL,
  MemberType tinyint(4) default NULL,
  Credit bigint(20) default NULL,
  MemberNumber varchar(12) default NULL,
  PRIMARY KEY  (MemberID)
);

CREATE TABLE titles (
  Title varchar(255) default NULL,
  YearPublished int(11) default NULL,
  ISBN varchar(20) default NULL,
  PubID bigint(20) default NULL,
  Description varchar(50) default NULL,
  Notes varchar(50) default NULL,
  Subject varchar(50) default NULL,
  Comments longtext,
  isOnLoan tinyint(4) default NULL,
  PRIMARY KEY  (ISBN)
);
```

## Rad u MySQL konzoli

MySQL klijent za rad sa komandne linije (mysql) služi za interaktivno izvršavanje SQL iskaza. Konfigurisan je da se priključi na server sa root nalogom, pa se prilikom pokretanja od korisnika zahteva unos lozinke za root nalog ukoliko je definisana. Može se pokrenuti na sledeći način: WAMP → MySQL → MySQL Console. Nakon pokretanja neophodno je uneti lozinku za root nalog i kliknuti na taster Enter. Korisnik može MySQL i SQL komande upisivati direktno u MySQL Monitor.

## Import / Export podataka iz MySQL konzole

Nakon prijave može se videti koje sve baze podataka postoje na serveru korišćenjem komande SHOW.

```
show databases;
```

### Izrada rezervnih kopija pomoću mysqldump

Izradu rezervne kopije (engl. backup) postojeće baze podataka možemo postići pomoću *mysqldump* programa koji se nalazi na sledećoj putanji:

```
C:\wamp\bin\mysql\mysql5.6.12\bin
```

Opšta sintaksa je:

```
mysqldump --opt -u ime_korisnika -p ime_baze > ime_baze.sql
```

ili npr:

```
mysqldump --opt -u admin -p sample > sample_backup.sql
```

U ovom primeru zadali smo samo opciju --opt; ona grupiše nekoliko drugih opcija. Naveli smo ime baze podataka, a rezultat preusmeravamo u datoteku rezervne kopije koju želimo da napravimo.

### Rekonstrukcija baze podataka

Bazu podataka možemo da rekonstruišemo na nekom drugom računaru ako napravimo bazu sa odgovarajućim imenom, i pri tome učitamo datoteku sa rezervnom kopijom podataka.

Opšta sintaksa je:

```
mysql -u ime_korisnika -p ime_baze < ime_baze.sql
```

ili npr:

```
mysql -u admin –p

create database sample2;

\q
```

```
mysql -u admin -p sample2 < sample_backup.sql
```

# Korisnici MySQL baze podataka

### Dodavanje novog korisnika

Dodavanje novog korisnika *mysql* baze se vrši u okviru komande *mysql* koju pokrećemo kao *root* sa sledećim parametrima:

```
mysql –u root –p
```

Opšta sintaksa je:

```
create user 'ime_korisnika'@'localhost' identified by 'lozinka';
```

ili npr:

```
create user 'student'@'localhost' identified by 'student';
```

### Prikazivanje korisnika

```
select user, host from mysql.user;
```

### Brisanje korisnika

Opšta sintaksa je:

```
drop user 'ime_korisnika'@'localhost';
```

ili npr:

```
drop user 'student'@'localhost';
```

### Privilegije korisnika

Opšta sintaksa za dodavanje privilegije je:

```
grant all on ime_baze.* to 'ime_korisnika'@'localhost';
```

ili npr:

```
grant select, insert on sample.* to 'student'@'localhost';
grant all on sample.* to 'student'@'localhost';
```

Opšta sintaksa za brisanje privilegije je:

```
revoke all on ime_baze.* from 'ime_korisnika'@'localhost';
```

ili npr:

```
revoke insert on sample.* from 'student'@'localhost';
revoke all on sample.* from 'student'@'localhost';
```

### Pregled privilegija korisnika

Pregled postojećih privilegija datog korisnika možemo dobiti pomoću sledeće komande:

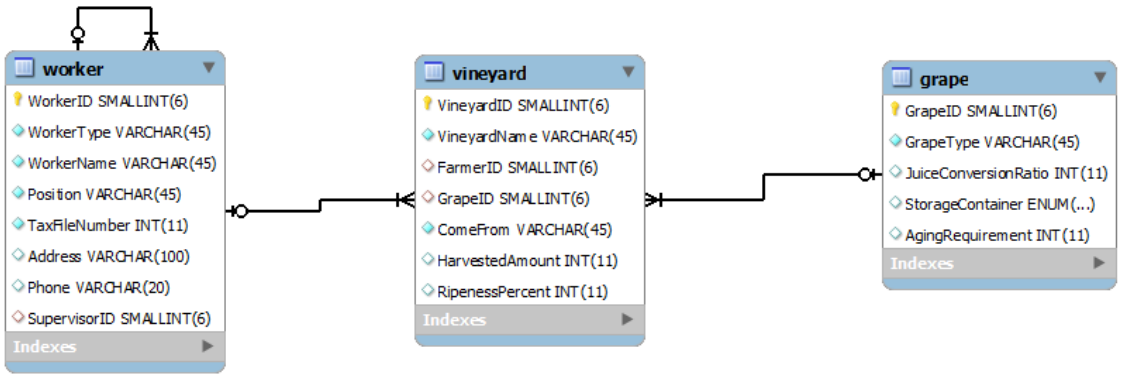
```
show grants for 'student'@'localhost';
```

Iz MySQL monitora korisnik se može odjaviti tako što otkuca \q (slovo q potiče od reči quit). Ova komanda se ne završava znakom tačka i zarez. Postoji grupa komandi koje počinju znakom \ (obrnuta kosa crta ili backslash). Nijedna od njih se ne završava znakom tačka i zarez. Spisak tih komandi se može dobiti ako se otkuca \h (slovo h potiče od reči help).Za napuštanje *mysql* komande potrebno je uneti sledeće:

```
mysql> \q
```

### Kreiranje baze podataka iz konzole

U slučaju da želimo kreirati bazu pod imenom test i stukturom kao što je prikazano na slici, potrebno je iz konzole ulucati sledeće:




*Sl. 1.6 Primer test baze sa stranim ključevima*

```
mysql -u admin -p
create database test;
use test;
```

```
CREATE TABLE Worker (
 WorkerID smallint auto_increment,
 WorkerType  varchar(45) NOT NULL,
 WorkerName  varchar(45) NOT NULL,
 Position    varchar(45) NOT NULL,
 TaxFileNumber int NOT NULL,
 Address    varchar(100) ,
 Phone      varchar(20) ,
 SupervisorID  smallint ,
 PRIMARY KEY (WorkerID),
 FOREIGN KEY (SupervisorID) REFERENCES Worker(WorkerID)
    ON DELETE SET NULL
    ON UPDATE CASCADE
)Engine=InnoDB;

CREATE TABLE Grape (
    GrapeID smallint NOT NULL,
    GrapeType varchar(45) NOT NULL,
    JuiceConversionRatio int,
```

```
    StorageContainer ENUM('Stainless Steel Tank','Oak Barrel'),
    AgingRequirement int,
    PRIMARY KEY (GrapeID)
)Engine=InnoDB;

CREATE TABLE Vineyard (
    VineyardID smallint auto_increment,
    VineyardName VARCHAR(45) NOT NULL,
    FarmerID    smallint,
    GrapeID smallint,
    ComeFrom    varchar(45) NOT NULL,
    HarvestedAmount int,
    RipenessPercent int,
    PRIMARY KEY (VineyardID),
    FOREIGN KEY (FarmerID) REFERENCES Worker(WorkerID)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (GrapeID) REFERENCES Grape(GrapeID)
        ON DELETE SET NULL
        ON UPDATE CASCADE
)Engine=InnoDB;
```

```
drop database test;
```

# 2. POGLAVLJE

## Tipovi podataka u kolonama

U MySQL-u postoje tri osnovna tipa kolona: numerički, znakovni ili tekstualni i datumsko/vremenski. Objasnićemo ih jedan po jedan.

### Numerički tipovi podataka

Numerički tipovi se koriste za skladištenje brojeva. U našem primeru, upotrebili smo tipove INT (celobrojne vrednosti) i FLOAT (vrednosti s pokretnim zarezom).To su dva primera podtipova numeričkih tipova: tačni numerički tipovi i aproksimirani (približni) numerički tipovi. Za numeričke tipove možemo zadati ukupan broj cifara koji se prikazuje (širina, M) i, za tipove s pokretnim zarezom, broj decimalnih mesta, D. Vrednosti tih parametara zadaju se iza deklaracije tipa podatka; na primer:

```
plata decimal(10,2)
```

Ova komanda omogućava prikazivanje vrednosti sa ukupno 12 cifara i dva decimalna mesta. Možemo se opredeliti da ne zadamo nijedan od ovih parametara, ili možemo zadati samo ukupnu širinu za prikazivanje, ili i širinu i broj decimala. Uz numeričke tipove možemo zadati rezervisane reči UNSIGNED i/ili ZEROFILL. Rezervisana reč UNSIGNED znači da kolona može sadržati samo nule ili pozitivne vrednosti. Rezervisana reč ZEROFILL znači da će se vrednosti iz kolone prikazivati s vodećim nulama. Tačni numerički tipovi opisani su u nastavku teksta.

**Tip NUMERIC ili DECIMAL**

Obe reči označavaju potpuno isti tip podataka, a DECIMAL se može skratiti na DEC Ovi tipovi omogućavaju čuvanje tačnih vrednosti s pokretnim zarezom i obično se koriste za rad s novčanim vrednostima. Opseg mogućih vrednosti jednak je kao za brojeve s pokretnim zarezom dvostruke preciznosti.

**Tip INTEGER i varijante**

Ovaj tip se može skratiti na INT. To je standardni tip za celobrojne vrednosti, koje se smeštaju u četiri bajta, što daje $2^{32}$ moguće vrednosti. Tip INT ima nekoliko varijanti:

- TINYINT zauzima jedan bajt ($2^8$ mogućih vrednosti). Rezervisane reči BIT i BOOL su sinonimi za TINYINT.
- SMALLINT zauzima dva bajta ($2^{16}$ mogućih vrednosti).
- MEDIUMINT zauzima tri bajta ($2^{24}$ mogućih vrednosti).
- BIGINT zauzima osam bajtova ($2^{64}$ mogućih vrednosti). Aproksimirani numerički tipovi opisani su u nastavku teksta.

**Tip FLOAT**

Ovaj tip je namenjen za rad s brojevima s pokretnim zarezom jednostruke preciznosti. Može predstavljati pozitivan broj u opsegu od $1.18 \times 10^{-38}$ do $3.40 \times 10^{38}$ i sličan opseg negativnih brojeva.

**Tip DOUBLE**

Ovaj tip je namenjen brojevima s pokretnim zarezom dvostruke preciznosti. Sinonimi za DOUBLE su REAL i DOUBLE PRECISION. Mogu predstavljati pozitivan broj u opsegu od $2.23 \times 10^{-308}$ do $1.80 \times 10^{308}$ i sličan opseg negativnih brojeva.

## Znakovni i tekstualni tipovi podataka

MySQL podržava više znakovnih i tekstualnih tipova podataka. Osnovni tekstualni tipovi su CHAR, VARCHAR, TEXT, BLOB, ENUM i SET. Opisaćemo ih jedan po jedan u nastavku teksta.

**Tip CHAR**

Tip CHAR omogućava skladištenje znakovnih vrednosti fiksne dužine. Kao u primeru baze podataka employee, rezervisanoj reči CHAR obično sledi dužina (broj znakova) znakovne vrednosti, na primer CHAR(20). Ako ne zadamo dužinu, podrazumeva se CHAR(1). Maksimalna dužina podatka tipa CHAR je 255 znakova. Kada se podatak tipa CHAR upiše u kolonu tabele, on uvek ima dužinu koju smo zadali u definiciji kolone. To se postiže dopunjavanjem podatka u koloni razmacima. Ti razmaci se automatski uklanjanju pri učitavanju podatka iz kolone tipa CHAR.

Očigledno je da podaci tipa CHAR zauzimaju više prostora na disku od ekvivalentnih znakovnih vrednosti promenljive dužine. Prednost im je to što se podaci brže učitavaju iz tabele čije su sve kolone fiksne širine (tj. CHAR, numerički ili DATE). Budući daje brzina učitavanja podataka često važnija od prostora koji oni zauzimaju na disku, možda ćemo se opredeliti da tekstualna polja u kojima se vrednosti ne razlikuju mnogo po dužini deklarišete kao CHAR da bismo (malo) optimizovali sistem.

Ispred deklaracija oba tipa, CHAR i VARCHAR, možemo dodati rezervisanu reč NATIONAL, što znači da želite da ograničite sadržaj na standardni skup znakova. Pošto se ova opcija podrazumeva u MySQL-u, korisna je samo ako nam je potrebna kompatibilnost između različitih platformi.

Deklaracijama tipova CHAR i VARCHAR može slediti rezervisana reč BINARY, što znači da se pri poređenju znakovnih vrednosti pravi razlika između malih i velikih slova. Podrazumevani način poređenja je da se ta razlika ne pravi.

**Tip VARCHAR**

Tip VARCHAR omogućava skladištenje znakovnih nizova promenljive dužine. Dužina podataka zadaje se između zagrada iza imena tipa, na primer, VARCHAR(10). Opseg mogućih vrednosti je od 0 do 255.

**Tipovi TEXT, BLOB i njihove varijante**

Tipovi TEXT omogućavaju skladištenje tekstualnih podataka dužih od onog što može da stane u tipove CHAR i VARCHAR. BLOB je skraćenica za Binary Large OBject (veliki binarni objekat). Ovi tipovi se međusobno ni po čemu ne razlikuju, jedino je BLOB namenjen čuvanju binarnih a ne tekstualnih podataka. Pri poređenju podataka tipa BLOB pravi se razlika između malih i velikih slova, dok se to ne čini pri poređenju podataka tipa TEXT. Oba tipa su promenljive dužine i za oba postoje varijante raznih veličina:

- Tip TINYTEXT ili TINYBLOB može sadržati najviše 255 (to je $2^8$-1) znakova ili bajtova.
- Tip TEXT ili BLOB može sadržati najviše 65.535 ($2^{16}$-1) znakova ili bajtova (64 KB).
- Tip MEDIUMTEXT ili MEDIUMBLOB može sadržati najviše 16,777,215 ($2^{24}$-1) znakova ili bajtova (16 MB).
- Tip LONGTEXT ili LONGBLOB može sadržati najviše 4,294,967,295 ($2^{32}$-1) znakova ili bajtova (4 GB).

**Tip ENUM**

Ovaj tip podataka omogućava da zadate listu mogućih vrednosti. Kolona tabele može sadržati jednu vrednost iz nabrojanog skupa mogućih. Tip podataka ENUM deklariše se na sledeći način:

```
pol enum('m','ž')
```

Pošto vrednost tipa ENUM može biti i NULL, moguće vrednosti kolone pol su: m, ž, NULL ili error.

**Tip SET**

Tip SET je sličan tipu ENUM s tom razlikom što kolone u redovima tabele mogu sadržati i više vrednosti iz nabrojanog skupa mogućih.

### Datumski i vremenski tipovi podataka

MySQL podržava više tipova za rad s datumima i vremenima, koji su opisani u daljem tekstu.

**Tip DATE**

Tip DATE omogućava skladištenje datuma. MySQL očekuje da datum bude u ISO redosledu godina-mesec-dan, čime se izbegavaju problemi usled različitih formata datuma s obe strane Atlantika. Datumi se prikazuju u formatu GGGG-MM-DD.

**Tip TIME**

Ovaj tip omogućava skladištenje podataka koji predstavljaju vreme, koje se prikazuje u formatu ČČ:MM:SS.

**Tip DATETIME**

Ovaj tip je kombinacija dva prethodna. Format je GGGG-MM-DD ČČ:MM:SS.

**Tip TIMESTAMP**

Ovo je koristan tip podataka za kolone tabele. Ako u određenom redu ne zadamo vrednost za kolonu ovog tipa, ili zadamo NULL, u kolonu se upisuje vreme kada je red dodat tabeli ili kada je poslednji put izmenjen sadržaj reda. Kada učitamo podatak tipa TIMESTAMP, prikazuje se u istom formatu kao tip DATETIME,To je značajna razlika između MySQL-ovih verzija 4.0 i 4.1. Ranije smo u deklaraciji kolone tipa TIMESTAMP mogli da zadamo širinu na kojoj se prikazuju podaci iz te kolone.

**Tip YEAR**

Ovaj tip omogućava skladištenje podataka koji predstavljaju godine. Kada deklarišemo kolonu ovog tipa, možemo zadati YEAR (2) ili YEAR (4) da bismo zadali broj cifara. Podrazumeva se YEAR (4). YEAR (2) predstavlja opseg godina od 1970. do 2069.

## Naredbe za šemu baze podataka (CREATE, ALTER, DROP) u MySQL okruženju

Mnogi DBMS-ovi poseduju interaktivne grafičke alatke koje nam omogućuju da pravimo tabele i da rukujemo njima i njihovim svojstvima, kao što su definicije kolona i ograničenja. U ovom poglavlju se objašnjava kako ove zadatke obaviti programski, korišćenjem SQL-a:

- Iskaz CREATE TABLE pravi novu tabelu.
- Iskaz ALTER TABLE menja strukturu postojeće tabele.
- Iskaz DROP TABLE uništava tabelu i sve njene podatke.

Ovi iskazi ne vraćaju rezultat, ali bi naš DBMS mogao ispisati poruku koja ukazuje na to da li je iskaz uspešno izvršen. Ovi iskazi menjaju objekte baze podataka i podatke, tako da će možda biti potrebno da nam administrator naše baze podataka odobri pravo da ih izvršavamo.

### Pravljenje nove tabele pomoću CREATE TABLE

U ovom odeljku se govori o tome kako se pravi nova tabela korišćenjem minimalnog iskaza CREATE TABLE. U narednim odeljcima pokazano je kako se u iskaz CREATE TABLE dodaju ograničenja kolona i ograničenja tabele. Da bismo napravili novu tabelu upišimo sledeće:

Definisati prost primarni ključ za tabelu publishers u primeru baze podataka, koristeći imenovano ograničenje tabele.

```
CREATE TABLE publishers
(
pub_id CHAR(3) NOT NULL,
pub_name VARCHAR(20) NOT NULL,
city VARCHAR(15) NOT NULL,
state CHAR(2) ,
country VARCHAR(15) NOT NULL,
CONSTRAINT publishers_pk
PRIMARY KEY (pub_id)
);
```

### Menjanje tabele pomoću ALTER TABLE

Koristimo iskaz ALTER TABLE za izmenu definicije tabele dodavanjem, menjanjem ili odbacivanjem kolona i ograničenja. Uprkos SQL standardu, implementacija ALTER TABLE veoma se menja od DBMS-a do DBMS-a. Da bismo odredili šta možemo da menjamo i uslove pod kojima su izmene dozvoljene, potražimo u dokumentaciji našeg DBMS-a pojam ALTER TABLE. U zavisnosti od našeg DBMS-a, neke od izmena koje možemo da napravimo pomoću ALTER TABLE su:

- dodavanje ili izbacivanje kolone;
- izmena tipa podataka kolone;
- dodavanje, izmena ili izbacivanje podrazumevane vrednosti ili ograničenja nulabilnosti kolone;
- dodavanje, izmena ili izbacivanje ograničenja kolone ili tabele kao što su primarni ključ, spoljni ključ, ograničenja jedinstvenosti i kontrolna ograničenja;
- preimenovanje kolone;
- preimenovanje tabele.

Dodati kolonu email_address tabeli publishers.

```
ALTER TABLE publishers
ADD email_address CHAR(25);
```

Izbaciti kolonu email_address iz tabele publishers.

```
ALTER TABLE publishers
DROP COLUMN email_address;
```

Ukoliko iskaz ALTER TABLE našeg DBMS-a ne podržava radnju koja nam je potrebna (kao što je, recimo, odbacivanje ili preimenovanje kolone ili ograničenja), proverimo da li naš DBMS nudi tu radnju u okviru drugog SQL iskaza ili kao izdvojenu (ne-SQL) komandu preko komandne linije ili grafičkog korisničkog interfejsa. Kao poslednje pribežište, možemo ručno ponovo da napravimo i popunimo tabelu u željenom obliku.

### Odbacivanje tabele pomoću DROP TABLE

Iskaz DROP TABLE koristimo za uklanjanje tabele iz baze podataka. Kada odbacujemo tabelu, neka važna razmatranja su:

- Možemo da odbacimo baznu tabelu ili privremenu tabelu.

- Neki DBMS-ovi nam omogućuju da povratimo odbačenu tabelu opozivom. Ako odbačena tabela nije bila deo transakcije, možemo je povratiti iz najsvežije rezervne kopije (iako bi mogla biti zastarela).
- Odbacivanjem tabele uništavaju se njena struktura, podaci, indeksi, ograničenja, dozvole i tako dalje.
- Odbacivanje tabele nije isto što i brisanje svih njenih redova. Možemo izbaciti sve redove iz tabele, ali je ne uništiti, pomoću DELETE FROM tabela.
- Odbacivanjem tabele ne odbacuju se prikazi koji referenciraju tu tabelu.
- Imaćemo problema sa spoljnim ključevima ili prikazima koji referenciraju odbačenu tabelu osim ako i oni sami nisu izmenjeni ili odbačeni.

Odbaciti tabelu publishers.

```
DROP TABLE publishers;
```

## Naredbe za podatke (SELECT, INSERT, UPDATE, DELETE)

Do ove tačke, objasnili smo kako se koristi iskaz za definiciju strukture baze i tabele. U ovom poglavlju objasnićemo kako se SQL iskazi koriste za menjanje podataka u tabelama:

- Iskaz INSERT dodaje nove redove tabeli.
- Iskaz UPDATE menja vrednosti u postojećim redovima tabele.
- Iskaz DELETE uklanja redove iz tabele.

Ovi iskazi ne vraćaju rezultat, ali će naš DBMS obično ispisati poruku koja govori da li je iskaz uspešno izvršen i, ako jeste, ispisaće broj redova pogođenih izmenom. Da bismo videli stvaran učinak koji je iskaz imao nad tabelom, upotrebimo iskaz SELECT poput SELECT * FROM tabela;.

Za razliku od iskaza SELECT, koji samo pristupa podacima, ovi iskazi menjaju podatke, tako da će možda biti potrebno da nam administrator naše baze podataka odobri pravo da ih izvršavamo.

### Umetanje redova pomoću INSERT

Iskaz INSERT dodaje nove redove tabeli. U ovom odeljku objašnjeno je kako se koristi nekoliko varijanti iskaza INSERT za:

- umetanje reda korišćenjem pozicija kolona (INSERT VALUES);
- umetanje reda korišćenjem imena kolona (INSERT VALUES);
- umetanje redova iz jedne tabele u drugu tabelu (INSERT SELECT).

Važna svojstva iskaza INSERT su:

- Pri pozicionom umetanju, uređene vrednosti umećemo u novi red istim redosledom kojim se kolone pojavljuju u tabeli (pročitajmo deo o umetanju reda korišćenjem pozicija kolona kasnije u ovom odeljku). Pri umetanju sa imenovanim kolonama, za svaku vrednost imenujemo određenu kolonu u koju se vrednost umeće u novom redu (pročitajmo deo o umetanju reda korišćenjem imena kolona kasnije u ovom odeljku). Uvek bi trebalo da koristimo umetanje sa imenovanim kolonama tako da naš SQL kod radi i ako neko preuredi kolone tabele ili doda nove kolone.

- Pomoću INSERT VALUES defnišemo eksplicitne vrednosti koje će biti umetnute u tabelu. Pomoću INSERT SELECT biramo redove iz druge tabele koji će biti umetnuti u predmetnu tabelu.
- INSERT VALUES dodaje jedan red tabeli. INSERT SELECT dodaje tabeli nula ili više redova.
- Svaka umetnuta vrednost mora imati isti tip podataka kao njoj odgovarajuća kolona ili mora postojati mogućnost implicitne konverzije u isti tip.
- Da bi se sačuvao referencijalni integritet, umetnuta vrednost spoljnog ključa mora sadržati ili null (ako je dozvoljeno) ili postojeću vrednost ključa iz kolone primarnog ili jedinstvenog ključa koju referencira spoljni ključ.
- Umetnuta vrednost ne srne narušiti kontrolno ograničenje.
- Nijedan izraz ne sme prouzrokovati aritmetičku grešku (prekoračenje ili deljenje nulom, na primer).

Da bismo umetnuli red korišćenjem pozicija kolona, upišimo sledeće:

```
INSERT INTO tabela
VALUES vrednost1 ,vrednost2,...,
vrednostN);
```

tabela je ime tabele u koju se red umeće. Vrednost1, vrednost2,…,vrednostN je spisak zarezom odvojenih literala ili izraza, obuhvaćen zagradama, koji obezbeđuje vrednost za svaku kolonu u novom redu. Broj vrednosti mora biti jednak broju kolona u tabeli, a vrednosti moraju biti navedene istim redosledom kao kolone u tabeli. DBMS ubacuje svaku vrednost u kolonu koja odgovara poziciji vrednosti u tabeli, vrednosti ubacuje se u prvu kolonu tabele u novom redu, vrednost2 u drugu kolonu i tako dalje. Ovaj iskaz dodaje jedan red tabeli.

Ovaj iskaz INSERT dodaje novi red tabeli authors, navodeći vrednosti istim redosledom kojim su poređane kolone u tabeli authors.

```
INSERT INTO authors
VALUES('A08','Michael','Polk','512-953-1231','4028 Guadalupe
St','Austin','TX','78701');
```

### Ažuriranje redova pomoću UPDATE

Iskaz UPDATE menja vrednosti u postojećim redovima tabele. UPDATE možemo da koristimo za menjanje:

- svih redova u tabeli;
- određenih redova u tabeli.

Da bismo ažurirali redove, navodite:

- tabelu koju ažurirate;
- imena kolona koje treba ažurirati i njihove nove vrednosti;
- opcioni uslov pretrage koji određuje koje redove treba ažurirati.

Važna svojstva iskaza UPDATE su:

- UPDATE uzima opcionu rečenicu WHERE koja određuje koje redove treba ažurirati. Bez rečenice WHERE, UPDATE menja sve redove u tabeli.
- UPDATE je opasan, jer se rečenica WHERE lako može greškom izostaviti (čime bi se ažurirali svi redovi), ili se može pogrešno navesti uslov pretrage uz WHERE (čime bi se ažurirali pogrešni redovi). Pametno je pre izvršavanja stvarnog iskaza UPDATE izvršiti iskaz SELECT koji koristi istu rečenicu WHERE. Upotrebimo SELECT * da bi se prikazali svi redovi koje će DBMS izmeniti kada izvršimo UPDATE, ili upotrebimo SELECT COUNT(*) da bi se prikazao samo broj redova koji će biti izmenjeni.
- Svaka vrednost kojom se ažurira mora imati isti tip podataka kao njoj odgovarajuća kolona ili mora postojati mogućnost implicitne konverzije u isti tip.
- Da bi se sačuvao referencijalni integritet, možemo da definišemo radnju koju DBMS automatski obavlja kada pokušamo da ažuriramo vrednost ključa na koju ukazuju vrednosti spoljnog ključa.
- Vrednost kojom se ažurira ne srne prekršiti kontrolno ograničenje.
- Nijedan izraz ne sme prouzrokovati aritmetičku grešku (prekoračenje ili deljenje nulom, na primer).

Da bismo ažurirali redove upišimo sledeće:

```
UPDATE tabela
SET kolona = izraz
[WHERE uslov_pretrage];
```

tabela je ime tabele koja se ažurira. kolona je ime kolone u tabeli koja sadrži redove koji će biti izmenjeni. izraz je literal, izraz ili podupit obuhvaćen zagradama koji vraća jednu vrednost. Vrednost koju vrati izraz zamenjuje postojeću vrednost u koloni. Da bismo izmenili vrednosti u više kolona, upišimo spisak zarezom odvojenih izraza kolona = izraz u rečenici SET. Izraze kolona = izraz možemo ispisati bilo kojim redosledom.
uslov_pretrage određuje uslove koje redovi moraju zadovoljavati da bi bili ažurirani. Uslovi uslov_pretrage mogu biti uslovi WHERE (operatori poređenja, LIKE, BETWEEN, IN i IS NULL ili uslovi podupita (operatori poređenja, IN, ALL, ANY i EXISTS, kombinovani sa AND, OR i NOT. Ako se rečenica WHERE izostavi, ažurira se svaki red u tabeli.

Udvostručiti cene knjiga iz oblasti istorije.

```
UPDATE titles
SET price = price * 2.0
WHERE type = 'history';
```

Izmeniti datum objavljivanja za sve knjige Sare Buchman na 1. januar 2014.

```
UPDATE titles
SET pubdate = '2014-01-01'
WHERE title_id IN
(SELECT title_id
FROM title_authors
WHERE au_id IN
(SELECT au_id
FROM authors
WHERE au_fname = 'Sarah'
AND au_lname = 'Buchman'));
```

### Brisanje redova pomoću DELETE

Iskaz DELETE uklanja redove iz tabele. DELETE možemo da koristimo za uklanjanje:

- svih redova tabele;
- određenih redova tabele.

Da bismo brisali redove, navodimo:

- tabelu čiji se redovi brišu;
- opcioni uslov pretrage koji određuje redove koji se brišu.

Važna svojstva iskaza DELETE su:

- Za razliku od INSERT i UPDATE, DELETE ne uzima imena kolona, jer uklanja celokupne redove.
- DELETE uklanja redove iz tabele, ali nikad ne briše definiciju tabele. Čak i kad uklonimo sve redove iz tabele, sama tabela i dalje postoji. Ako želimo da obrišemo definiciju tabele (i sve njoj pridružene podatke, indekse i tako dalje).
- DELETE uzima opcionu rečenicu WHERE koja određuje koji se redovi brišu. Bez rečenice WHERE, DELETE uklanja sve redove u tabeli.
- Iskaz DELETE je opasan, jer se rečenica WHERE lako može greškom izostaviti (čime bi se obrisali svi redovi), ili se može pogrešno navesti uslov pretrage uz WHERE (čime bi se obrisali pogrešni redovi). Pametno je pre izvršavanja stvarnog iskaza DELETE izvršiti iskaz SELECT koji koristi istu rečenicu WHERE. Upotrebimo SELECT * da bi se prikazali svi redovi koje će DBMS ukloniti kada izvršimo DELETE, ili upotrebimo SELECT COUNT(*) da bi se prikazao samo broj redova koji će nestati.
- Da bi se sačuvao referencijalni integritet, možemo da definišemo radnju koju DBMS automatski obavlja kada pokušamo da obrišemo vrednost ključa na koju ukazuju vrednosti spoljnog ključa.
- Nijedan izraz ne sme prouzrokovati aritmetičku grešku (prekoračenje ili deljenje nulom, na primer).

Da bismo obrisali redove, upišimo sledeće:

```
DELETE FROM tabela
[WHERE uslov_pretrage];
```

tabela je ime tabele čiji se redovi brišu. uslov_pretrage određuje uslove koje redovi koji se brišu moraju zadovoljavati. Uslovi uslov_pretrage mogu biti uslovi WHERE (operatori poređenja, LIKE, BETWEEN, IN i IS NULL ili uslovi podupita (operatori poređenja, IN, ALL, ANY i EXISTS), kombinovani sa AND, OR i NOT. Ako se rečenica WHERE izostavi, briše se svaki red u tabeli.
U sledećim primerima ćemo zanemariti ograničenja referencijalnog integriteta - što, naravno, ne bih činio u proizvodnoj bazi podataka.

Obrisati sve redove iz tabele royalties.

```
DELETE FROM royalties;
```

Iz tabele title_authors obrisati redove za one knjige koje je objavio izdavač P01 ili P04.

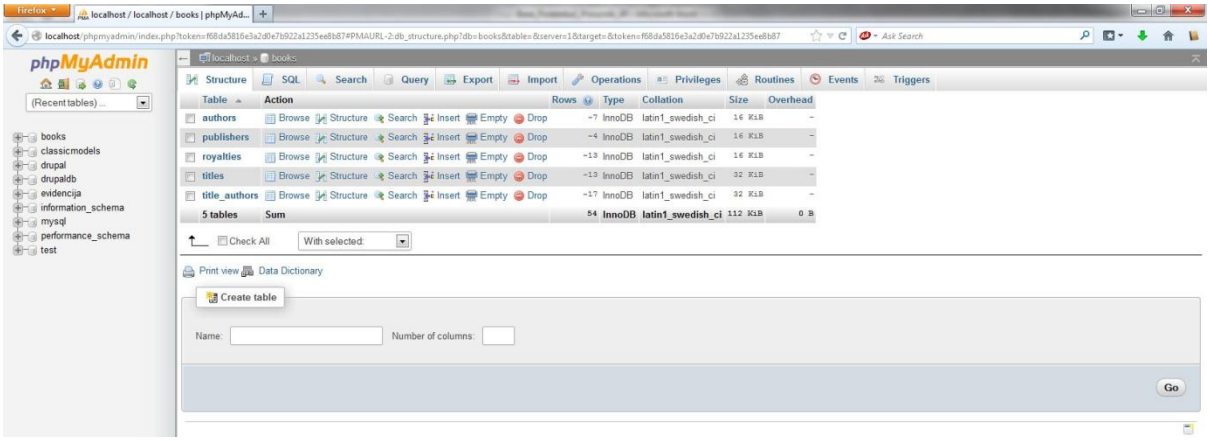
```
DELETE FROM title_authors
WHERE title_id IN
(SELECT title_id
FROM titles
WHERE pub_id IN ('P01', 'P04'));
```

## Import / Export podataka pomoću phpMyAdmin

Za izvršenje backup foldera i datoteka putem FTP-a sa servera za potpuni backup svoje web stranice rađene u CMS sistemu nije dovoljno samo preuzeti datoteke već je potrebno izvršiti i backup SQL baze. Nakon upisa pristupne lozinke i otvoriće nam se Control Panel phpMyAdmin interfejs gde imamo mogućnosti uvoza i izvoza baze podataka.

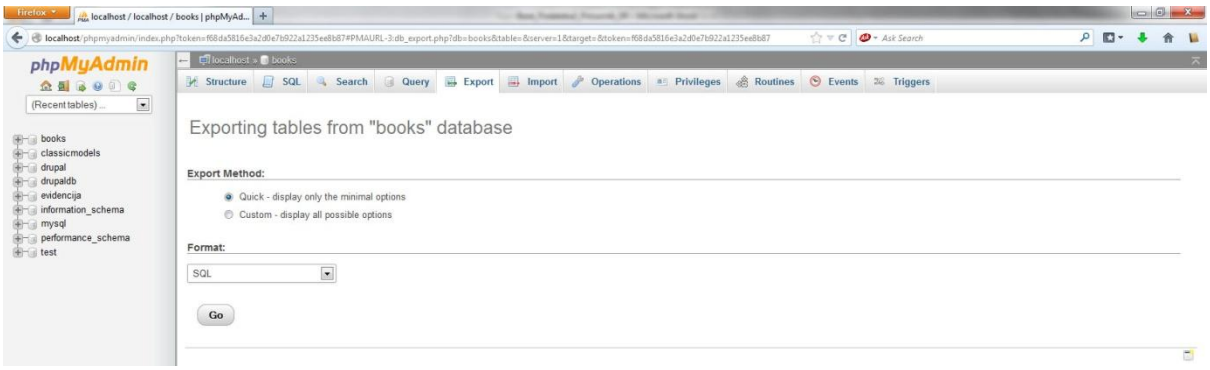
### Export podataka

Nakon pristupanja SQL bazi na levoj strani prozora na meniju imamo popis svih baza ( ako ih je više ). Kliknimo na bazu koju želimo Exportovati (izvesti). U ovom slučaju baza će biti pod nazivom books. Kliknimo na dugme *Export*. Ovaj primer je prikazan na localhost-u tako da nas to ne buni ali to je isti postupak i na udaljenom serveru.




*Sl. 2.1 Export baze podataka pod imenom books*

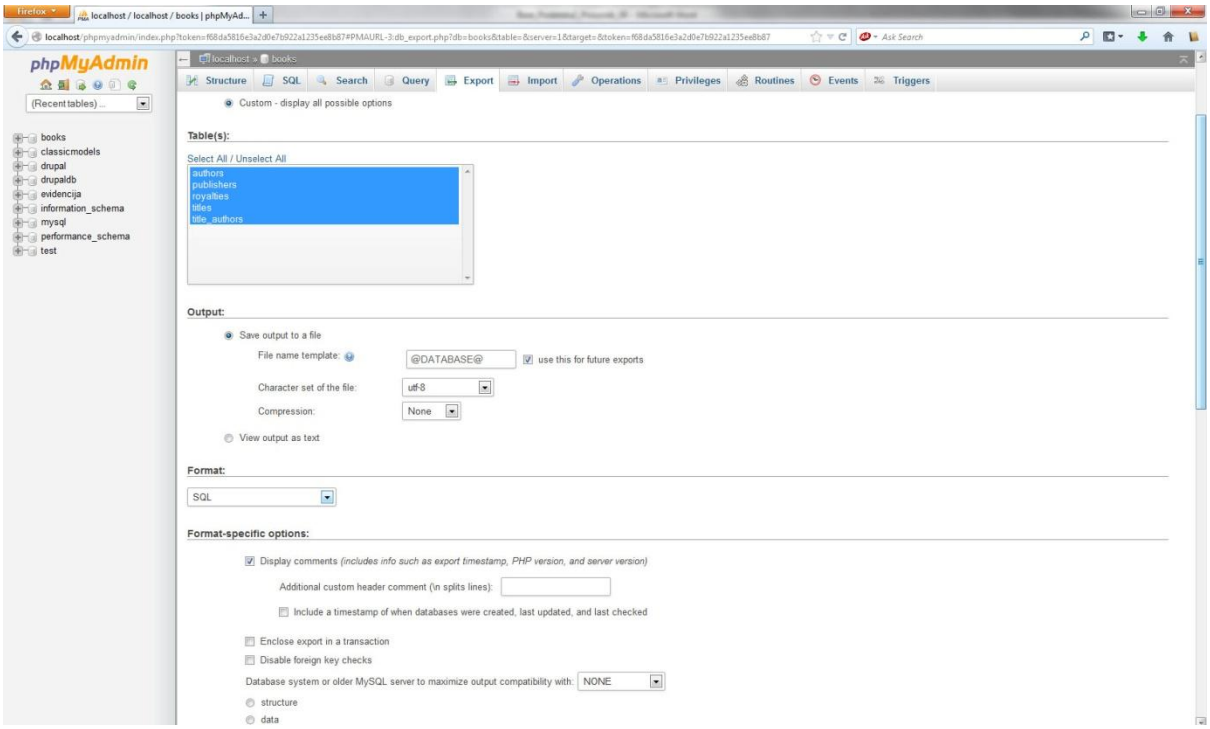
Nakon toga nam se javlja mogućnost izbora *Quick* ili *Custom* izvoza baze kao što je prikazano na Sl. 2.2.




*Sl. 2.2 Izbor Quick ili Custom izvoza baze*

Tokom izbora *Quick* izvoza, otvara nam se dijalog prozor za potvrdu preuzimanja na naš računar . Kliknimo na dugme Save. U ovom dijalog prozoru možemo videti naziv datoteke koju preuzimamo i

server odakle je skidamo. U slučaju da izaberemo *Custom* izvoz, imaćemo niz mogućnosti za podešavanje izvoza kao što je prikazano na Sl. 2.3.

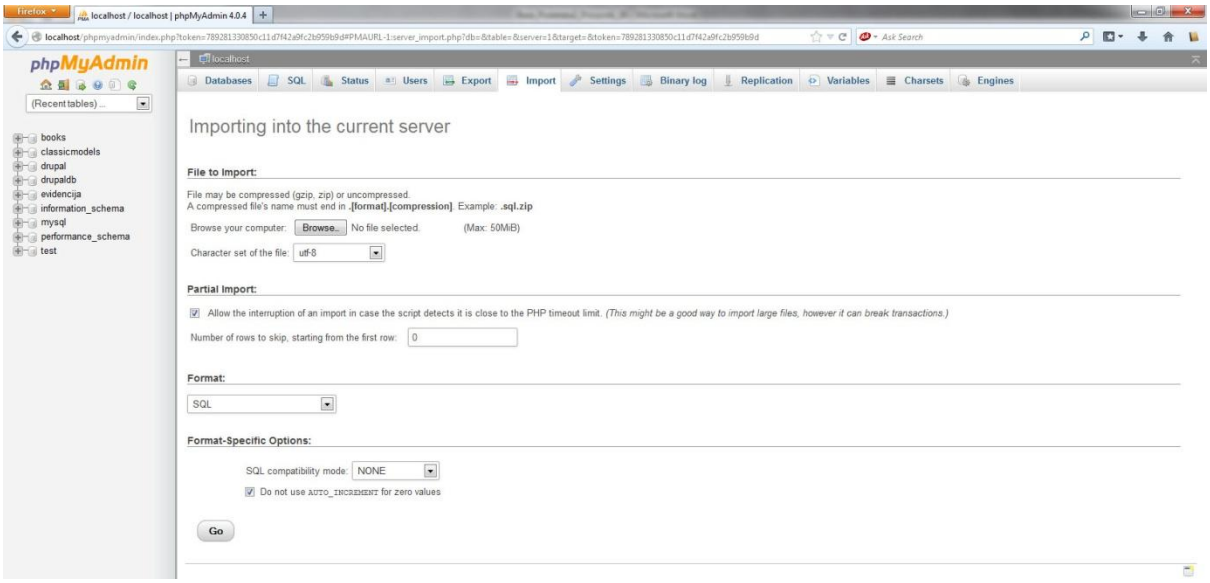



*Sl. 2.3 Custom izvoz baze podataka*

Prilikom *Custom* izvoza baze imaćemo mogućnosti izvoza u raznim popularnim formatima ili izvoz pojedinih elemenata baze.

## Import podataka

Nakon pristupanja SQL bazi na levoj strani prozora na meniju imamo popis svih baza ( ako ih je više ). Kliknimo na bazu koju želimo Importovati (uvesti). U ovom slučaju baza će biti pod nazivom books. Kliknimo na dugme Import. Ovaj primer je prikazan na localhost-u tako da nas to ne treba da buni ali to je isti postupak i na udaljenom serveru.




*Sl. 2.4 Import baze pod imenom books*

Na gornjoj slici uočimo dugme *Browse*, kliknimo na njega i imaćemo prozor za odabir sql datoteke za uvoz. Na njemu je potrebno pronaći folder u kojem se nalazi naša sačuvana SQL baza sa web-servera. Kliknimo na nju i potvrdimo na dugme *Open*. Nakon odabira datoteke kliknimo na dugme *Go*. U slučaju uspešnog izvršenja uvoza baze podataka dobićemo sledeću poruku.

*Import has been successfully finished.*

## Konekcija sa MySQL bazom pomoću PHP okruženja

PHP i MySQL su popularne tehnologije otvorenog koda, savršene za brz i efikasan razvoj Web aplikacija koje rade s bazama podataka. PHP je moćan jezik za pisanje skriptova, koji omogućava programerima da brzo prave složene Web aplikacije; MySQL je brz i pouzdan sistem za upravljanje bazama podataka, koji se dobro integriše u PHP i pogodan je za dinamičke aplikacije koje rade na Internetu. Proveru funkcionisanja MySQL servera možemo izvesti pomoću sledećeg primera:

testmysql.php

```
<?php
$link = mysql_connect('localhost','user','pass');
if (!$link) {
        die('Could not connect to MySQL: ' . mysql_error());
}
echo 'Connection OK'; mysql_close($link);
?>
```

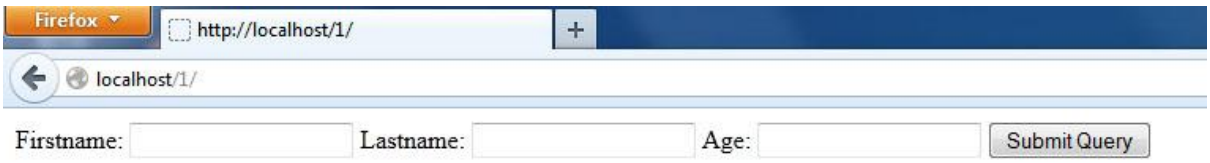
# 3. POGLAVLJE

## Rad sa HTML formama

Forme se koriste u većini kod manjih a posebno većih web aplikacija, nezavisno u kojem je jeziku aplikacija napisana. Zbog ovoga je potrebno imati znanje samih HTML formi (tipove polja, koja su za koju namenu najbolja, načine grupisanja elemenata… ). Uz poznavanje HTML dela formi, potrebno je znati kako prihvatiti pomoću PHP-a vrednosti koje su unošene u formu. Ovaj deo se bavi tom problematikom od izrade jednostavnih formi i prihvaćanja informacija, do izrade komplikovanije forme. Sa tehničke strane gledano, forma je HTML element pomoću kojeg grupišemo više elemenata za unos podataka (tipa text box, select, opcije … ). U te elemente korisnik može uneti ili izabrati informacije koje se kasnije koriste za bilo koju svrhu (unos informacija u bazu podataka, slanje e-maila sa unetim informacijama… ).

### Prvi primer: Unos podataka u MySQL bazu putem HTML forme

Ovaj primer prikazuje korišćenje HTML forme, preuzimanje unetih podataka pomoću php programskog jezika, konekciju i unos podataka u MySQL bazu.




*Sl. 3.1 HTML forma prvog primera*

test.sql

```
CREATE DATABASE test;

USE DATABASE test;

CREATE TABLE Persons (
    PID INT NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (PID),
    FirstName CHAR(15),
    LastName CHAR(15),
    Age INT
);
```

index.html

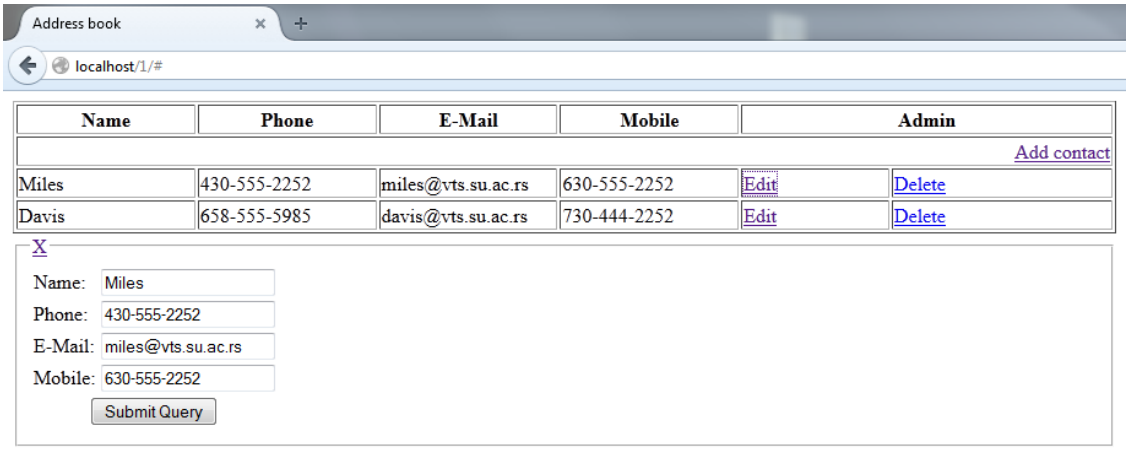
```
<html>
<body>

<form action="insert.php" method="post">
Firstname: <input type="text" name="firstname">
Lastname: <input type="text" name="lastname">
Age: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

insert.php

```
<?php
$con=mysqli_connect("localhost","admin","admin","test");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }

$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";

if (!mysqli_query($con,$sql))
  {
  die('Error: ' . mysqli_error($con));
  }
echo "1 record added";

mysqli_close($con);
?>
```

## Listanje i unos podataka u MySQL bazu pomoću PHP

### Drugi primer: Čitanje, unos, promena i brisanje podataka iz baze




*Sl. 3.2 HTML forma drugog primera*

test.sql

```
CREATE TABLE contacts (
    id INT(4) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30),
    phone VARCHAR(30),
    email VARCHAR(30),
    mobile varchar(15)
);

INSERT INTO contacts (name, phone, email, mobile)
VALUES
( "Miles", "430-555-2252", "miles@vts.su.ac.rs", "630-555-2252"),
( "Davis", "658-555-5985", "davis@vts.su.ac.rs", "730-444-2252");
```

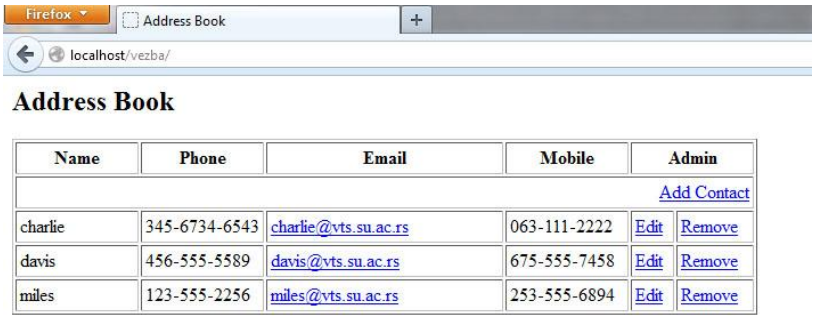
index.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Address book</title>
<script type="text/javascript">
data = [];
function showDiv(what,where){
      if(what == 'add'){
            document.getElementById('form').style.display = 'block';
            document.getElementsByName('name').item(0).value = '';
            document.getElementsByName('phone').item(0).value = '';
            document.getElementsByName('email').item(0).value = '';
            document.getElementsByName('mobile').item(0).value = '';
            document.getElementsByName('id').item(0).value = '#';
      }
      if(what == 'edit'){
            document.getElementById('form').style.display = 'block';
            document.getElementsByName('name').item(0).value =
data[where]['name'];
            document.getElementsByName('phone').item(0).value =
data[where]['phone'];
            document.getElementsByName('email').item(0).value =
data[where]['email'];
            document.getElementsByName('mobile').item(0).value =
data[where]['mobile'];
            document.getElementsByName('id').item(0).value = where;
      }
}

function hideDiv(){
      document.getElementById('form').style.display = 'none';
}
</script>
</head>

<body>
<?php
$dbhost = "localhost";
$dbuser = "root";
$dbpass = "";
$db = "test";
$connection = mysqli_connect($dbhost,$dbuser,$dbpass,$db) or
die(mysqli_error($connection));
if(isset($_POST['id'])){
      if($_POST['id'] == '#'){
            $sql = "INSERT INTO contacts (name,phone,email,mobile) VALUES
('".$_POST['name']."','".$_POST['phone']."','".$_POST['email']."','".$_POST
['mobile']."');";
            $query = mysqli_query($connection,$sql);
      }else{
            $sql = "UPDATE contacts SET name = '".$_POST['name']."', phone
= '".$_POST['phone']."', email = '".$_POST['email']."', mobile =
'".$_POST['mobile']."'
            WHERE id = '".$_POST['id']."' LIMIT 1";
            $query = mysqli_query($connection,$sql);
      }
      echo mysqli_error($connection);
}
```

```
if(isset($_GET['delete'])){
      $sql = "DELETE FROM contacts WHERE id = '".$_GET['delete']."';";
      $query = mysqli_query($connection,$sql);
}
?>
<table width="900" border="1">
<tr>
<th width="16.6%">Name</th>
<th width="16.6%">Phone</th>
<th width="16.6%">E-Mail</th>
<th width="16.6%">Mobile</th>
<th colspan="2" width="33.3%">Admin</th>
</tr>
<tr>
<td colspan="6" style="text-align:right;"><a href="#"
onclick="showDiv('add')">Add contact</a></td>
</tr>
<?php
$sql = "SELECT * FROM contacts ORDER BY id";
$query = mysqli_query($connection,$sql);
while($data = mysqli_fetch_assoc($query)){
      echo
'<tr><td>'.$data['name'].'</td><td>'.$data['phone'].'</td><td>'.$data['emai
l'].'</td><td>'.$data['mobile'].'</td>
      <td><a href="#"
onclick="showDiv(\'edit\','.$data['id'].')">Edit</a></td>
      <td><a href="?delete='.$data['id'].'" onclick="return confirm(\'Are
you sure to delete '.$data['name'].'?\');">Delete</a></td>';
      echo '<script type="text/javascript">
      data['.$data['id'].'] = {
      name: \''.$data['name'].'\',
      phone: \''.$data['phone'].'\',
      email: \''.$data['email'].'\',
      mobile: \''.$data['mobile'].'\'
      }
      </script>';
}
?>
</table>
<div id="form" style="display:none; width:900px;">
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<fieldset>
<legend><a href="#" onclick="hideDiv()">X</a></legend>
<table><tr>
<td>Name:</td> <td><input type="text" name="name" /></td>
</tr><tr>
<td>Phone:</td> <td><input type="tel" name="phone" /></td>
</tr><tr>
<td>E-Mail:</td> <td><input type="text" name="email" /></td>
</tr><tr>
<td>Mobile:</td> <td><input type="tel" name="mobile" /></td>
</tr><tr>
<td colspan="2"><center><input type="hidden" name="id" value="#" /><input
type="submit" /></center></td>
</tr></table>
</fieldset>
</form>
</div>
</body>
</html>
```

**Vežba:**

Napraviti izmenu strukture baze i aplikacije sa dodatkom adrese u adresar kao što je prikazano na sledećoj slici.




*Sl. 3.3 HTML forma zadatka*

## Treći primer: Pretraga baze pomoću php

Na ovom primeru se prikazuje jednostavna pretraga baze po imenu.


*Sl. 3.4 HTML forma trećeg primera*

test.sql

```
CREATE TABLE IF NOT EXISTS tbl_students (
    student_id int(11) NOT NULL,
    student_name varchar(21) NOT NULL,
    student_course varchar(21) NOT NULL,
    student_age int(11) NOT NULL
)  ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Dumping data for table tbl_students

INSERT INTO tbl_students (student_id, student_name, student_course,
student_age) VALUES
(1, 'Charlie', 'IT support', 32),
(2, 'Parker', 'Manager', 27);
```

Search-form.html

```
<html>
<head>
<meta charset="UTF-8" />
<title>Search</title>
</head>
<body>
<h1>Search by name</h1>
<form action="search.php" method="get">
  <label>Name:
```

```
  <input type="text" name="keyname" />
  </label>
  <input type="submit" value="Search" />
</form>
</body>
</html>
```

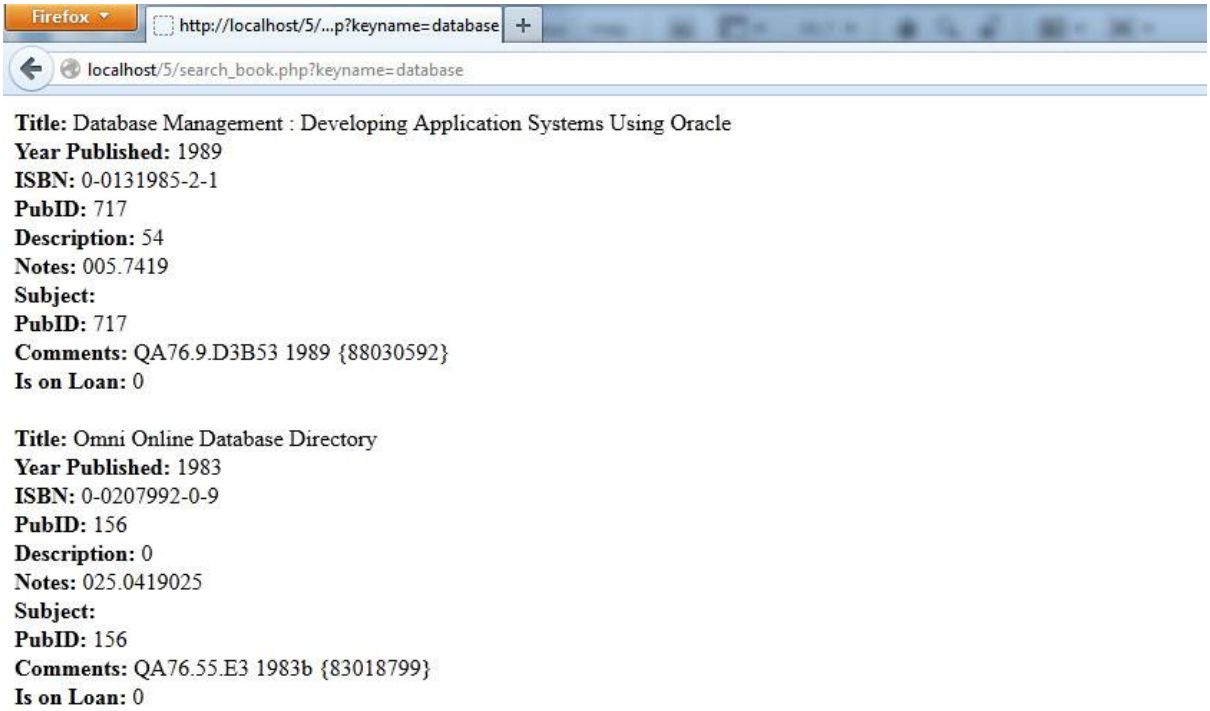
search.php

```
<?php

//capture search term and remove spaces at its both ends if the is any
$searchTerm = trim($_GET['keyname']);

//check whether the name parsed is empty
if($searchTerm == "")
{
      echo "Enter name you are searching for.";
      exit();
}

//database connection info
$host = "localhost"; //server
$db = "test"; //database name
$user = "admin"; //dabases user name
$pwd = "admin"; //password

//connecting to server and creating link to database
$link = mysqli_connect($host, $user, $pwd, $db);

//MYSQL search statement
$query = "SELECT * FROM tbl_students WHERE student_name LIKE
'%$searchTerm%'";

$results = mysqli_query($link, $query);

/* check whethere there were matching records in the table
by counting the number of results returned */
if(mysqli_num_rows($results) >= 1)
{
      $output = "";
      while($row = mysqli_fetch_array($results))
      {
            $output .= "Student ID: " . $row['student_id'] . "<br />";
            $output .= "Name: " . $row['student_name'] . "<br />";
            $output .= "Course: " . $row['student_course'] . "<br />";
            $output .= "Age: " . $row['student_age'] . "<br /><br />";
      }
      echo $output;
}
else
      echo "There was no matching record for the name " . $searchTerm;
?>
```

**Vežba:**

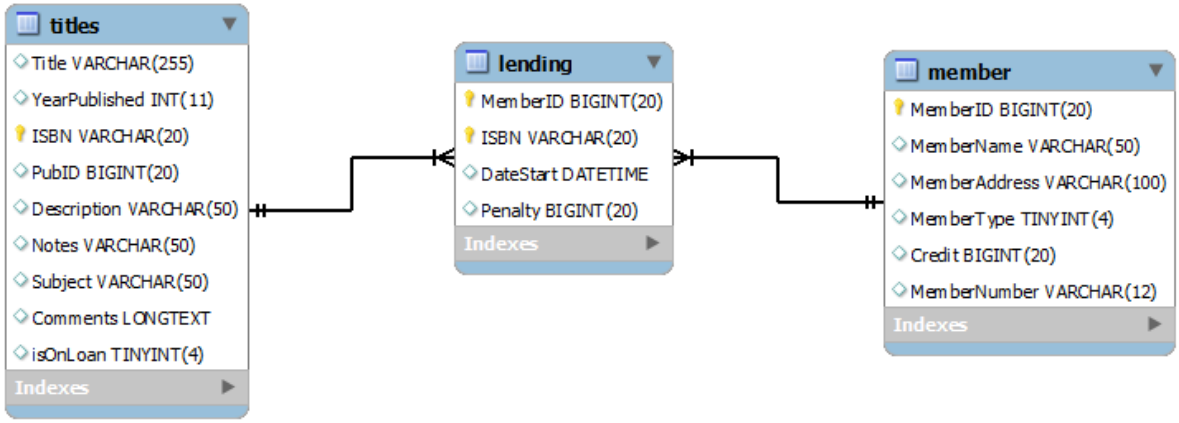
Napraviti pretragu za sledeću bazu po naslovima knjige.


*Sl. 3.5 Html forma za pretragu baze sa knjigama*




*Sl. 3.6 Rezultat pretrage*

Struktura baze je sledeća:




*Sl. 3.7 Struktura baze pod nazivom biblio*

Biblio.sql

```
-- -----------------------------------------------------
-- Table titles
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS titles (
  Title VARCHAR(255) NULL DEFAULT NULL,
```

```
  YearPublished INT(11) NULL DEFAULT NULL,
  ISBN VARCHAR(20) NOT NULL DEFAULT '',
  PubID BIGINT(20) NULL DEFAULT NULL,
  Description VARCHAR(50) NULL DEFAULT NULL,
  Notes VARCHAR(50) NULL DEFAULT NULL,
  Subject VARCHAR(50) NULL DEFAULT NULL,
  Comments LONGTEXT NULL DEFAULT NULL,
  isOnLoan TINYINT(4) NULL DEFAULT NULL,
  PRIMARY KEY (ISBN))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table member
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS member (
  MemberID BIGINT(20) NOT NULL DEFAULT '0',
  MemberName VARCHAR(50) NULL DEFAULT NULL,
  MemberAddress VARCHAR(100) NULL DEFAULT NULL,
  MemberType TINYINT(4) NULL DEFAULT NULL,
  Credit BIGINT(20) NULL DEFAULT NULL,
  MemberNumber VARCHAR(12) NULL DEFAULT NULL,
  PRIMARY KEY (MemberID))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table lending
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS lending (
  MemberID BIGINT(20) NOT NULL DEFAULT '0',
  ISBN VARCHAR(20) NOT NULL DEFAULT '',
  DateStart DATETIME NULL DEFAULT NULL,
  Penalty BIGINT(20) NULL DEFAULT NULL,
  PRIMARY KEY (MemberID, ISB),
  INDEX fk_lending_titles_idx (ISBN ASC),
  CONSTRAINT fk_lending_titles
    FOREIGN KEY (ISBN)
    REFERENCES titles (ISBN)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_lending_member1
    FOREIGN KEY (MemberID)
    REFERENCES member (MemberID)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

insert into titles values
('dBASE III : A Practical Guide21', 1900, '0-0038307-6-4', 472, '22.5',
null, null, ' {}', 0),
('The dBASE Programming Language', 1986, '0-0038326-7-8', 471, '29.5',
'005.756520', null, 'QA76.9.D3D424 1986 {997011870}', 0),
('dBASE III Plus', 1987, '0-0038337-8-X', 469, '29.5', null, null, ' {}',
0),
('Database Management : Developing Application Systems Using Oracle', 1989,
'0-0131985-2-1', 717, '54', '005.7419', null, 'QA76.9.D3B53 1989
{88030592}', 0),
('Wordstar 4.0-6.0 Quick Reference Guide', 1990, '0-0133656-1-4', 460,
'14.95', null, null, ' {}', 0),
('Oracle Triggers and Stored Procedure Programming', 1996, '0-0134436-3-1',
715, '0', '005.756520', null, 'QA76.9.D3O93 1996 {95035147}', 0),
```

```
('Programming in Clipper', 1988, '0-0201145-8-3', 9, '0', '005.756520',
'2nd', 'QA76.76.C65S77 1988 {88019843}', 0),
('Inside MacIntosh', 1994, '0-0201406-7-3', 9, '99.01', '006.676520', null,
'QA76.8.M3I529 1994 {93048597}', 0),
('Structured C for Engineering and Technology/Book and Diskette', 1995, '0-
0230081-2-1', 715, '54', '005.26520', '2nd Bk&dsk', 'QA76.73.C15A33 1995
{94021783}', 0),
('An Introduction to Assembly Language Programming for the Intel 8088
Microprocessor', 1995, '0-0230362-0-6', 715, '60', '005.26520',
'Book&Disk', 'QA76.73.A8A36 1995 {94022171}', 0);
```

search_book.html

```
<html>
<head>
<title>Search</title>
</head>
<body>
<h1>Search books by title</h1>
<form action="search_book.php" method="get">
      <label>Title:</label>
      <input type="text" name="keyname" />
      <input type="submit" value="Search" />
</form>
</body>
</html>
```

search_book.php

```
<?php
      $searchTerm = trim($_GET['keyname']);

      if($searchTerm == "") {
            echo "Enter the title you are searching for.";
            exit();
      }
      $host= "localhost";
      $db = "biblio";
      $user = "admin";
      $pwd = "admin";

      $link = mysqli_connect($host, $user, $pwd, $db);
      $query = "SELECT * FROM titles WHERE Title LIKE '%$searchTerm%'";
      $results = mysqli_query($link, $query);

      if(mysqli_num_rows($results) > 0) {
            $output = "";
            while($row = mysqli_fetch_array($results)) {
                  $output .= "<b>Title:</b> ".$row['Title']."<br />";
                  $output .= "<b>Year Published:</b>
".$row['YearPublished']."<br />";
                  $output .= "<b>ISBN:</b> ".$row['ISBN']."<br />";
                  $output .= "<b>PubID:</b> ".$row['PubID']."<br />";
                  $output .= "<b>Description:</b>
".$row['Description']."<br />";
                  $output .= "<b>Notes:</b> ".$row['Notes']."<br />";
                  $output .= "<b>Subject:</b> ".$row['Subject']."<br />";
                  $output .= "<b>Comments:</b> ".$row['Comments']."<br />";
                  $output .= "<b>Is on Loan:</b> ".$row['isOnLoan']."<br
/><br />";
            }
```

```
            echo $output;
      }
      else {echo "There was no matching record for the title
<b>".$searchTerm."</b>";}
?>
```

# 4. POGLAVLJE

## Uskladištene procedure (engl. STORED PROCEDURES)

Iako predstavljaju novu mogućnost u okviru MySQL-a, odavno postoje u ostalim RDBMS. Uskladištene procedure su brze, a efekat brzine se postiže pre svega kroz smanjenje mrežnog saobraćaja. Naročito su pogodne za ponavljajuće zadatke koji zahtevaju proveru, iteraciju, sa malo ili bez interakcije sa korisnikom Ako promenimo jezik za pristup bazi podataka ne bi trebalo da bude problema jer je logika u bazi podataka a ne aplikaciji. Sintaksa uskladištenih procedura MySQL-a je bliska SQL:2003 standardu, tako da se lako mogu primeniti i na drugim RDBMS. S obzirom da su uskladištene procedure uvedene u verziji 5, neophodno je prvo proveriti koja verzija je instalirana na računaru, da bismo bili sigurni da ih uopšte možemo koristiti.

```
SELECT VERSION();
```

Uskladištena procedura (stored procedure) je procedura (potprogram ili metod u programskim jezicima) koja je smeštena u bazi podataka. MySQL podržava dve vrste ovakvih procedura:

- uskladištene procedure koje ne vraćaju vrednost,
- funkcije koje vraćaju vrednosti na isti način kao i funkcije ugrađene u MySQL.

Sintaksa kreiranja funkcije:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parametri[,...]])
    [karakteristike ...] telo_rutine

proc_parametri:
  [ IN | OUT | INOUT ] ime_param tip
tip:
  MySQL tip podatka
karakteristike:
  COMMENT 'string'
 | LANGUAGE SQL
 | [NOT] DETERMINISTIC
 | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
 | SQL SECURITY { DEFINER | INVOKER }
telo_rutine:
  Validne SQL rečenice
```

Primer Hello World:

```
DELIMITER //
CREATE PROCEDURE HelloWorld()
BEGIN
SELECT 'Hello World';
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL HelloWorld();
```

Uskladištena procedura ima naziv, listu parametara i sadrži jednu ili više SQL naredbi.

```
DELIMITER //
CREATE PROCEDURE pr1()
BEGIN
DECLARE a INT Default 5;
select a;
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL pr1();
```

Naziv uskladištene procedure nije case senzitivan. U jednoj bazi sve uskladištene procedure moraju imati različite nazive, što znači da preklapanje procedura nije moguće. Naziv uskladištene procedure može sadržati maksimalno 64 znaka uključujući i praznine (space).

```
delimiter //
create procedure povrsinakruga (in r double, out a double)
begin
set a = r * r * pi();
end; //
delimiter ;
```

Pozivanje uskladištene procedure:

```
call povrsinakruga(10, @a);
select @a;
```

Uzimanje podataka iz tabele (books.sql)

```
DELIMITER //
CREATE PROCEDURE pr2()
BEGIN
SELECT * FROM authors;
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL pr2();
```

Uslovno uzimanje podataka iz tabele (books.sql)

```
DELIMITER //
CREATE PROCEDURE pr3(in p char(2))
BEGIN
SELECT * FROM authors WHERE state = p ;
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL pr3('NY');
```

Dodavanje podataka u tabelu sa proverom parametra (books.sql)

```
DELIMITER //
CREATE PROCEDURE add_author(au_id CHAR(3), au_fname VARCHAR(15), au_lname
VARCHAR(15), phone VARCHAR(12), address VARCHAR(20) , city VARCHAR(15) ,
state CHAR(2) , zip CHAR(5))

SQL SECURITY DEFINER
BEGIN
IF au_id='' OR au_fname='' OR au_lname='' OR phone='' OR address='' OR
city='' OR state='' OR zip=''
THEN
SELECT 'Warning: Bad parameters';
END IF;
INSERT INTO authors VALUES(au_id, au_fname, au_lname, phone, address, city,
state, zip);
END; //
DELIMITER ;
```

```
CALL add_author ('A08','Michael','Polk','512-953-1231','4028 Guadalupe
St','Austin','TX','78701');
```

test.sql

```
CREATE DATABASE test;
USE test;
CREATE TABLE t (s1 INT);
INSERT INTO t VALUES (5);
```

Uskladištena procedura sa WHILE strukturom (test.sql)

```
DELIMITER //
CREATE PROCEDURE pr4()
BEGIN
DECLARE v INT;
SET v = 0;
WHILE v < 5 DO
INSERT INTO t VALUES (v);
SET v = v + 1;
END WHILE;
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL pr4();
```

Nakon pozivanja procedure pr4, u tabeli t se zapisuju brojevi od 0 do 4.

Uskladištena procedura sa CASE strutkturom (test.sql)

```
DELIMITER //
CREATE PROCEDURE pr5(IN parameter1 INT)
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
CASE variable1
WHEN 1 THEN INSERT INTO t VALUES (17);
WHEN 2 THEN INSERT INTO t VALUES (18);
ELSE
INSERT INTO t VALUES (19);
```

```
END CASE;
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL pr5(0);
CALL pr5(1);
CALL pr5(11);
CALL pr5(NULL);
```

Uskladištena procedura sa IF THEN ELSE strukturom (test.sql)

```
DELIMITER //
CREATE PROCEDURE pr6 (IN parameter1 INT)
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
IF variable1 = 1 THEN
INSERT INTO t VALUES (17);
END IF;
IF parameter1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
CALL pr6(0);
CALL pr6(1);
```

Sledeći primer kreira proceduru ‘pop_tabele’ kojom se popunjava tabela ‘korisnici’ sa brojem korisnika koji je određen ulaznim parametrom. Ime, prezime i email se dobijaju spajanjem stringova naredbom CONCAT() a iznos se slučajno generiše naredbom RAND().

```
CREATE TABLE korisnici (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  ime varchar(45) NOT NULL DEFAULT '',
  prezime varchar(45) NOT NULL DEFAULT '',
  iznos double(10,2) DEFAULT NULL,
  email varchar(100) DEFAULT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY email (email)
)
ENGINE=InnoDB AUTO_INCREMENT=8
DEFAULT CHARSET=utf8
COMMENT='Test tabela za ugradjenu proceduru'
```

Uskladištena procedura za popunjavanje tabele podacima

```
DELIMITER //
CREATE PROCEDURE pop_tabele (IN p_broj_korisnika INT, OUT p_poruka
VARCHAR(255))
COMMENT 'Ova procedura sluzi za popunjavanje tabele ''korisnici''  '
BEGIN
  -- deklaracija potrebnih varijabli
  DECLARE v_counter INT DEFAULT 0;
```

```
  -- petlja za unos podataka
  WHILE v_counter < p_broj_korisnika  DO

   -- povecaj brojac za 1
   SET v_counter = v_counter + 1;

   INSERT INTO korisnici (id, ime, prezime, iznos, email)
   VALUES ( NULL,
            CONCAT('ime_', v_counter),
            CONCAT('prezime_', v_counter),
            RAND() * (10/RAND()) * 1000,
            CONCAT('email', MOD(v_counter, 5), '@domena', MOD(v_counter,
7), '.com')
          );
  END WHILE;

  -- sastavimo OUT poruku
  SET p_poruka = CONCAT('Broj unesenih korisnika: ', v_counter);

  -- prikazimo sadrzaj poruke:
  SELECT p_poruka;

END; //
DELIMITER ;
```

Pozivanje uskladištene procedure:

```
call pop_tabele(5,@poruka);
```

Brisanje procedure

```
drop procedure pop_tabele;
```

## Uskladištene funkcije

Funkcije su programi koji nakon pozivanja vraćaju vrednost, moraju uvek da vrate vrednost i uvek vraćaju samo jednu vrednost. Mogu biti pozvane iz SQL naredbe. DETERMINISTIC karakteristika se koristi samo unutar funkcije. Funkcija je deterministička ako vraća istu vrednost svaki put za isti skup parametara.

Sintaksa kreiranja funkcije:

```
CREATE FUNCTION function_name (parameter[,...])
RETURNS datatype
[LANGUAGE SQL]
[ [NOT] DETERMINISTIC ]
[ {CONTAINS SQL | NO SQL | MODIFIES SQL DATA | READS SQL DATA} ]
[ SQL SECURITY {DEFINER|INVOKER} ]
[ COMMENT comment_string ]
function_statements
```

```
DELIMITER //
CREATE FUNCTION fnc1 (parametar VARCHAR(10))
RETURNS varchar(10)
DETERMINISTIC
BEGIN
declare izlaz VARCHAR(10) default "Nije A";
   if parametar = "A" then
      set izlaz := "Jeste A";
   end if;
   return izlaz;
END; //
DELIMITER ;
```

Pozivanje uskladištene funkcije.

```
select fnc1('A');
select fnc1('B');
```

Brisanje uskladištenih funkcija

```
drop function fnc1;
```

Funkcija za izračunavanje obima kruga za dati prečnik.

```
delimiter //
create function obimkruga (r double)
returns double
deterministic
begin
declare c double;
set c = 2 * r * pi();
return c;
end //
delimiter ;
```

Pozivanje uskladištene funkcije.

```
select obimkruga(3);
```

Funkcija za izračunavanje faktorijala datog broja.

```
DELIMITER //
CREATE FUNCTION factorial (n DECIMAL(3,0))
RETURNS DECIMAL(20,0)
DETERMINISTIC
BEGIN
DECLARE factorial DECIMAL(20,0) DEFAULT 1;
DECLARE counter DECIMAL(3,0);
SET counter = n;
factorial_loop: REPEAT
SET factorial = factorial * counter;
SET counter = counter - 1;
UNTIL counter = 1
END REPEAT;
RETURN factorial;
END; //
DELIMITER ;
```

```
select factorial(9);
```

Funkcija za brojanje karatera.

```
DELIMITER //
CREATE FUNCTION charcount(addressee TEXT)
  RETURNS TEXT
DETERMINISTIC
BEGIN
  DECLARE strlen INT;
  SET strlen = LENGTH(addressee);
  RETURN CONCAT('Hello ', addressee, ' - your parameter has ', strlen, '
characters');
END; //
DELIMITER ;
```

```
select charcount ('Subotica-Tech');
```

Funkcija konvertovanje niza karaktera u ASCII vrednosti.

```
DELIMITER //
CREATE FUNCTION ascii_string (in_string VARCHAR(80) )
RETURNS VARCHAR(256)
DETERMINISTIC
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE string_len INT;
DECLARE out_string VARCHAR(256) DEFAULT '';
SET string_len=LENGTH(in_string);
WHILE (i<string_len) DO
SET out_string=CONCAT(out_string,ASCII(SUBSTR(in_string,i,1)),' ');
SET i=i+1;
END WHILE;
RETURN (out_string);
END; //
DELIMITER ;
```

```
SELECT ascii_string('MySQL Rocks!')
```

Funkcija za brojanje podstringova u nizu.

```
DELIMITER //
CREATE FUNCTION count_strings
(in_string VARCHAR(256),in_substr VARCHAR(128))
RETURNS INT
DETERMINISTIC
BEGIN
DECLARE l_count INT DEFAULT 0;
DECLARE l_start INT DEFAULT 1;
DECLARE l_pos INT;
MainLoop:
LOOP
SET l_pos=LOCATE(in_substr,in_string,l_start);
IF l_pos=0 THEN
LEAVE MainLoop;
ELSE
SET l_count=l_count+1;
SET l_start=l_pos+1;
END IF;
END LOOP;
RETURN(l_count);
END; //
DELIMITER ;
```

```
SELECT count_strings('She sells sea shells by the sea shore','sea') as
count
```

### Pozivanje uskladištenih procedura pomoću php

```
DELIMITER //
CREATE PROCEDURE HelloWorld()
BEGIN
SELECT 'Hello World';
END; //
DELIMITER ;
```

Stored1.php

```
<?php
$mysqli = new mysqli("localhost", "admin", "admin", "books");
if (mysqli_connect_errno( )) {
printf("Connect failed: %s <p>", mysqli_connect_error( ));
exit ( );
} else {
printf("Connect succeeded <p>");
}

$sql = 'call HelloWorld( )';
$mysqli->query($sql);
if ($mysqli->errno) {
die("Execution failed: ".$mysqli->errno.": ".$mysqli->error);
}
else {
printf("Stored procedure execution succeeded <p>");
}
?>
```

**Uskladištena procedura sa ulaznim parametrom i rezultatima niza**

```
DELIMITER //

CREATE PROCEDURE authors_for_rep(in_state CHAR(2))
      READS SQL DATA
BEGIN
       SELECT au_fname,au_lname
         FROM authors
            WHERE state=in_state;

END; //
DELIMITER ;
```

Stored2.php

```
<?php
$mysqli = new mysqli("localhost", "admin", "admin", "books");
if (mysqli_connect_errno( )) {
printf("Connect failed: %s <p>", mysqli_connect_error( ));
exit ( );
} else {
printf("Connect succeeded <p>");
}

      $sql = "CALL authors_for_rep(?)";
      $stmt = $mysqli->prepare($sql);
      if ($mysqli->errno) {die($mysqli->errno.":: ".$mysqli->error);}

      $stmt->bind_param("s", $authors_rep_id);
      $authors_rep_id = 'CA';
      $stmt->execute();
      if ($mysqli->errno) {die($mysqli->errno.": ".$mysqli->error);}

      $stmt->bind_result($au_fname,$au_lname);
      while ($stmt->fetch()) {
            printf("%s %s <p>", $au_fname,$au_lname);
      }
?>
```

# 5. POGLAVLJE

## Pogledi (engl. VIEW)

Prikaz je uskladišten iskaz SELECT koji vraća tabelu čiji su podaci izvedeni iz jedne ili više drugih tabela, koje se nazivaju podložne (engl. underlying) tabele. Neka važna svojstva prikaza su:

- Podložne tabele prikaza mogu da budu bazne tabele, privremene tabele ili drugi prikazi.
- Prikaz se naziva virtuelnom ili izvedenom tabelom da bi se razlikovao od bazne ili privremene tabele.
- DBMS skladišti prikaz samo kao SQL iskaz, ne kao skup vrednosti podataka, sprečavajući na taj način redundantnost podataka.
- Prikaz se dinamički materijalizuje kao fizička tabela kada se referencira imenom u SQL iskazu. On postoji samo za vreme trajanja iskaza i iščezava kada se iskaz završi.
- Prikaz je skup imenovanih kolona i redova podataka, tako da ga možemo koristiti skoro svuda gde bismo upotrebili pravu tabelu.
- Nema ograničenja u pogledu pravljenja upita (SELECT) kroz prikaze. U nekim slučajevima, prikazi se mogu ažurirati, što za posledicu ima prenošenje izmena na podložne bazne tabele.
- Radi zatvorenosti, prikaz je uvek jedna tabela bez obzira na to koliko podložnih tabela referencira ili kako su te tabele kombinovane.

## Pravljenje prikaza pomoću CREATE VIEW

Posmatrajmo prikaz kao iskrojenu predstavu u obliku rešetkastog prozora koji gleda u jednu ili više baznih tabela. Prozor može da prikaže celokupnu baznu tabelu, deo bazne tabele ili kombinaciju baznih tabela (ili njihovih delova). Prikaz takođe može da odrazi podatke u baznim tabelama kroz druge prikaze - prozori u prozorima. Uglavnom, SQL programeri koriste prikaze za predstavljanje podataka krajnjim korisnicima u aplikacijama baza podataka. Prikazi nude sledeće prednosti:

**Pojednostavljen pristup podacima** Prikazi sakrivaju složenost podataka i pojednostavljuju iskaze, tako da korisnici mogu obavljati operacije nad prikazom lakše nego direktno na baznoj tabeli. Ako napravimo složen prikaz - jedan koji obuhvata, recimo, više baznih tabela, spojeva i podupita - korisnici mogu da vrše upite u ovom prikazu, a da ne moraju razumeti složene relacione koncepte ili uopšte znati da se radi o više tabela.

**Automatsko ažuriranje** Kada se bazna tabela ažurira, svi prikazi koji referenciraju tabelu automatski odražavaju promenu. Ako u tabelu authors umetnemo red koji predstavlja novog autora, na primer, svi prikazi definisani nad tabelom authors automatski će odraziti novog autora. Ovakvom šemom čuva se prostor diska i sprečava se redundantnost, jer bi bez prikaza DBMS morao da skladišti izvedene podatke da bi ih održao sinhronizovanim.

**Povećana bezbednost** Jedna od najčešćih upotreba prikaza jeste sakrivanje podataka od korisnika putem filtriranja podložnih tabela. Pretpostavimo da tabela employees sadrži kolone salary i commission. Ako nad tabelom employees napravimo prikaz koji izostavlja ove dve kolone, ali sadrži ostale bezazlene kolone (poput email_address), administrator baze podataka može da korisnicima

odobri dozvole za pregledanje prikaza, ali ne i pregledanje podložne tabele, čime se podaci o novčanoj nadoknadi sakrivaju od radoznalih.

**Logička nezavisnost podataka** Bazne tabele obezbeđuju stvaran prikaz baze podataka. Međutim, kada koristimo SQL da bismo izgradili aplikaciju baze podataka, mi krajnjim korisnicima ne želimo da predstavimo stvaran prikaz, već virtuelni prikaz specifičan za aplikaciju. Virtuelni prikaz sakriva one delove baze podataka (celokupne tabele ili određene redove ili kolone) koji nisu relevantni za aplikaciju. Na taj način korisnici ostvaruju interakciju sa virtuelnim prikazom, koji je izveden iz - premda nezavisan od - stvarnog prikaza koji predstavljaju bazne tabele.

Da bismo napravili prikaz upišimo sledeće:

```
CREATE VIEW prikaz [(kolone_prikaza)]
AS iskaz_select;
```

*prikaz* je ime prikaza koji se pravi. Ime prikaza mora biti jedinstveno unutar baze podataka.

*kolone_prikaza* je opcioni spisak, obuhvaćen zagradama, od jednog ili više zarezom odvojenih imena koja će se koristiti za kolone u prikazu. Broj kolona u spisku kolone_prikaza mora odgovarati broju kolona u rečenici SELECT iskaza_select. (Ako jednu kolonu imenujemo na ovaj način, moramo ih sve imenovati na ovaj način.) Navedimo kolone_prikaza kada je kolona u iskazu_select izvedena iz aritmetičkog izraza, funkcije ili literala; kada bi inače dve ili više kolona prikaza imale isto ime (obično usled spoja); ili da bismo koloni u prikazu dali ime različito od imena kolone iz koje je izvedena. Ako se spisak kolone_prikaza izostavi, prikaz nasleđuje imena kolona od iskaza_select. Imena kolona mogu se takođe dodeliti u iskazu_select putem rečenica AS. Svako ime kolone mora biti jedinstveno unutar prikaza.

*iskaz_select* je iskaz SELECT koji određuje kolone i redove tabele (ili tabela) na kojima se prikaz zasniva.

Napraviti prikaz koji sakriva lične podatke o autorima (telefonske brojeve i adrese).

```
CREATE VIEW au_names
AS
SELECT au_id, au_fname, au_lname
FROM authors;
```

Napraviti prikaz koji daje spisak autora koji žive u gradu u kojem se nalazi izdavač. Primetimo da u prikazu se koriste imena kolona au_city i pub_city. Preimenovanjem ovih kolona razrešava se neodređenost do koje bi došlo da su obe kolone nasledile isto ime kolone city od podložnih tabela.

```
CREATE VIEW cities
(au_id, au_city, pub_id, pub_city)
AS
SELECT a.au_id, a.city, p.pub_id, p.city
FROM authors a
INNER JOIN publishers p
ON a.city = p.city;
```

Napraviti prikaz koji daje ukupan prihod (= cena x prodaja) grupisan po tipu knjige u okviru izdavača. Ovaj prikaz će kasnije biti lako upitivati, jer rezultat aritmetičkog izraza eksplicitno imenujemo, umesto da smo pustili da DBMS dodeli podrazumevano ime.

```
CREATE VIEW revenues
(Publisher, BookType, Revenue)
AS
SELECT pub_id, type, SUM(price * sales)
FROM titles
GROUP BY pub_id, type;
```

Napraviti prikaz koji olakšava štampanje poštanskih adresa autora. Primetimo da smo dodelili imena kolona u rečenici SELECT, a ne u rečenici CREATE VIEW.

```
CREATE VIEW mailing_labels
AS
SELECT
CONCAT(au_fname, ' ', au_lname)
AS "address1",
TRIM(address)
AS "address2",
CONCAT(city , ', ' , state ,' ' , zip)
AS "address3"
FROM authors;
```

Napraviti prikaz koji daje spisak prezimena autora A02 i A05 i knjiga koje je svaki od njih napisao (ili učestvovao u pisanju). Obratimo pažnju na to da ovaj iskaz koristi ugnežden prikaz: on referencira prikaz au_names napravljen ranije.

```
CREATE VIEW au_titles (LastName, Title)
AS
SELECT an.au_lname, t.title_name
FROM title_authors ta
INNER JOIN au_names an
ON ta.au_id = an.au_id
INNER JOIN titles t
ON t.title_id = ta.title_id
WHERE an.au_id in ('A02','A05');
```

## Dobijanje podataka kroz prikaz

Kada napravimo prikaz, ne prikazuje se ništa. Nailaskom na CREATE VIEW, DBMS skladišti prikaz kao imenovani iskaz SELECT i to je sve. Da bismo videli podatke kroz prikaz, moramo izvršiti upit nad prikazom pomoću SELECT, kao što bismo to učinili nad tabelom. Možemo da:

- preuredimo redosled prikazanih kolona pomoću rečenice SELECT;
- koristimo operatore i funkcije da bismo obavili izračunavanje;
- izmenimo zaglavlje kolone pomoću AS;
- filtriramo redove pomoću WHERE;
- grupišemo redove pomoću GROUP BY;
- filtriramo grupisane redove pomoću HAVING;
- spojimo prikaz sa drugim prikazima, tabelama i privremenim tabelama pomoću JOIN;
- sortiramo rezultat pomoću ORDER BY.

Da bismo dobili podatke kroz prikaz upišimo sledeće:

```
SELECT kolone
FROM prikaz
[JOIN spojevi]
[WHERE uslov_pretrage]
[GROUP BY kolone_grupe]
[HAVING uslov_pretrage]
[ORDER BY sort_kolone]
```

*prikaz* je ime prikaza koji se upituje. Rečenice rade sa prikazima na isti način kao sa tabelama.

Dati spisak svih redova i kolona prikaza au_titles.

```
SELECT *
FROM au_titles;
```

Dati spisak svih različitih gradova u prikazu cities.

```
SELECT DISTINCT au_city
FROM cities;
```

Dati spisak tipova knjiga čiji prosečni prihod premašuje milion dolara.

```
SELECT BookType,
AVG(Revenue) AS "AVG(Revenue)"
FROM revenues
GROUP BY BookType
HAVING AVG(Revenue) > 1000000;
```

Ispisati treći red poštanske adrese svakog autora čije ime sadrži string Kell.

```
SELECT address3
FROM mailing_labels
WHERE address1 LIKE '%Kell%';
```

Dati spisak imena svih autora koji barem u jednoj knjizi nisu bili vodeći autori.

```
SELECT DISTINCT an.au_fname, an.au_lname
FROM au_names an
INNER JOIN title_authors ta
ON an.au_id = ta.au_id
WHERE ta.au_order > 1;
```

Dati spisak imena autora koji su iz Californije.

```
SELECT au_fname, au_lname
FROM au_names
WHERE state = 'CA';
```

Prikaz au_names referencira tabelu authors, ali sakriva kolonu state, tako da obraćanje koloni state kroz prikaz uzrokuje **grešku**.

## Ažuriranje podataka kroz prikaz

Prikaz koji se može ažurirati je prikaz na koji možemo da primenimo operacije INSERT, UPDATE i DELETE da bi se izmenili podaci u podložnoj tabeli (ili tabelama). Sve promene učinjene u prikazu koji

se može ažurirati uvek nedvosmisleno prolaze do bazne tabele (tabela). Sintaksa iskaza INSERT, UPDATE i DELETE ista je za prikaze kao za tabele.
Prikaz koji se ne može ažurirati (ili prikaz samo za čitanje) je onaj koji ne podržava operacije INSERT, UPDATE i DELETE, jer bi izmene bile dvosmislene. Da bismo izmenili podatke koji se pojavljuju u prikazu koji je samo za čitanje, moramo direktno da izmenimo podložnu tabelu (ili tabele).
Svakom redu u prikazu koji se može ažurirati pridružen je tačno jedan red u podložnoj baznoj tabeli. Prikaz se ne može ažurirati ako njegov iskaz SELECT koristi GROUP BY, HAVING, DISTINCT ili agregatne funkcije, na primer.
Standard SQL-92 kazivao je da prikaz koji se može ažurirati mora biti definisan nad samo jednom tabelom, što je strogo, ali veoma bezbedno. SQL:1999 popustio je ovo ograničenje, jer se pojavilo mnogo više tipova prikaza koji se mogu ažurirati. Do vremena kad je taj standard objavljen, proizvođači DBMS-ova su već ponudili proširen skup prikaza koji se mogu ažurirati. Prikazi nad jednom tabelom uvek se mogu ažurirati. DBMS-ovi takođe ispituju spojeve podložnih tabela i ograničenja referencijalnog integriteta višetabelarnog prikaza kako bi odredili da li se prikaz srne ažurirati. Evo nekoliko tipova upita koji mogu da definišu prikaze koji se mogu ažurirati:

- unutrašnji spojevi jedan-jedan;
- spoljašnji spojevi jedan-jedan;
- unutrašnji spojevi jedan-prema-više;
- spoljašnji spojevi jedan-prema-više;
- spojevi vise-prema-vise;
- upiti sa UNION i EXCEPT.

Primeri u ovom odeljku koriste prikaze koji se mogu ažurirati i koji referenciraju samo jednu podložnu tabelu. Pogledajmo dokumentaciju našeg DBMS-a da bismo saznali koje višetabe-larne prikaze možemo da ažuriramo i kako ovo ažuriranje utiče na svaku baznu tabelu.

Napraviti i izložiti prikaz ny_authors, koji ispisuje ID, ime i državu samo onih autora koji su iz države New York.

```
CREATE VIEW ny_authors
AS
SELECT au_id, au_fname, au_lname, state
FROM authors
WHERE state = 'NY';
SELECT *
FROM ny_authors;
```

Umetnuti novi red kroz prikaz ny_authors.

```
INSERT INTO ny_authors
VALUES('A08','Don','Dawson','NY');
```

Umetnuti novi red kroz prikaz ny_authors. DBMS bi poništio ovo umetanje da je rečenica WITH CHECK OPTION korišćena kada je napravljen prikaz ny_authors.

```
INSERT INTO ny_authors
VALUES('A09','Jill','LeFlore','CA');
```

**Umetanje reda kroz prikaz**

Razmotrimo prikaz ny_authors koji sadrži ID, ime i državu samo onih autora koji su iz države New York. ny_authors referencira samo baznu tabelu authors.

INSERT umeće novi red kroz prikaz. DBMS umeće novi red u tabelu authors. Red sadrži A08 u koloni au_id, Don u koloni au_fname, Dawson u koloni au_lname i NY u koloni state. Ostale kolone u redu - phone, address, city i zip - postavljaju se na null (ili na njihove podrazumevane vrednosti ako postoji ograničenje DEFAULT).
U slučaju da je novi autor iz Californije, a ne države New York, čime se krši uslov WHERE u definiciji prikaza postavlja se sledeće pitanje. Da li DBMS umeće red ili poništava operaciju? Odgovor zavisi od toga kako je prikaz napravljen. U ovom posebnom slučaju, umetanje je dozvoljeno, jer u iskazu CREATE VIEW nema rečenice WITH CHECK OPTION, tako da DBMS nije primoran da održava doslednost sa originalnom definicijom prikaza. Informacije o rečenici WITH CHECK OPTION možemo naći u DBMS savetu u odeljku o pravljenju prikaza pomoću CREATE VIEW ranije u ovom poglavlju. DBMS bi poništio umetanje daje prikaz ny_authors bio definisan ovako:

```
CREATE VIEW ny_authors
AS
SELECT au_id, au_fname, au_lname, state
FROM authors
WHERE state = 'NY'
WITH CHECK OPTION;
```

**Ažuriranje reda kroz prikaz**

Sledeći listing ažurira postojeći red kroz prikaz. DBMS ažurira red za autora A01 u tabeli authors izmenom imena autora od Sarah Buchman u Yasmin Howcomely. Vrednosti u ostalim kolonama u redu - au_id, phone, address, city, state i zip - ne menjaju se.
Ali pretpostavimo da listing izgleda ovako:

```
UPDATE ny_authors
SET au_fname = 'Yasmin',
au_lname = 'Howcomely',
state = 'CA'
WHERE au_id = 'A01';
```

željena promena prouzrokovala bi da Yasminin red više ne zadovoljava uslove pripadnosti prikazu. Ponovo, DBMS će prihvatiti ili odbaciti UPDATE u zavisnosti od toga da li je rečenica WITH CHECK OPTION navedena kad je prikaz napravljen. Ako se koristi WITH CHECK OPTION, redovi se ne mogu izmeniti na način koji uzrokuje da oni nestanu iz prikaza.

Ažurirati postojeći red kroz prikaz ny_authors.

```
UPDATE ny_authors
SET au_fname = 'Yasmin',
au_lname = 'Howcomely'
WHERE au_id = 'A01';
```

**Brisanje reda kroz prikaz**

Ažuriranje prikaza može da ima povratne posledice po integritet, naravno. DBMS neće dozvoliti brisanje ako se uklanjanjem reda narušava ograničenje referencijalnog integriteta. Ako obrišemo red, sva podložna ograničenja FOREIGN KEY u pridruženim tabelama moraju i dalje biti zadovoljena da bi brisanje uspelo. Neka ažuriranja mogu se sprovesti pomoću opcije CASCADE (ako je navedena)

DBMS briše red autora A05 u tabeli authors. (Svaka kolona u redu se briše, ne samo one koje su u prikazu.) Kao posledica, red nestaje iz prikaza ny_authors.

Obrisati red kroz prikaz ny_authors.

```
DELETE FROM ny_authors
WHERE au_id = 'A05';
```

U gornjem listingu, na primer, DBMS će **poništiti** DELETE ako prvo ne izmenimo ili obrišem vrednosti spoljnog ključa u tabeli title_authors koje pokazuju na autora A05 u tabeli authors.

## Odbacivanje prikaza pomoću DROP VIEW

Koristimo iskaz DROP VIEW da bismo uništili prikaz. Kako je prikaz fizički nezavisan od svoje podložne tabele (ili tabela), možemo u bilo kom trenutku odbaciti prikaz bez uticaja na tabelu (tabele). Međutim, svi SQL programi, aplikacije i drugi prikazi koji referenciraju odbačeni prikaz će pasti.

Da bismo odbacili prikaz upišimo sledeće:

```
DROP VIEW prikaz;
```

*prikaz* je ime prikaza koji se odbacuje.

Odbaciti prikaz ny_authors.

```
DROP VIEW ny_authors;
```

Odbacivanjem tabele ne odbacuju se prikazi koji referenciraju tu tabelu, tako da eksplicitno moramo odbaciti prikaze pomoću DROP VIEW.

# 6. POGLAVLJE

## Transakcije (START TRANSACTION, COMMIT, ROLLBACK)

Transakcija predstavlja sekvencu od jednog ili više SQL iskaza koji se izvršavaju kao jedna logička celina posla. DBMS transakciju smatra nedeljivom, kao poduhvat sve-ili-ništa: on izvršava sve iskaze transakcije kao grupu ili ne izvršava nijedan. Kanonički zakon nalaže nam da slikovito prikažemo važnost transakcija na bankovnom primeru. Pretpostavimo da klijent želi da prenese 500 EUR sa svog štednog računa na svoj čekovni račun. Ova operacija sastoji se od dve odvojene radnje, koje se izvršavaju uzastopno:

1. Umanjiti saldo štednje za 500 EUR.
2. Povećati saldo na čekovnom računu za 500 EUR.

```
UPDATE savings_accounts
SET balance = balance - 500.00
WHERE account_number = 1009;

UPDATE checking_accounts
SET balance = balance + 500.00
WHERE account_number = 6482;
```

Potrebna su dva SQL iskaza kada klijent banke obavlja transfer novca sa štednje na čekovni račun.

Gornji primer prikazuje dva SQL iskaza za ovu transakciju. Sada zamislimo da DBMS zataji - nestanak napajanja, slom sistema, problem sa hardverom - nakon što izvrši prvi iskaz, ali pre drugog. Došlo bi do poremećaja salda bez našeg znanja. Ubrzo bi usledile optužbe za kršenje zakona i pretnje zatvorom.
Da bismo izbegli dosije u policiji, koristimo transakciju kako bismo garantovali da su oba SQL iskaza izvršena da bi se održao pravilan saldo na računima. Kada nešto spreči da se jedan od iskaza u transakciji izvrši, DBMS poništava (opoziva) druge iskaze transakcije. Ako ne dođe do greške, izmene postaju trajne (predate).

## Izvršavanje transakcije

Da bismo naučili kako transakcije funkcionišu, treba da znamo nekoliko pojmova:

**Predaja** Predavanjem transakcije sve pro-mene nad podacima počinjene od početka transakcije postaju trajni deo baze podataka. Posle predaje transakcije, sve izmene nastale usled dejstva transakcije postaju vidljive dragim korisnicima i garantovano su trajne za slučaj da dođe do pucanja sistema ili drugog otkaza.

**Opoziv** Opozivom transakcije povlače se sve promene koje su proizveli SQL iskazi unutar transakcije. Posle opoziva transakcije, pogođeni podaci ostaju neizmenjeni, kao da se SQL iskazi unutar transakcije nikad nisu izvršili.

**Dnevnik transakcije** Datoteka dnevnika transakcije, ili samo dnevnik, je serijski zapis svih izmena koje su se desile u bazi podataka putem transakcija. Dnevnik transakcija beleži početak svake transakcije, izmene nad podacima i dovoljno informacija da se povuku ili ponovo počine izmene koje

obavlja transakcija (ako je kasnije potrebno). Dnevnik neprestano raste kako se transakcije pojavljuju u bazi podataka.

Premda je odgovornost DBMS-a da osigura fizički integritet svake transakcije, naša je odgovornost da započnemo i završimo transakcije u tačkama koje su značajne za logičku doslednost podataka, prema pravilima naše organizacije, odnosno posla. Transakcija treba da sadrži samo SQL iskaze koji su neophodni za pravljenje dosledne izmene - ništa više i ništa manje. Podaci u svim referenciranim tabelama moraju biti u konzistentnom stanju pre nego što transakcija počne i nakon što se ona završi.

Kada planiramo i izvršavamo transakcije, neka važna razmatranja su:

- SQL iskazi vezani za transakcije menjaju podatke, tako da će možda biti potrebno da nam administrator naše baze podataka odobri dozvolu da ih izvršavate.
- Izvršavanje u obliku transakcije primenjuje se za iskaze koji menjaju podatke ili objekte baze podataka (INSERT, UPDATE, DELETE, CREATE, ALTER, DROP - spisak se menja u zavisnosti od DBMS-a). Za proizvodne baze podataka, svaki ovakav iskaz trebalo bi izvršavati u okviru transakcije.
- Za predatu transakciju kaže se da je po-stojana, što znači da njene izmene trajno ostaju na svom mestu, čak i ako sistem otkaže.
- DBMS-ov mehanizam oporavka podataka zavisi od transakcija. Kada se DBMS vrati u život posle otkaza, on proverava svoj dnevnik transakcija kako bi video da li su sve transakcije predate bazi podataka. Ako naiđe na nepredate (delimično obavljene) transakcije, on ih prema dnevniku opoziva (vraća unazad). Moramo ponovo da podnesemo opozvane transakcije (iako neki DBMS-ovi mogu automatski da dovedu do kraja nezavršene transakcije).
- DBMS-ova sredstva za rezervne kopije/ oporavak zavise od transakcija. Mehanizam rezervnih kopija uzima redovne snimke baze podataka i čuva ih sa (sledećim) dnevnicima transakcija na rezervnom disku. Pretpostavimo da dođe do oštećenja proizvodnog diska, i to takvog da podaci i dnevnik transakcija postanu nečitljivi. Možemo da pokrenemo mehanizam oporavka, koji će iskoristiti najsvežiju rezervnu kopiju baze podataka i zatim izvršiti, odnosno pokrenuti unapred, sve predate transakcije u dnevniku od trenutka kada je uzet snimak do poslednje transakcije iza koje je usledio otkaz. Ova operacija oporavka dovodi bazu podataka u njeno ispravno stanje pre otkaza. (Opet, moraćemo da ponovo podnesemo nepredate transakcije.)
- Iz očiglednih razloga, bazu podataka i njen dnevnik transakcija treba da skladištimo na odvojenim fizičkim diskovima.

Da bi se transakcija izvršila u maniru sve-ili-ništa, granice transakcije (početna i krajnja tačka) moraju biti jasne. Ove granice omogućuju DBMS-u da izvrši iskaze kao jednu ne-deljivu celinu posla. Transakcija može početi implicitno, sa prvim izvršivim SQL iskazom, ili eksplicitno, sa iskazom BEGIN TRANSACTION. Transakcija se eksplicitno završava iskazom COMMIT ili ROLLBACK (nikad se ne završava implicitno). Ne možemo opozvati transakciju nakon što je predamo.

**Da bismo eksplicitno započeli transakciju:**
Upišimo:

```
START TRANSACTION;
```

**Da bismo predali transakciju:**

Upišite:

```
COMMIT;
```

**Da bismo opozvali transakciju:**

Upišite:

```
ROLLBACK;
```

Unutar bloka transakcije, operacije UPDATE (kao i operacije INSERT i DELETE) nikad nisu konačne.

MySQL konzola (books)

```
USE books
SELECT SUM(pages), AVG(price) FROM titles;

START TRANSACTION;
/*BEGIN;*/
UPDATE titles SET pages = 0;
UPDATE titles SET price = price * 2;

SELECT SUM(pages), AVG(price) FROM titles;
ROLLBACK;

SELECT SUM(pages), AVG(price) FROM titles;
```

Iskazi SELECT u gornjem primeru pokazuju da je DBMS obavio operacije UPDATE, a da su one zatim poništene iskazom ROLLBACK.

Upotrebiti transakciju za brisanje izdavača P04 iz tabele publishers i brisanje redova u drugim tabelama koji su povezani sa ovim izdavačem.

```
START TRANSACTION;

DELETE FROM title_authors
WHERE title_id IN
(SELECT title_id
FROM titles
WHERE pub_id = 'P04');

DELETE FROM royalties
WHERE title_id IN
(SELECT title_id
FROM titles
WHERE pub_id = 'P04');

DELETE FROM titles
WHERE pub_id = 'P04';

DELETE FROM publishers
WHERE pub_id = 'P04';

COMMIT;
```

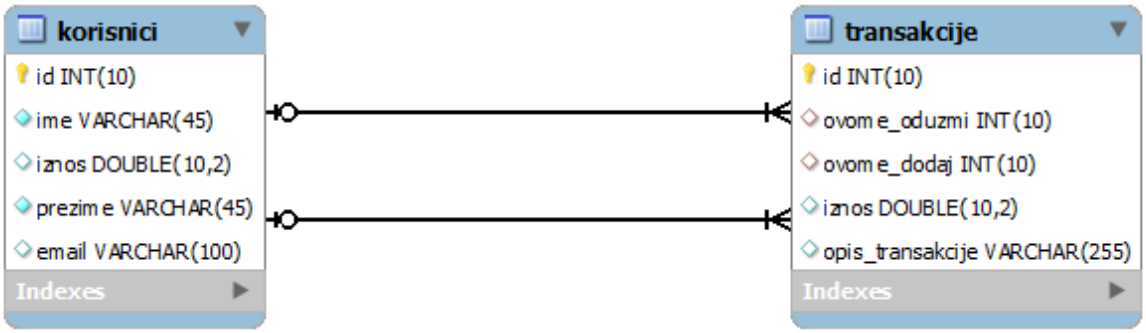
Gornji primer pokazuje praktičniji primer transakcije. Želimo da obrišemo izdavača P04 iz tabele publishers, a da se ne javi greška u referencijalnom integritetu. Pošto neke od vrednosti spoljnog ključa u tabeli titles pokazuju na izdavača P04 u tabeli publishers, najpre moramo da obrišemo povezane redove iz tabela titles, titles_authors i royalties. Koristimo transakciju kako bi bili sigurni da su svi iskazi DELETE izvršeni. Da su samo neki od iskaza bili uspešni, podaci bi bili ostavljeni u nedoslednom stanju.

**ACID -** je akronim koji iskazuje poželjna svojstva transakcije:

**Nedeljivost** (engl. atomicity) ili se obave sve izmene podataka unutar transakcije, ili nijedna.
**Doslednost** (engl. consistency) Obavljena transakcija ostavlja podatke u doslednom stanju kojim se održava integritet podataka.
**Izolacija** (engl. isolation) Dejstvo transakcije izolovano je od delovanja drugih transakcija. Obratimo pažnju na naslov „Kontrola konkurentnosti" ranije u ovom poglavlju.
**Postojanost** (engl. durability) Nakon što se transakcija završi, njen učinak je trajan čak i u slučaju otkaza sistema.

### Prvi primer

Za primer ćemo uzeti jednu novčanu transakciju, odnosno prebacivanje novca sa računa jednog korisnika na račun drugog korisnika. Kreirajmo dve jednostavne tabele ‘korisnici‘ i ‘transakcije‘, u kojima ćemo držati podatke o korisnicima i obavljenim transakcijama:




testdb.sql

```
CREATE TABLE korisnici (
       id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
       ime VARCHAR(45) NOT NULL DEFAULT '',
       iznos DOUBLE(10,2),
       prezime VARCHAR(45) NOT NULL DEFAULT '',
       email VARCHAR(100),
       PRIMARY KEY(id),
       UNIQUE INDEX (email)
     )
       ENGINE = InnoDB
       CHARACTER SET utf8
       COLLATE utf8_general_ci
       COMMENT = 'Test tabela sa korisnicima';

    CREATE TABLE transakcije (
       id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
       ovome_oduzmi INTEGER UNSIGNED,
       ovome_dodaj INTEGER UNSIGNED,
       iznos DOUBLE(10,2),
```

```
       opis_transakcije VARCHAR(255),
       PRIMARY KEY(id),
       CONSTRAINT FK_transakcije_oduzmi
         FOREIGN KEY FK_transakcije_oduzmi (ovome_oduzmi)
         REFERENCES korisnici (id)
         ON DELETE SET NULL,
       CONSTRAINT FK_transakcije_dodaj
        FOREIGN KEY FK_transakcije_dodaj (ovome_dodaj)
        REFERENCES korisnici (id)
        ON DELETE SET NULL
     )
      ENGINE = InnoDB
      CHARACTER SET utf8
      COLLATE utf8_general_ci
      COMMENT = 'Lookup tabela sa transakcijama';
```

Da bismo koristili ACID transakcije u MySQL-u, moramo imati aktiviran InnoDB engine. Nakon sto smo kreirali te dve tabele, popunićemo ih nekim test podacima, a za to ćemo upotrebiti jednu uskladištenu proceduru 'popuni_tabele'. Kreirajmo tu proceduru koristeći priloženi izvorni kôd i pozovimo je sa odgovarajućim parametrima – prvi parametar označava broj korisnika, koje želimo kreirati, a u drugom parametru će se nalaziti izlazna poruka procedure:

popuni_tabele.sql

```
DELIMITER //
CREATE PROCEDURE popuni_tabele (IN p_broj_korisnika INT, OUT p_poruka
VARCHAR(255))
LANGUAGE SQL
NOT DETERMINISTIC
MODIFIES SQL DATA
SQL SECURITY DEFINER
COMMENT 'Ova procedura sluzi za popunjavanje tabela ''korisnici'' i
''transakcije'' '
BEGIN
  -- deklaracija potrebnih varijabli
  DECLARE v_counter INT DEFAULT 0;
  DECLARE v_broj_transakcija INT DEFAULT 0;
  DECLARE v_oduzmi_id INT;
  DECLARE v_dodaj_id INT;
  DECLARE v_iznos DOUBLE(10,2);
  DECLARE nema_vise_redova BOOLEAN DEFAULT false;
  DECLARE cur_korisnici CURSOR FOR
      SELECT k1.id AS oduzmi, k2.id AS dodaj
        FROM korisnici k1, korisnici k2
       WHERE k1.id <> k2.id;

  DECLARE continue HANDLER FOR SQLSTATE '02000' SET nema_vise_redova =
true;
  DECLARE exit HANDLER FOR SQLEXCEPTION
    BEGIN
      ROLLBACK;
      SET p_poruka = 'Unos prekinut zbog SQL Exceptiona';
    END;
  -- oznacimo pocetak transakcije
  START TRANSACTION;
  -- petlja_za_unos_podataka
  WHILE v_counter < p_broj_korisnika
  DO
   -- povecaj brojac za 1
```

```
   SET v_counter = v_counter + 1;
   INSERT INTO korisnici (id, ime, prezime, iznos, email)
   VALUES ( NULL,
            CONCAT('ime_', v_counter),
            CONCAT('prezime_', v_counter),
            RAND() * (10/RAND()) * 1000,
            CONCAT('email', MOD(v_counter, 5), '@domena', MOD(v_counter,
7), '.com')
          );

  END WHILE;

  -- otvori cursor sa korisnicima
  OPEN cur_korisnici;

  -- otvori petlju sa korisnicima
  petlja_sa_korisnicima: LOOP
     -- pokupi vrednosti iz cursora
     FETCH cur_korisnici INTO v_oduzmi_id, v_dodaj_id;

     -- ako cursor ne sadrzi vise redova, zavrsi sa petljom
     IF nema_vise_redova THEN
        LEAVE petlja_sa_korisnicima;
     END IF;

     -- povecaj brojac transakcija za 1
     SET v_broj_transakcija = v_broj_transakcija + 1;

     SET v_iznos = RAND() * (MOD(v_broj_transakcija, 33) + 10);

     -- necemo vrsiti proveru koliko je trenutni korisnikov iznos,
     -- nego cemo samo uneti podatke radi demonstracije funkcionisanja:
      UPDATE korisnici
         SET iznos = iznos - v_iznos
       WHERE id = v_oduzmi_id;

      UPDATE korisnici
         SET iznos = iznos + v_iznos
       WHERE id = v_dodaj_id;

      -- unesi transakciju u tabelu
      INSERT INTO transakcije(id, ovome_oduzmi, ovome_dodaj, iznos,
opis_transakcije)
      VALUES (NULL, v_oduzmi_id, v_dodaj_id, v_iznos, CONCAT('Transakcija
broj ', v_broj_transakcija));

  -- zatvori petlju sa korisnicima
  END LOOP petlja_sa_korisnicima;

  CLOSE cur_korisnici;

  -- potvrdimo unos i oznacimo kao kraj transakcije
  COMMIT;

  -- sastavimo OUT poruku
  SET p_poruka = CONCAT('Broj unesenih korisnika: ', v_counter, '; Broj
obavljenih transakcija: ', v_broj_transakcija);
  -- prikazimo sadrzaj poruke:
  SELECT p_poruka;
END //
DELIMITER ;
```

Pozivanje procedure

```
call popuni_tabele(30, @poruka);
```

Sada bi se u tabelama trebalo nalaziti dovoljno podataka za testiranje. Za obavljanje jedne novčane transakcije, kreiraćemo proceduru 'obavi_transakciju', koja prihvata sledeće parametre:

- **p_oduzmi_id** (broj korisnika, kojemu se skida određeni iznos sa računa),
- **p_dodaj_id** (broj korisnika, kojemu se dodaje određeni iznos na račun),
- **p_iznos** (iznos, koji ce se jednom korisniku skinuti sa računa, a drugom dodati),
- **p_opis** (opis transakcije) i
- **p_poruka** (izlazna poruka procedure).

obavi_transakciju.sql

```
DELIMITER //
CREATE PROCEDURE obavi_transakciju(IN p_oduzmi_id INT,
                                   IN p_dodaj_id INT,
                                   IN p_iznos DOUBLE,
                                   IN p_opis VARCHAR(255),
                                   OUT p_poruka VARCHAR(255))
BEGIN
  DECLARE v_trenutno_stanje DOUBLE(10, 2) DEFAULT 0;

  BEGIN
   DECLARE exit HANDLER FOR NOT FOUND
   SET p_poruka = 'Korisnik nije pronadjen pod zadanim brojem!';
             DECLARE exit HANDLER FOR SQLEXCEPTION
        BEGIN
          ROLLBACK;
          SET p_poruka = 'Greska';
        END;

              -- oznaci pocetak transakcije
              START TRANSACTION;

              -- proveri da li korisnik, kojem se skida sa racuna, postoji
i da li
              -- ima dovoljno sredstva da mu se zadani iznos moze skinuti:
      SELECT iznos
                INTO v_trenutno_stanje
                FROM korisnici
               WHERE id = p_oduzmi_id;

              -- ako ima, onda obavi transakciju:
              IF v_trenutno_stanje > p_iznos THEN
                 UPDATE korisnici
                    SET iznos = iznos - p_iznos
                  WHERE id = p_oduzmi_id;

                 UPDATE korisnici
                    SET iznos = iznos + p_iznos
                  WHERE id = p_dodaj_id;

                  -- sacuvaj informacije o obavljenoj transakciji:
                  INSERT INTO transakcije (id, ovome_oduzmi, ovome_dodaj,
iznos, opis_transakcije)
                  VALUES (NULL, p_oduzmi_id, p_dodaj_id, p_iznos, p_opis);
```

```
      ELSE
         SET p_poruka = 'Korisnik nema dovoljno sredstva';
      END IF;
  END;

  -- potvrdi unos i oznaci kraj transakcije
  -- ili ponisti unos
  IF p_poruka != '' THEN
     ROLLBACK;
  ELSE
     COMMIT;
     SET p_poruka = 'SUCCESS! Transakcija je uspesno obavljena';
  END IF;

  SELECT p_poruka;
END //
DELIMITER ;
```

Najpre ćemo prikazati rezultat nekoliko testova, a onda sledi objašnjenje kompletne procedure.

**Test 1**

Sa računa korisnika 1 potrebno je prebaciti 500 EUR na račun korisnika 2. Pogledajmo prvo trenutno stanje računa oba korisnika:

```
SELECT * FROM korisnici WHERE id IN(1, 2);
```

Pozovimo proceduru *obavi_transakciju*:

```
CALL obavi_transakciju(1, 2, 500.00, 'Prebaci 500 EUR sa racuna 1 korisnika
na racun korisnika 2', @poruka);
```

Proverimo opet trenutno stanje oba korisnika nakon obavljene transakcije:

```
SELECT * FROM korisnici WHERE id IN(1, 2);
```

Iznos kod prvog korisnika se smanjio za 500, a kod drugog se povećao za 500 EUR. Proverimo i u tabeli *transakcije*, da li je transakcija uspešno obavljena:

```
SELECT * FROM transakcije ORDER BY id DESC LIMIT 1;
```

**Test 2**

Sa računa korisnika 1 prebacimo na račun korisnika 2 veći iznos, nego što korisnik 1 poseduje. Transakcija ne sme biti obavljena. Trenutno stanje je isto kao nakon obavljene transakcije u prvom testu. Pozovimo proceduru *obavi_transakciju*:

```
CALL obavi_transakciju(1, 2, 99999.99, 'Sa racuna korisnika 1 prebaci na
racun korisnika 2 veci iznos nego sto korisnik 1 poseduje', @poruka);
```

Proverimo trenutno stanje:

```
SELECT * FROM korisnici WHERE id IN(1, 2);
```

```
Korisnik nema dovoljno sredstva
```

Kao što vidimo, stanje je nepromenjeno. Proveravanjem tabele *transakcije*, uverićemo se da transakcija nije obavljena.

**Test 3**

Pokušajmo prebaciti 500 EUR sa računa nepostojećeg korisnika na račun korisnika 2. Transakcija neće biti obavljena i pojaviće se greška o nepostojećem korisniku. Pozovimo proceduru *obavi_transakciju*:

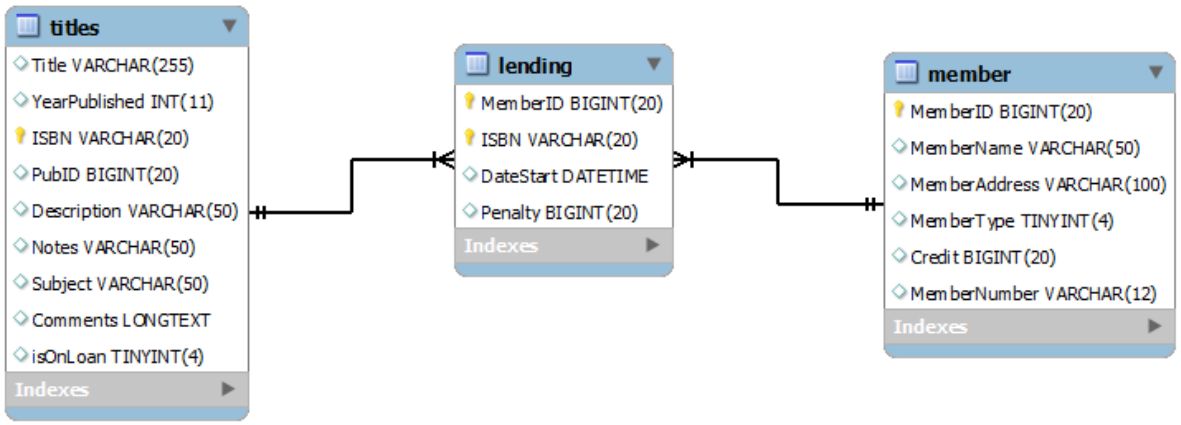
```
CALL obavi_transakciju(31, 2, 500.00, 'Prebaci 500 EUR sa racuna korisnika
31 na racun korisnika 2', @poruka);
```

```
Korisnik nije pronadjen pod zadanim brojem!
```

Proveravanjem trenutnog stanja i liste obavljenih transakcija, uverićemo se da transakcija nije obavljena, jer zadani korisnik ne postoji.

## Drugi primer

Za bazu podataka pod nazivom Biblio sa strukturom kao što je prikazano na slici, napisati sledeće procedure:




1. Napraviti uskladištenu proceduru za iznajmljivanje knjige registrovanim korisnicima, sa proverom dostupnosti knjige, proverom postojanja korisnika pod zadanim brojem u obliku transakcije.

getbook.sql

```
DELIMITER //
CREATE PROCEDURE getbook(in memberid BIGINT(20), in isbn VARCHAR(20))
BEGIN
	DECLARE isOnLoan TINYINT(4);

	DECLARE EXIT HANDLER FOR SQLEXCEPTION
	BEGIN
		ROLLBACK;
		SELECT "Unknown exception occured." AS ERROR;
	END;

	DECLARE EXIT HANDLER FOR NOT FOUND
		SELECT "No record was found." AS ERROR;

	START TRANSACTION;

	-- if it exists
	SELECT ISBN INTO isbn FROM titles t WHERE t.ISBN = isbn;
	SELECT MemberID INTO memberid FROM member m WHERE m.MemberID =
```

```
memberid;

      SELECT t.isOnLoan INTO isOnLoan FROM titles t WHERE t.ISBN = isbn;

      IF isOnLoan = 0 THEN
            INSERT INTO lending VALUES (memberid, isbn, NOW(), 0);
            UPDATE titles t SET t.isOnLoan = 1 WHERE t.ISBN = isbn;

      ELSE
            SELECT "The book has already been loaned.";
      END IF;

      -- ellenorzes
      SELECT t.isOnLoan INTO isOnLoan FROM titles t WHERE t.ISBN = isbn;

      IF isOnLoan != 1 THEN
            ROLLBACK;
      END IF;

      COMMIT;
END; //
```

```
call getbook (’0-0038307-6-4’,5)
```

2. Napraviti uskladištenu proceduru za vraćanje knjiga sa proverom validnosti zapisa o iznajmljivanju.

giveback.sql

```
DELIMITER //

CREATE PROCEDURE giveback(in memberid BIGINT(20), in isbn VARCHAR(20))
BEGIN

      DECLARE EXIT HANDLER FOR SQLEXCEPTION
      BEGIN
            ROLLBACK;
            SELECT "Unknown exception occured." AS ERROR;
      END;

      DECLARE EXIT HANDLER FOR NOT FOUND
            SELECT "Record was not found." AS ERROR;

      START TRANSACTION;

      -- test if present
      SELECT ISBN INTO isbn FROM lending l WHERE l.ISBN = isbn AND
l.MemberID = memberid;

      DELETE FROM lending WHERE lending.ISBN = isbn AND lending.MemberID =
memberid;
      UPDATE titles t SET isOnLoan = 0 WHERE t.ISBN = isbn;

      COMMIT;

END; //

DELIMITER ;
```

```
call giveback (’0-0038307-6-4’,5)
```

3. Napraviti uskladištenu proceduru za dodavanje novih korisnika, sa proverom validnosti podataka.

addmember.sql

```
DELIMITER //

CREATE PROCEDURE addmember(in id BIGINT, in name VARCHAR(50), in address
VARCHAR(100), in type TINYINT(4), in credit BIGINT(20), in number
VARCHAR(20))
BEGIN
      DECLARE message VARCHAR(75);
        SET message = "Successful addition.";

      IF id = '' OR name = '' OR address = '' OR credit = '' AND credit !=
0 OR number = '' OR type = '' AND type != 0
      THEN
            SET message = "Fill in all the needed data.";
      ELSE
            INSERT INTO member
            VALUES
            (id, name, address, type, credit, number);
      END IF;

    SELECT message;
END; //

DELIMITER ;
```

```
call addmember (5,'T. Denise','London',0,300,'456-123')
```

### Treći primer

Ovaj primer prikazuje realizaciju transakcije pomoću php programskog jezika. Primer je podešen za bazu pod imenom *books*. Vrši se unos novog sloga u bazu pomoću transakcije. Ukoliko taj unos već postoji u bazi, transakcija se poništava.

trans.php --> (books.sql)

```
<?php
function begin()
{
mysql_query("BEGIN");
}
function commit()
{
mysql_query("COMMIT");
}
function rollback()
{
mysql_query("ROLLBACK");
}
mysql_connect("localhost","admin", "admin") or die(mysql_error());
mysql_select_db("books") or die(mysql_error());

$query = "INSERT INTO authors VALUES('A08','Michael','Polk','512-953-
1231','4028 Guadalupe St','Austin','TX','78701')";

begin(); // transaction begins
```

```
$result = mysql_query($query);
if(!$result)
{
rollback(); // transaction rolls back
echo "transaction rolled back";
exit;
}
else
{
commit(); // transaction is committed
echo "Database transaction was successful";
}
?>
```

# 7. POGLAVLJE

## Indeksiranje i pretraga podataka

Redovi su uskladišteni u tabeli neuređeni, kao što to zahteva relacioni model. Ovaj nedostatak redosleda omogućuje DBMS-u lako i brzo umetanje, ažuriranje i brisanje redova, ali nažalost, postoji i sporedni efekat koji se ogleda u neefikasnom pretraživanju i sortiranju. Pretpostavimo da pokrenemo ovaj upit:

```
SELECT *
FROM authors
WHERE au_lname = 'Hull';
```

Da bi se izvršio ovaj upit, DBMS mora da uzastopno pretraži celokupnu tabelu authors, poredeći vrednost kolone au_lname u svakom redu sa stringom Huli. Pretraživanje celokupne tabele u maloj bazi podataka je trivijalno, međutim tabele u proizvodnoj bazi podataka mogu imati milione redova. DBMS-ovi obezbeđuju mehanizam koji se zove indeks i koji ima istu svrhu kao indeks u knjizi ili biblioteci: ubrzavanje procesa dobijanja podataka. Na pojednostavljenom nivou, indeks je uređen spisak u kojem se svaka posebna vrednost u indeksiranoj koloni (ili skupu kolona) čuva sa adresom na disku (fizičkom lokacijom) redova koji sadrže tu vrednost. Umesto da čita celokupnu tabelu da bi pronašao određene redove, DBMS skenira samo indeks da bi adresama pristupio direktno. Indeksirane pretrage su tipično nekoliko redova veličine brže od sekvencijalnih pretraga, međutim tu postoje i neki kompromisi, o čemu će biti reči u ovom poglavlju.

## Pravljenje indeksa pomoću CREATE INDEX

Indeksi su složeni: njihov oblik i učinak na performanse zavise od ponašanja optimizatora našeg DBMS-a. U ovom odeljku daću smernice, a mi potražimo u dokumentaciji našeg DBMS-a pojam index da bismo saznali kako naš DBMS implementira i koristi indekse. U opštem slučaju, indeksi su podesni za kolone koje se često:

- pretražuju (WHERE);
- sortiraju (ORDER BY);
- grupišu (GROUP BY);
- koriste u spojevima (JOIN);
- koriste za izračunavanje redne statistike (MIN(). MAX() ili medijana, na primer).

U opštem slučaju, indeksi su nepodesni za kolone:

- koje prihvataju samo nekoliko posebnih vrednosti (gender, marital_status ili state, na primer);
- koje se retko koriste u upitima;
- koje su deo male tabele sa nekoliko redova.

Kada pravimo indeks, neka važna razmatranja su:

- SQL-ovi indeksirajući iskazi menjaju objekte baze podataka, tako da će možda biti potrebno da nam administrator naše baze podataka odobri dozvolu za njihovo izvršavanje.
- Indeks nikad ne menja podatke; on samo predstavlja brzu pristupnu putanju do podataka.
- Tabela može imati nula ili više indeksa.

- U idealnom slučaju, sve indekse tabele pravimo kada pravimo tabelu. U praksi, upravljanje indeksima je iterativan proces. Tipično se samo najhitniji indeksi prave zajedno sa tabelom. Ostali indeksi dodaju se ili brišu tokom vremena, kako problemi u performansama rastu ili opadaju, a pristupne putanje korisnika se menjaju. Većina DBMS-ova pruža alatke za testiranje i merenje da bi se odredila delotvornost indeksa.
- Nemojmo da pravimo više indeksa nego što nam je nesumnjivo potrebno. DBMS mora da ažurira (i moguće reorganizuje) indeks nakon što umetnete, ažuriramo ili obrišemo redove u tabeli. Sa rastom broja indeksa u tabeli, performanse pri izmeni redova opadaju, jer DBMS troši sve više i više vremena na održavanje indeksa. Uglavnom, ne treba da pravimo više od desetak indeksa po tabeli.
- Naš DBMS će automatski održavati i koristiti indekse nakon što se naprave. Nema potrebe da korisnici ili SQL programeri preduzimaju bilo kakve dodatne radnje da bi se promene u podacima odrazile na sve relevantne indekse.
- Indeksi su transparentni za korisnika i SQL programera. Odsustvo ili prisustvo indeksa ne zahteva promenu formulacije bilo kog SQL iskaza.
- Indeks može da referencira jednu ili više kolona u tabeli. Indeks koji referencira jednu kolonu je prost indeks; indeks koji referencira više kolona je složen indeks. Kolone u složenom indeksu ne moraju da budu susedne u tabeli. Jedan indeks ne može da obuhvata više tabela.

Da bismo napravili indeks upišimo sledeće:

```
CREATE [UNIQUE] INDEX indeks
ON tabela (kolone_indeksa);
```

*indeks* je ime indeksa koji se pravi i predstavlja valjan SQL identifikator. Imena indeksa moraju biti jedinstvena unutar tabele.

Sledeći primer pravi prost indeks pod imenom pub_id_idx nad kolonom pub_id za tabelu titles. pub_id je spoljni ključ i predstavlja dobrog kandidata za indeks jer:

- promene nad ograničenjima PRIMARY KEY proveravaju se preko ograničenja FOREIGN KEY u povezanim tabelama;
- kolone spoljnog ključa često se koriste u kriterijumima spojeva kada se podaci iz povezanih tabela kombinuju u upitima slaganjem kolone (kolona) FOREIGN KEY iz jedne tabele sa kolonom (kolonama) PRIMARY KEY ili UNIQUE u drugoj tabeli.

Napraviti prost indeks nad kolonom pub_id za tabelu titles.

```
CREATE INDEX pub_id_idx
ON titles (pub_id);
```

Napraviti prost jedinstveni indeks nad kolonom title_name za tabelu titles.

```
CREATE UNIQUE INDEX title_name_idx
ON titles (title_name);
```

Gornji primer pravi prost jedinstveni indeks nazvan title_name_idx nad kolonom title_name za tabelu titles. DBMS će napraviti ovaj indeks samo ako u koloni title_name ne postoje ponovljene

vrednosti. Ovaj indeks takođe zabranjuje umetanje naslova koji nije jedinstven ili ažuriranje takvim naslovom.

Napraviti složeni indeks nad kolonama state i city za tabelu authors.

```
CREATE INDEX state_city_idx
ON authors (state, city);
```

Gornji primer pravi složen indeks nazvan state_city_idx nad kolonama state i city za tabelu authors. DBMS koristi indeks kada sortiramo redove prema redosledu koji određuje kombinacija vrednosti state i city. Ovaj indeks je beskoristan za sortiranje i pretrage samo prema vrednosti state, ili samo city, ili city i state; za ove svrhe moramo da napravimo posebne indekse.

Grupisući indeks (engl. clustering index) je indeks u kojem logički redosled ključnih vrednosti određuje fizički redosled odgovarajućih redova u tabeli. Kod negrupišućeg indeksa se logički redosled indeksa razlikuje od fizičkog redosleda redova uskladištenih na disku. Tabela može imati najviše jedan grupisući indeks. Grupisući indeksi obično poboljšavaju performanse, ali u nekim slučajevima oni pretrage čine mnogo bržim, a umetanje, ažuriranje i brisanje mnogo sporijim. Većina indeksa implementiraju se kao balansirana stabla, ili B-stabla. B-stablo je napredna struktura podataka koja minimizuje operacije čitanja i pisanja na disku. Neki DBMS-ovi omogućuju nam da navedemo strukturu podataka koja se koristi prilikom građenja indeksa.

## Odbacivanje indeksa pomoću DROP INDEX

Za uništavanje indeksa koristimo iskaz DROP INDEX. Budući daje indeks logički i fizički nezavisan od podataka u pridruženoj tabeli, možemo da u bilo kom trenutku odbacimo indeks, a da to nema uticaja na tabelu (ili druge indekse). Svi SQL programi i druge aplikacije nastaviće da rade kada odbacimo indeks, ali će pristup prethodno indeksiranim podacima sada biti sporiji.
Uobičajeni razlozi za odbacivanje indeksa su:

- Indeks više nije potreban, jer je pridružena tabela mnogo manja (ili je odbačena), ili korisnici više ne pristupaju kolonama indeksa tako često.
- Dodatno vreme koje je potrebno DBMS-u za održavanje indeksa posle operacija umetanja, ažuriranja ili brisanja značajnije je u odnosu na poboljšanje brzine koje indeks pruža pri operacijama povlačenja podataka.

SQL standard izostavlja indekse, tako da se SQL iskazi vezani za indekse menjaju u zavisnosti od DBMS-a. U ovom odeljku se govori o tome kako odbaciti indeks, za svaki DBMS obuhvaćen ovom knjigom. Ako koristimo drugačiji DBMS, potražimo u njegovoj dokumentaciji pojam index i saznajmo kako se indeks odbacuje.

Da bismo odbacili indeks u MySQL-u, upišimo sledeće:

```
DROP INDEX tabela.indeks;
```

*indeks* je ime indeksa koji se odbacuje, a *tabela* je ime tabele kojoj je indeks pridružen (listing 11.4).

Odbaciti indeks pub_id_idx (MySQL).

```
DROP INDEX pub_id_idx
ON titles;
```

# 8. POGLAVLJE

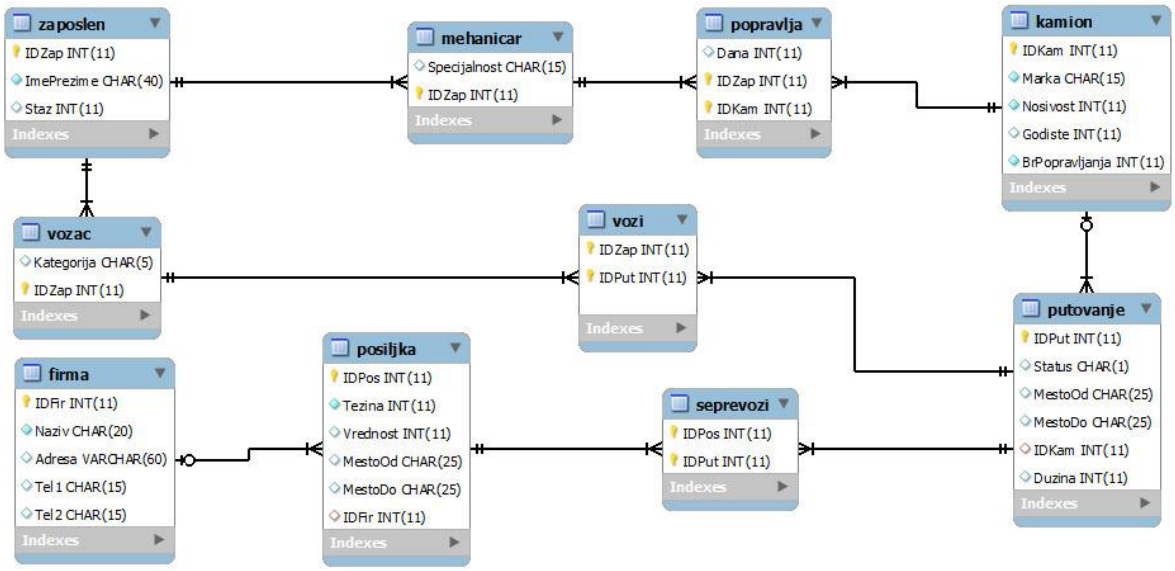
## Složeni podupiti

Podupit ili pododabir, je iskaz SELECT umetnut u drugi SQL iskaz. Podupit možemo da ugnezdimo u:

- rečenicu SELECT, FROM, WHERE ili HAVING iskaza SELECT;
- drugi podupit;
- iskaz INSERT, UPDATE ili DELETE.

U opštem slučaju, podupit možemo da koristimo na bilo kom mestu gde je izraz dozvoljen, ali bi naš DBMS mogao da ograniči mesta gde se mogu pojaviti. U ovom poglavlju obrađuju se podupiti ugnežđeni u iskaz SELECT ili drugi podupit. Budući da podupiti mogu biti teški za korišćenje i pronalaženje grešaka, možda će nam se više sviđati da koristimo spojeve, ali neka pitanja možemo da postavimo samo kao podupite. U slučajevima kada podupite i spojeve možemo da menjamo jedne drugima, treba da testiramo podupite na svom DBMS-u da bismo videli postoji li razlika u performansi između iskaza koji koristi podupit i semantički ekvivalentne verzije koja koristi spoj.

**Prvi primer:**




Kompanija za prevoz

Ispisati težinu svih pošiljki firme "Tropik".

```
SELECT SUM(Tezina) FROM Posiljka P, Firma F
WHERE F.IDFir = P.IDFir AND F.Naziv='Tropik'
```

| SUM(Tezina) |
|---|
| 6700 |

Ispisati koja firma je imala pošiljku najveće vrednosti.

```
SELECT F.Naziv FROM Posiljka P, Firma F
WHERE F.IDFir = P.IDFir
AND
P.Vrednost=(SELECT MAX(Vrednost) FROM Posiljka)
```

| Naziv |
|---|
| Petrolept |

Dati pregled naziva firmi i prosečne vrednosti pošiljki koje su imali.

```
SELECT F.Naziv, AVG(P.Vrednost) FROM Posiljka P, Firma F
WHERE F.IDFir = P.IDFir
GROUP BY F.Naziv
```

| Naziv | AVG(P.Vrednost) |
|---|---|
| Meldovo | 10000.0000 |
| Petrolept | 14800.0000 |
| Radoje Dakic | 10000.0000 |
| Tropik | 3500.0000 |

Dati ime i prezime mehaničara koji trenutno ne popravljaju kamione.

```
SELECT ImePrezime FROM Zaposlen Z, Mehanicar M
WHERE Z.IDZap=M.IDZap
AND
NOT EXISTS(SELECT * FROM Popravlja P WHERE P.IDZap=M.IDZap)
```

| ImePrezime |
|---|
| Marko Pesic |

Dati pregled imena i prezimena mehaničara koji popravljaju kamione, kao i marku i nosivost kamiona kog popravljaju kao i koliko dana će učestvovati u popravci.

```
SELECT Z.ImePrezime, K.Marka, K.Nosivost, P.Dana
FROM Zaposlen Z, Mehanicar M, Popravlja P, Kamion K
WHERE Z.IDZap=M.IDZap AND P.IDZap=M.IDZap AND P.IDKam=K.IDKam
```

| ImePrezime | Marka | Nosivost | Dana |
|---|---|---|---|
| Dusan Rajic | Mercedes | 4 | 2 |
| Obrad Zimonjic | MAN | 8 | 3 |
| Drasko Petkovic | MAN | 8 | 3 |

Ispisati koje firme imaju pošiljke koje još nisu poslate a ni isplanirane, kao i ukupnu težinu tih pošiljki.

```
SELECT F.Naziv,SUM(P.Tezina) FROM Firma F, Posiljka P
WHERE F.IDFir=P.IDFir
AND
NOT EXISTS(SELECT * FROM sePrevozi SP WHERE SP.IDPos=P.IDPos)
GROUP BY F.Naziv
```

| Naziv | SUM(P.Tezina) |
|---|---|
| Radoje Dakic | 5000 |
| Tropik | 1900 |

Ispisati koji kamioni (marka, nosivost) su korišćeni za prevoz pošiljki u ukupnoj težini do 7000 kg.

```
SELECT DISTINCT Marka, Nosivost FROM Kamion Kam, Putovanje PUT
WHERE Kam.IDKam=PUT.IDKam
AND
(
SELECT SUM(P.Tezina) FROM Posiljka P, sePrevozi SP
WHERE P.IDPos=SP.IDPos AND PUT.IDPut=SP.IDPut
) <=7000
```

| Marka | Nosivost |
|---|---|
| Mercedes | 6 |
| FAP | 5 |
| FAP | 3 |

Ispisati ime i prezime svih vozača koji su nekada vozili pošiljku od firme "Petrolept" kao i marku kamiona koji je korišćen tom prilikom i koliko puta je ta marka korišćena u tom slučaju.

```
SELECT Z.ImePrezime,K.Marka, COUNT(*)
FROM Zaposlen Z, Vozac V, Vozi VO, Putovanje PU, Kamion K,
Posiljka P, Firma F, sePrevozi SP
WHERE
Z.IDZap=V.IDZap AND
V.IDZap=VO.IDZap AND
VO.IDPut=PU.IDPut AND
K.IDKam=PU.IDKam AND
PU.IDPut=SP.IDPut AND
SP.IDPos=P.IDPos AND
P.IDFir=F.IDFir AND
F.Naziv='Petrolept'
GROUP BY Z.ImePrezime, K.Marka
```

| ImePrezime | Marka | COUNT(*) |
|---|---|---|
| Dragan Milutinovic | FAP | 1 |
| Mihajlo Janjic | Mercedes | 1 |
| Milos Miljkovic | FAP | 1 |
| Milos Miljkovic | Mercedes | 2 |

Ispisati koliko je tereta prevezao vozač sa najvećim radnim stažom među vozačima.

```
SELECT SUM(P.Tezina)
FROM Zaposlen Z, Vozac V, Vozi VO, Putovanje PU, Posiljka P,
sePrevozi SP
WHERE
Z.IDZap=V.IDZap AND
V.IDZap=VO.IDZap AND
VO.IDPut=PU.IDPut AND
PU.IDPut=SP.IDPut AND
SP.IDPos=P.IDPos AND
Z.Staz
=
(SELECT MAX(Z2.Staz)
FROM Zaposlen Z2, Vozac V2
WHERE Z2.IDZap=V2.IDZap)
```

| SUM(P.Tezina) |
|---|
| 2900 |

Ispisati koliko su ukupno tereta u težini i vrednosti prevezli kamioni koji se trenutno popravljaju, da su već imali manje od prosečnog broja popravki svih postojećih kamiona, a da ih je u nekom trenutku vozio Dragan Milutinović.

```
SELECT SUM(POS.Vrednost), SUM(POS.Tezina)
FROM Kamion K, Popravlja POP, Posiljka POS, Putovanje PUT,
sePrevozi SP, Zaposlen Z, Vozac V, Vozi VO
WHERE
K.IDKam=POP.IDKam AND
PUT.IDKam=K.IDKam AND
PUT.IDPut=VO.IDPut AND
```

```
VO.IDZap=V.IDZap AND
V.IDZap=Z.IDZap AND
Z.ImePrezime='Dragan Milutinovic' AND
PUT.IDPut=SP.IDPut AND
SP.IDPos=POS.IDPos AND
K.BrPopravljanja<(SELECT AVG(BrPopravljanja) FROM Kamion)
```

| SUM(POS.Vrednost) | SUM(POS.Tezina) |
|---|---|
| 16000 | 15000 |

Napraviti pogled koji će prikazivati sve informacije o vozačima (ime i prezime, staž, kategorija)

```
CREATE VIEW Vozaci
AS
SELECT ImePrezime, Staz, Kategorija
FROM Zaposlen Z, Vozac V
WHERE Z.IDZap=V.IDZap
```

Napraviti pogled koji će prikazivati sve informacije o popravci kamiona

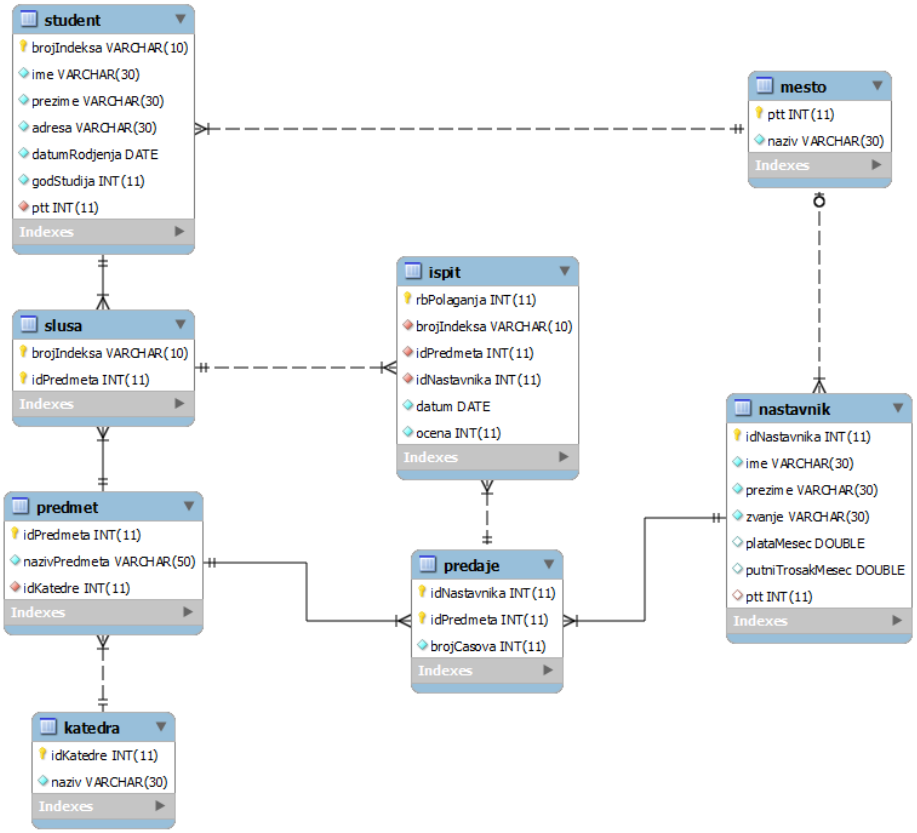
```
CREATE VIEW KamioniPopravka
AS
SELECT
IDKam, Marka, Nosivost, BrPopravljanja,
(SELECT COUNT(*) FROM Popravlja P WHERE P.IDKam=K.IDKam)
LjudiPopravlja,
(SELECT MAX(Dana) FROM Popravlja P WHERE P.IDKam=K.IDKam)
MaxDana
FROM Kamion K
```

Napraviti pogled koji će prikazivatipređene kilometre pojedinih vozača i to kada se status promeni na "O" (obavljeno)

```
CREATE VIEW PredjeniKilometri
AS
SELECT ImePrezime, SUM(Duzina) FROM
Zaposlen Z,Vozac V, Vozi VO, Putovanje P
WHERE
Z.IDZap=V.IDZap AND
V.IDZap=VO.IDZap AND
VO.IDPut=P.IDPut AND
P.Status='O'
GROUP BY ImePrezime
```

# Napredne tehnike pretraživanja

**Drugi primer:**




Fakultet

Prikazati prezimena, imena i brojeve indeksa studenata treće godine studija iz Subotice.

```
select godstudija, prezime, ime, brojindeksa, naziv, mesto.ptt
from student, mesto
where mesto.ptt=student.ptt
and godstudija=3 and naziv='Subotica'
```

Izdvojiti podatke za nastavnika koji ima najveću platu.

```
SELECT prezime, ime, zvanje, platamesec as plata
FROM nastavnik
WHERE platamesec = (SELECT MAX(platamesec)
                    FROM nastavnik)
```

Prikazati koje predmete sa određenim fondom časova predaje svaki nastavnik.

```
SELECT nastavnik.idnastavnika, prezime, ime, zvanje,
          brojcasova as fond, nazivpredmeta
FROM nastavnik, predaje, predmet
WHERE nastavnik.idnastavnika=predaje.idnastavnika
and predmet.idPredmeta = predaje.idPredmeta
```

Izračunati koliko predmeta predaje svaki nastavnik.

```
SELECT nastavnik.idnastavnika, prezime, ime,
      COUNT(predaje.idnastavnika) as predmeta
FROM nastavnik, predaje
WHERE nastavnik.idnastavnika=predaje.idnastavnika
GROUP BY nastavnik.idnastavnika, prezime, ime
```

Prikazati koje predmete na katedri za informatiku predaje nastavnik Petar Petrović.

```
SELECT nastavnik.idnastavnika, prezime, ime, nazivpredmeta
FROM nastavnik, predaje, predmet, katedra
WHERE nastavnik.idnastavnika=predaje.idnastavnika
and predmet.idPredmeta = predaje.idPredmeta
and prezime='Petrovic' and ime='Petar'
and katedra.naziv='Katedra za informatiku'
and predmet.idkatedre=katedra.idkatedre
```

Kreirati pogled koji izdvaja predmete na katedri za informatiku koje predaje nastavnik Petar Petrović.

```
CREATE VIEW Predmeti_Radulovic
AS
SELECT nastavnik.idnastavnika, prezime, ime, nazivpredmeta
FROM nastavnik, predaje, predmet, katedra
WHERE nastavnik.idnastavnika=predaje.idnastavnika
and predmet.idPredmeta = predaje.idPredmeta
and prezime='Petrovic' and ime='Petar'
and katedra.naziv='Katedra za informatiku'
and predmet.idkatedre=katedra.idkatedre
```

Izdvojiti podatke o studentima koji su položili ispit iz Baza podataka 1.

```
SELECT S.BROJINDEKSA, S.PREZIME, S.IME, P.NAZIVPREDMETA, I.OCENA, I.DATUM
FROM STUDENT S
INNER JOIN ISPIT I
ON I.BROJINDEKSA = S.BROJINDEKSA
INNER JOIN PREDMET P
ON P.idPredmeta = I.idPredmeta
WHERE P.NAZIVPREDMETA = 'Baze podataka 1' AND OCENA>5;
```

Kreirati pogled koji izdvaja podatke o studentima koji su polagali ispit iz Baza podataka 1 i urediti ga hronološki.

```
CREATE VIEW ISPITI
AS
SELECT STUDENT.BROJINDEKSA, STUDENT.PREZIME, IME,
            NAZIVPREDMETA, OCENA, DATUM
FROM STUDENT
INNER JOIN ISPIT
     ON ISPIT.BROJINDEKSA=STUDENT.BROJINDEKSA
INNER JOIN PREDMET
     ON PREDMET.IDPREDMETA=ISPIT.IDPREDMETA
WHERE NAZIVPREDMETA='Baze podataka 1'
```

Poziv pogleda:

```
SELECT * FROM ISPITI
ORDER BY datum asc
```

Izdvojiti iz baze podatke o predmetima koje je odslušao a nije položio student Pajovic Predrag.

```
SELECT STUDENT.BROJINDEKSA, STUDENT.PREZIME, IME,
            NAZIVPREDMETA
FROM STUDENT, SLUSA, ISPIT, PREDMET
WHERE PREDMET.IDPREDMETA=SLUSA.IDPREDMETA
     AND SLUSA.BROJINDEKSA=STUDENT.BROJINDEKSA
     AND (SLUSA.BROJINDEKSA=ISPIT.BROJINDEKSA
     AND SLUSA.IDPREDMETA=ISPIT.IDPREDMETA)
AND ime='Predrag' and prezime ='Pajovic'
and ocena=5
UNION
SELECT STUDENT.BROJINDEKSA, STUDENT.PREZIME, IME,
```

```
            NAZIVPREDMETA
FROM STUDENT, SLUSA, PREDMET
WHERE PREDMET.IDPREDMETA=SLUSA.IDPREDMETA
     AND SLUSA.BROJINDEKSA=STUDENT.BROJINDEKSA
AND ime='Predrag' and prezime ='Pajovic'
and SLUSA.IDPREDMETA NOT IN (SELECT IDPREDMETA FROM ISPIT)
```

Prikazati koliko studenata je položilo neki predmet. Prikazati podatke i za one predmete koje nijedan student nije (još) položio! Sortirati izveštaj po broju studenata koji su položili predmete.

```
SELECT NAZIVPREDMETA, COUNT(ISPIT.idPredmeta) AS BRSTUDENATA
FROM PREDMET LEFT JOIN ISPIT
ON ISPIT.idPredmeta = PREDMET.idPredmeta
GROUP BY NAZIVPREDMETA
```

Izdvojiti broj indeksa, prezime i ime studenta, broj položenih ispita i prosečnu ocenu za one studente čiji je prosek veći od 9. Sortirati podatke po prosecnoj oceni od najvece do najmanje.

```
SELECT student.brojIndeksa, student.ime, student.prezime, AVG(ispit.ocena)
AS prosek,
COUNT(ispit.idPredmeta) AS 'broj polozenih ispita'
FROM  ispit
INNER JOIN student
ON ispit.brojIndeksa = student.brojIndeksa
WHERE  (ispit.ocena > 5)
GROUP BY student.brojIndeksa, student.ime, student.prezime
HAVING  (AVG(ispit.ocena) > 9)
ORDER BY prosek DESC
```

Prikazati koliko studenata je slušalo neki predmet. Spisak sortirati od najviše do najmanje slušanih predmeta. Izostaviti podatke o predmetima koje je slušalo manje od 5 studenata.

```
SELECT slusa.idPredmeta, predmet.nazivPredmeta,
            COUNT(slusa.brojIndeksa) AS studenata
FROM   predmet INNER JOIN slusa
ON predmet.idPredmeta = slusa.idPredmeta
INNER JOIN student
ON slusa.brojIndeksa = student.brojIndeksa
GROUP BY slusa.idPredmeta, predmet.nazivPredmeta
HAVING COUNT(slusa.brojIndeksa) >=5
ORDER BY COUNT(slusa.brojIndeksa) desc
```

Prikazati koje ispite je položio student Pajovic Predrag.

```
SELECT student.brojIndeksa, student.ime, student.prezime,
predmet.nazivPredmeta, ispit.ocena
FROM  predmet INNER JOIN ispit
ON predmet.idPredmeta = ispit.idPredmeta
INNER JOIN student
ON ispit.brojIndeksa = student.brojIndeksa
WHERE  student.ime='Pajovic' and student.prezime='Predrag'
```

Izračunati prosečnu ocenu svih studenata koji su položili bar jedan ispit i prikazati koliko su predmeta položili.

```
SELECT student.brojIndeksa, student.ime, student.prezime, AVG(ispit.ocena)
AS prosek,
COUNT(ispit.idPredmeta) AS 'broj polozenih predmeta'
FROM  ispit INNER JOIN
      student ON ispit.brojIndeksa = student.brojIndeksa
WHERE  (ispit.ocena > 5)
GROUP BY student.brojIndeksa, student.ime, student.prezime
```

Da li postoje ispiti koje do sada nije položio nijedan student? Kreirati pogled. Napisati SQL komandu za brisanje pogleda iz baze podataka.

```
CREATE VIEW NEPOLOZENIISPITI
AS
SELECT nazivPredmeta
FROM PREDMET
WHERE IDPredmeta NOT IN (SELECT IDPredmeta
                                          FROM ISPIT
                                          WHERE OCENA>5)
```

```
DROP VIEW NEPOLOZENIISPITI
```

Prikazati brojeve indeksa, prezimena, imena studenata prve godine iz Zrenjanina ili Novog Sada koji nisu položili nijedan ispit.

```
SELECT student.brojIndeksa, student.ime, student.prezime, mesto.naziv
FROM mesto INNER JOIN student ON mesto.ptt = student.ptt
WHERE student.brojIndeksa not IN (SELECT brojIndeksa
                                          FROM ISPIT
                                          WHERE OCENA>5)
and (mesto.naziv='Zrenjanin' or mesto.naziv='Novi Sad')
and godstudija=1
```

Koji nastavnici imaju platu veću od prosečne plate nastavnika za grad/mesto iz kojeg putuju.

```
select ime, prezime, zvanje, naziv, platamesec
from nastavnik n1 inner join mesto
on n1.ptt=mesto.ptt
where platamesec > (select avg(platamesec)
                        from nastavnik n2
                        where n2.ptt=n1.ptt)
```

Prikazati brojeve indeksa, prezimena, imena studenata, nazive predmeta koje su ti studenti položili sa ocenama, ali samo za one studente koji su te predmete položili ocenom koja je veća od prosečne ocene za taj predmet.

```
SELECT i1.brojIndeksa, student.prezime, student.ime,
      predmet.nazivPredmeta, i1.ocena
FROM ispit i1
INNER JOIN     predmet
ON i1.idPredmeta = predmet.idPredmeta
INNER JOIN     student
ON i1.brojIndeksa = student.brojIndeksa
WHERE ocena>5 and ocena>(select AVG(ocena)
                              from ispit i2
                              where i2.idPredmeta=i1.idpredmeta)
```

Spajanje podataka iz tabela NASTAVNIK, PREDAJE I PREDMET preko upita sa podupitom

```
SELECT n1.idnastavnika, prezime, ime, zvanje
FROM nastavnik n1
WHERE n1.idnastavnika IN (select idNastavnika
                            from predaje
                            where predaje.idPredmeta in (select idpredmeta
                                                                  from
predmet))
```

Prikazati koje ispite su položili studenti čije prezime počinje slovom P (operator LIKE).

```
SELECT student.brojIndeksa, student.ime, student.prezime,
predmet.nazivPredmeta, ispit.ocena
FROM  predmet
INNER JOIN ispit
```

```
ON predmet.idPredmeta = ispit.idPredmeta
INNER JOIN student
ON ispit.brojIndeksa = student.brojIndeksa
WHERE  student.prezime LIKE 'P%'
```

Spisak studenata koji su položili neki ispit u junu 2010. godine.

```
SELECT STUDENT.BROJINDEKSA, STUDENT.PREZIME, IME, NAZIVPREDMETA, OCENA,
DATUM
FROM STUDENT
INNER JOIN ISPIT
 ON ISPIT.BROJINDEKSA=STUDENT.BROJINDEKSA
INNER JOIN PREDMET
 ON PREDMET.IDPREDMETA=ISPIT.IDPREDMETA
WHERE year(DATUM)=2010 and MONTH(DATUM)=6
```

# 9. POGLAVLJE

## Okidači (engl. TRIGGERS)

Trigeri su objekti (uskladištene procedure) pridruženi tabeli, koji se automatski aktiviraju kada se desi neki događaj vezan za tu tabelu. Uvedeni su u verziji 5.02 Alpha. Mogu se uspešno koristiti za validaciju podataka i druge operacije direktno nad tabelama baze podataka. Trigeri se aktiviraju kada se desi neka promena u pridruženoj tabeli, preciznije prilikom izvršenja Insert / Update / Delete naredbi.

Sintaksa kreiranja trigera:

```
CREATE TRIGGER naziv_trigera
vreme_izvršavanja_trigera
događaj
ON tbl_name
FOR EACH ROW trigger_statement
```

Vreme izvršavanja trigera:

- BEFORE
- AFTER

Događaj je ona akcija čije izvršavanje na pridruženoj tabeli uzrokuje aktiviranje trigera. Može biti:

- INSERT
- UPDATE
- DELETE

Kako trigeri funkcionišu? Ključna reč BEFORE opredeljuje vreme izvršenja trigera. U ovom slučaju, triger će biti aktiviran pre svakog novog upisa zapisa u tabelu test. Ključne reči FOR EACH ROW označavaju da će triger biti izvršen za svaki novi slog. Ključne reči OLD i NEW nam omogučavaju pristup vrednostima polja u zapisima na koje se odnosi triger. U INSERT trigeru može se koristiti samo NEW.ime_polja U DELETE trigeru može se koristiti samo OLD.ime_polja

Kompleksni trigeri:

- Trigeri koji sadrže više od jedne linije koda
- Tada se koriste ključne reči BEGIN i END koje imaju svrhu da označe kompajleru početak i kraj koda trigera

### Prikazivanje trigera

```
SHOW TRIGGERS;
```

### Upotreba trigera za kreiranje log fajlova

**Prvi primer:**

Test.sql

```
drop table if exists Important_data;
create table Important_data (
```

```
id int not null primary key auto_increment,
Data text not null
);

drop table if exists Event_log;
create table Event_log (
id int not null primary key auto_increment,
Data_ID int not null,
OLD_Data text not null,
NEW_Data text not null,
time timestamp not null,
Active_user varchar(20) not null
);

insert into Important_data(Data) values ('top_secret1');
insert into Important_data(Data) values ('top_secret2');
insert into Important_data(Data) values ('top_secret3');
```

Trigger1.sql (UPDATE)

```
drop trigger if exists trig_Dataupdate;
delimiter //
create trigger trig_Dataupdate before update on Important_data for each row
begin
        if OLD.Data != NEW.Data then
                insert into Event_log(Data_ID, OLD_Data, NEW_Data, time)
values(NEW.id, OLD.Data, NEW.Data, now());
        end if;
end //
delimiter ;
```

```
update Important_data
set Data='changed data'
where id = 2;

update Important_data
set Data='changed data2'
where id = 1;
```

Trigger2.sql (UPDATE with user data)

```
drop trigger if exists trig_Dataupdate;
delimiter //
create trigger trig_Dataupdate before update on Important_data for each row
begin
        if OLD.Data != NEW.Data then
                insert into Event_log(Data_ID, OLD_Data, NEW_Data, time,
Active_user) values(NEW.id, OLD.Data, NEW.Data, now(),@user_data);
        end if;
end //
delimiter ;
```

```
set @user_data = 'Student';
update Important_data
set Data = 'Important data'
where id = 1;
```

Trigger3.sql (DELETE)

```
drop trigger if exists trig_Datadelete;
delimiter //
create trigger trig_Datadelete after delete on Important_data for each row
begin
       set @DelData.id = OLD.id;
       set @DelData.Data = OLD.Data;
insert into Event_log(Data_ID, OLD_Data, time, Active_user)
values(@DelData.id, @DelData.Data, now(), @user_data);
end //
delimiter ;
```

```
set @user_data = 'Student';
DELETE FROM Important_data
where id = 3;

set @user_data = 'root';
DELETE FROM Important_data
where id = 2;
```

Trigger4.sql (INSERT)

```
drop trigger if exists trig_Datainsert;
delimiter //
create trigger trig_Datainsert after insert on Important_data for each row
begin
  insert into Event_log(Data_ID, NEW_Data, time, Active_user)
values(NEW.id, NEW.Data, now(), @user_data);
end //
delimiter ;
```

```
set @user_data = 'Admin';
insert into Important_data(Data) values ('top_secret');
```

**Drugi primer:**

trigger_db.sql

```
CREATE TABLE blog (
      id mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
      title text,
      content text,
      deleted tinyint(1) unsigned NOT NULL DEFAULT '0',
      PRIMARY KEY (id),
      KEY ix_deleted (deleted)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='Blog posts';

CREATE TABLE audit (
      id mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
      blog_id mediumint(8) unsigned NOT NULL,
      changetype enum('NEW','EDIT','DELETE') NOT NULL,
      changetime timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
      PRIMARY KEY (id),
      KEY ix_blog_id (blog_id),
      KEY ix_changetype (changetype),
      KEY ix_changetime (changetime),
      CONSTRAINT FK_audit_blog_id FOREIGN KEY (blog_id) REFERENCES blog
(id) ON DELETE CASCADE ON UPDATE CASCADE
```

```
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

```
DELIMITER $$
CREATE
      TRIGGER blog_after_insert AFTER INSERT
      ON blog
      FOR EACH ROW BEGIN

      IF NEW.deleted THEN
            SET @changetype = 'DELETE';
      ELSE
            SET @changetype = 'NEW';
      END IF;

      INSERT INTO audit (blog_id, changetype) VALUES (NEW.id, @changetype);
    END$$
DELIMITER ;
```

```
DELIMITER $$
CREATE
      TRIGGER blog_after_update AFTER UPDATE
      ON blog
      FOR EACH ROW BEGIN

      IF NEW.deleted THEN
            SET @changetype = 'DELETE';
      ELSE
            SET @changetype = 'EDIT';
      END IF;

      INSERT INTO audit (blog_id, changetype) VALUES (NEW.id, @changetype);
END$$
DELIMITER ;
```

Testiranje trigera

```
INSERT INTO blog (title, content) VALUES ('Article One', 'Initial text.');

UPDATE blog SET content = 'Edited text' WHERE id = 1;

UPDATE blog SET deleted = 1 WHERE id = 1;
```

### Brisanje trigera

```
DROP TRIGGER naziv_trigera;
```

## Operacije nad skupovima

Teorija skupova je fundamentalna za relacioni model. Međutim, dok su matematički skupovi nepromenljivi, skupovi baze podataka su dinamički - oni rastu, smanjuju se i na drugi način menjaju u vremenu. U ovom poglavlju obrađeni su sledeći SQL-ovi operatori nad skupovima, koji rezultate dva iskaza SELECT kombinuju u jedan rezultat:

- UNION vraća sve redove koje su vratila oba upita, sa uklonjenim duplikatima.
- INTERSECT vraća sve redove zajedničke za oba upita (to jest, sve različite redove koje su vratila oba upita).

- EXCEPT vraća sve redove iz prvog upita bez redova koji se pojavljuju u drugom upitu, sa uklonjenim duplikatima.

Ove operacije sa skupovima nisu spojevi, ali možemo da ih mešamo i ulančavamo da bismo kombinovali dve ili više tabela.

### Kombinovanje redova pomoću UNION

Operacija UNION kombinuje rezultate dva upita u jedan rezultat koji sadrži redove koje su vratila oba upita. (Ova operacija razlikuje se od spoja, koji kombinuje kolone iz dve tabele.) Izraz UNION uklanja ponovljene redove iz rezultata; izraz UNION ALL ne uklanja duplikate.

Unije su jednostavne, ali imaju neka ograničenja:

- Liste rečenice SELECT u dva upita moraju imati isti broj kolona (imena kolona, aritmetički izrazi, agregatne funkcije itd).
- Odgovarajuće kolone u dva upita moraju biti navedene istim redosledom u ta dva upita.
- Odgovarajuće kolone moraju biti istog tipa podataka ili mora postojati mogućnost implicitne konverzije u isti tip.
- Ukoliko se imena odgovarajućih kolona poklapaju, to ime kolone koristi se u rezultatu. Ukoliko se imena odgovarajućih kolona razlikuju, na DBMS-u je da odredi ime kolone u rezultatu. Većina DBMS-ova imena kolona rezultata uzima iz prvog pojedinačnog upita u iskazu UNION. Ako želimo da preimenujemo kolonu u rezultatu, upotrebimo rečenicu AS u prvom upitu.
- Rečenica ORDER BY može se pojaviti samo u konačnom upitu u iskazu UNION. Sortiranje se primenjuje na konačni, kombinovani rezultat. S obzirom na to da imena kolona u rezultatu zavise od DBMS-a, često je najlakše koristiti relativne pozicije kolona da bi se odredio redosled sortiranja.
- GROUP BY i HAVING mogu se navesti samo u pojedinačnim upitima; ne mogu se koristiti tako da pogađaju konačni rezultat.

Da bismo kombinovali redove, upišimo sledeće:

```
select_iskaz1
UNION [ALL]
select_iskaz2;
```

select_iskaz1 i select_iskaz2 su iskazi SELECT. Broj i redosled kolona mora biti identičan u oba iskaza, a tipovi podataka odgovarajućih kolona moraju biti kompatibilni. Ukoliko se ne navede ALL, ponovljeni redovi biće uklonjeni iz rezultata.

Dati spisak država u kojima se nalaze autori i izdavači.

```
SELECT state FROM authors
UNION
SELECT state FROM publishers;
```

| state |
|---|
| NY |
| CO |
| CA |
| FL |
| TX |
| *NULL* |

Dati spisak država u kojima se nalaze autori i izdavači, uključujući ponovljene države.

```
SELECT state FROM authors
UNION ALL
SELECT state FROM publishers;
```

| state |
|---|
| NY |
| CO |
| CA |
| CA |
| NY |
| CA |
| FL |
| TX |
| NY |
| CA |
| *NULL* |

Dati spisak imena svih autora i izdavača.

```
SELECT CONCAT(au_fname,' ',au_lname) AS "Name"
FROM authors
UNION
SELECT pub_name
FROM publishers
ORDER BY 1 ASC;
```

| Name |
|---|
| Kellsey |
| Abatis Publishers |
| Christian Kells |
| Core Dump Books |
| Hallie Hull |
| Klee Hull |
| Paddy O'Furniture |
| Sarah Buchman |
| Schadenfreude Press |
| Tenterhooks Press |
| Wendy Heydemark |

Dati spisak imena svih autora i izdavača koji se nalaze u državi New York, sortiranih po tipu, a zatim po imenu.

```
SELECT
'author' AS "Type",
CONCAT(au_fname,' ', au_lname) AS "Name",
state
FROM authors
WHERE state = 'NY'
UNION
SELECT
'publisher',
```

```
pub_name,
state
FROM publishers
WHERE state = 'NY'
ORDER BY 1 ASC, 2 ASC;
```

| Type | Name | state |
|---|---|---|
| author | Christian Kells | NY |
| author | Sarah Buchman | NY |
| publisher | Abatis Publishers | NY |

Dati spisak imena svih autora i izdavača koji se nalaze u državi New York i naslova knjiga objavljenih u državi New York, sortiranih po tipu, a zatim po imenu.

```
SELECT
'author' AS "Type",
CONCAT(au_fname,' ', au_lname)  AS "Name"
FROM authors
WHERE state = 'NY'
UNION
SELECT
'publisher',
pub_name
FROM publishers
WHERE state = 'NY'
UNION
SELECT
'title',
title_name
FROM titles t
INNER JOIN publishers p
ON t.pub_id = p.pub_id
WHERE p.state = 'NY'
ORDER BY 1 ASC, 2 ASC;
```

| Type | Name |
|---|---|
| author | Christian Kells |
| author | Sarah Buchman |
| publisher | Abatis Publishers |
| title | 1977! |
| title | How About Never? |
| title | Not Without My Faberge Egg |
| title | Spontaneous, Not Annoying |

Dati broj svih autora i izdavača koji se nalaze u državi New York i naslova knjiga objavljenih u državi New York, sortirano po tipu.

```
SELECT
'author' AS "Type",
COUNT(au_id) AS "Count"
FROM authors
WHERE state = 'NY'
UNION
SELECT
'publisher',
COUNT(pub_id)
FROM publishers
WHERE state = 'NY'
```

```
UNION
SELECT
'title',
COUNT(title_id)
FROM titles t
INNER JOIN publishers p
ON t.pub_id = p.pub_id
WHERE p.state = 'NY'
ORDER BY 1 ASC;
```

| Type | Count |
|---|---|
| author | 2 |
| publisher | 1 |
| title | 4 |

Povećati cenu knjiga iz oblasti istorije za 10 procenata, cenu knjiga iz psihologije za 20 procenata, a cene ostalih knjiga ostaviti neizmenjene.

```
SELECT title_id, type, price,
price * 1.10 AS "New price"
FROM titles
WHERE type = 'history'
UNION
SELECT title_id, type, price, price * 1.20
FROM titles
WHERE type = 'psychology'
UNION
SELECT title_id, type, price, price
FROM titles
WHERE type NOT IN ('psychology','history')
ORDER BY type ASC, title_id ASC;
```

| title_id | type | price | New price |
|---|---|---|---|
| T06 | biography | 19.95 | 19.9500 |
| T07 | biography | 23.95 | 23.9500 |
| T10 | biography | *NULL* | *NULL* |
| T12 | biography | 12.99 | 12.9900 |
| T08 | children | 10.00 | 10.0000 |
| T09 | children | 13.95 | 13.9500 |
| T03 | computer | 39.95 | 39.9500 |
| T01 | history | 21.99 | 24.1890 |
| T02 | history | 19.95 | 21.9450 |
| T13 | history | 29.99 | 32.9890 |
| T04 | psychology | 12.99 | 15.5880 |
| T05 | psychology | 6.95 | 8.3400 |
| T11 | psychology | 7.99 | 9.5880 |

### Pronalaženje zajedničkih redova pomoću INTERSECT

Operacija INTERSECT kombinuje rezultate dva upita u jedan rezultat koji sadrži sve redove zajedničke za oba upita. Preseci imaju ista ograničenja kao i unije.

Da bismo pronašli zajedničke redove upišimo sledeće:

```
select_iskaz1
INTERSECT
select_iskaz2;
```

select_iskaz1 i select_iskaz2 su iskazi SELECT. Broj i redosled kolona mora biti identičan u oba iskaza, a tipovi podataka odgovarajućih kolona moraju biti kompatibilni. Ponovljeni redovi biće uklonjeni iz rezultata. Sledeći primer koristi INTERSECT za dobijanje spiska gradova u kojima se nalaze i autor i izdavač.

Dati spisak gradova u kojima se nalaze i autor i izdavač.

```
SELECT city
FROM authors
INTERSECT
SELECT city
FROM publishers;
```

**Napomena**: MySQL ne podržava INTERSECT.

### Pronalaženje različitih redova pomoću EXCEPT

Operacija EXCEPT, poznata i kao razlika, kombinuje rezultate dva upita u jedan rezultat koji sadrži redove koji pripadaju samo prvom upitu. Suočimo li INTERSECT i EXCEPT, A INTERSECT B sadrži redove iz tabele A koji se ponavljaju u tabeli B, dok A EXCEPT B sadrži redove iz tabele A koji nisu ponovljeni u tabeli B. Razlike imaju ista ograničenja kao unije.
Da bismo pronašli različite redove upišimo sledeće:

```
select_iskaz1
EXCEPT
select_iskaz2;
```

select_iskaz1 i select_iskaz2 su iskazi SELECT. Broj i redosled kolona mora biti identičan u oba iskaza, a tipovi podataka odgovarajućih kolona moraju biti kompatibilni. Ponovljeni redovi biće uklonjeni iz rezultata.

Primena EXCEPT za dobijanje spiska gradova u kojima živi neki autor, ali nema izdavača.

```
SELECT city
FROM authors
EXCEPT
SELECT city
FROM publishers;
```

**Napomena**: MySQL ne podržava EXCEPT.

# 10. POGLAVLJE

## MySQL Workbench

MySQL Workbench dolazi u dve varijante: u slobodnoj (Comunity Edition), koja je namenjena open source zajednici i standardnoj (Standard Edition), koja spada u red komercijalnih aplikacija. Nakon instalacije i pokretanja programa, postaje više nego očigledno da MySQL Workbench, na radost velikog broja obožavatelja DBDesignera, u potpunosti preslikava izgled i način rada iz njihovog omiljenog RDBMS alata. Od rasporeda komandi i izgleda korisničkog interfejsa preko logičke organizacije raspoređivanja entiteta i relacija sve je tu. Što se tiče razlika u funkcionalnosti između comunity i standardne verzije programa sigurno je da će kod većine open source zaljubljenika, koji su navikli na obilje opcija koje je DBDesigner imao da ponudi, MySQL Workbench u slobodnoj varijanti izazvati malo razočaranje. Još jedna od stvari koja počinje užasno da iritira već posle samo nekoliko minuta korišćenja programa jeste katastrofalna tromost, koja, prema rečima autora, proizilazi iz činjenice da je u trenutnoj verziji, zbog određenih bagova u programu, onemogućena hardverska akceleracija. To praktično znači da se aplikacija, koja najverovatnije po prvi put uvodi broj frejmova u sekundi kao relevantan faktor jednog ostvarenja za razvoj relacionih baza podataka, zahvaljujući softverskom renderovanju, sporija od očekivanog. Pa ipak, da stvari nisu toliko crne dokazuje i prisustvo gomile inovacija i podrška za funkcionalnosti koje su nedostajale DBDesigneru, zbog čega, realno gledano, na duge staze MySQL Workbench ima potencijala da postane najpopularnija alatka za rad sa MySQL-om. Uvođenje podrške za prikaz i modeliranje skladištenih rutina, triggera i pogleda samo su neke od inovacija koje kod DBDesignera, razume se, nisu postojale ni pod tačkom razno. Veoma pregledan editor SQL skripti, kao i gomila sitnih opcija koje mogu znatno da pomognu u optimizaciji relacionih modela namenjenih MySQL platformi, predstavljaju nešto što nijedan ozbiljan korisnik MySQL-a ne sme da propusti. Od notacija ER modela u open source varijanti programa dostupna je samo crows foot notacija, dok je za korišćenje drugih, kao što je recimo IDEF1X, potrebno nabaviti standardnu verziju. Od ostalih opcija, osim integrisanog GRT shella, sve su manje-više standardne za bilo koju alatku ovog tipa, uz napomenu da je primetno da se u toku razvoja povelo dosta računa o važnim specifičnostima MySQL RDBMS-a, što je, naravno, logično, s obzirom na to da oba paketa dolaze iz iste kuće. MySQL Workbench 6.0 CE je grafički alat za rad sa MySQL serverima i bazama podataka. Što se tiče funkcionalnosti pokriva sledeće oblasti:

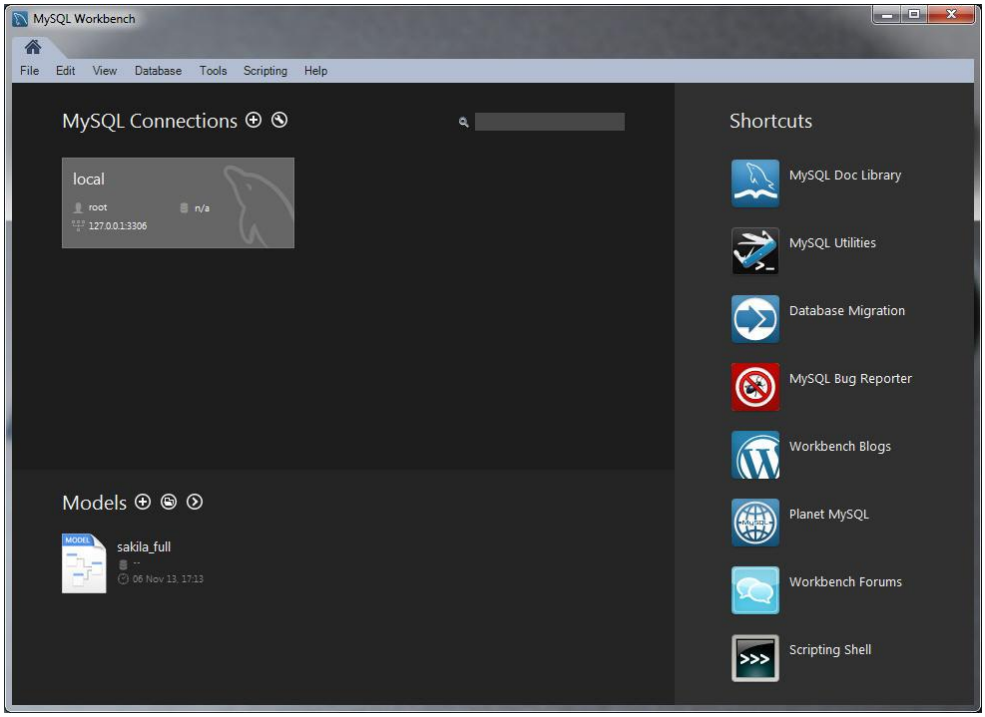
- SQL razvoj: Omogućava korisniku kreiranje i upravljanje konekcijama sa serverima baza podataka, kao i konfigurisanje parametara konekcije. Takođe omogućava izvršavanje SQL upita nad bazom podataka uz pomoć ugrađenog SQL editora.
- Modeliranje podataka: Omogućava kreiranje modela šeme baze podataka u grafičkom
- okruženju, reverzan i direktan inženjering između šeme i baze podataka, kao i izmenu
- svih aspekata baze podataka uz pomoć editora tabela. Editor tabela nudi jednostavne za korišćenje alate za izmenu tabela, kolona, indeksa, okidača, particionisanja, opcija, umetanja, privilegija, uskladištenih procedura i pogleda.
- Administracija servera: Omogućava kreiranje i administraciju instanci servera.

Za primere u ovom uputstvu je korišćen lokalno instaliran MySQL server. Ako imamo pristup samo udaljenom MySQL serveru, neophodno je da unesemo odgovarajuće parametre za konekciju.

## Administracija MySQL servera

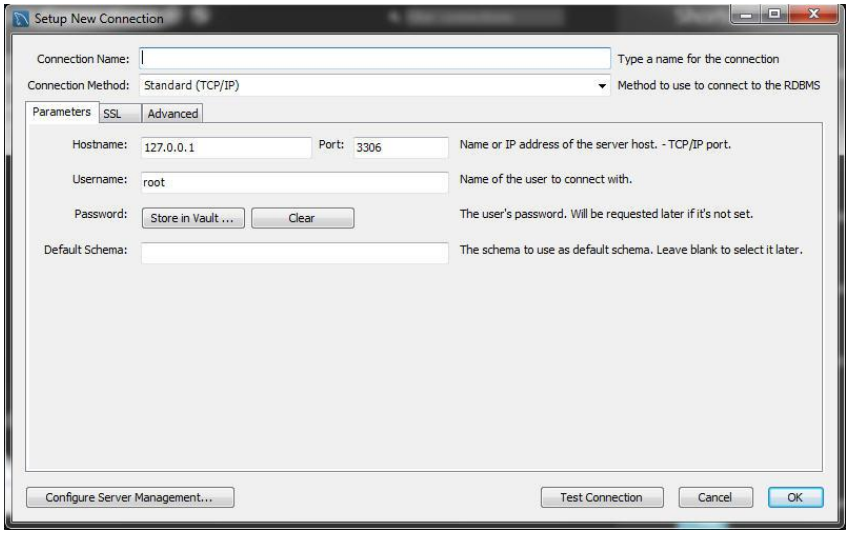
U ovom odeljku ćemo koristiti MySQL Workbench za obavljanje administrativnih funkcija, poput pokretanja i zaustavljanja servera.

1. Pokrenimo MySQL Workbench. Pojaviće se Home prozor.



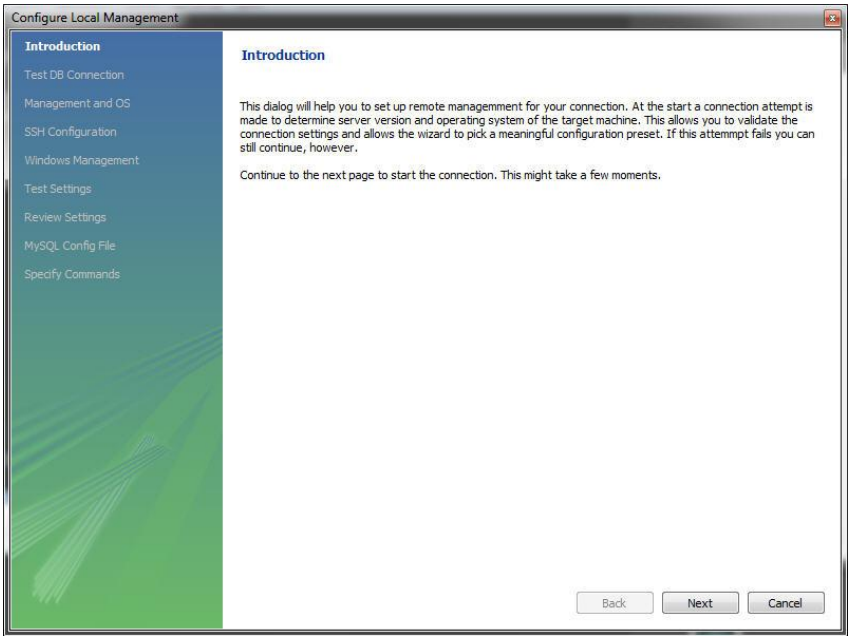

*Sl. 10.1 MySQL Workbench home*

2. Da bismo administrirali naš MySQL server moramo prvo kreirati instancu servera. Instanca sadrži informacije u vezi ciljnog servera, uključujući i one neophodne za povezivanje na isti. Na Home prozoru je potrebno kliknuti na MySQL Connections +. Biće prikazana prva stranica čarobnjaka Create New Server Instance Profile.




*Sl. 10.2 Setup new connection*

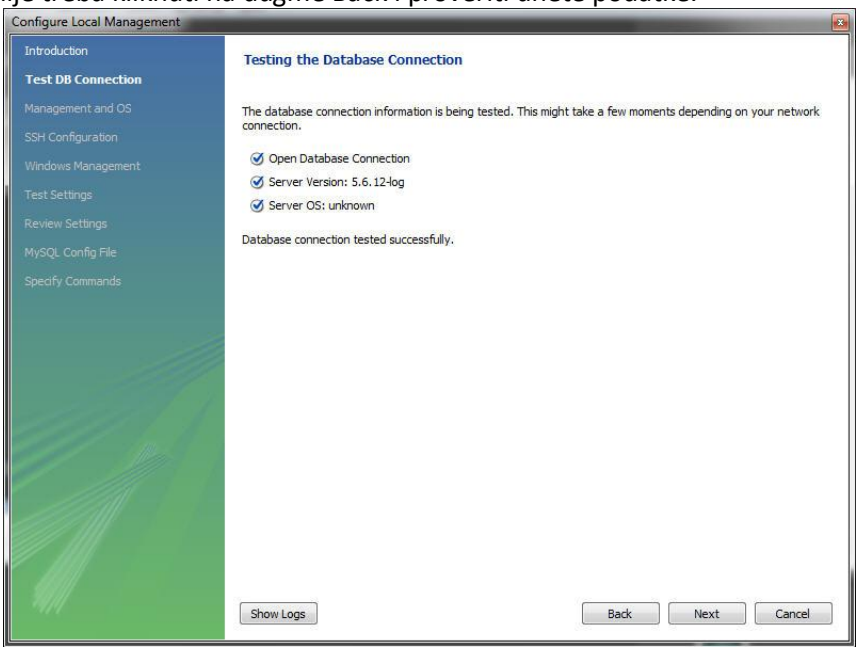
3. Sada je potrebno definisati konekciju ili izabrati postojeću konekciju koja će biti korišćena za povezivanje na server. Možemo koristiti podrazumevane vrednosti na stranici (pretpostavlja se da konekcija nije bila prethodno kreirana). Ukoliko MySQL server ima definisanu lozinku za root nalog istu je moguće uneti klikom na dugme Store in Vault ... Na ovaj način će nam biti omogućeno da se konektujemo na server bez potrebe da svaki put unosimo lozinku. Ako je neophodno, na ovoj stranici je moguće koristiti i drugi nalog za konektovanje na MySQL server (moguće je definisati drugo korisničko ime i odgovarajuću lozinku).
4. Biće izvršeno povezivanje na lokalni server, pa na prvoj stranici čarobnjaka treba kliknuti na dugme Configure Server Management.




*Sl. 10.3 Introduction*

Sada možemo kliknuti na dugme Next.

5. Konekcija će sada biti testirana. Na kraju testiranja će jasno biti vidljivo da li je konekcija bila uspešna. Ako nije treba kliknuti na dugme Back i proveriti unete podatke.



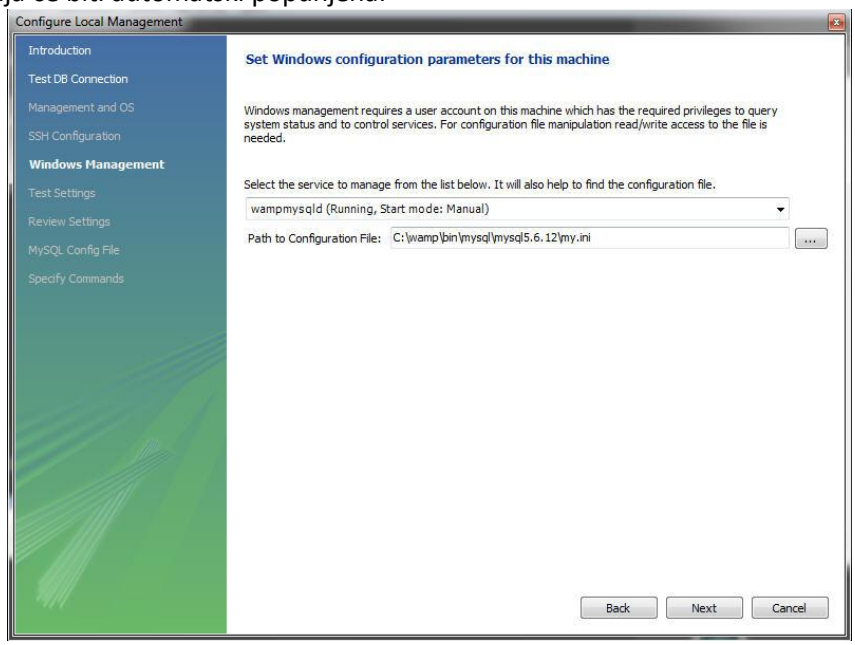

*Sl. 10.4 Test DB connection*

Ako je konekcija bila uspešna kliknuti na dugme Next.

6. Opciono možete konfigurisati metod za udaljeno upravljanje ako je prethodno bio specificiran Remote Host. Ove opcije omogućavaju MySQL Workbench-u da odredi lokaciju konfiguracionih fajlova i ispravne komande za pokretanje i zaustavljanje servera.
SSH login bazirano upravljanje i nativno Windows udaljeno upravljanje su na raspolaganju. U padajućim listama Operating System i MySQL Installation Type su izabrane vrednosti za SSH login bazirano upravljanje.

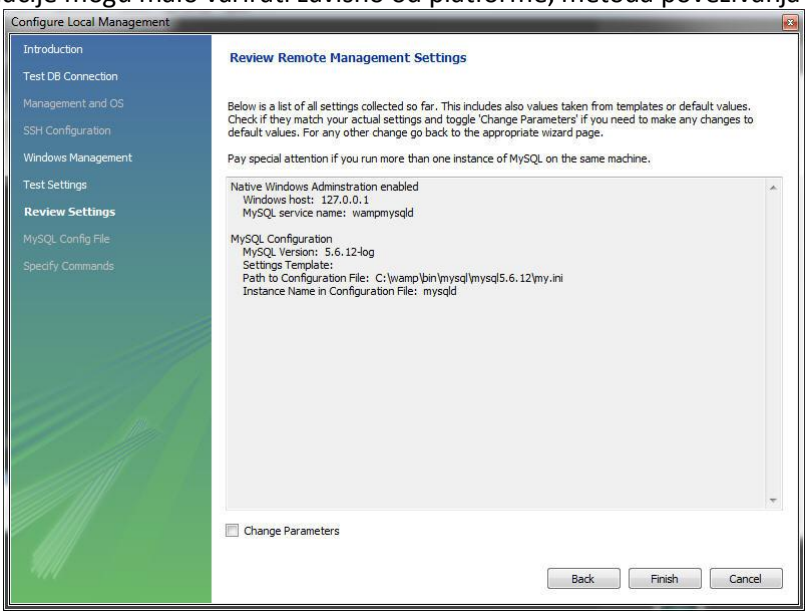
Definisati metod za udaljeno upravljanje i kliknuti na dugme Next.

7. Ako je bilo izabrano SSH login bazirano upravljanje, na ovoj stranici je neophodno konfigurisati njegove parametre koji uključuju naziv host-a, korisničko ime i opciono SSH ključ za autentifikaciju. Potebno je proveriti da li su podaci ispravno uneseni i kliknuti na dugme Next.
8. Ako je MySQL instaliran kao Windows servis, ovde treba definisati Windows konfiiguracione parametre. Polja će biti automatski popunjena.




*Sl. 10.5 Windows management*

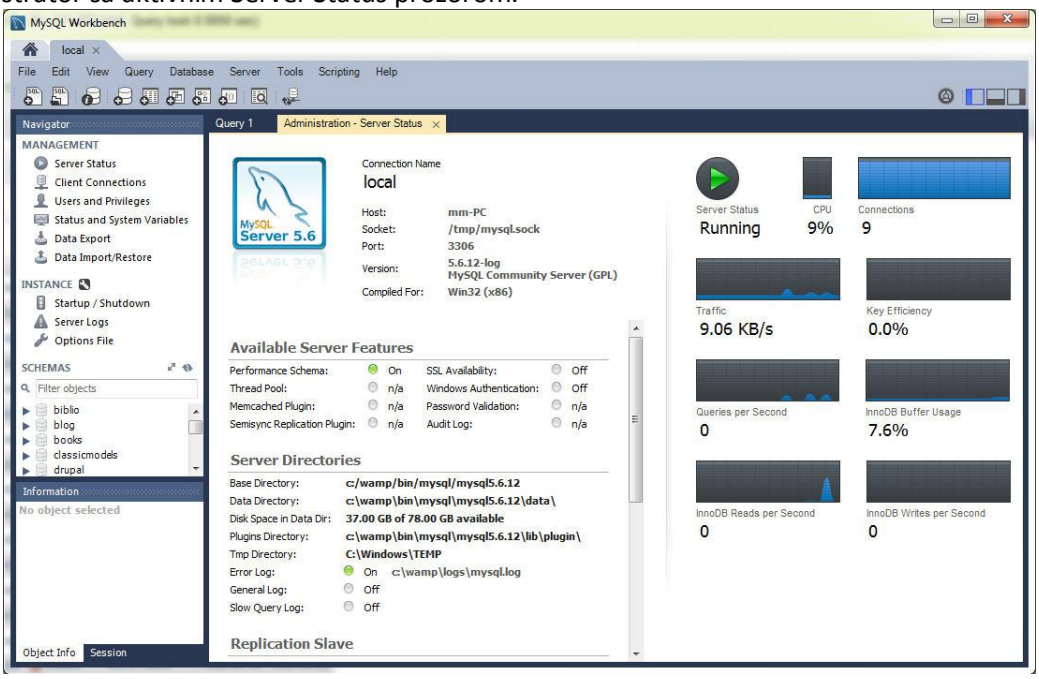
Proveriti da li su podaci ispravno uneseni i kliknuti na dugme Next.
9. Čarobljak će sada proveriti da li je u stanju da pristupi konfiguracionom fajlu MySQL servera i komandama za pokretanje i zaustavljanje servera.
Kliknuti na dugme Next.
10. Sada imamo priliku da pregledamo do sada definisana podešavanja za instancu servera. Prikazane informacije mogu malo varirati zavisno od platforme, metoda povezivanja i tipa instalacije.




*Sl. 10.6Review settings*

Pregledati podatke i kliknuti na dugme Next.

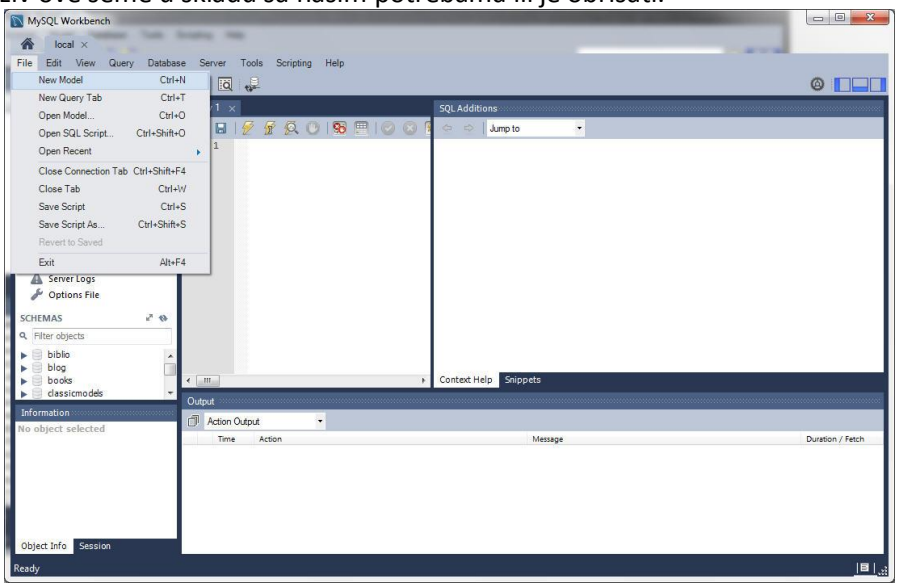
11. Konačno možemo dati pogodan naziv instanci servera. Ovaj naziv će biti korišćen prilikom biranja instance servera iz liste raspoloživih instanci.
Definisati željeni naziv i kliknuti na dugme Finish da bi se završio proces kreiranja instance servera.
12. Sada će ponovo biti prikazan Home prozor. Videćemo novu instancu servera koju smo kreirali zajedno sa novom konekcijom koju ste kreirali u prethodnoj proceduri.
Sada smo spremni da testiramo novu instancu servera.
13. Dvaput kliknimo levim tasterom miša na instancu servera koju smo kreirali. Otvara se MySQL Administrator sa aktivnim Server Status prozorom.




*Sl. 10.7 Server status*

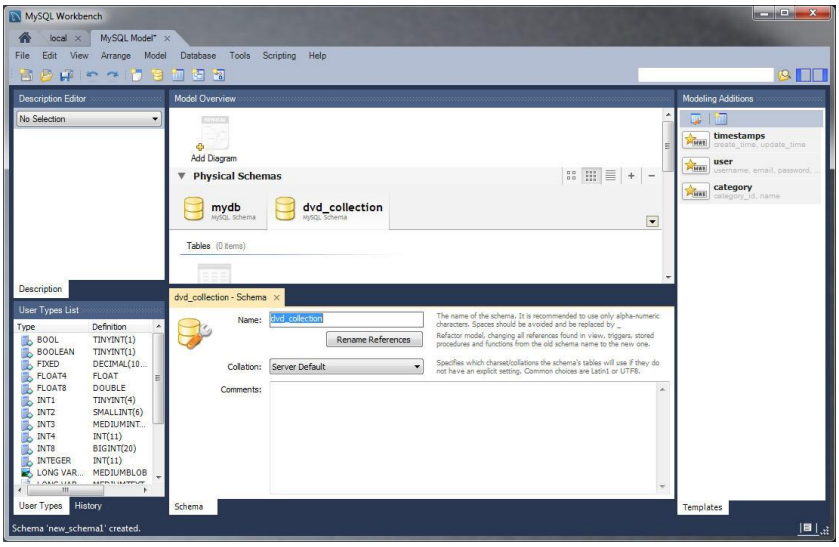
## Kreiranje modela

U ovom odeljku ćemo naučiti kako da kreiramo novi EER model baze podataka, novi EER dijagram i tabele i zatim izvršimo direktan inženjering između modela i pokrenutog servera.
1. Pokrenuti MySQL Workbench. U Home prozoru kliknuti na link File / Create New Model. Model može sadržati više šema. Kada kreiramo novi model on podrazumevano sadrži mydb šemu. Možemo promeniti naziv ove šeme u skladu sa našim potrebama ili je obrisati.



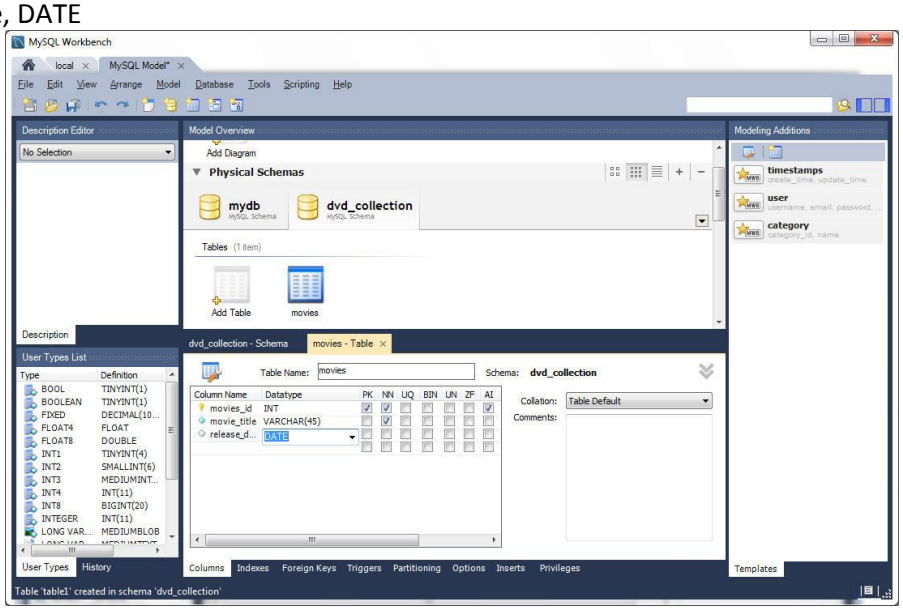

*Sl. 10.8 New model*

Na Physical Schemadata paleti alatki kliknite na dugme + kako biste dodali novu šemu. Na ovaj način će biti kreirana nova šema i biće prikazan njen prozor. U prozoru nove šeme promenite naziv šeme (polje Name) na „dvd_collection". Proverite da li je nova šema sa upravo definisanim nazivom dodata u prozor Physical Schemadata. Sada ste spremni da dodate tabelu u šemu. Ukolio se u ovom trenutku pojavi poruka u kojoj se korisnik pita da li da preimenuje sva pojavljivanja šeme, treba kliknuti na dugme Yes.



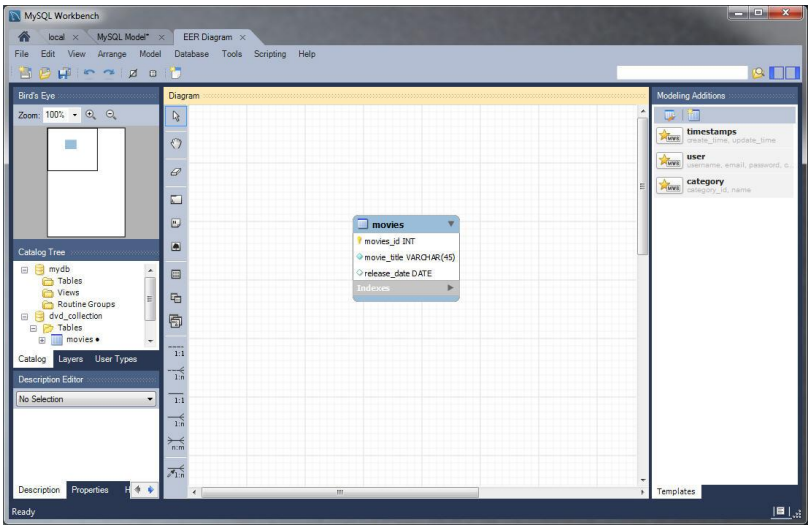

*Sl. 10.9 Physical schemas*

3. U prozoru Physical Schemadata dvaput kliknuti levim tasterom miša na dugme Add Table.
4. U editoru tabela promeniti naziv tabele u „movies" (polje Table Name).
5. Promeniti naziv prve kolone u „movie_id". Izabrati tip podataka INT. Sada je za ovi kolonu potrebno definisati sledeća svojstva: Primary Key, Not Null, Auto Incremental. Čekirati opcije PK, NN i AI.
6. Dodati još dve kolone:
movie_title, VARCHAR(45), NN
release_date, DATE



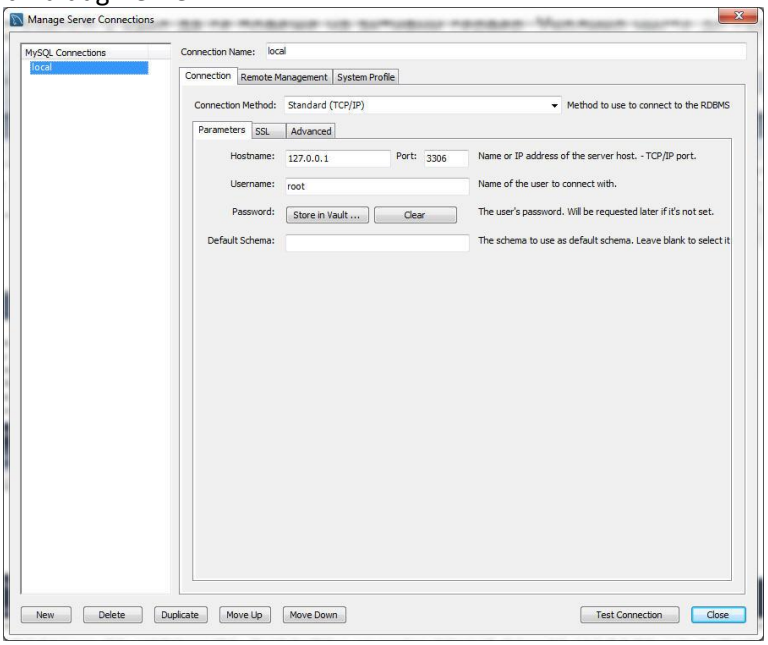

*Sl. 10.10 Add table*

7. Sada ćemo preći na vizuelnu reprezentaciju šeme. U glavnom meniju kliknuti na meni Model i izabrati komandu Create Diagram from Catalog Objects. EER dijagram će biti kreiran i prikazan.




*Sl. 10.11 EER diagram*

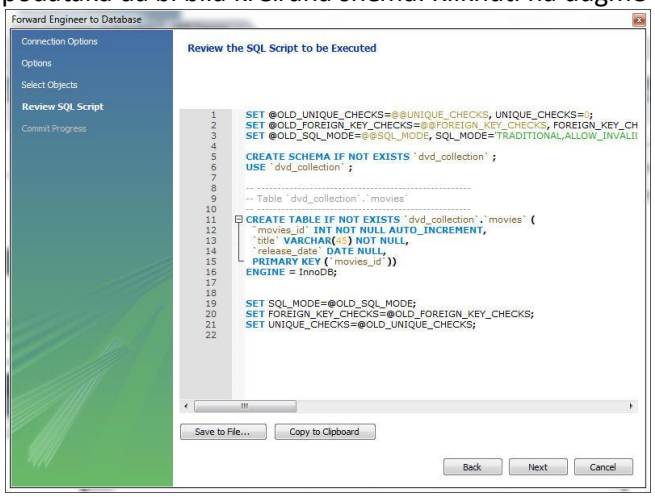
8. U editoru tabela promeniti naziv kolone „movie_title" u „title". EER dijagram biva automatski ažuriran.
9. U ovom trenutku možemo sačuvati model. Kliknuti na dugme Save Model to Current File. Pošto je ovo prvo snimanje fajla biće neophodno definisati naziv modela. Uneti naziv „Home_Media". „Home_Media" model može sadržati i druge šeme pored „dvd_collection" šeme (na primer, „cd_collection" šemu). Kliknuti na dugme Save da bi model bio snimljen.
10. Možemo sihnronizovati model sa aktivnim serverom baza podataka. Prvo moramo reći MySQL Workbench-u kako da se poveže na aktivni server. Ukoliko nismo do sada definisali konekciju sa serverom baza podataka to možemo uraditi tako što ćemo u glavnom meniju kliknuti na meni Database i izabrati komandu Manage Connections ... i na kraju u okviru za dijalog Manage DB Connections kliknuti na dugme New.




*Sl. 10.12 Manage server connection*

11. Sada smo spremni da izvršimo direktan inženjering našeg modela na aktivan server baza podataka. U glavnom meniju kliknuti na meni Database i izabrati komandu Forward Engineer ... Pokreće se čarobnjak Forward Engineer to Database.

12. Na prvoj stranici čarobnjaka je potrebno izabrati prethodno definisanu konekciju iz padajuće liste Stored Connections i kliknuti na dugme Next.
13. Na stranici sa opcijama su prikazane različite napredne opcije vezane za direktan inženjering. Treba ih zanemariti i kliknuti na dugme Next.
14. Na sledećoj stranici je moguće izabrati objekte koji se žele izvesti na aktivan server baza podataka. U ovom slučaju, pošto je kreirana samo jedna tabela, nije neophodno birati druge objekte. Kliknuti na dugme Next.
15. Na sledećoj stranici - Review SQL Script - je prikazan sql skript fajl koji će biti pokrenut na aktivnom serveru baza podataka da bi bila kreirana shema. Kliknuti na dugme Next.




*Sl. 10.13 SQL script review*

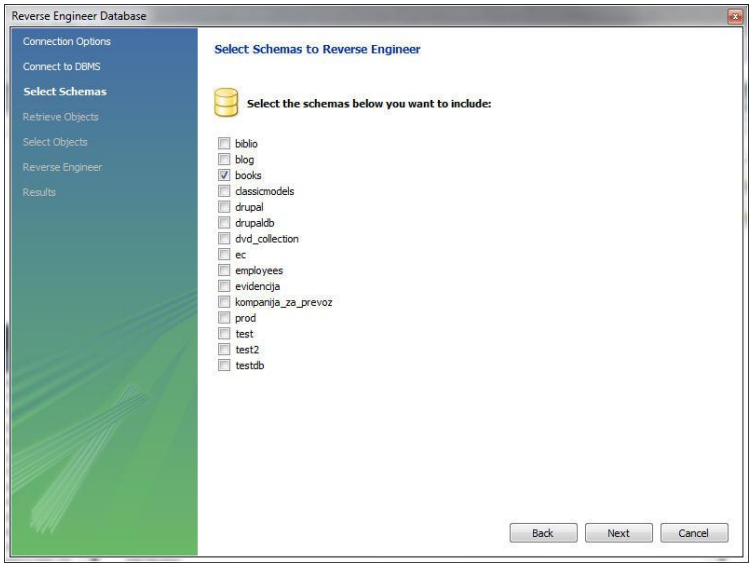
16. Kliknuti na dugme Next. Kliknuti na dugme Close.

17. Kliknuti na dugme Save Model to Current File.

## Obrnuti inženjering (engl. REVERSE ENGINEERING)

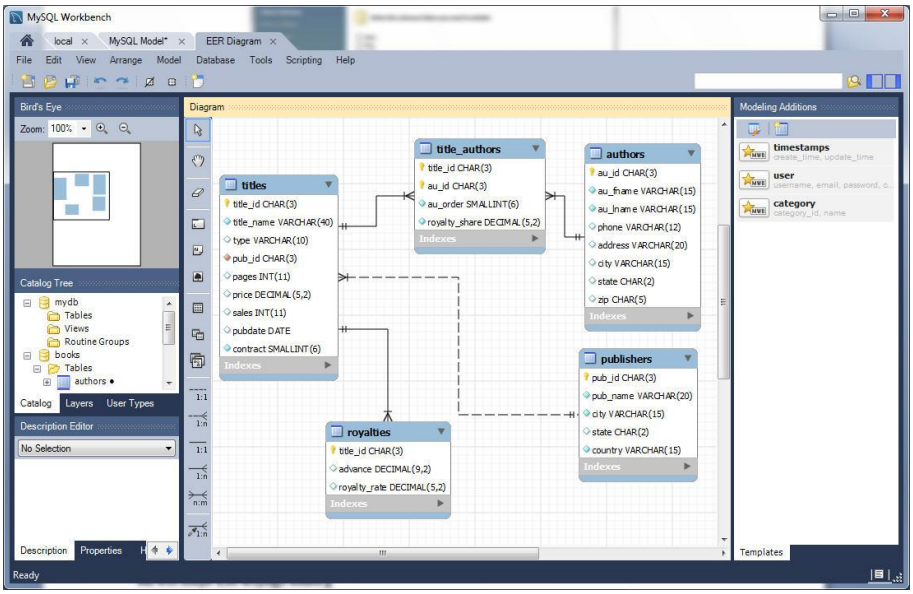
Obrnuti inženjering je postupak kojim se baza podataka koja se nalazi na SQL serveru vraća u ER dijagram. Postupak dobijanja ER modela se vrši prema sledećim koracima:

Nakon uspešne konekcije na SQL server iz menija Database biramo opciju Reverse Engineer... Pojaviće nam se dijalog o izboru baze podataka.




*Sl. 10.14 Reverse engineering database*

Tokom dijaloga postoji mogućnost izbora pojedinih tabela ili rutina. Nakon završetka dijaloga dobijamo ER model kao što je prikazano na sledećoj slici.




*Sl. 10.15 ER dijagram baze books*

Postoji mogućnost promene stukture baze pomoću ER modela. Promene možemo sinhronizovati sa SQL serverom izborom opcije Synchronize model… iz Database menija. U slučaju sinhronizacije moraćemo ispratiti odgovarajući dialog prozor gde ukazujemo na SQL server i bazu podataka.

## Kreiranje ER dijagrama

ER dijagram se dodaje dvostrukim klikom na ikonu Add Diagram. Možemo kreirati proizvoljan broj ER dijagrama baš kao što možemo kreirati proizvoljan broj fizičkih šema podataka. Svaki ER dijagram biva prikazan kao kartica ispod palete alatki. Željeni ER dijagram se bira klikom na karticu.

Klikom na karticu željenog ER dijagrama se dolazi do okruženja koje se koristi za grafičko manipulisanje objektima baze podataka. Vertikalna paleta alatki se nalazi sa leve strane prozora.

### Paleta alatki

Klikom na neku od ikona na paleti alatki se menja izgled pokazivača tako da odgovara samoj ikoni. Ukoliko se pokazivač dovede na neku od ikona pojaviće se kratak opis funkcije ikone.


1. Select Object(s) - Standardni pokazivač, lociran na vrhu vertikalne palete alatki, je podrazumevani pokazivač operativnog sistema. Uz pomoć ove ikone se može vratiti na standardni pokazivač nakon korišćenja nekog od raspoloživih alata. Vraćanje na standardni pokazivač uz pomoć tastature se može izvršiti pritiskom na taster Esc.
2. Move Model - Uz pomoć ovog alata se može pomerati ceo ER dijagram. Treba kliknuti na ovu ikonu levim tasterom miša i zatim kliknuti levim tasterom miša na bilo koju lokaciju u radnom prostoru. Pomeranje dijagrama se vrši pomeranjem miša uz pritisnut levi taster miša.
3. Delete Object - Uz pomoć ovog alata je moguće obrisati neki od objekata ER dijagrama. Treba kliknuti na ovu ikonu i zatim kliknuti na željeni objekat. U zavisnosti od podešavanja može se pojaviti okvir za dijalog u kome se od korisnika traži da potvrdi brisanje. Objekat je moguće obrisati i ako se klikne desnim tasterom miša na isti i iz kontekstualnog menija izabere komanda Delete.
4. Place a New Layer at Selected Area - Uz pomoć ovog alata se mogu organizovati objekti ER dijagrama (mogu se, na primer, grupisati slični objekti). Treba kliknuti na ovu ikonu i nacrtati pravougaonik u radnom prostoru. Nakon toga se treba vratiti na standardni pokazivač i izabrati objekte koji se žele smestiti u layer.
5. Place a New Text Object - Uz pomoć ovog alata je moguće dodati tekstualni objekat ER dijagramu. Treba kliknuti na ikonu i zatim kliknuti na željenu lokaciju u radnom prostoru. Nakon dodavanja tekstualnog objekta standardni pokazivač postaje aktivan. Da bi se objaktu dodao tekst treba kliknuti desnim tasterom miša na isti i iz kontekstualnog menija izabrati komandu Edit Note.
6. Place a New Image - Uz pomoć ovog alata je moguće dodati sliku u radni prostor. Kada se klikne na ovu ikonu i zatim klikne u radni prostor otvara se okvir za dijalog uz pomoć koga korisnik može da izabere željeni fajl.
7. Place a New Table - Uz pomoć ovog alata je moguće kreirati novu tabelu u okviru ER dijagrama. Treba kliknuti na ovu ikonu i zatim kliknuti na željenu lokaciju u radnom prostoru. Tabelu je moguće izmeniti uz pomoć editora tabela tako što se klikne desnim tasterom miša na novokreiranu tabelu i iz kontekstualnog menija izabere komanda Edit Table ...
8. Place a New View - Uz pomoć ovog alata je moguće kreirati novi pogled. Treba kliknuti na ovu ikonu i zatim kliknuti na željenu lokaciju u radnom prostoru. Pogled je moguće izmeniti tako što se klikne desnim tasterom miša na novokreirani pogled i iz kontekstualnog menija izabere komanda Edit View ...
9. Place a New Routine Group - Uz pomoć ovog alata je moguće kreirati grupu uskladištenih procedura. Treba kliknuti na ovu ikonu i zatim kliknuti na željenu lokaciju u radnom prostoru. Grupu uskladištenih procedura je moguće izmeniti tako što se klikne desnim tasterom miša na novokreiranu grupu i iz kontekstualnog menija izabere komanda Edit Routine Group ...
10. Place a New 1:1 Non-Identifying Relationship - Nova 1:1 neidentifikujuća veza
11. Place a New 1:n Non-Identifying Relationship - Nova 1:N neidentifikujuća veza
12. Place a New 1:1 Identifying Relationship - Nova 1:1 identifikujuća veza
13. Place a New 1:n Identifying Relationship - Nova 1:N identifikujuća veza
14. Place a New n:m Identifying Relationship - Nova N:M identifikujuća veza
15. Place a Reletionship Using Existing Columns - Nova 1:N veza preko postojećih kolona

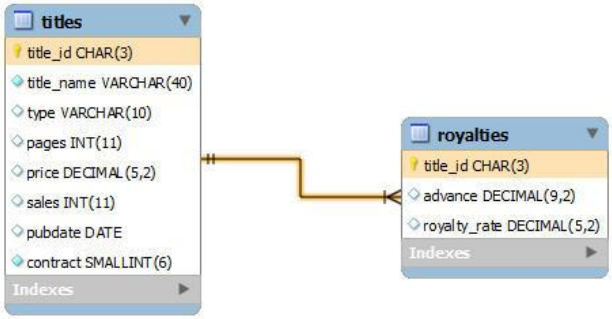
### Dodavanje veza u okviru EER dijagrama

Identifikujuća veza je ona kod koje dete tabela ne može biti jedinstveno (jednoznačno) identifikovana bez roditelj tabele. Ovo se obično dešava u slučaju kada se kreira pomoćna tabela da

bi se realizovala više-prema-više veza. U takvim slučajevima primarni ključ se obično sastoji od primarnih ključeva tih dveju tabela. Kliknuti na odgovarajuću ikonu veze koja se želi kreirati. Ako se kreira jedan-prema-više veza prvo treba kliknuti na tabelu koja je na „više" strani veze, a zatim na tabelu koja sadrži referencirani ključ. Na ovaj način se kreira kolona u tabeli koja je na „više" strani veze.

Da bi se izmenila svojstva stranog ključa treba kliknuti dvaput levim tasterom miša na bilo koju lokaciju na liniji kojom je predstavljena veza. Na ovaj način se otvara editor u donjem delu prozora sa karticama Relationship i Foreign Key.

Dovođenjem pokazivača na liniju kojom je predstavljena veza bivaju istaknuti veza i povezani ključevi (pogledati sliku ispod).




U MySQL Workbench-u, prilikom kreiranja dijagrama mogu se odabrati sledeći tipovi veza:

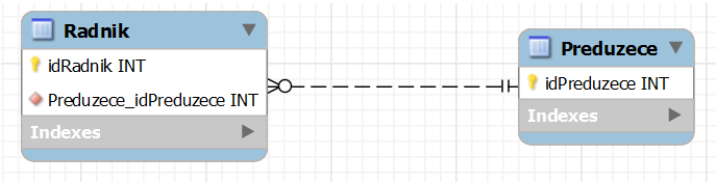
1. 1:1 Non identifying relationship
2. 1:N Non identifying relationship
3. n:m Identifying relationship (agregacija)
4. 1:n Identifying relationship (slab entitet)
5. 1:1 Identifying relationship (specijalizacija)

Osobine veza mogu da se promene selektovanjem veze na dijagramu, pritiskom desnog tastera misa, i izborom opcije Edit relationship. U okviru Caption polja može se zadati ime veze. Izborom Foreign Key dugmeta mogu se postaviti dodatne opcije kao na primer, da li je kardinalnost (1:1) ili (1:n), da li je donja vrednost u kardinalnosti veze 0 ili 1 (mandatory), da li je veza identifikujuća ili ne (identifying relationship), odnosno da li primarni ključ jednog tipa entiteta postaje primarni ključ drugog tipa entiteta sa kojim stupa u vezu i drugo.

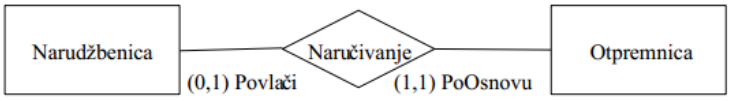
# Primeri EER dijagrama u MySQL Workbench-u

**Prvi primer:**

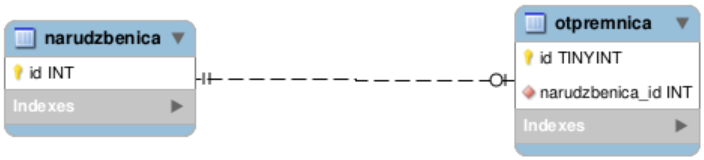

Rešenje u MySQL Workbench-u:




**Drugi primer:**

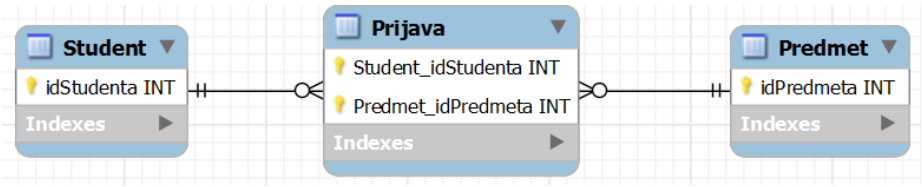



Rešenje u MySQL Workbench-u:

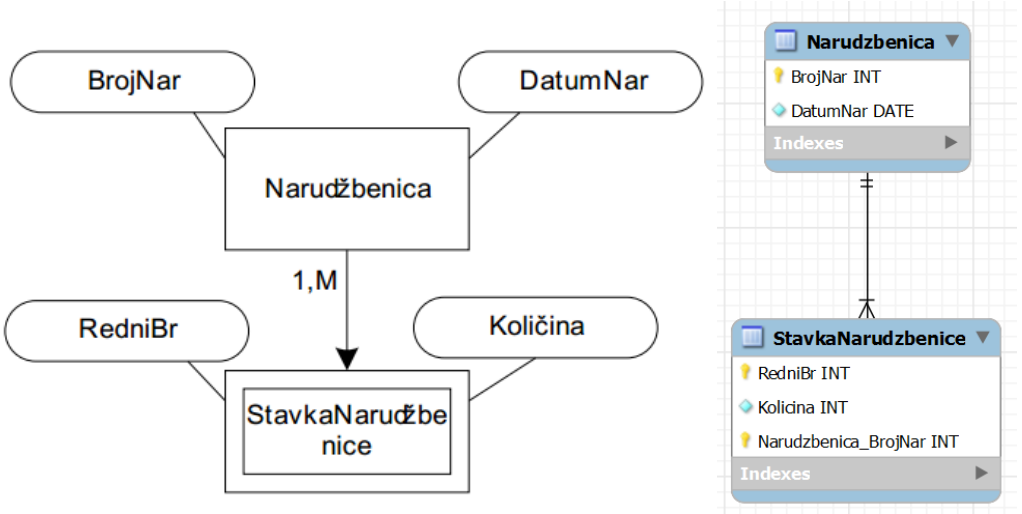



**Treći primer:**

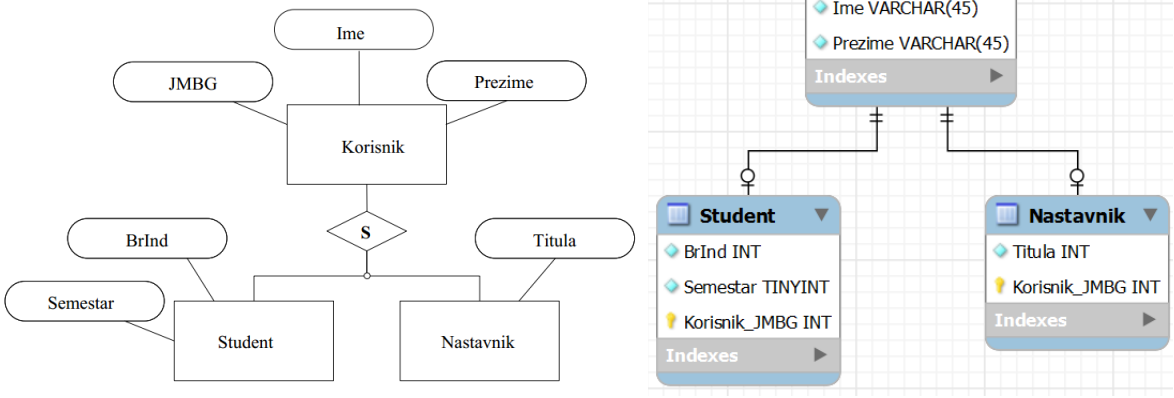

Rešenje u MySQL Workbench-u:



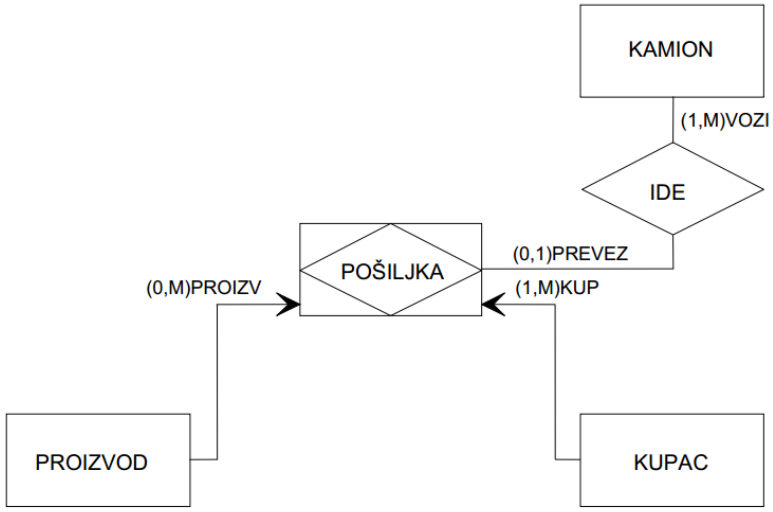

**Četvrti primer:**



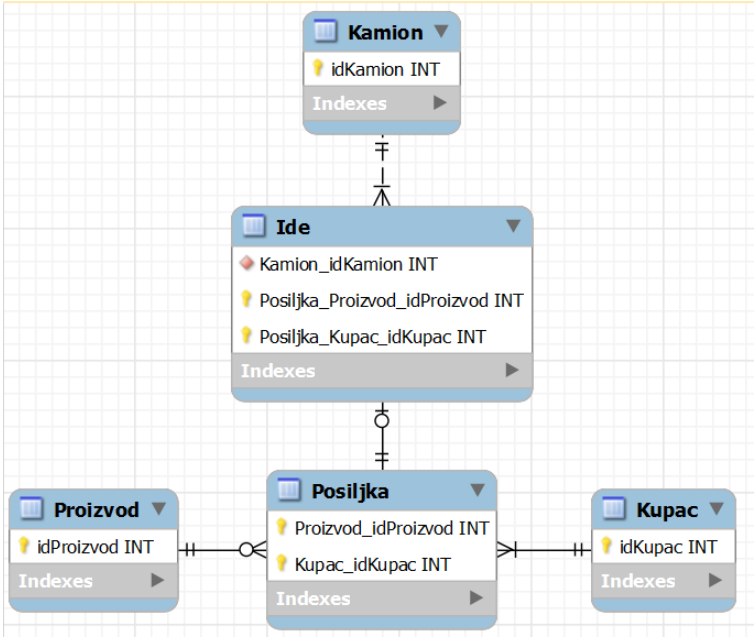
**Peti primer:**



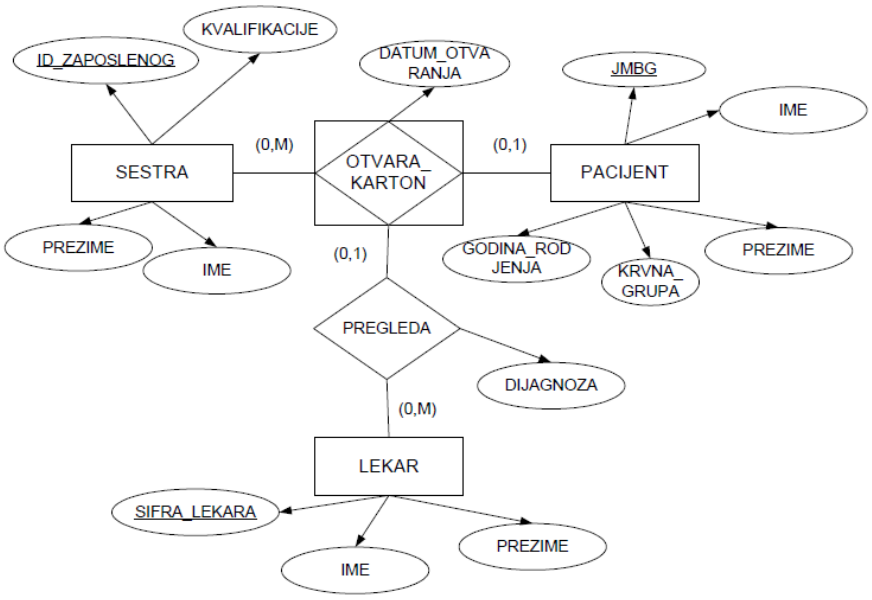

**Šesti primer:**



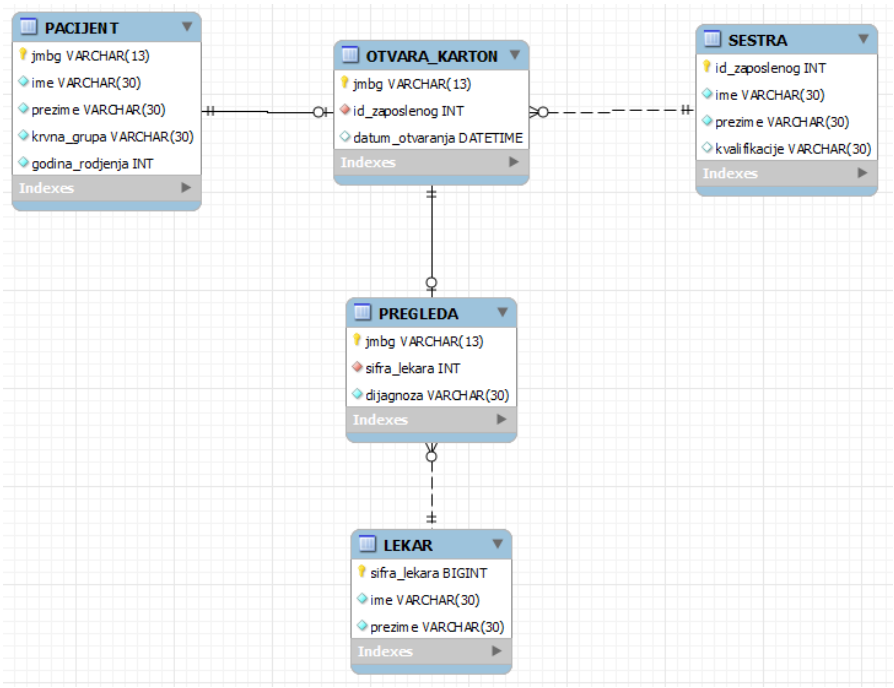

Rešenje u MySQL Workbench-u:



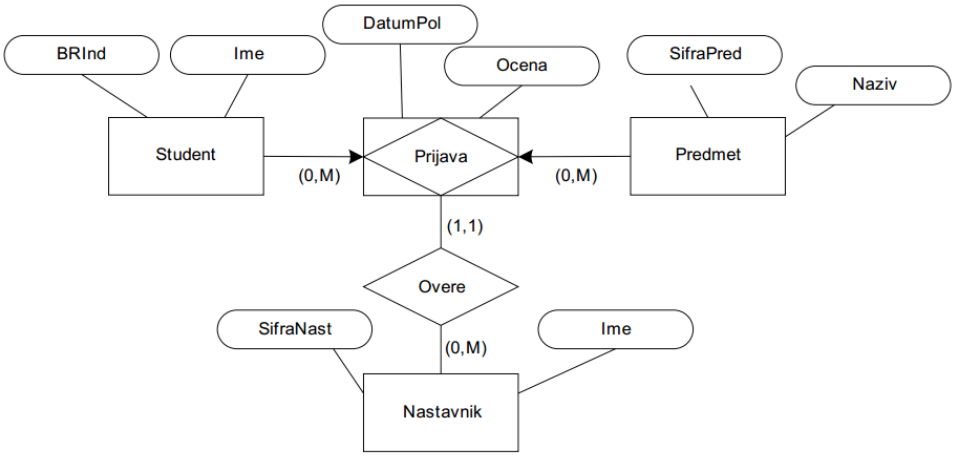
**Sedmi primer:**



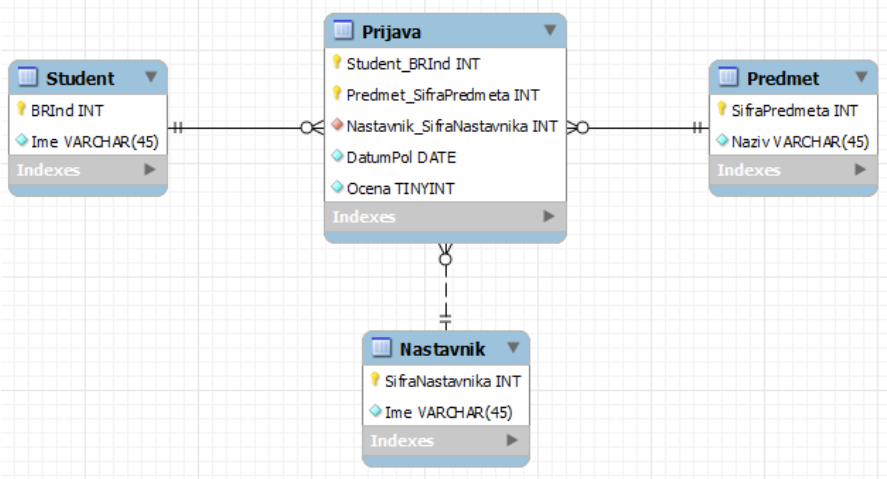

Rešenje u MySQL Workbench-u:



**Osmi primer:**

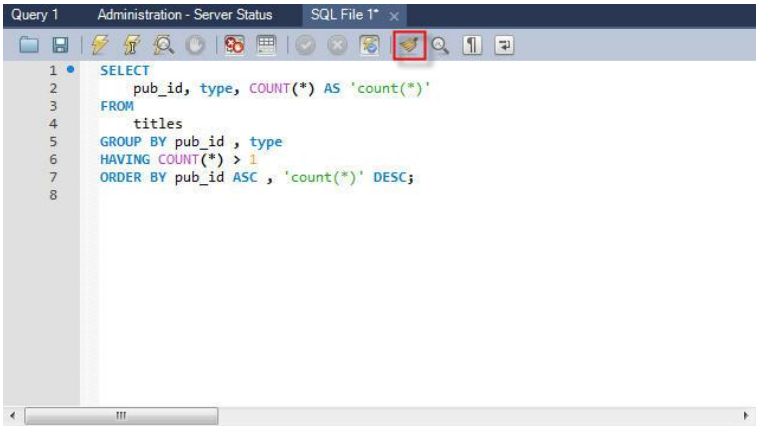



Rešenje u MySQL Workbench-u:




## Formatiranje SQL iskaza

MySQL Workbench omogućava automatsko formatiranje SQL iskaza. Iz **Edit** menija se bira opcija **Format**, zatim **Beutify Query**. Na istom mestu nalazimo i opciju **UPCASE keywords**. Komandom **Beutify** dobijamo pravilno napisan SQL iskaz.




*Sl. 10.16 SQL script editor*

# 11. POGLAVLJE

## XML struktura podataka i upiti

XML je akronim od eXtensible Markup Language, a konzorcijum W3C prihvatio ga je kao standard za označavanje (engl. markup) dokumenata. On definiše opštu sintaksu za označavanje podataka jednostavnim, svima razumljivim oznakama (engl. tags). XML obezbeđuje standardan format za računarske dokumente, dovoljno fleksibilan da se može prilagoditi za najrazličitije oblasti kao što su Web lokacije, elektronska razmena podataka, vektorska grafika, ponuda nekretnina, serijalizacija objekata, pozivanje udaljenih procedura, sistemi glasovne pošte itd. Možemo pisati sopstvene programe za rad s podacima u XML dokumentima i njihovu obradu. Ako to uradimo, postaće nam dostupan veliki broj besplatnih biblioteka na raznim jezicima na kojima se može čitati i pisati XML, pa čemo moći da se usredsredimo baš na ono što je potrebno našem programu. Za rad sa XML dokumentima možemo upotrebiti i gotov softver, kao što su čitači Weba (engl. Web browsers) ili programi za obradu teksta. Jedan deo alatki može da radi sa svim XML dokumentima. Druge su prilagođene za podršku određenoj XML aplikaciji u nekoj oblasti - poput vektorske grafike - i izvan nje su najčešće neupotrebljive. No, u svim slučajevima koristi se ista sintaksa, čak i ako je namerno sakrivena u alatkama koje se lakše koriste ili ograničena na jednu aplikaciju.

XML je metajezik za označavanje tekstualnih dokumenata. Podaci se smeštaju u XML dokumente u obliku znakovnih nizova (engl. strings), između tekstualnih oznaka koje ih opisuju. Osnovne jedinice podataka i oznaka u XML-u nazivaju se elementi. XML specifikacija precizno definiše sintaksu kojeg se moramo pridržavati pri pisanju oznaka: kako su elementi razgraničeni oznakama, kako oznaka izgleda, kakva imena elementi mogu imati, gde se stavljaju atributi itd. Površno gledano, oznake XML dokumenta mnogo liče na oznake HTML dokumenta, ali među njima postoje bitne razlike.

### Osnove XML-a

U primeru 1 prikazan je jednostavan XML dokument, kakav se sreće u sistemu za upravljanje inventarom ili u bazi podataka skladišta. Njegovi podaci su obeleženi oznakama (engl. tags) i atributima koji opisuju boju, veličinu, broj bar-koda, ime proizvođača, ime proizvoda itd.

Primer 1. XML dokument

```
<?xml version="1.0"?>
<product barcode="2394287410">
<manufacturer>Verbatim</manufacturer>
<name>DataLife MF 2HD</name>
<quantity>10</quantity>
<size>3.5"</size>
<color>black</color>
<description>floppy disks</description>
</product>
```

Ovaj dokument je tekst, pa se može smestiti u tekstualnu datoteku koju možemo uređivati u svakom standardnom programu za obradu teksta, kao što su BBEdit, jE-dit, UltraEdit, Emacs ili vi. Za XML nam nije potreban specijalan program za obradu teksta. Štaviše, smatramo da većina XML editora opšte namene više košta nego što vredi i da ih je mnogo teže koristiti od klasičnih programa za obradu teksta. Za razliku od HTML-a, XML uzima u obzir razliku između malih i velikih slova. Element OSOBA nije jednak elementu osoba, niti elementu Osoba. Ako element otvorimo oznakom <osoba>,

ne možemo ga zatvoriti oznakom </0S0BA>. Možemo upotrebljavati i mala i velika slova po svom izboru, ali svaki element moramo dosledno ispisati.

Primer 2. cd_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
      <CD>
            <TITLE>Empire Burlesque</TITLE>
            <ARTIST>Bob Dylan</ARTIST>
            <COUNTRY>USA</COUNTRY>
            <COMPANY>Columbia</COMPANY>
            <PRICE>10.90</PRICE>
            <YEAR>1985</YEAR>
      </CD>
      <CD>
            <TITLE>Hide your heart</TITLE>
            <ARTIST>Bonnie Tyler</ARTIST>
            <COUNTRY>UK</COUNTRY>
            <COMPANY>CBS Records</COMPANY>
            <PRICE>9.90</PRICE>
            <YEAR>1988</YEAR>
      </CD>
      <CD>
            <TITLE>Greatest Hits</TITLE>
            <ARTIST>Dolly Parton</ARTIST>
            <COUNTRY>USA</COUNTRY>
            <COMPANY>RCA</COMPANY>
            <PRICE>9.90</PRICE>
            <YEAR>1982</YEAR>
      </CD>
      <CD>
            <TITLE>Still got the blues</TITLE>
            <ARTIST>Gary Moore</ARTIST>
            <COUNTRY>UK</COUNTRY>
            <COMPANY>Virgin records</COMPANY>
            <PRICE>10.20</PRICE>
            <YEAR>1990</YEAR>
      </CD>
</CATALOG>
```

**Deklaracija XML-a**

XML dokumenti bi trebalo da otpočnu deklaracijom XML-a (ali ne moraju). Dekla-racija XML-a izgleda kao instrukcija za obradu nazvana xml, koja sadrži pseudoatri-bute version, standalone i encoding. Strogo uzev, to nije instrukcija za obradu, nego deklaracija XML-a - ni više ni manje od toga. Ilustraciju daje primer 3.

Primer 3. Veoma jednostavan XML dokument s deklaracijom XML-a

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<osoba>
Alen Tjuring </osoba>
```

XML dokumenti ne moraju imati deklaraciju XML-a, ali ako je imaju, ona mora biti prva stavka dokumenta. Pre nje ne sme biti komentara, razmaka (belina), instrukcija za obradu itd. Razlog za to je što XML analizator na osnovu prvih pet znakova (< ?xml) zaključuje kakvo je kodiranje znakova u

dokumentu - recimo, da li je upotrebljen jednobajtni ili višebajtni skup znakova. Pre deklaracije XML-a sme biti samo nevidljiva Unicode oznaka redosleda bajtova.

### Definicija tipa dokumenta (DTD-ovi)

DTD-ovi omogućavaju nametanje ograničenja obliku XML dokumenata, ali unutar njih može biti prilično mnogo fleksibilnosti. DTD nikada ništa ne kazuje o dužini, strukturi, značenju, dozvoljenim vrednostima ili drugim aspektima tekstualnog sadržaja elemenata ili atributa.

Validnost je opciona. Analizator koji čita XML dokument može, ali ne mora proveravati njegovu validnost. Ako ne proverava validnost, program koji od njega prima podatke može primiti ili odbiti nevalidan dokument. U nekim slučajevima, kao što je slanje zapisa bazi podataka, greška validnosti može biti sasvim ozbiljna i ukazati, recimo, na to da zapisu nedostaje određeno obavezno polje. U drugim slučajevima, na primer pri prikazivanju Web stranica, greška validnosti ne mora biti toliko važna i program je može zaobići. Svi XML dokumenti moraju biti dobro oblikovani, ali ne moraju svi biti validni. Naši dokumenti i programi mogu upotrebljavati validaciju kako nam odgovara.

U DTD-u za lične dokumente bilo bio navedeno da element osoba sadrži jedan element potomak, ime_i_prezime, iza koga slede nula ili više elemenata potomaka zanimanje. Nadalje bi bilo navedeno da svaki element ime_i_prezime sadrži tačno jedan element potomak, ime, iza koga sledi tačno jedan element potomak, prezime. Najzad bi bilo navedeno da elementi ime, prezime i zanimanje sadrže tekst. U primeru 4, DTD opisuje takav element osoba.

Primer 4. DTD za element osoba

```
<!ELEMENT osoba   (ime_i_prezime, zanimanje*)>
<!ELEMENT ime_i_prezime (ime, prezime)>
<!ELEMENT ime     (#PCDATA)>
<!ELEMENT prezime (#PCDATA)>
<!ELEMENT zanimanje       (#PCDATA)>
```

Ovaj DTD bi verovatno bio smešten u zasebnu datoteku u odnosu na dokument koji opisuje. Tako bi ga moglo referencirati više XML dokumenata. Međutim, ako je zgodnije, on može biti uključen u XML dokument unutar njegove deklaracije tipa dokumenta, koju ćemo razmotriti u nastavku poglavlja. Ukoliko je DTD smešten u zasebnu datoteku, ona bi verovatno bila nazvana osoba.dtd ili slično tome. Nastavak imena .dtd je uobičajen, premda se to ne zahteva izričito u specifikaciji XML-a. Da tu datoteku servira Web server, dodelio bi joj MIME tip medija application/xml-dtd.

Svaki red primera 4 je deklaracija elementa (engl. element declaration). Prvi red deklariše element *osoba*, drugi - element *ime_i_prezime*, treći - element *ime* itd. Ipak, deklaracije nisu morale biti poslagane svaka u svoj red - to je učinjeno samo radi veće čitljivosti. Mada je uobičajeno da se u svaki red stavlja samo po jedna deklaracija, to nije obavezno. Dugačke deklaracije mogu se protezati na više redova.

Prvi element deklaracije u primeru 4 utvrđuje da svaki element osoba mora sadržati tačno jedan element potomak ime_i_prezime, iza koga sledi nula ili više elemenata zanimanje. **Zvezdica nakon zanimanja znači „nula ili više".** Svaka osoba mora imati ime i prezime, a može ali ne mora imati jedno ili više zanimanja. Dakle, pre svih zanimanja mora doći ime i prezime. Na primer, sledeći element osoba je validan:

```
<osoba> <ime_i_prezime>
<ime>Alen</ime>
<prezime>Tjuring</prezime> </ime_i_prezime>
<zanimanje>naučnik u oblasti računarstva</zanimanje>
```

```
<zanimanje>matematičar</zanimanje>
<zanimanje>kriptograf</zanimanje>
</osoba>
```

Pošto su elementi zanimanje deklarisani kao opcioni, validan je i sledeći element osoba:

```
<osoba>
<ime_i_prezime>
<ime>Alen</ime>
<prezime>Tjuring</prezime>
</ime_i_prezime>
</osoba>
```

Sledeći element osoba **nije validan**, zato što mu nedostaje zahtevani element potomak <ime_i_prezime>:

```
<osoba>
<zanimanje>naučnik u oblasti računarstva</zanimanje>
 <zanimanje>matematičar</zanimanje>
 <zanimanje>kriptograf</zanimanje>
</osoba>
```

Sledeći element osoba **nije validan**, zato što je element zanimanje naveden pre elementa ime_i_prezime:

```
<osoba>
<zanimanje>naučnik u oblasti računarstva</zanimanje>
<ime_i_prezime>
<ime>Alen</ime>
<prezime>Tjuring</prezime>
</ime_i_prezime>
<zanimanje>matematičar</zanimanje>
<zanimanje>kriptograf</zanimanje>
</osoba>
```

Element osoba ne može sadržati elemente koji nisu navedeni u njegovoj deklaraciji. Jedini dodatni znakovni podatak koji može sadržati jeste razmak (belina). Na primer, sledeći element osoba **nije validan** zato što sadrži i element publikacija:

```
<osoba>
<ime_i_prezime>
<ime>Alen</ime>
<prezime>Tjuring</prezime>
</ime_i_prezime>
<zanimanje>naučnik u oblasti računarstva</zanimanje>
<zanimanje>matematičar</zanimanje> <zanimanje>kriptograf</zanimanje>
<publikacija>On Computable Numbers...</publikacija>
</osoba>
```

Sledeći element osoba **nije validan**, zato što sadrži neki tekst izvan dozvoljenih potomaka:

```
<osoba>
<ime_i_prezime>
<ime>Alen</ime>
<prezime>Tjuring</prezime>
</ime_i_prezime>
```

```
bio je
<zanimanje>naučnik u oblasti računarstva</zanimanje>,
<zanimanje>matematičar</zanimanje> i
<zanimanje>kriptograf</zanimanje>. </osoba>
```

U svim ovim primerima nevalidnih elemenata, mogli bismo izmeniti DTD da bismo te elemente učinili validnim. Na kraju krajeva, svi ti primeri su dobro oblikovani, nisu validni uz DTD iz primera 4.

Deklaracija elementa ime_i_prezime kazuje da svaki element ime_i_prezime mora sadržati tačno jedan element ime, iza koga sledi tačno jedan element prezime. Sve varijacije su zabranjene. Preostale tri deklaracije - ime, prezime i zanimanje - kazuju da njihovi elementi moraju sadržati #PCDATA. To je rezervisana reč DTD-a i skraćenica od *parsed character data* (raščlanjeni znakovni podaci). Drugim recima, #PCDATA označava sirov tekst koji može sadržati reference entiteta kao što su & i <, ali ne oznake niti elemente potomke.

Primer 4. Validan dokument osoba

```
<?xml version="1.0" standalone="no"?>
<!D0CTYPE osoba SYSTEM "http://www.cafeconleche.org/dtds/osoba.dtd">
<osoba>    <ime_i_prezime>    <ime>Alen</ime>    <prezime>Tjuring</prezime>
</ime_i_prezime>
<zanimanje>naučnik u oblasti računarstva</zanimanje>
<zanimanje>matematičar</zanimanje> <zanimanje>kriptograf</zanimanje>
</osoba>
```

Ako se dokument nalazi na istoj osnovnoj lokaciji kao i DTD, možemo zadati relativnu URL adresu umesto apsolutne. Na primer:

```
<!DOCTYPE osoba SYSTEM "/dtd/osoba.dtd">
```

Ukoliko je DTD u istom direktorijumu s dokumentom, možete zadati samo ime datoteke:

```
<!DOCTYPE osoba SYSTEM "osoba.dtd">
```

U slučaju da pišemo svoj prvi DTD, koristiće nam ako u istoj datoteci budemo držali njega i kanonski primerak dokumenta, da bismo mogli da ih istovremeno menjamo i proveravamo. U tom slučaju, u deklaraciju tipa dokumenta upišimo DTD unutar uglastih zagrada, umesto da upućujemo na njega preko spoljne URL adrese. Ilustracija je data u primeru 5.

Primer 5. Validan dokument osoba sa internim DTD-om

```
<?xml version="1.0"?>
<!DOCTYPE osoba [
<!ELEMENT ime     (#PCDATA)>
<!ELEMENT prezime (#PCDATA)>
<!ELEMENT zanimanje       (#PCDATA)>
<!ELEMENT ime_i_prezime (ime, prezime)>
<!ELEMENT osoba   (ime_i_prezime, zanimanje*)>
] >
<osoba>
<ime_i_prezime>
<ime>Alen</ime>
<prezime>Tjuring</prezime>
</ime_i_prezime>
```

```
<zanimanje>naučnik u oblasti računarstva</zanimanje>
<zanimanje>matematičar</zanimanje>
<zanimanje>kriptograf</zanimanje>
</osoba>
```

Neke deklaracije tipa dokumenta sadrže deo deklaracija neposredno, a na druge upućuju pomoću SYSTEM ili PUBLIC identifikatora.

### XML na Web-u

XML je počeo kao pokušaj da se celokupna funkcionalnost i struktura SGML-a prenesu na Web u obliku toliko jednostavnom da ga mogu koristiti i laici. Kao i wecina velikih izuma, ispostavilo se da se XML može upotrebiti na mnogo načina koji daleko prevazilaze ono što su njegovi izumitelji predvideli. Staviše, izvan Weba ima mnogo više XML-a nego na njemu. Bez obzira na to, XML je još uvek veoma privlačan jezik za pisanje i serviranje Web stranica. Pošto XML dokumenti moraju biti dobro oblikovani, jer analizatori moraju odbiti one koji nisu takvi, manje je verovatno da će XML stranice biti nekompatibilne s raznim čitačima. Kako XML dokumenti imaju jasno definisanu strukturu, robotima je mnogo lakše da ih analiziraju. Pošto imena XML elemenata i atributa odražavaju prirodu svog sadržaja, robotima pretraživača je lakše da utvrde značenje XML stranica.

Artist: Bob Dylan
Title: Empire Burlesque
Year: 1985
Country: USA
Company: Columbia
Price: 10.90

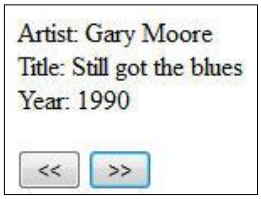
| | |
|---|---|
| Bob Dylan | Empire Burlesque |
| Bonnie Tyler | Hide your heart |
| Dolly Parton | Greatest Hits |
| Gary Moore | Still got the blues |

*Sl. 11.1 Jednostavna XML aplikacija*

cd_catalog_app.html

```
<!DOCTYPE html>
<html>
<head>
<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.open("GET","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
x=xmlDoc.getElementsByTagName("CD");

function displayCDInfo(i)
{
artist=(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
title=(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
year=(x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue);
```

```
country=(x[i].getElementsByTagName("COUNTRY")[0].childNodes[0].nodeValue);
company=(x[i].getElementsByTagName("COMPANY")[0].childNodes[0].nodeValue);
price=(x[i].getElementsByTagName("PRICE")[0].childNodes[0].nodeValue);
txt="Artist: "+artist+"<br>Title: "+title+"<br>Year: "+year+"<br>Country:
"+country+"<br>Company: "+company+"<br>Price: "+price  ;
document.getElementById("showCD").innerHTML=txt;
}
</script>
</head>

<body>
<div id='showCD'>Click on a CD to display album information.</div><br>
<script>
document.write("<table border='1'>");
for (var i=0;i<x.length;i++)
  {
  document.write("<tr onclick='displayCDInfo(" + i + ")'>");
  document.write("<td>");
  document.write(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
  document.write("</td><td>");
  document.write(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
  document.write("</td></tr>");
  }
document.write("</table>");
</script>

</body>
</html>
```




*Sl. 11.2 Listanje XML zapisa*

cd_catalog_nav.html

```
<!DOCTYPE html>
<html>
<head>

<script>
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.open("GET","cd_catalog.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

x=xmlDoc.getElementsByTagName("CD");
i=0;

function displayCD()
{
artist=(x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue);
title=(x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue);
year=(x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue);
txt="Artist: " + artist + "<br>Title: " + title + "<br>Year: "+ year;
```

```
document.getElementById("showCD").innerHTML=txt;
}

function next()
{
if (i<x.length-1)
  {
  i++;
  displayCD();
  }
}

function previous()
{
if (i>0)
  {
  i--;
  displayCD();
  }
}
</script>
</head>
<body onload="displayCD()">
<div id='showCD'></div><br>
<input type="button" onclick="previous()" value="<<" />
<input type="button" onclick="next()" value=">>" />
</body>
</html>
```

### Kaskadni opisi stilova (CSS) i XML

Imena većine elemenata opisuju semantičko značenje njihovog sadržaja. Međutim, taj sadržaj često treba formatirati i prikazati korisnicima. Da bi se to uradilo, mora postojati korak u kome se podaci o formatiranju primenjuju na XML dokument, a semantička obeležja pretvaraju u prezentacijska. Postoji više varijanata sintakse tog prezentacijskog sloja, od kojih su dve naročito vredne pažnje:

- kaskadni opisi stilova (CSS)
- XSL objekti za formatiranje (XSL-FO)

CSS je ne-XML sintaksa za opisivanje izgleda određenih elemenata u dokumentu. CSS je veoma jednostavan jezik; ne obavlja se nikakva transformacija. Raščlanjeni znakovni podaci dokumenta prikazuju se manje-više onako kao što izgledaju u XML dokumentu, mada, razume se, dokument uvek možete da transformišete pomoću XSLT-a i da zatim na njega primenimo CSS opis stilova ukoliko sadržaj dokumenta želimo da preuredimo pre prikazivanja korisniku. CSS opis stilova uopšte ne menja oznake XML dokumenta; on samo primenjuje stilove na postojeći sadržaj.


*Sl. 11.3 Formatiranje XML zapisa*

cd_catalog.css

```
CATALOG
{
background-color: #ffffff;
width: 100%;
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
display: block;
color: #000000;
margin-left: 20pt;
}
```

cd_catalog_with_css.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
	<CD>
		<TITLE>Empire Burlesque</TITLE>
		<ARTIST>Bob Dylan</ARTIST>
		<COUNTRY>USA</COUNTRY>
		<COMPANY>Columbia</COMPANY>
		<PRICE>10.90</PRICE>
		<YEAR>1985</YEAR>
	</CD>
	<CD>
		<TITLE>Hide your heart</TITLE>
		<ARTIST>Bonnie Tyler</ARTIST>
		<COUNTRY>UK</COUNTRY>
		<COMPANY>CBS Records</COMPANY>
		<PRICE>9.90</PRICE>
		<YEAR>1988</YEAR>
	</CD>
	<CD>
		<TITLE>Greatest Hits</TITLE>
		<ARTIST>Dolly Parton</ARTIST>
		<COUNTRY>USA</COUNTRY>
		<COMPANY>RCA</COMPANY>
		<PRICE>9.90</PRICE>
		<YEAR>1982</YEAR>
	</CD>
	<CD>
		<TITLE>Still got the blues</TITLE>
		<ARTIST>Gary Moore</ARTIST>
		<COUNTRY>UK</COUNTRY>
		<COMPANY>Virgin records</COMPANY>
		<PRICE>10.20</PRICE>
		<YEAR>1990</YEAR>
	</CD>
</CATALOG>
```

## XSL objekti za formatiranje

Za razliku od CSS-a, XSL-FO je kompletna XML aplikacija za precizno opisivanje postavke teksta na stranici. On ima elemente za predstavljanje sekvenci stranica, blokova teksta na stranicama, grafike, horizontalnih linija itd. Međutim, XSL-FO se uglavnom ne piše direktno, nego pišemo XSLT opis stilova koji matične oznake našeg dokumenta pretvara u XSL-FO. Aplikacija koja prikazuje dokument, čita XSL-FO i prikazuje ga korisniku.


*Sl. 11.4 XSL farmatiranje*

simplexsl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="simple.xsl" ?>
<breakfast_menu>
	<food>
		<name>Belgian Waffles</name>
		<price>$5.95</price>
		<description>two of our famous Belgian Waffles with plenty of real
maple syrup</description>
		<calories>650</calories>
	</food>
	<food>
		<name>Strawberry Belgian Waffles</name>
		<price>$7.95</price>
		<description>light Belgian waffles covered with strawberries and
whipped cream</description>
		<calories>900</calories>
	</food>
	<food>
		<name>Berry-Berry Belgian Waffles</name>
		<price>$8.95</price>
		<description>light Belgian waffles covered with an assortment of fresh
berries and whipped cream</description>
		<calories>900</calories>
	</food>
	<food>
		<name>French Toast</name>
		<price>$4.50</price>
		<description>thick slices made from our homemade sourdough
bread</description>
		<calories>600</calories>
	</food>
	<food>
		<name>Homestyle Breakfast</name>
		<price>$6.95</price>
		<description>two eggs, bacon or sausage, toast, and our ever-popular
hash browns</description>
```

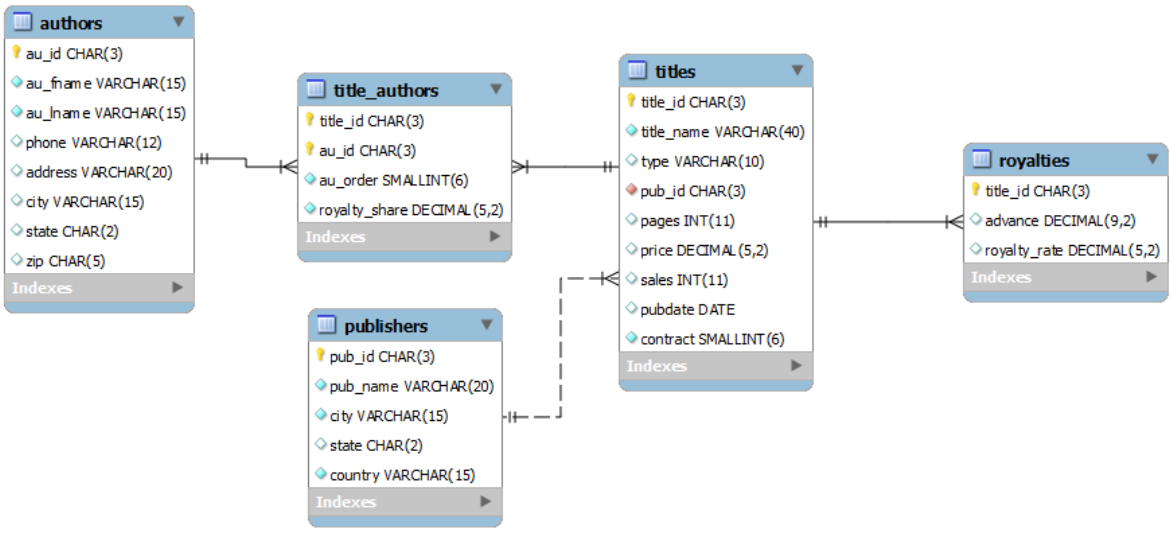
```
            <calories>950</calories>
      </food>
</breakfast_menu>
```

simple.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
    <span style="font-weight:bold"><xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories
per serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

# DODATAK A - Primeri SQL baze podataka

## 1. Books

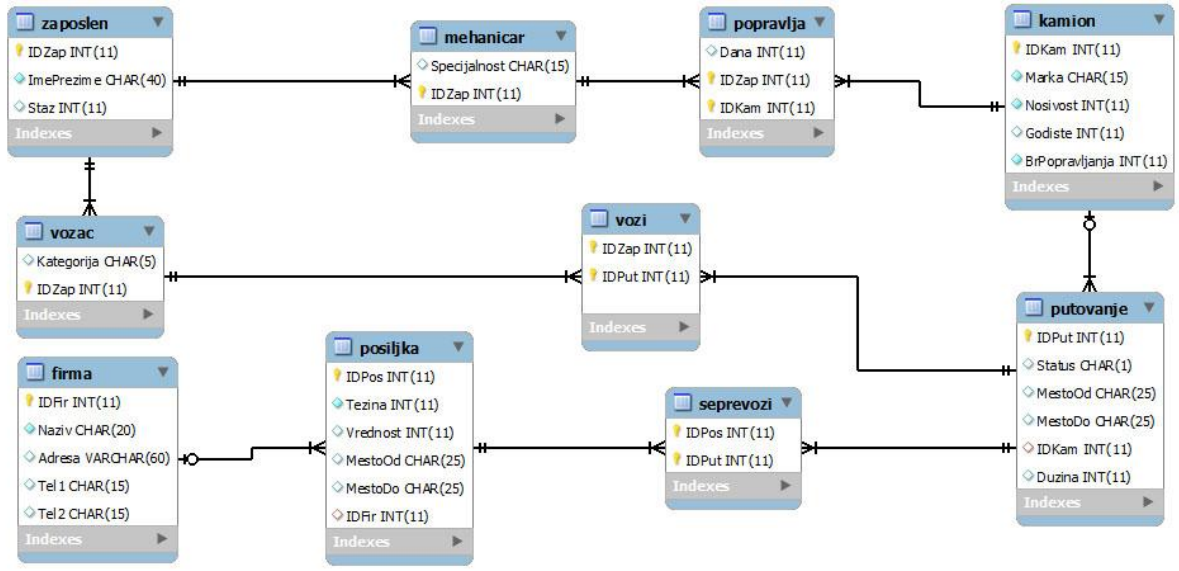



```
CREATE TABLE authors
(
au_id CHAR(3) NOT NULL ,
au_fname VARCHAR(15)NOT NULL ,
au_lname VARCHAR(15)NOT NULL ,
phone VARCHAR(12)NULL ,
address VARCHAR(20)NULL ,
city VARCHAR(15)NULL ,
state CHAR(2) NULL ,
zip CHAR(5) NULL ,
CONSTRAINT authors_pk PRIMARY KEY (au_id)
);

CREATE TABLE publishers
(
pub_id CHAR(3) NOT NULL ,
pub_name VARCHAR(20) NOT NULL ,
city VARCHAR(15) NOT NULL ,
state CHAR(2) NULL ,
country VARCHAR(15) NOT NULL ,
CONSTRAINT publishers_pk PRIMARY KEY (pub_id)
);

CREATE TABLE titles
(
title_id CHAR(3) NOT NULL
PRIMARY KEY ,
title_name VARCHAR(40) NOT NULL,
type VARCHAR(10) ,
pub_id CHAR(3) NOT NULL,
CONSTRAINT title_publisher_fk1
FOREIGN KEY (pub_id)
REFERENCES publishers(pub_id) ,
pages INTEGER ,
price DECIMAL(5,2) ,
sales INTEGER ,
pubdate DATE ,
contract SMALLINT NOT NULL
);

CREATE TABLE title_authors
(
title_id CHAR(3) NOT NULL,
au_id CHAR(3) NOT NULL,
au_order SMALLINT NOT NULL,
```

```
royalty_share DECIMAL(5,2) NOT NULL,
CONSTRAINT title_authors_pk
PRIMARY KEY (title_id, au_id),
CONSTRAINT title_authors_fk1
FOREIGN KEY (title_id)
REFERENCES titles(title_id),
CONSTRAINT title_authors_fk2
FOREIGN KEY (au_id)
REFERENCES authors(au_id)
);

CREATE TABLE royalties
(
title_id CHAR(3) NOT NULL,
advance DECIMAL(9,2) ,
royalty_rate DECIMAL(5,2) ,
CONSTRAINT royalties_pk
PRIMARY KEY (title_id),
CONSTRAINT royalties_title_id_fk
FOREIGN KEY (title_id)
REFERENCES titles(title_id)
);

INSERT INTO authors VALUES('A01','Sarah','Buchman','718-496-7223','75 West 205
St','Bronx','NY','10468');
INSERT INTO authors VALUES('A02','Wendy','Heydemark','303-986-7020','2922 Baseline
Rd','Boulder','CO','80303');
INSERT INTO authors VALUES('A03','Hallie','Hull','415-549-4278','3800 Waldo Ave, #14F','San
Francisco','CA','94123');
INSERT INTO authors VALUES('A04','Klee','Hull','415-549-4278','3800 Waldo Ave, #14F','San
Francisco','CA','94123');
INSERT INTO authors VALUES('A05','Christian','Kells','212-771-4680','114 Horatio St','New
York','NY','10014');
INSERT INTO authors VALUES('A06','','Kellsey','650-836-7128','390 Serra Mall','Palo
Alto','CA','94305');
INSERT INTO authors VALUES('A07','Paddy','O''Furniture','941-925-0752','1442 Main
St','Sarasota','FL','34236');

INSERT INTO publishers VALUES('P01','Abatis Publishers','New York','NY','USA');
INSERT INTO publishers VALUES('P02','Core Dump Books','San Francisco','CA','USA');
INSERT INTO publishers VALUES('P03','Schadenfreude Press','Hamburg',NULL,'Germany');
INSERT INTO publishers VALUES('P04','Tenterhooks Press','Berkeley','CA','USA');

INSERT INTO titles VALUES('T01','1977!','history','P01',107,21.99,566, '2000-08-01',1);
INSERT INTO titles VALUES('T02','200 Years of German Humor','history','P03',14,19.95,9566,
'1998-04-01',1);
INSERT INTO titles VALUES('T03','Ask Your System
Administrator','computer','P02',1226,39.95,25667, '2000-09-01',1);
INSERT INTO titles VALUES('T04','But I Did It
Unconsciously','psychology','P04',510,12.99,13001, '1999-05-31',1);
INSERT INTO titles VALUES('T05','Exchange of Platitudes','psychology','P04',201,6.95,201440,
'2001-01-01',1);
INSERT INTO titles VALUES('T06','How About Never?','biography','P01',473,19.95,11320, '2000-
07-31',1);
INSERT INTO titles VALUES('T07','I Blame My Mother','biography','P03',333,23.95,1500200,
'1999-10-01',1);
INSERT INTO titles VALUES('T08','Just Wait Until After School','children','P04',86,10.00,4095,
'2001-06-01',1);
INSERT INTO titles VALUES('T09','Miss My Yoo-Yoo','children','P04',22,13.95,5000, '2002-05-
31',1);
INSERT INTO titles VALUES('T10','Not Without My Faberge
Egg','biography','P01',NULL,NULL,NULL,0);
INSERT INTO titles VALUES('T11','Perhaps It''s a Glandular
Problem','psychology','P04',826,7.99,94123, '2000-11-30',1);
INSERT INTO titles VALUES('T12','Spontaneous, Not
Annoying','biography','P01',507,12.99,100001, '2000-08-31',1);
INSERT INTO titles VALUES('T13','What Are The Civilian
Applications?','history','P03',802,29.99,10467, '1999-05-31',1);

INSERT INTO title_authors VALUES('T01','A01',1,1.0);
INSERT INTO title_authors VALUES('T02','A01',1,1.0);
INSERT INTO title_authors VALUES('T03','A05',1,1.0);
INSERT INTO title_authors VALUES('T04','A03',1,0.6);
INSERT INTO title_authors VALUES('T04','A04',2,0.4);
INSERT INTO title_authors VALUES('T05','A04',1,1.0);
INSERT INTO title_authors VALUES('T06','A02',1,1.0);
```

```
INSERT INTO title_authors VALUES('T07','A02',1,0.5);
INSERT INTO title_authors VALUES('T07','A04',2,0.5);
INSERT INTO title_authors VALUES('T08','A06',1,1.0);
INSERT INTO title_authors VALUES('T09','A06',1,1.0);
INSERT INTO title_authors VALUES('T10','A02',1,1.0);
INSERT INTO title_authors VALUES('T11','A03',2,0.3);
INSERT INTO title_authors VALUES('T11','A04',3,0.3);
INSERT INTO title_authors VALUES('T11','A06',1,0.4);
INSERT INTO title_authors VALUES('T12','A02',1,1.0);
INSERT INTO title_authors VALUES('T13','A01',1,1.0);

INSERT INTO royalties VALUES('T01',10000,0.05);
INSERT INTO royalties VALUES('T02',1000,0.06);
INSERT INTO royalties VALUES('T03',15000,0.07);
INSERT INTO royalties VALUES('T04',20000,0.08);
INSERT INTO royalties VALUES('T05',100000,0.09);
INSERT INTO royalties VALUES('T06',20000,0.08);
INSERT INTO royalties VALUES('T07',1000000,0.11);
INSERT INTO royalties VALUES('T08',0,0.04);
INSERT INTO royalties VALUES('T09',0,0.05);
INSERT INTO royalties VALUES('T10',NULL,NULL);
INSERT INTO royalties VALUES('T11',100000,0.07);
INSERT INTO royalties VALUES('T12',50000,0.09);
INSERT INTO royalties VALUES('T13',20000,0.06);
```

## 2. Kompanija za prevoz



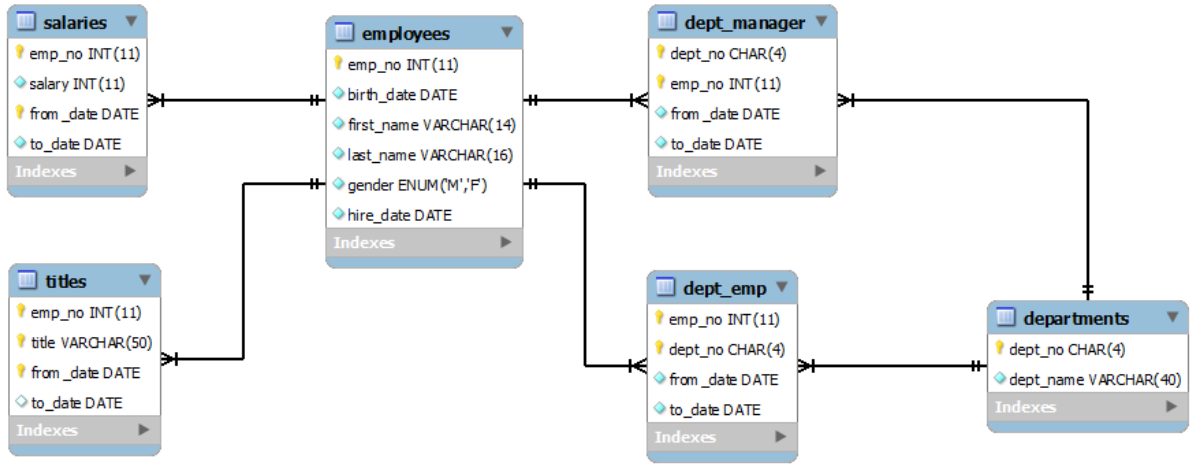

```
CREATE TABLE Kamion (
    IDKam INTEGER NOT NULL,
    Marka CHAR(15) NOT NULL,
    Nosivost INTEGER NOT NULL CHECK (Nosivost>0),
    Godiste INTEGER CHECK (Godiste>1900),
    BrPopravljanja INTEGER DEFAULT 0 NOT NULL,
    PRIMARY KEY (IDKam)
);

CREATE TABLE Zaposlen (
    IDZap INTEGER NOT NULL,
    ImePrezime CHAR(40) NOT NULL,
    Staz INTEGER CHECK (Staz>=0),
    PRIMARY KEY (IDZap)
);

CREATE TABLE Vozac (
    Kategorija CHAR(5),
    IDZap INTEGER NOT NULL,
    PRIMARY KEY (IDZap)
);
```

```
CREATE TABLE Mehanicar (
    Specijalnost CHAR(15),
    IDZap INTEGER NOT NULL,
    PRIMARY KEY (IDZap)
);

CREATE TABLE Firma (
    IDFir INTEGER NOT NULL,
    Naziv CHAR(20) UNIQUE NOT NULL,
    Adresa VARCHAR(60),
    Tel1 CHAR(15),
    Tel2 CHAR(15),
    PRIMARY KEY (IDFir)
);

CREATE TABLE Posiljka (
    IDPos INTEGER NOT NULL,
    Tezina INTEGER NOT NULL,
    Vrednost INTEGER,
    MestoOd CHAR(25),
    MestoDo CHAR(25),
    IDFir INTEGER,
    PRIMARY KEY (IDPos)
);

CREATE TABLE Putovanje (
    IDPut INTEGER NOT NULL,
    Status CHAR(1),
    MestoOd CHAR(25),
    MestoDo CHAR(25),
    IDKam INTEGER,
    Duzina INTEGER,
    PRIMARY KEY (IDPut)
);

CREATE TABLE Popravlja (
    Dana INTEGER,
    IDZap INTEGER NOT NULL,
    IDKam INTEGER NOT NULL,
    PRIMARY KEY (IDZap, IDKam)
);

CREATE TABLE Vozi (
    IDZap INTEGER NOT NULL,
    IDPut INTEGER NOT NULL,
    PRIMARY KEY (IDZap, IDPut)
);

CREATE TABLE sePrevozi (
    IDPos INTEGER NOT NULL,
    IDPut INTEGER NOT NULL,
    PRIMARY KEY (IDPos, IDPut)
);

ALTER TABLE Vozac
    ADD FOREIGN KEY (IDZap) REFERENCES Zaposlen (IDZap) ON DELETE CASCADE ON UPDATE NO ACTION;
ALTER TABLE Mehanicar
    ADD FOREIGN KEY (IDZap) REFERENCES Zaposlen (IDZap) ON DELETE CASCADE ON UPDATE NO ACTION;
ALTER TABLE Posiljka
    ADD FOREIGN KEY (IDFir) REFERENCES Firma (IDFir) ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE Putovanje
    ADD FOREIGN KEY (IDKam) REFERENCES Kamion (IDKam) ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE Popravlja
    ADD FOREIGN KEY (IDZap) REFERENCES Mehanicar (IDZap) ON DELETE NO ACTION;
ALTER TABLE Popravlja
    ADD FOREIGN KEY (IDKam) REFERENCES Kamion (IDKam) ON DELETE CASCADE ON UPDATE NO ACTION;
ALTER TABLE Vozi
    ADD FOREIGN KEY (IDZap) REFERENCES Vozac (IDZap) ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE Vozi
    ADD FOREIGN KEY (IDPut) REFERENCES Putovanje (IDPut) ON UPDATE NO ACTION ON DELETE NO
ACTION;
ALTER TABLE sePrevozi
    ADD FOREIGN KEY (IDPos) REFERENCES Posiljka (IDPos) ON UPDATE NO ACTION ON DELETE NO
ACTION;
ALTER TABLE sePrevozi
    ADD FOREIGN KEY (IDPut) REFERENCES Putovanje (IDPut) ON UPDATE NO ACTION ON DELETE NO
```

```
ACTION;

INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (0,'Mercedes',4,1990,8);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (1,'Mercedes',6,1994,8);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (2,'Mercedes',5,2000,8);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (3,'Mercedes',8,2001,8);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (4,'Mercedes',15,2005,0);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (5,'FAP',5,2002,3);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (6,'FAP',3,2000,4);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (7,'MAN',12,2004,0);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (8,'MAN',10,1999,3);
INSERT INTO Kamion (IDKam,Marka, Nosivost, Godiste, BrPopravljanja)
VALUES  (9,'MAN',8,1997,3);

/*------------vozaci--------------*/
INSERT INTO Zaposlen (IDZap,ImePrezime, Staz)
VALUES (0,'Marko Jagodic',12);
INSERT INTO Vozac (IDZap, Kategorija)
VALUES (0,'BCDE');
INSERT INTO Zaposlen  (IDZap, ImePrezime, Staz)
VALUES (1,'Mihajlo Janjic',4);
INSERT INTO Vozac (IDZap, Kategorija)
VALUES (1,'BC');
INSERT INTO Zaposlen  (IDZap,ImePrezime, Staz)
VALUES (2,'Dragan Milutinovic',6);
INSERT INTO Vozac (IDZap, Kategorija)
VALUES (2,'BC');
INSERT INTO Zaposlen  (IDZap,ImePrezime, Staz)
VALUES (3,'Milos Miljkovic',7);
INSERT INTO Vozac (IDZap, Kategorija)
VALUES (3,'BCD');

/*-----------mehanicari---------------*/
INSERT INTO Zaposlen  (IDZap,ImePrezime, Staz)
VALUES (4,'Dusan Rajic',4);
INSERT INTO Mehanicar (IDZap,Specijalnost)
VALUES (4,'Mercedes');
INSERT INTO Zaposlen  (IDZap,ImePrezime, Staz)
VALUES (5,'Marko Pesic',8);
INSERT INTO Mehanicar (IDZap,Specijalnost)
VALUES (5,'FAP');
INSERT INTO Zaposlen  (IDZap,ImePrezime, Staz)
VALUES (6,'Obrad Zimonjic',7);
INSERT INTO Mehanicar (IDZap,Specijalnost)
VALUES (6,'MAN');
INSERT INTO Zaposlen  (IDZap,ImePrezime, Staz)
VALUES (7,'Drasko Petkovic',8);
INSERT INTO Mehanicar (IDZap,Specijalnost)
VALUES (7,'MAN');

INSERT INTO Firma (IDFir,Naziv, Adresa, Tel1, Tel2)
VALUES (0,'Petrolept','Beograd, Milana Rakica 5','011/258-221','063/223-223');
INSERT INTO Firma (IDFir,Naziv, Adresa, Tel1)
VALUES (1,'Radoje Dakic', 'Podgorica, Njegoseva 10/1','069/123-123');
INSERT INTO Firma (IDFir,Naziv, Adresa, Tel1, Tel2)
VALUES (2,'Tropik', 'Beograd, Dusana Vukasovica 50/1','011/123-1234','064/111-1111');
INSERT INTO Firma (IDFir,Naziv, Adresa, Tel1)
VALUES (3,'Meldovo', 'Beograd, Dusana Kovacevica 15/bb','064/222-3334');
/*Prva posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (0,1900,2000,'Kragujevac','Kraljevo',2);
/*Druga posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (1,5000,10000,'Kragujevac','Kraljevo',1);
/*Treca posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (2,2500,25000,'Beograd','Berlin',0);
```

```
INSERT INTO Putovanje(IDPut,Status,MestoOd,MestoDo,Duzina,IDKam)
VALUES (0,'T','Beograd','Berlin',1500,6);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (2,0);
INSERT INTO vozi(IDZap,IDPut)
VALUES(3,0);
INSERT INTO vozi(IDZap,IDPut)
VALUES(2,0);
/*Cetvrta i peta posiljka (obavljeno u jednom putovanju)*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (3,4150,8000,'Beograd','Sombor',0);
INSERT INTO Putovanje(IDPut,Status,MestoOd,MestoDo,Duzina,IDKam)
VALUES (1,'O','Beograd','Sombor',190,1);
INSERT INTO vozi(IDZap,IDPut)
VALUES(3,1);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (3,1);
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (4,1900,4000,'Beograd','Sombor',2);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (4,1);
/*Sesta posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (5,12000,12000,'Beograd','Subotica',3);
INSERT INTO Putovanje(IDPut,Status,MestoOd,MestoDo,Duzina,IDKam)
VALUES (2,'P','Beograd','Subotica',240,7);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (5,2);
/*Sedma posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (6,7500,8000,'Beograd','Arandjelovac',3);
INSERT INTO Putovanje(IDPut,Status,MestoOd,MestoDo,Duzina,IDKam)
VALUES (3,'O','Beograd','Arandjelovac',80,9);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (6,3);
INSERT INTO vozi(IDZap,IDPut)
VALUES(2,3);
/*Osma posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (7,2900,4500,'Beograd','Leskovac',2);
INSERT INTO Putovanje(IDPut,Status,MestoOd,MestoDo,Duzina,IDKam)
VALUES (4,'O','Beograd','Leskovac',300,5);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (7,4);
INSERT INTO vozi(IDZap,IDPut)
VALUES(0,4);
/*Deveta posiljka*/
INSERT INTO Posiljka(IDPos,Tezina,Vrednost,MestoOd,MestoDo,IDFir)
VALUES (8,9000,11400,'Beograd','Niksic',0);
INSERT INTO Putovanje(IDPut,Status,MestoOd,MestoDo,Duzina,IDKam)
VALUES (5,'O','Beograd','Niksic',460,4);
INSERT INTO sePrevozi(IDPos,IDPut)
VALUES (8,5);
INSERT INTO vozi(IDZap,IDPut)
VALUES(1,5);
INSERT INTO vozi(IDZap,IDPut)
VALUES(3,5);
INSERT INTO Popravlja(IDZap,IDKam,Dana)
VALUES(6,9,3);
INSERT INTO Popravlja(IDZap,IDKam,Dana)
VALUES(7,9,3);
INSERT INTO Popravlja(IDZap,IDKam,Dana)
VALUES(4,0,2);
```

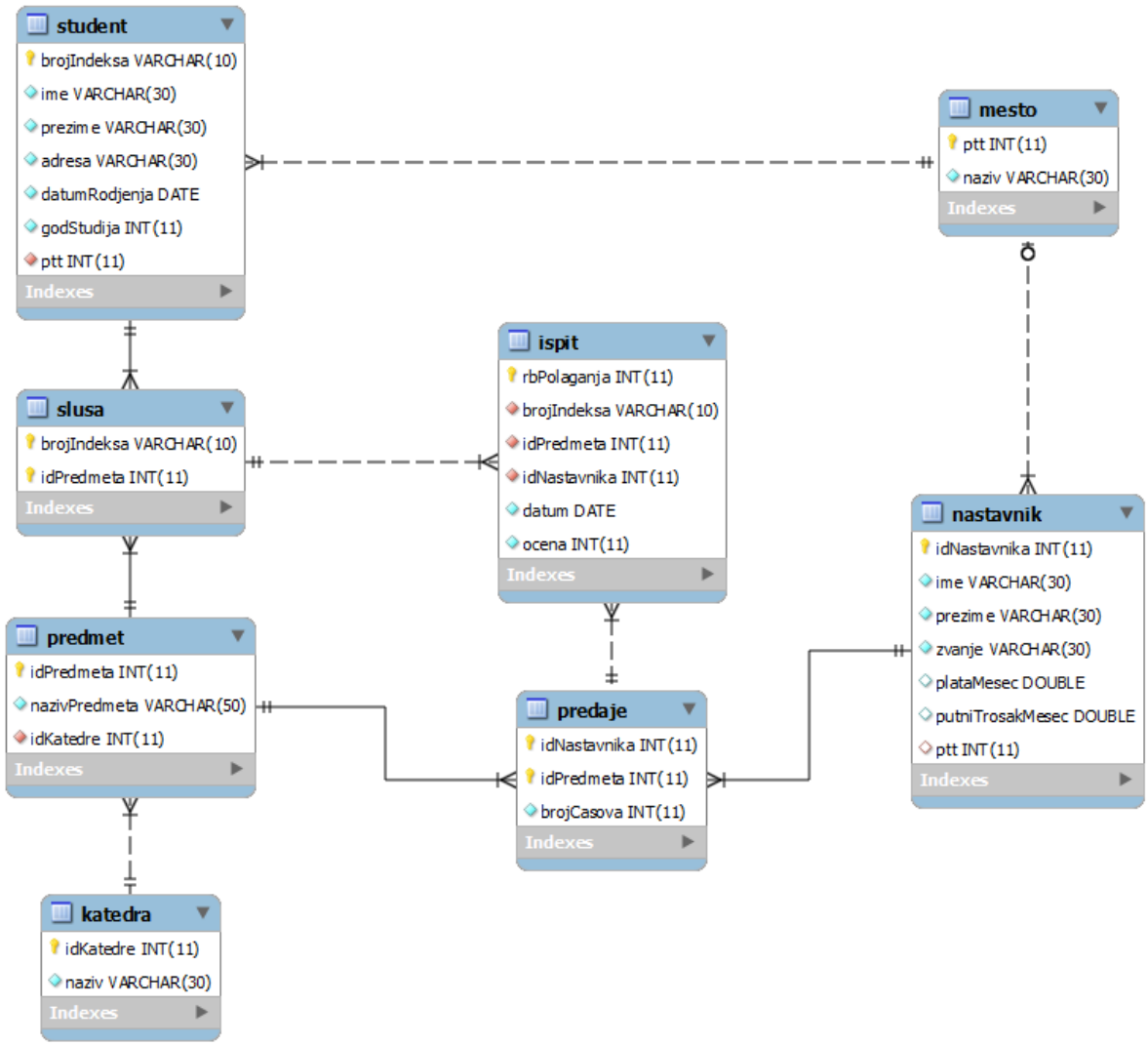
## 3. Employees




```
DROP DATABASE IF EXISTS employees;
CREATE DATABASE IF NOT EXISTS employees;
USE employees;

SELECT 'CREATING DATABASE STRUCTURE' as 'INFO';

DROP TABLE IF EXISTS dept_emp,
                     dept_manager,
                     titles,
                     salaries,
                     employees,
                     departments;

   set storage_engine = InnoDB;

select CONCAT('storage engine: ', @@storage_engine) as INFO;

CREATE TABLE employees (
    emp_no      INT             NOT NULL,
    birth_date  DATE            NOT NULL,
    first_name  VARCHAR(14)     NOT NULL,
    last_name   VARCHAR(16)     NOT NULL,
    gender      ENUM ('M','F')  NOT NULL,
    hire_date   DATE            NOT NULL,
    PRIMARY KEY (emp_no)
);

CREATE TABLE departments (
    dept_no     CHAR(4)         NOT NULL,
    dept_name   VARCHAR(40)     NOT NULL,
    PRIMARY KEY (dept_no),
    UNIQUE  KEY (dept_name)
);

CREATE TABLE dept_manager (
   dept_no      CHAR(4)         NOT NULL,
   emp_no       INT             NOT NULL,
   from_date    DATE            NOT NULL,
   to_date      DATE            NOT NULL,
   KEY          (emp_no),
   KEY          (dept_no),
   FOREIGN KEY (emp_no)  REFERENCES employees (emp_no)    ON DELETE CASCADE,
   FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE CASCADE,
   PRIMARY KEY (emp_no,dept_no)
);

CREATE TABLE dept_emp (
    emp_no      INT             NOT NULL,
    dept_no     CHAR(4)         NOT NULL,
    from_date   DATE            NOT NULL,
    to_date     DATE            NOT NULL,
    KEY         (emp_no),
    KEY         (dept_no),
```

```
    FOREIGN KEY (emp_no)  REFERENCES employees   (emp_no)  ON DELETE CASCADE,
    FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no,dept_no)
);

CREATE TABLE titles (
    emp_no      INT             NOT NULL,
    title       VARCHAR(50)     NOT NULL,
    from_date   DATE            NOT NULL,
    to_date     DATE,
    KEY         (emp_no),
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no,title, from_date)
);

CREATE TABLE salaries (
    emp_no      INT             NOT NULL,
    salary      INT             NOT NULL,
    from_date   DATE            NOT NULL,
    to_date     DATE            NOT NULL,
    KEY         (emp_no),
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no, from_date)
);
```

## 4. Fakultet



```
CREATE TABLE ispit
(
        rbPolaganja int NOT NULL,
        brojIndeksa nvarchar(10) NOT NULL,
        idPredmeta int NOT NULL,
        idNastavnika int NOT NULL,
        datum date NOT NULL,
        ocena int NOT NULL
);

CREATE TABLE katedra
(
        idKatedre int NOT NULL,
        naziv nvarchar(30) NOT NULL
);

CREATE TABLE mesto
(
        ptt int NOT NULL,
        naziv nvarchar(30) NOT NULL
);

CREATE TABLE nastavnik
(
        idNastavnika int NOT NULL,
        ime nvarchar(30) NOT NULL,
        prezime nvarchar(30) NOT NULL,
        zvanje nvarchar(30) NOT NULL,
        plataMesec real NULL,
        putniTrosakMesec real NULL,
        ptt int NULL
);

CREATE TABLE predaje
(
        idNastavnika int NOT NULL,
        idPredmeta int NOT NULL,
        brojCasova int NOT NULL
);

CREATE TABLE predmet
(
        idPredmeta int NOT NULL ,
        nazivPredmeta nvarchar(50) NOT NULL,
        idKatedre int NOT NULL
);

CREATE TABLE  slusa
(
        brojIndeksa nvarchar(10) NOT NULL,
        idPredmeta int NOT NULL
);

CREATE TABLE  student
(
        brojIndeksa nvarchar(10) NOT NULL,
        ime nvarchar(30) NOT NULL,
        prezime nvarchar(30) NOT NULL,
        adresa nvarchar(30) NOT NULL,
        datumRodjenja date  NOT NULL,
        godStudija int NOT NULL,
        ptt int NOT NULL
);

ALTER TABLE ispit ADD CONSTRAINT PK_ispit PRIMARY KEY (rbPolaganja);
ALTER TABLE katedra ADD CONSTRAINT PK_katedra PRIMARY KEY (idKatedre);
ALTER TABLE mesto ADD CONSTRAINT PK_mesto PRIMARY KEY (ptt);
ALTER TABLE nastavnik ADD CONSTRAINT PK_nastavnik  PRIMARY KEY  (idNastavnika);
ALTER TABLE predaje  ADD CONSTRAINT PK_predaje  PRIMARY KEY
(
                idNastavnika , idPredmeta
);

ALTER TABLE  predmet  ADD CONSTRAINT  PK_predmet  PRIMARY KEY
       (
                idPredmeta
```

```
        );

 ALTER TABLE  slusa  ADD CONSTRAINT  PK_slusa  PRIMARY KEY
        (
                brojIndeksa , idPredmeta
        );

 ALTER TABLE  student  ADD CONSTRAINT  PK_student  PRIMARY KEY
        (
                brojIndeksa
        );

ALTER TABLE  predmet  ADD CONSTRAINT  AK_predmet UNIQUE (nazivPredmeta);

ALTER TABLE  predaje  ADD CONSTRAINT  CK_predaje CHECK (brojCasova>0);

ALTER TABLE  ispit  ADD CONSTRAINT  FK_ispit_predaje  FOREIGN KEY
        (
                idNastavnika , idPredmeta
        ) REFERENCES  predaje  ( idNastavnika , idPredmeta );

ALTER TABLE  ispit  ADD CONSTRAINT  FK_ispit_slusa  FOREIGN KEY
        (
                brojIndeksa , idPredmeta
        ) REFERENCES  slusa ( brojIndeksa , idPredmeta );

ALTER TABLE  nastavnik  ADD CONSTRAINT  FK_nastavnik_mesto  FOREIGN KEY
        (
                ptt
        ) REFERENCES  mesto  ( ptt );

ALTER TABLE  predaje  ADD CONSTRAINT  FK_predaje_nastavnik  FOREIGN KEY
        (
                idNastavnika
        ) REFERENCES  nastavnik  ( idNastavnika );

ALTER TABLE  predaje  ADD CONSTRAINT  FK_predaje_predmet  FOREIGN KEY
        (
                idPredmeta
        ) REFERENCES  predmet  ( idPredmeta );

ALTER TABLE  predmet  ADD CONSTRAINT  FK_predmet_katedra  FOREIGN KEY
        (
                idKatedre
        ) REFERENCES  katedra  ( idKatedre );

ALTER TABLE  slusa  ADD CONSTRAINT  FK_slusa_predmet  FOREIGN KEY
        (
                idPredmeta
        ) REFERENCES  predmet ( idPredmeta );

ALTER TABLE  slusa  ADD CONSTRAINT  FK_slusa_student  FOREIGN KEY
        (
                brojIndeksa
        ) REFERENCES  student  ( brojIndeksa );

ALTER TABLE  student  ADD CONSTRAINT  FK_student_mesto  FOREIGN KEY
        (
                ptt
        ) REFERENCES  mesto  ( ptt );
```

# LITERATURA


[1] A. Butsher, "**Naučite MySQL za 21 dan**", Kompjuterska biblioteka, 2003.

[2] A. Davies, „**High Availability MySQL Cookbook**”, Packt Publishing, 2002.

[3] A. Molinaro, „**SQL Cookbook**“, O'Reilly, 2005.

[4] B. Radulović, Z. Kazi, Z. Subić, „**Baze podataka kroz primere i zadatke**“, Tehnički fakultet - Mihajlo Pupin, 2007.

[5] B. Schwartz, P. Zaitsev, V. Tkachenko, J. D. Zawodny, A. Lentz, D. J. Balling „**High Performance MySQL**”, O'Reilly, 2004.

[6] C. Fehily, "**Visual Quickstart Guide: SQL**", Peachpit Press, 2005.

[7] D. Lane, H. E. Williams , „**Web Database Application with PHP and MySQL, 2nd Edition**”, O'Reilly, 2004.

[8] E. R. Harold ,W. S. Means, „**XML in a Nutshell**”, O'Reilly, 2004.

[9] G. Harrison, „**MySQL Stored Procedure Programming**”, O'Reilly, 2006.

[10] G. Smith, „**PostgreSQL 9.0 High Performance**“, Packt Publishing, 2010.

[11] H. E. Williams, D. Lane, „**Web Database Applications with PHP and MySQL**”, O'Reilly, 2002.

[12] I. Petkovics, „**Adatbázisok**“, Szabadkai Műszaki Szakfőiskola, 2013.

[13] L. Gheorghe, H. Hayder, J. P. Maia, „**Smarty PHP Template programming and Application**”, Packt Publishing, 2006.

[14] M. E. Davis, J. A. Phillips, „**Learning PHP and MySQL**”, O'Reilly, 2006.

[15] M. Rochkind, „**Expert PHP and MySQL: Application Design and Development**“, Apress, 2013.

[16] P. Walmsley, „**XQuery**”, O'Reilly, 2007.

[17] R. K. Stephens, R. R. Plew, B. Morgan, J. Perkins, "**Naučite SQL za 21 dan**", Kompjuterska biblioteka, 2004.

[18] S. Suehring, „**MySQL™ Bible**”, Wiley Publishing, 2002.

[19] W. J. Gilmore, „**Beginning PHP and MySQL: From Novice to Professional, Fourth Edition**“, Apress, 2010.