Baze podataka

Autori:
Goran Savić
Milan Segedinac

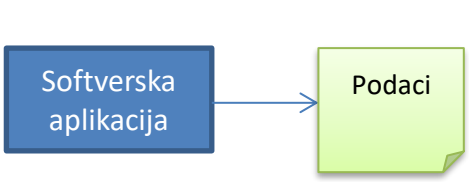
1. Baze podataka

U toku izvršavanja programa, podaci koje program koristi se nalaze u glavnoj memoriji računara. Podaci uskladišteni u glavnoj memoriji više nisu dostupni kada se program završi. Da bi podaci koje je program evidentirao bili dostupni i nakon završetka programa, pa i isključenja računara, neophodno je da obezbedimo dugotrajno skladištenje (perzistenciju) podataka. Podaci se dugotrajno skladište na masovnoj (sekundarnoj) memoriji za šta se trenutno najčešće koristi hard disk.

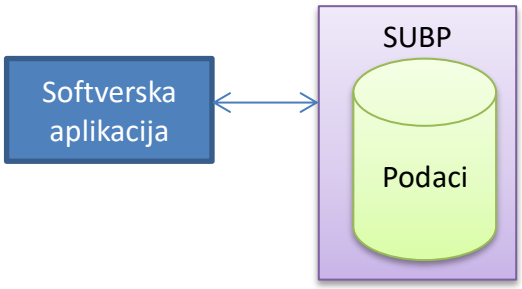
Ranije smo videli da aplikacija može uskladištiti podatke na masovnoj memoriji u fajlove. U takvoj realizaciji skladištenja, aplikacija sama vodi računa o pristupu fajlovima na niskom nivou. Dakle, aplikacija bira format i način skladištenja i odgovorna je da implementira operacije unosa, izmene, brisanja i pretrage podataka u fajlu. Ovakav pristup je vrlo zahtevan za programera aplikacije jer je, u situaciji kada se skladišti velika količina podataka i kada se zahtevaju visoke performanse operacija nad skladištem, realizacija ovih funkcionalnosti komplikovana. Na primer, u pomenutom pristupu programer mora da reši problem efikasnog pronalaženja određenog podatka u fajlu, čak i kada fajl sadrži milione podataka. Obzirom na zahtevnost samostalne realizacije podsistema za skladištenje, u praksi se aplikacije najčešće oslanjaju na standardne modele organizacije podataka, kao i na specijalizovane softverske aplikacije zadužene za realizaciju operacija nad uskladištenim podacima.

Kada je reč o načinu organizacije uskladištenih podataka, organizovanu kolekciju podataka nazivamo **baza podataka**. Pomenimo da koncept baza podataka nije nužno vezan za dugotrajno skladištenje. Baza podataka može organizovano skladištiti podatke i isključivo privremeno u glavnoj memoriji. Takve baze podataka nazivamo *in memory* baze podataka. U nastavku je naš fokus postavljen na rad sa bazama podataka tako da se omogući dugotrajno skladištenje.

Koncept baza podataka nam omogućuje da podatke uskladištimo na organizovan način i time dobijemo efikasniji podsistem za skladištenje. Do sada je razvijen značajan broj specijalizovanih softverskih aplikacija koje obavljaju funkcije skladištenja podataka korišćenjem baze podataka. Ovakve aplikacije nazivamo **sistem za upravljanje bazom podataka** (SUBP) (eng. *Database Management System* – DBMS). Dakle, za realizaciju operacija nad skladištem podataka, moguće je koristiti SUBP i time osloboditi aplikaciju odgovornosti da sama realizuje ovu funkcionalnost. U toj varijanti, aplikacija funkcionalnosti vezane za pristup podacima u dugotrajnom skladištu delegira sistemu za upravljanje bazom podataka. Aplikacija ne mora da se bavi internom realizacijom skladištenja. Dovoljno je da postoji dogovoren protokol komunikacije kojim aplikacija može da uputi zahtev SUBP-u i da od njega dobije određeni rezultat. Na slici je ilustrovana razlika u arhitekturi aplikacije koja samostalno vrši perzistenciju naspram aplikacije koja se oslanja na SUBP.



Samostalna perzistencija korišćenjem fajlova



Perzistencija korišćenjem SUBP

Vidimo da u slučaju korišćenja SUBP aplikacija ne pristupa direktno uskladištenim podacima, već samo komunicira sa SUBP. Treba naglasiti da se i interna realizacija skladištenja u bazi podataka od strane SUBP najčešće realizuje korišćenjem fajlova. Ipak odgovornost za realizaciju samog pristupa fajlovima, izbora formata fajla i načina organizacije podataka preuzima SUBP umesto sama aplikacija.

Obzirom da baza podataka predstavlja organizovanu kolekciju podataka, podrazumeva se da je organizaciju podataka moguće izvršiti na različite načine. Tako postoje različiti tipovi baza podataka. Najzastupljeniji tipovi baza podataka i njihovi podtipovi su:

- Relacione baze podataka
- Objektne baze podataka
- Dokument-orijentisane baze podataka
    - XML-bazirane baze podataka
    - JSON-bazirane baze podataka

U nastavku teksta posvećeni smo relacionim bazama podataka, kao najčešćoj organizaciji podataka u informacionim sistemima. Iako nisu tema ovog teksta, ne treba gubiti iz vida dokument-orijentisane baze podataka, posebno njihov podtip koji skladišti dokumente u JSON formatu, obzirom da je ovaj tip skladištenja podataka sve dominantniji u određenim scenarijima.

2. Relacione baze podataka

Relacione baze podataka organizuju podatke u skladu sa relacionim modelom podataka. Prema ovom modelu, podaci se organizuju u tabele koje se sastoje od redova i kolona. Između tabela uspostavljaju se relacije. Tabele koje skladište srodne i povezane podatke SUBP organizuje u bazu podataka. Specifikacija tabela od kojih se baza podataka sastoji nazivamo šema baze podataka. Jedan SUBP sadrži više baza podataka.

Do sada je razvijen veliki broj različitih relacionih SUBP. Od toga neki su komercijalni, a neki besplatni. Besplatni su često i otvorenog koda. Neki od popularnih SUBP su:

- Besplatni
    - MySQL
    - PostgreSQL
- Komercijalni
    - Oracle Database
    - Microsoft SQL Server

Princip organizacije podataka kod svih pomenutih sistema je isti. Takođe, sa svim relacionim SUBP se komunicira na standardizovan način. Tako da je nastavak teksta primenljiv na bilo koji relacioni SUBP. U situacijama kada predstavljamo detalje koji su specifično realizovani za različite SUBP, izabrali smo realizaciju koju koristi MySQL SUBP.

Pogledajmo sada kako se prema relacionom modelu skladište podaci u tabelama. Svaka tabela predstavlja jedan tip entiteta i skladišti podatke tog tipa. Tako se na primer, u jednoj tabeli skladište podaci o državama, a u drugoj tabeli podaci o gradovima. Svaki entitet je predstavljen jednim redom u tabeli, a podaci konkretnog entiteta se skladište u ćelijama tog reda u skladu sa kolonama

definisanim u tabeli. Slika ilustruje skladištenje podataka o studentima u tabeli relacione baze podataka.

Broj indeksa	Ime	Prezime
9254	Milan	Petrić
9344	Jelena	Zorić
9356	Milan	Ilić

Prema terminologiji relacionih baza podataka, redove nazivamo slogovi tabele. Kolone nazivamo polja tabele. U zaglavlju prikazane tabele navedeni su nazivi polja. Informacije o poljima tabele (naziv, tip vrednosti koji se može upisati u polje, ...) nazivamo šema tabele.

3. Primarni ključ

Od svih polja koje slog sadrži, treba da postoji polje koje jednoznačno identifikuje taj slog u odnosu na sve ostale slogove u tabeli. To znači da u tabeli treba da postoji kolona u kojoj je za svaki slog upisana vrednost drugačija od vrednosti u ostalim slogovima u toj koloni. U prikazanom primeru sa tabelom studenata, polje *Broj indeksa* je takvo polje. Moguće je i da kombinacija vrednosti više polja jednoznačno identifikuje slog. Na primer, redni broj studenta i godina upisa pojedinačno nisu jedinstveni, ali ne postoje dva studenta koji imaju isti i redni broj i godinu upisa. Dakle, broj studenta i godina upisa u kombinaciji jednoznačno identifikuju studenta.

Kolona (ili kombinacija kolona) tabele relacione baze podataka koja jedinstveno identifikuje svaki slog tabele se naziva **primarni ključ** tabele (eng. *primary key*). Primarni ključ tabele mora da sadrži jedinstvenu vrednost za svaki slog tabele. Dakle, dva sloga tabele ne mogu imati upisanu istu vrednost za polje koje je primarni ključ tabele. Kada primarni ključ čini kombinacija više kolona tabele, takav primarni ključ nazivamo **kompozitni primarni ključ**.

Dakle, neophodno je da tabela ima primarni ključ. U primeru sa tabelom studenata je očigledno da je broj indeksa odgovarajući izbor za primarni ključ. Kada među podacima koje evidentiramo o entitetu postoji podatak čije su vrednosti takve da su jedinstvene za svaki slog, takav primarni ključ nazivamo **prirodni primarni ključ**. Broj indeksa u studentu je takav primarni ključ. Ukoliko takav podatak ne postoji, potrebno je uvesti kolonu u tabelu čija je svrha da skladišti vrednosti koje jedinstveno identifikuju svaki slog. Za razliku od drugih podataka koje evidentiramo o entitetu, ovo polje ne predstavlja neku osobinu entiteta, već je reč o veštački uvedenom polju da bi se omogućila jednoznačna identifikacija sloga. Kada ovakvu kolonu uvedemo i postavimo za primarni ključ tabele, takva kolona se naziva **surogat primarni ključ**. Obzirom da vrednosti u koloni koja je surogat primarni ključ nemaju posebno značenje, već imaju za cilj samo da jedinstveno identifikuju slog, za ovu kolonu bi mogli izabrati bilo koje vrednosti proizvoljnog tipa koje će obezbediti jedinstvenu identifikaciju. Zbog jednostavnosti i efikasnosti najčešće se kao surogat primarni ključ definiše kolona sa celobrojnim vrednostima, pri čemu vrednosti u slogovima uzimaju redom prirodne brojeve (1, 2, 3, 4, 5, ...). Obzirom na ovo jednostavno pravilo određivanja vrednosti primarnog ključa za sledeći slog, sistemi za upravljanje bazom podataka mogu automatski da dodele slogu vrednost primarnog

ključa uzimajući sledeći prirodan broj. U tom slučaju, reč je o *auto increment* surogat primarnom ključu.

Iako nije neophodno uvoditi surogat primarni ključ kada postoji kolona koja može da bude prirodni primarni ključ, česta je praksa da se surogat ključ ipak uvede. Naime, konzistentno korišćenje *auto increment* surogat primarnog ključa u svim tabelama baze podataka omogućuje uniforman rad sa primarnim ključevima u svim tabelama. Takođe, kod ovog tipa ključa, SUBP automatski određuje vrednost ključa koja je jedinstvena čime se pojednostavljuje određivanje ključa za nove slogove. Očigledna mana uvođenja surogat ključa čak i kada to nije neophodno je što skladištenje vrednosti kolone koja je surogat ključ dodatno zauzima memoriju. Ipak, obzirom na prednosti koje donosi uvođenje surogat ključa i relativno nisku cenu masovne memorije po jedinici skladištenja, zauzimanje dodatne memorije za skladištenje još jednog celog broja po slogu nije značajan gubitak.

4. Strani ključ

Ranije smo naveli da relacioni model predviđa i uspostavljanje relacija između tabela. Relacije između tabela se uvode u situaciji kada je određeni entitet, osim podacima uskladištenim u tabeli koja predstavlja taj entitet, opisan i podacima koji su definisani u nekoj drugoj tabeli. Tada je potrebno specificirati da se određeni podaci vezani za slog nalaze u drugoj tabeli. Pojasnimo to primerom. Ranije smo videli tabelu koja definiše podatke o studentima. Uzmimo da baza podataka sadrži i drugu tabelu u kojoj su podaci o gradovima. Zbog lakšeg razumevanja, ove dve tabele su ilustrovane na slici.

Broj indeksa	Ime	Prezime
9254	Milan	Petrić
9344	Jelena	Zorić
9356	Milan	Ilić

Ptt broj	Naziv
21000	Novi Sad
11000	Beograd

Vidimo da je u ovom primeru grad opisan ptt brojem i nazivom. Primarni ključ tabele sa spiskom gradova je polje ptt broj. Kako bismo sada specificirali u kojem gradu student živi? Kako da, na primer, evidentiramo informaciju da student Milan Petrić živi u Novom Sadu?

Jedno moguće rešenje je da tabelu student proširimo dodatnom kolonom u kojoj bismo evidentirali naziv grada u kojem student živi. Ovo rešenje je ilustrovano narednom slikom.

Broj indeksa	Ime	Prezime	Prebivalište
9254	Milan	Petrić	Novi Sad
9344	Jelena	Zorić	Beograd
9356	Milan	Ilić	Novi Sad

Prikazano rešenje ima više nedostataka. Novouvedena kolona je obična tekstualna kolona u koju je moguće upisati bilo kakav tekst. To znači da je moguće evidentirati da student živi u gradu koji uopšte nije evidentiran u spisku gradova. Dakle, prebivalište nije ograničeno na skup postojećih gradova („postojećih“ u smislu gradova koje aplikacija ima u svojoj evidenciji). Takođe, obzirom da kolona dopušta unos slobodnog teksta, moguće je i greškom uneti netačno ime grada. Tada se, na primer, pri pretrazi studenata koji žive u Beogradu ne bi pronašli oni studenti kod kojih je naziv Beograd neispravno unesen (npr. ima slovna greška u nazivu).

Iz pomenutih razloga, potrebno je drugačije rešenje u kojem je kao prebivalište studenta moguće naznačiti samo neki od gradova koji već postoji evidentiran u spisku gradova. Drugim rečima, potrebno je napraviti vezu između evidencije studenata i evidencije gradova. Ovo je ilustrovano na sledećoj slici.

Broj indeksa	Ime	Prezime
9254	Milan	Petrić
9344	Jelena	Zorić
9356	Milan	Ilić

Ptt broj	Naziv
21000	Novi Sad
11000	Beograd

Vidimo da student sada „ukazuje“ na grad u kojem živi. Obzirom da se podaci u relacionoj bazi podataka moraju evidentirati u nekoj od kolona tabele, treba da vidimo kako da na nivou baze podataka realizujemo vezu koja je na slici ilustrovana strelicom.

Ovo se vrši uvođenjem novog polja u tabelu studenata. U tom polju je potrebno evidentirati grad u kojem student živi. Tačnije, potrebno je evidentirati podatak koji jednoznačno identifikuje grad u kojem student živi. Podatak koji jednoznačno identifikuje neki grad u tabeli gradova je upisan u koloni koja je primarni ključ te tabele, a to je polje ptt broj. Za prebivalište studenta se definiše ograničenje da se kao prebivalište mora uneti identifikator nekog od gradova iz evidencije. Dakle, u trenutnom primeru moguće je uneti samo brojeve 21000 i 11000, jer su to dve vrednosti koje postoje u tabeli gradova u polju ptt broj, koje je primarni ključ u tabeli gradova. Ovo rešenje je ilustrovano na slici.

Broj indeksa	Ime	Prezime	Prebivalište
9254	Milan	Petrić	21000
9344	Jelena	Zorić	11000
9356	Milan	Ilić	21000

U opštem slučaju, veza između tabela se ostvaruje time što se u jednoj tabeli definiše kolona koja može da sadrži samo vrednosti koje su primarni ključ slogova iz druge tabele. Time ova kolona faktički referencira slog iz druge tabele. Takva kolona se naziva **strani ključ** ili spoljni ključ (eng. *foreign key*). Korišćenje stranog ključa obezbeđuje da je moguće referencirati samo neki od slogova

koji postoje u drugoj tabeli. Ovaj koncept uspostavljanja konzistentnih podataka pri povezivanju tabela se naziva **referencijalni integritet** (eng. *referential integrity*).

Podrazumeva se da, u opštem slučaju, tip vrednosti u koloni koja je strani ključ mora da bude isti kao tip vrednosti u koloni koja je primarni ključ u tabeli na koju strani ključ ukazuje. Konkretno, u primeru sa prebivalištem studenta reč je bila o celobrojnom tipu, obzirom da je ptt grada broj. Spomenimo ovde još i da je moguće definisati strani ključ koji se sastoji od više kolona. Ovo se koristi u slučaju kada se referencira tabela koja ima kompozitni primarni ključ.

5. SQL jezik

U prethodnom delu teksta smo objasnili osnovne koncepte relacionih baza podataka. Rekli smo da je komunikacija sa relacionim SUBP standardizovana. Ovo je obezbeđeno korišćenjem SQL jezika (eng. *structured query language*). U nastavku ćemo prikazati osnove ovog jezika.

SQL jezik se sastoji od:

- jezika za definisanje podataka (eng. *data definition language*) – DDL
- jezika za manipulaciju podacima (eng. *data manipulation language*) – DML
- jezika za kontrolu pristupa podacima (eng. *data control language*) - DCL

Definisanje podataka

Data Definition Language (DDL) je podskup SQL jezika koji omogućuje definisanje strukture podataka. U nastavku ćemo opisati glavne konstrukte DDL.

SQL jezik omogućuje kreiranje nove baze podataka u koju ćemo smestiti tabele. Prikazana je SQL naredba za kreiranje nove baze podataka na primeru baze podataka pod nazivom studentskasluzba.

```
create database studentskasluzba;
```

Postojeća baza podataka se može obrisati naredbom `drop database`, za koju je primer korišćenja prikazan u nastavku.

```
drop database studentskasluzba;
```

Nakon kreiranja baze podataka, moguće je izvršavati SQL upite nad tom bazom podataka. Za početak, izvršićemo upit kreiranja jedne nove tabele u bazi podataka. Pre upita nad bazom podataka, potrebno je naznačiti koju bazu podataka koristimo među svim bazama kreiranim u okviru tog SUBP. Drugim rečima, potrebno je da naznačimo na koju bazu podataka će se svi naredni upiti odnositi, kako bismo izbegli potrebu da pri svakom SQL upitu navodimo na koju se bazu podataka odnosi. Prikazana je naredba `use` kojom specificiramo koju bazu podataka ćemo koristiti.

```
use studentskasluzba;
```

Sada možemo da kreiramo novu tabelu u bazi podataka. Sintaksu kreiranja nove tabele ćemo pokazati na primeru kreiranja ranije ilustrovane tabele koja evidentira gradove. Dat je SQL kod za kreiranje tabele `grad` koja sadrži polja `ptt_broj` i `naziv`.

```
create table grad (
      ptt_broj integer,
```

```
      naziv varchar(100),
      primary key (ptt_broj)
);
```

Vidimo da je pri kreiranju tabele potrebno navesti njen naziv i polja koja sadrži. Kao što smo ranije objasnili, tabela mora da ima primarni ključ. U prikazanom primeru je specificirano da je polje `ptt_broj` primarni ključ tabele `grad`. Za svako polje se navodi naziv i tip. U tabeli su navedeni najvažniji tipovi podataka u SQL jeziku.

Naziv tipa	Opis
INTEGER	Celi broj
BIGINT	Veliki celi broj. Načešće se koristi kao tip podatka za *auto increment* surogat primarni ključ, obzirom da tabela može imati veliki broj slogova
DECIMAL(M,N)	Decimalni broj. *M* predstavlja ukupan broj cifara, a *N* broj decimalnih mesta. Npr. u polje tipa DECIMAL(6,2) možemo smestiti broj koji ima 4 cifre pre decimalne tačke i 2 nakon tačke
BIT	Logička vrednost. Moguće vrednosti su 0 ili 1
CHAR(M)	String fiksne dužine od M karaktera
VARCHAR(M)	String promenljive dužine, između 1 i M karaktera. Maksimum je 255 karaktera.
DATETIME	Vrednost koja sadrži datum i vreme u formatu YYYY-MM-DD HH:MM:SS. Na primer 2017-05-14 17:36:27. Postoje i tipovi DATE i TIME, koji skladište samo datum, odnosno vreme
TEXT	Tekst od najviše 65535 karaktera
BLOB	Binarni podatak. U ovakvo polje možemo da smestimo neki binarni dokument. Na primer sliku.

Treba naglasiti da ovo svakako nisu svi podržani SQL tipovi podataka. Takođe, za mnoge tipove postoje sinonimi, pa je moguće koristiti prikazane tipove kroz drugačije nazive (npr. INT umesto INTEGER).

Postojeća tabela se uklanja iz baze podataka naredbom drop table koja je ilustrovana narednim primerom.

```
drop table grad;
```

U primeru sa kreiranjem tabele grad smo videli definisanje klasičnog primarnog ključa. Naredni primer pokazuje MySQL sintaksu za definisanje *auto increment* primarnog ključa.

```
create table grad (
      id integer auto_increment,
      ptt_broj integer,
      naziv varchar(100),
      primary key (id)
);
```

Dakle, uveli smo novo polje id koje vrednosti dobija po *auto increment* principu i proglasili ga primarnim ključem tabele.

Pokazaćemo još i sintaksu za definisanje kompozitnog primarnog ključa na primeru tabele koja evidentira studente. Primarni ključ se formira kao kombinacija godine upisa i rednog broja studenta.

```
create table student (
      godina_upisa integer,
```

```
      redni_broj integer,
      ime varchar(50),
      prezime varchar(50),
      primary key (godina_upisa, redni_broj)
);
```

Pogledajmo sada kako možemo definisati strani ključ pri kreiranju tabele. Ilustrovaćemo to kroz ranije pokazani primer evidencije studenata koji imaju prebivalište predstavljeno putem stranog ključa na tabelu gradova. Prikazan je SQL kod za kreiranje tabele studenata.

```
create table student (
      broj_indeksa varchar(10),
      ime varchar(50),
      prezime varchar(50),
      prebivaliste integer,
      primary key (broj_indeksa),
      foreign key (prebivaliste) references grad(ptt_broj)
);
```

Vidimo da je informacija o prebivalištu uskladištena u standardno integer polje. Zatim je definisano da to polje referencira neku od vrednosti iz polja `ptt_broj` u tabeli `grad`.

Ranije smo objasnili da uspostavljanjem stranih ključeva baza obezbeđuje referencijalni integritet. Dakle, vrednosti u jednoj tabeli uvek ukazuju na neku od postojećih vrednosti iz druge tabele. Postavlja se pitanja šta će se desiti ako želimo da obrišemo slog koji je referenciran od strane sloga u nekoj drugoj tabeli. Na primer, pokušavamo da obrišemo grad koji je prebivalište određenog broja studenata. To zavisi od kaskadne strategije koju izaberemo pri definisanju stranog ključa. Naime, moguće je definisati kako se akcije nad slogom odnose na slogove koji taj slog referenciraju putem stranog ključa. Sledeće varijante su dostupne:

- CASCADE – kaskadno se primenjuje akcija nad povezanim slogovima. Dakle, brisanje sloga će inicirati i brisanje svih slogova koji ga referenciraju
- RESTRICT – biće onemogućena akcija ukoliko ima slogova koji referenciraju slog. Konkretno, SUBP neće izvršiti brisanje sloga ako postoje slogovi koji ga referenciraju. Ovo je podrazumevana kaskadna strategija.
- SET NULL – biće dozvoljena akcija nad slogom, a u svim slogovima koji referenciraju taj slog će u vrednost koja je strani ključ biti upisana null vrednost

Dat je primer veze između studenta i grada, pri čemu će pri brisanju grada biti obrisani svi studenti koji imaju prebivalište u tom gradu.

```
create table student (
      broj_indeksa varchar(10),
      ime varchar(50),
      prezime varchar(50),
      prebivaliste integer,
      primary key (broj_indeksa),
      foreign key (prebivaliste) references grad(ptt_broj)
            on delete cascade
);
```

Vidimo da je kod stranog ključa navođenjem `on delete cascade` definisano kaskadno brisanje.

U prethodnim primerima smo videli da se pri definisanju polja navodi tip polja. Osim tipa polja, moguće je definisati i dodatne informacije o polju. Jedna od informacija je da li je obavezno da slog sadrži vrednost za to polje. Drugim rečima, možemo označiti da li je polje obavezno. Ako polje ne sadrži vrednost, sadržaj polja je null. Dakle, možemo da specificiramo da li je dozvoljen unos null vrednosti u polje. Ukoliko su null vrednosti dozvoljene, za polje se navodi oznaka `null`. Ovo je i podrazumevano ponašanje ako se nikakva oznaka ne navede. Ukoliko nije dozvoljen unos null vrednosti u polje, potrebno je postaviti oznaku `not null`. Ovo je ilustrovano na primeru tabele gradova u kojoj je navođenje naziva grada obavezno.

```
create table grad (
      ptt_broj integer,
      naziv varchar(100) not null,
      primary key (ptt_broj)
);
```

Ranije smo objasnili da ako neko polje proglasimo primarnim ključem, tada je vrednost u polju različita za sve redove u tabeli. Bez obzira da li je polje primarni ključ, možemo ovakvo ponašanje specificirati nad poljem korišćenjem oznake `unique`. Korišćenje ključne reči `unique` je ilustrovano na primeru tabele gradova u kojoj nije dozvoljen unos dva grada sa istim nazivom.

```
create table grad (
      ptt_broj integer,
      naziv varchar(100) unique,
      primary key (ptt_broj)
);
```

Jednom definisanu šemu tabele moguće je kasnije izmeniti naredbom `alter table`. Dat je primer naknadne izmene šeme ranije prikazane tabele `grad`, tako da polje naziv umesto inicijalno definisanih maksimalno 100 karaktera može da sadrži maksimalno 150 karaktera.

```
alter table grad modify naziv varchar(150);
```

Slično, u nastavku je dat SQL kod za naknadno ubacivanje, odnosno izbacivanje kolone iz tabele.

```
alter table grad add broj_stanovnika integer;
```

```
alter table grad drop column broj_stanovnika;
```

Manipulacija podacima

Data Manipulation Language (DML) je podskup SQL jezika koji omogućuje rukovanje podacima koji se skladište u tabelama baze podataka.

Najpre ćemo objasniti kako možemo korišćenjem SQL jezika dodati novi slog u tabelu baze podataka. Ovo vršimo naredbom insert, čije je korišćenje ilustrovano u narednom primeru u kojem se u tabelu grad ubacuje novi slog sa podacima o Novom Sadu, čiji je ptt broj 21000.

```
insert into grad (ptt_broj, naziv) values (21000, 'Novi Sad');
```

U upitu se najpre navodi ime tabele, a zatim spisak polja za koje ćemo specificirati vrednosti koje želimo da upišemo u polja. Jasno je da polja koja su označena sa `not null` ne smemo da izostavimo jer je za njih obavezno da navedemo vrednost. Nakon toga, navode se konkretne vrednosti koje će biti upisane u polja sloga. Treba primetiti da je string vrednosti neophodno navesti unutar jednostrukih navodnika. Isto važi i za vrednosti koje predstavljaju datum i vreme.

Sada ćemo analizirati preuzimanje podataka iz baze podataka. Zbog lakšeg razumevanja, uzmimo da u tabeli grad postoji uneseno 5 gradova. Sadržaj tabele je prikazan u nastavku.

Ptt_broj	Naziv
21000	Novi Sad
11000	Beograd
18000	Niš
24000	Subotica
34000	Kragujevac

Preuzimanje podataka iz tabele se vrši naredbom SELECT. Dat je primer korišćenja ove naredbe.

```
select ptt_broj, naziv from grad;
```

Prikazani upit preuzima sve slogove tabele `grad`. Za svaki slog preuzimaju se vrednosti iz polja `ptt_broj` i `naziv`. Rezultat upita je isti kao gore prikazani sadržaj tabele gradova.

Ukoliko preuzimamo vrednosti svih polja, nije neophodno da poimence navedemo sva polja. Umesto toga, možemo koristiti džoker znak `*`. Naredni upit je ekvivalentan prethodno prikazanom i preuzima sva polja iz redova tabele `grad`.

```
select * from grad;
```

Moguće je izvršiti izbor polja koja će se naći u rezultatu upita. Na primer, prikazan je upit koji preuzima samo nazive gradova.

```
select naziv from grad;
```

Rezultat ovog upita prikazan je u nastavku.

Naziv
Novi Sad
Beograd
Niš
Subotica
Kragujevac

Dva prikazana `select` upita vraćaju kao rezultat sve slogove tabele `grad`. Moguće je izvršiti i izbor slogova koji će se naći u rezultatu upita. Ovo vršimo `where` klauzulom `select` upita. Pogledajmo kako možemo preuzeti samo gradove čiji je ptt broj manji od 20000.

```
select * from grad where ptt_broj < 20000;
```

Rezultat upita dat je u tabeli.

Ptt_broj	Naziv
11000	Beograd
18000	Niš

Vidimo da su se u rezultati našli samo oni slogovi tabele koji ispunjavaju kriterijum definisan u `where` klauzuli.

Kao kriterijum izbora smo koristili relacioni operator `<`. U tabeli su navedeni ostali često korišćeni relacioni operatori u SQL jeziku.

Operator	Opis
=	Poredi da li su dva operanda jednaka
>	Poredi da li je prvi operand veći od drugog
<	Poredi da li je prvi operand manji od drugog
>=	Poredi da li je prvi operand veći ili jednak drugom
<=	Poredi da li je prvi operand manji ili jednak drugom
<>	Poredi da li su dva operanda različita
LIKE	Poredi string sa šablonom koji može da sadrži džoker znake. Džoker znaci su `*` (označava nula ili više bilo kojih karaktera) i _ (označava tačno jedan bilo koji karakter). Na primer, izraz `naziv like 'N%'` će biti istinit za svaki grad čiji naziv počinje na N.
IN()	Proverava da li se operand nalazi u skupu vrednosti razdvojenih zarezom. Na primer, izraz `naziv in ('Novi Sad', 'Beograd')` će biti istinit za gradove Novi Sad i Beograd.
IS NULL	Operator vraća true kada operand ima vrednost null
IS NOT NULL	Operator vraća true kada operand nema vrednost null

Moguće je koristiti i logičke operatore i time povezati više logičkih izraza. U tabeli su dati SQL simboli za standardne logičke operatore.

Operator	Opis
AND	Konjukcija
OR	Disjunkcija
NOT	Negacija

Dat je primer jednog složenijeg SQL upita koji koristi neke od prikazanih relacionih i logičkih operatora.

```
select * from grad where ptt_broj < 15000 OR
      (ptt_broj > 20000 AND naziv like '%c%');
```

Prikazani upit vraća gradove čiji je ptt broj manji od 15000 ili imaju ptt broj veći od 20000 i naziv im sadrži slovo c. Ti gradovi su prikazani u tabeli.

Ptt_broj	Naziv
11000	Beograd
24000	Subotica
34000	Kragujevac

Za slogove koje upit vraća možemo definisati kriterijum sortiranja. Ovo se definiše u `order by` klauzuli koja se navodi nakon `where` klauzule. Klauzula `order by` se sastoji od naziva polja po kojima je potrebno izvršiti sortiranje. Za svako polje se može navesti i redosled sortiranja (rastući ili opadajući). Rastući redosled se označava ključnom rečju `asc`. Ovaj redosled je podrazumevani. Opadajući redosled se specificira ključnom rečju `desc`. Dat je primer opadajućeg sortiranja gradova po nazivu.

```
select * from grad order by naziv desc;
```

Moguće je navesti i više polja u order by klauzuli. Tada se sortiranje vrši po prvonavedenom polju, a slogovi koji imaju istu vrednost tog polja se sortiraju po drugonavedenom polju i tako redom. Dat je primer opadajućeg sortiranja gradova po nazivu, pri čemu se gradovi sa istim nazivom sortiraju rastuće po ptt broju.

```
select * from grad order by naziv desc, ptt_broj asc;
```

Podaci koji su u tabeli uskladišteni `insert` upitom, mogu kasnije biti izmenjeni `update` upitom. U ovom upitu se navode nove vrednosti polja za određeni slog. Dat je primer izmene naziva grada.

```
update grad set naziv = 'Belgrade' where ptt_broj = 11000;
```

Treba primetiti `where` klauzulu, kojom se specificira na koji slog se izmena vrednosti polja odnosi. Moguće je jednim update upitom izmeniti vrednosti više polja. Ovo je prikazano u narednom SQL kodu na primeru izmene imena i prezimena studenta sa brojem indeksa 10784.

```
update student set ime = 'Milan', prezime = 'Milanovic'
      where broj_indeksa = 10784;
```

Postojeće slogove u tabeli je moguće obrisati `delete` upitom. Dat je primer brisanja grada sa ptt brojem 11000.

```
delete from grad where ptt_broj = 11000;
```

Slično `update` upitu, i ovde `where` klauzula specificira koji se slog briše.

Kada govorimo o DML, pomenimo još da je moguće realizovati i ugnježdene upite. Jedan SQL upit može da sadrži drugi upit.

Kontrola pristupa podacima

Data Control Language (DCL) definiše prava pristupa bazi i njenim tabelama. Za SUBP se definišu korisnici koji mogu da mu pristupaju. Putem DCL se za svakog korisnika mogu definisati akcije koje može da sprovodi nad bazom podataka. Dodela prava pristupa se vrši naredbom `grant`, a oduzimanje ranije dodeljenog prava naredbom `revoke`. Dat je primer dodele prava ubacivanja podataka u tabelu `grad` korisniku sa korisničkim imenom `petar`.

```
grant insert on grad to petar;
```

Ključna reč `insert` specificira da se pravo dodeljuje za operaciju ubacivanja podataka u tabelu. Na sličan način se korisniku oduzima određeno pravo. U primeru je ilustrovano oduzimanje prava dodeljenog u prethodnom primeru.

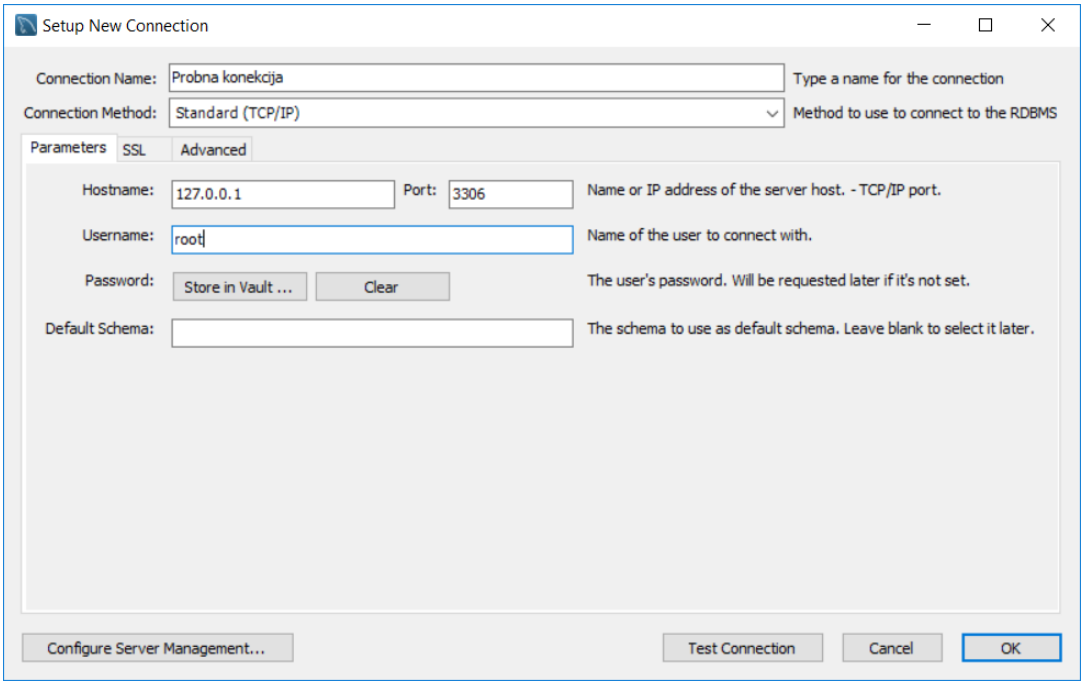
```
revoke insert on grad from petar;
```

Alati za komunikaciju sa SUBP

U prethodnom delu ovog poglavlja smo videli da SUBP-u možemo izdavati naredbe u SQL sintaksi. Ovde ćemo objasniti na koji način da tehnički realizujemo pisanje SQL komandi i njihovo slanje SUBP-u. SUBP može biti instaliran na bilo kojem računaru. Često to nije računar na kojem se izvršava aplikacija koja skladišti podatke koristeći SUBP. Dovoljno je da aplikacija može putem mreže da komunicira sa računarom na kojem je SUBP. Da bi aplikacija mogla da skladišti podatke u bazi podataka, neophodno je najpre kreirati bazu podataka i njene tabele, kao i ubaciti inicijalne podatke. Takođe, u toku razvoja aplikacije često je neophodno direktno pristupiti bazi podataka da bi se proverio njen sadržaj ili izvršile određene izmene u podacima ili šemi. Za ovu svrhu koriste se specijalizovani programi koji putem mreže šalju SQL komande SUBP-u, preuzimaju rezultate i prikazuju ih korisniku.

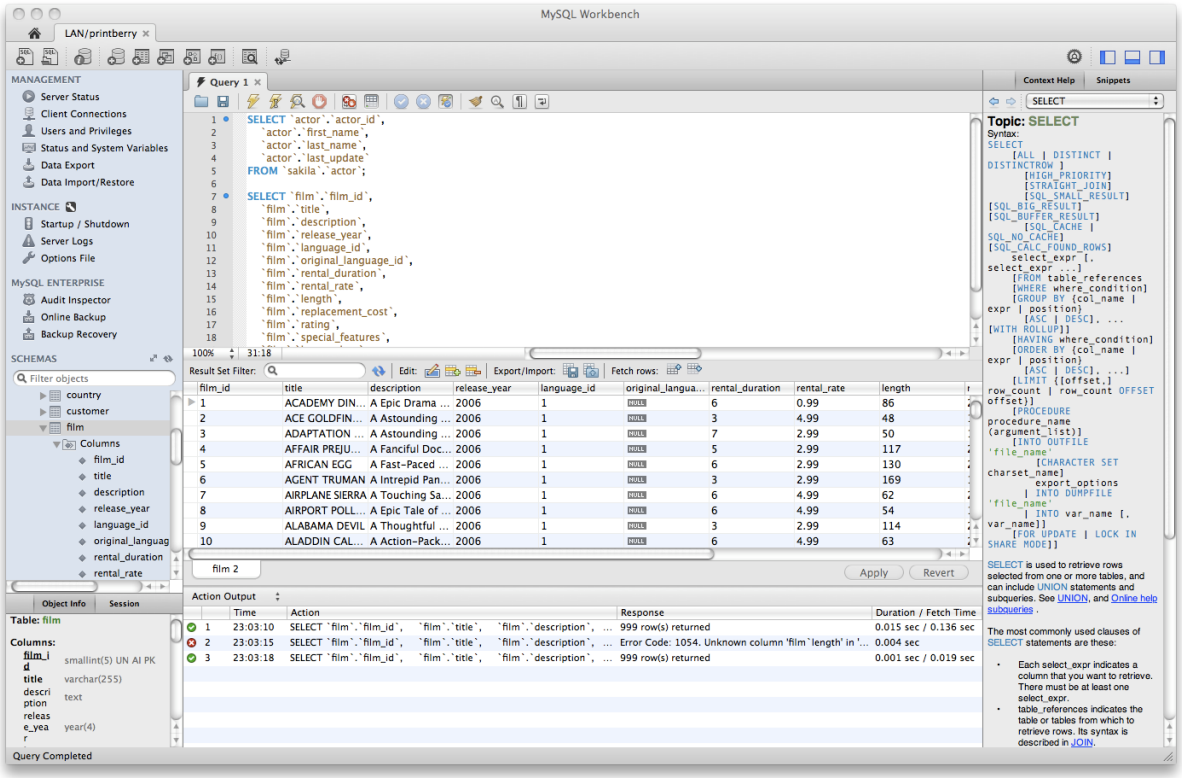
Za MySQL SUBP, postoji konzolna aplikacija pod nazivom mysql koja obavlja pomenute funkcionalnosti vršeći interakciju sa korisnikom putem komandne linije. Pored ove aplikacije, postoje i aplikacije iste namene koje sadrže grafički interfejs, pa korisniku prikazuju podatke u preglednijem obliku. U nastavku ćemo predstaviti MySQL Workbench aplikaciju, koja se najčešće koristi kao grafička aplikacija za komunikaciju sa MySQL SUBP-om.

Prvi korak u komunikaciji sa MySQL SUBP-om je povezivanje sa SUBP-om. U MySQL Workbench-u je za tu svrhu neophodno kreirati novu konekciju ka MySQL SUBP-u. Klikom na dugme + u sekciji MySQL Connections se otvara dijalog za kreiranje nove konekcije koji je prikazan na narednoj slici.



Polje hostname predstavlja mrežnu adresu računara na kojem je instaliran MySQL SUBP na koji se povezujemo. U prikazanom primeru, reč je o računaru sa adresom 127.0.0.1, što je lokalni računar (isti računar na kojem se izvršava i MySQL Workbench). Polje username se odnosi na korisničko ime korisnika koji se povezuje na SUBP.

Kada postoji konekcija, moguće je povezati se na SUBP i izvršavati SQL upite. MySQL Workbench pruža grafički editor za pisanje i izvršavanje upita. Moguće je više upita definisati u istom fajlu. Fajl koji sadrži SQL upite se u fajlu sistemu skladišti sa ekstenzijom .sql i naziva SQL skript.Na slici je prikazan prozor MySQL Workbench aplikacije u kojem je moguće pisati i izvršavati SQL kod.



* preuzeto sa www.mysql.com

Centralni deo prozora zauzima editor za pisansavje SQL koda. Napisani SQL kod se izvršava klikom na ikonicu za izvršavanje koja se nalazi iznad editora. Ispod editora se prikazuju rezultati izvršavanja koda koje SUBP vraća. Na dnu prozora su informacije o uspešnosti izvršavanja upita. Sa leve strane, na panelu SCHEMAS, se nalazi pregled svih baza podataka koje SUBP na koji smo povezani sadrži. Moguće je pregledati strukturu svake od baza podataka. Na panelu MANAGEMENT, koji se nalazi u gornjem levom uglu, korisniku su na raspolaganju akcije za administriranje SUBP. Pomenimo opciju *Data Export*, koja omogućuje eksportovanje podataka iz baze podataka u fajl. Opcijom *Data Import/Restore* je kasnije moguće eksportovane podatke ubaciti u proizvoljnu instancu MySQL SUBP-a.