Uvod u struktuirani upitni jezik

Ovo je kratak kurs o struktuiranom upitnom jeziku (poznatom i kao SQL) i predstavlja pionirski poduhvat jer je to prvi takav kurs koji se može naći na Internetu. SQL omogućava korisnicima pristup podacima u sistemima za upravljanje relacionim bazama podataka, kao što su Oracle, Sybase, Informix, Microsoft SQL Server, Access i drugi, tako što dopušta korisnicima da opišu podatke koje žele da dobiju. SQL takođe omogućava korisnicima da definišu podatke u nekoj bazi podataka i da manipulišu tim podacima. U ovom kursu biće opisano kako se koristi SQL, a to će sve biti potkrepljeno primerima. SQL koji se koristi u ovom dokumentu predstavlja ANSI verziju tog jezika, ili standardni SQL, i nijedna posebna mogućnost SQL-a koja je specifična za sisteme za upravljanje bazama podataka neće se razmatrati do odeljka „Nestandardni SQL“. Preporučujemo da odštampate ovaj dokument da biste lakše mogli da pogledate prethodne primere na koje se ukazuje u tekstu.

Sadržaj

Pregled sintakse naredbi
Vežbanja
Važniji sajtovi o SQL-u, bazama podataka i sličnim temama

Osnove naredbe SELECT

U relacionoj bazi podataka podaci se smeštaju u tabelama. Neka tabela, na primer, mogla bi da sadrži jedinstveni matični broj građana, imena, prezimena i adrese zaposlenih:

TabelaAdresaRadnika					
JMBG	**Ime**	**Prezime**	**Adresa**	**Grad**	**Republika**
512687458	Đorđe	Petrović	Kralja Petra 9	Beograd	Srbija
758420012	Marija	Simić	Bul Nikole Tesle 22	Jagodina	Srbija
102254896	Savo	Jovanović	Njegoševa 17	Podgorica	Crna Gora
876512563	Svetlana	Aćimović	Laze Lazarevića 10	Subotica	Srbija

Pretpostavimo da želite, recimo, da vidite adrese svih zaposlenih. Da biste to postigli, koristite naredbu SELECT:

```
SELECT Ime, Prezime, Adresa, Grad, Republika
FROM TabelaAdresaRadnika;
```

Rezultat ovog *upita* u bazu podataka je:

Ime	**Prezime**	**Adresa**	**Grad**	**Republika**
Đorđe	Petrović	Kralja Petra 9	Beograd	Srbija
Marija	Simić	Bul Nikole Tesle 22	Jagodina	Srbija
Savo	Jovanović	Njegoševa 17	Podgorica	Crna Gora
Svetlana	Aćimović	Laze Lazarevića 10	Subotica	Srbija

Da objasnimo sada šta ste upravo uradili: tražili ste sve podatke u tabeli TabelaAdresaRadnika – preciznije, tražili ste *kolone* pod nazivom Ime, Prezime, Adresa, Grad, Republika. Obratite pažnju da imena kolona i tabela ne sadrže razmake – ona se moraju navesti kao jedna reč, kao i da se naredba završava tačkom i zarezom (;). Opšti oblik naredbe SELECT kojom se dobijaju svi *redovi* u tabeli je:

```
SELECT ImeKolone, ImeKolone, ...
FROM ImeTabele;
```

Da biste dobili sve kolone neke tabele bez navođenja svih imena kolona, koristite:

```
SELECT * FROM ImeTabele;
```

Svaki sistem za upravljanje bazama podataka i softver za rad sa njima ima različite metode za prijavljivanje na bazu podataka i upisivanje SQL naredbi; potražite pomoć lokalnog administratora da biste se prijavili sistemu i mogli da koristite SQL.

Uslovna selekcija podataka

Da bismo dalje razmotrili naredbu SELECT, pogledajmo drugi primer (hipotetičke) tabele:

TabelaPrimanjaRadnika			
IDRadnika	**Plata**	**Prinadležnosti**	**Položaj**
010	75000	15000	rukovodilac
105	65000	15000	rukovodilac
152	60000	15000	rukovodilac
215	60000	12500	rukovodilac
244	50000	12000	činovnik
300	45000	10000	činovnik
335	40000	10000	činovnik
400	32000	7500	pripravnik
441	28000	7500	pripravnik

Relacioni operatori

U SQL-u postoji šest relacionih operatora i posle njihovog predstavljanja videćemo kako se koriste:

=	Jednako
< ili != (videti priručnik)	Različito
<	Manje
>	Veće
<=	Manje ili jednako
>=	Veće ili jednako

Da bi se prikazali samo oni redovi iz tabele koji zadovoljavaju određene kriterijume, koristi se *klauzula WHERE*. Ona se može najlakše razumeti ukoliko se pogleda nekoliko primera.

Ukoliko želite da dobijete ID brojeve onih zaposlenih koji zarađuju preko 50.000, koristite sledeću naredbu:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE PLATA >= 50000;
```

Obratite pažnju da se koristi znak >= (veće ili jednako), pošto smo želeli da izdvojimo one zaposlene koji zarađuju više od 50,000, ili jednako 50,000, i to prikazano zajedno. Kao rezultat dobijamo:

```
IDRADNIKA
------------
010
105
152
215
244
```

Opis klauzule WHERE, odnosno deo PLATA >= 50000, naziva se *uslov* (operacija koja kao rezultat daje vrednost True (tačno) ili False (netačno)). Isti tip operacije može se primeniti na tekstualne kolone:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE POLOŽAJ = 'rukovodilac';
```

Ova naredba prikazuje ID brojeve svih rukovodilaca. Generalno, u slučaju tekstualnih kolona, koristite operatore jednako ili različito, i obavezno ceo tekst koji se pojavljuje u naredbi navedite unutar apostrofa (').

Složeni uslovi

Operator AND kombinuje dva ili više uslova i prikazuje neki red tabele samo ukoliko podaci tog reda zadovoljavaju **sve** navedene uslove (tj. svi uslovi su tačni). Na primer, da biste prikazali sve činovnike koji zarađuju više od 40.000, koristite naredbu:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE PLATA > 40000 AND POLOŽAJ = 'činovnik';
```

Operator OR kombinuje dva ili više uslova, ali prikazuje red ukoliko je **neki** navedeni uslov zadovoljen. Da biste zajedno prikazali one zaposlene koji zarađuju manje od 40.000 ili su im prinadležnosti manje od 10.000, koristite ovaj upit:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE PLATA < 40000 OR PRINADLEŽNOSTI < 10000;
```

Operatori AND i OR mogu da se kombinuju, na primer:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
```

```
WHERE POLOŽAJ = 'rukovodilac' AND PLATA > 60000 OR PRINADLEŽNOSTI >
12000;
```

SQL najpre pronalazi (odvojeno) redove u kojima zaposleni ima platu veću od 60.000 i ima rukovodeći položaj, a zatim iz ove liste redova izdvaja one koji zadovoljavaju gornji AND uslov ili uslov da su prinadležnosti veće od 12.000. Na kraju, SQL prikazuje ovu drugu novu listu, pri čemu treba znati da će svako ko prima prinadležnosti veće od 12.000 biti uključen jer operator OR uključuje red ako je bar jedan uslov tačan. Takođe obratite pažnju da se najpre izračunava rezultat operatora AND.

Ako generalizujemo ovaj proces, SQL najpre izračunava AND uslove da bi odredio redove koji zadovoljavaju AND operacije (zapamtite: svi uslovi moraju biti tačni), zatim ovi redovi se proveravaju prema OR uslovima, a na kraju prikazuje samo one preostale redove koji zadovoljavaju neki od OR uslova (pri čemu se za izračunavanje operatora OR koristi par uslova ili rezultata operatora AND, a dobija se rezultat tačno ako je bilo koji međurezultat tačan). Matematički, SQL izračunava najpre sve uslove, zatim izračunava sve AND parove i na kraju OR parove (pri čemu se oba operatora izračunavaju sa leva na desno).

Da bismo ovo pojasnili na primeru, pretpostavimo da SQL izračunava klauzulu WHERE za dati red tabele kako bi odredio da li taj red treba uključiti u rezultat upita (tj. da li izračunavanje čitave klauzule WHERE daje tačno), da je izračunao rezultat svih uslova i da je spreman da primeni logičke operacije na ovaj rezultat:

```
True AND False OR True AND True OR False AND False
```

Najpre se uproste AND parovi:

```
False OR True OR False
```

Zatim OR parovi, sa leva na desno:

```
True OR False
```

```
True
```

Rezultat je tačno i red zadovoljava uslove upita. Obavezno pročitajte sledeći odeljak o operatoru NOT i redosledu izračunavanja logičkih operacija. Nadamo se da vam je ovaj odeljak pomogao u razumevanju operatora AND i OR, iako je teško to kratko objasniti.

Da biste primenili operatore OR pre operatora AND, kao npr. u slučaju kada želite da dobijete listu rukovodilaca koji imaju veliku platu (50.000) ili imaju velike prinadležnosti (10.000), koristite zagrade:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE POLOŽAJ = 'rukovodilac' AND (PLATA > 50000 OR PRINADLEŽNOSTI >
10000);
```

Operatori IN i BETWEEN

Lakši način za kombinovanje uslova je pomoću operatora IN ili BETWEEN. Na primer, ako želite da dobijete sve rukovodioce ili činovnike:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE POLOŽAJ IN ( 'rukovodilac', 'činovnik');
```

ili da prikažete sve zaposlene koji zarađuju više ili jednako 30.000, ali manje ili jednako 50,000:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE PLATA BETWEEN 30000 AND 50000;
```

Da biste prikazali sve čije plate nisu u ovom intervalu, pokušajte:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE PLATA NOT BETWEEN 30000 AND 50000;
```

Slično, NOT IN prikazuje sve redove koji nisu dobijeni u listi operatora IN.
Pored toga, operator NOT se može upotrebiti zajedno sa operatorima AND i OR, ali morate imati na umu da je on unarni operator (koristi se sa jednim uslovom, dajući kao rezultat njegovu suprotnu logičku vrednost, dok se operatori AND i OR koriste sa dva uslova), i da se svi operatori NOT izračunavaju pre bilo kojih operatora AND i OR.

Redosled izračunavanja logičkih operacija u SQL-u (svaka operacija izračunava se sa leva na desno) je :

1. NOT
2. AND
3. OR

Operator LIKE

Vratimo se na tabelu TabelaPrimanjaRadnika i pretpostavimo da želite da dobijete sve zaposlene čije prezime počinje sa „S“; pokušajte:

```
SELECT IDRADNIKA
FROM TABELAPRIMANJARADNIKA
WHERE PREZIME LIKE 'S%';
```

Procenat (%) se koristi ukoliko želimo da predstavimo bilo koji znak (cifru, slovo, znak interpunkcije) ili skup znakova koji mogu da slede iza slova „S“. Da biste pronašli zaposlene čija se prezimena završavaju sa „S“, koristite '%S', ili ako želite „S“ u sredini reči, pokušajte '%S%'. Znak '%' može biti upotrebljen umesto bilo kojih znakova koji se

nalaze na istoj poziciji relativno od datih znakova. Operator NOT LIKE prikazuje redove koji ne zadovoljavaju dati kriterijum. Postoje i druge mogućnosti za upotrebu operatora LIKE, ili onih drugih koje smo razmatrali, ali to umnogome zavisi od konkretnog sistema za upravljanje bazama podataka koji koristite; kao i obično, konsultujte priručnik ili administratora sistema da biste saznali koje mogućnosti postoje na vašem sistemu, ili da biste proverili da li ono što pokušavate da uradite jeste raspoloživo i dopušteno. Ova napomena važi i za mogućnosti SQL-a koje ćemo opisati u nastavku. Cilj ovog odeljka je samo da vam predoči ideju mogućnosti za upite koji se mogu pisati u SQL-u.

Spajanje tabela

U ovom odeljku govori se samo o *unutrašnjem* spajanju i spajanju *izjednačavanjem* pošto su oni najkorisniji u praksi. Da biste dobili više informacija o ovome, posetite neki od sajtova u vezi sa SQL-om na kraju dokumenta.

Dobar dizajn baze podataka preporučuje da svaka tabela sadrži podatke o samo jednom entitetu, a detaljnije informacije u relacionoj bazi podataka mogu se dobiti pomoću dodatnih tabela i njihovim *spajanjem*.

Uvedimo najpre sledeće tabele kao primer:

VlasniciAntikviteta

IDVlasnika	PrezimeVlasnika	ImeVlasnika
01	Jovanović	Branko
02	Simonović	Boban
15	Lazarević	Pava
21	Aćimović	Jelena
50	Filipović	Sima

Porudžbine

IDVlasnika	ŽeljeniKomad
02	sto
02	pisaći sto
21	stolica
15	ogledalo

Antikviteti

IDProdavca	IDKupca	KomadNameštaja
01	50	krevet
02	15	sto
15	02	stolica
21	50	ogledalo
50	01	pisaći sto

01	21	orman
02	21	stočić za kafu
15	50	stolica
01	15	kutija za nakit
02	21	posuđe
21	02	polica za knjige
50	01	vaza

Ključevi

Razmotrimo najpre koncept *ključeva*. Primarni ključ je kolona ili skup kolona koji jednoznačno određuju ostatak podataka u svakom redu. Na primer, kolona IDVlasnika u tabeli VlasniciAntikviteta jednoznačno određuje određeni red. Ovo znači dve stvari: nijedna dva reda ne smeju imati istu vrednost u koloni IDVlasnika, kao i da čak ukoliko dva vlasnika imaju isto ime i prezime, kolona IDVlasnika obezbeđuje da oni neće biti pomešani jer se za manipulisanje njima u celoj bazi koristi kolona IDVlasnika, a ne njihova imena.

Strani ključ je kolona u tabeli koja je primarni ključ u drugoj tabeli, što znači da sve vrednosti u koloni stranog ključa moraju imati odgovarajuće podatke u drugoj tabeli u kojoj je ta kolona primarni ključ. U terminologiji relacionih baza podataka, ova veza se naziva *referencijalni integritet*. Na primer, u tabeli Antikviteti obe kolone IDKupca i IDProdavca predstavljaju strane ključeve za primarni ključ tabele VlasniciAntikviteta (tj. kolonu IDVlasnika; u cilju izlaganja pretpostavljamo da neko mora biti vlasnik antikviteta pre nego što može da kupuje ili prodaje komade antikvitetnog nameštaja), pošto se u obe kolone ID brojeva koriste za identifikaciju vlasnika ili kupaca i prodavaca, a kolona IDVlasnika je primarni ključ tabele VlasniciAntikviteta. Drugim rečima, svi ovi ID brojevi se koriste za ukazivanje na same vlasnike, kupce i prodavce antikviteta, bez potrebe za korišćenjem njihovih stvarnih imena.

Obavljanje spajanja

Namena ovih ključeva je da podaci iz više tabela mogu da se kombinuju, bez potrebe da se podaci ponavljaju u svim tabelama – u ovome se sastoji snaga relacionih baza podataka. Na primer, možete da pronađete imena onih koji su kupili stolicu bez moranja da navedete njihova puna imena u tabeli Antikviteti. To ime možete da saznate povezujući one koji su kupili stolicu sa imenima u tabeli VlasniciAntikviteta preko kolone IDVlasnika, koja *uspostavlja odnos* između podataka u ovim dvema tabelama. Da biste dobili imena onih koji su kupili stolicu, koristite ovaj upit:

```
SELECT PREZIMEVLASNIKA, IMEVLASNIKA
FROM VLASNICIANTIKVITETA, ANTIKVITETI
WHERE IDKUPCA = IDVLASNIKA AND KOMADNAMEŠTAJA = 'stolica';
```

Obratite pažnju na sledeće činjenice o ovom upitu: obe tabele koje su u relaciji navedene su u klauzuli FROM naredbe SELECT. U klauzuli WHERE primetite da uslov KOMADNAMEŠTAJA = 'stolica' ograničava izbor na one koji su kupili (a u našem primeru, stoga poseduju) stolicu. Drugo, obratite pažnju kako se uspostavlja odnos između kolona sa ID brojevima iz dve tabele upotrebom uslova IDKUPCA = IDVLASNIKA. Samo ako se podudaraju ID brojevi u tabelama i kupljen komad nameštaja je stolica (zbog operatora AND), prikazaće se imena iz tabele . Pošto je u uslovu spajanja upotrebljen znak jednakosti, ovo spajanje se naziva *spajanje izjednačavanjem*. Rezultat ovog upita su dva imena: Simonović, Boban i Filipović, Sima.

Koristeći *zapis sa tačkama* da biste izbegli dvosmislenost, ispred imena kolona možete pisati imena tabela:

```
SELECT VLASNICIANTIKVITETA.PREZIMEVLASNIKA,
       VLASNICIANTIKVITETA.IMEVLASNIKA
FROM VLASNICIANTIKVITETA, ANTIKVITETI
WHERE ANTIKVITETI.IDKUPCA = VLASNICIANTIKVITETA.IDVLASNIKA
      AND KOMADNAMEŠTAJA = 'stolica';
```

Međutim, kako su u tabelama imena kolona različita, ovo nije bilo neophodno.

Klauzula DISTINCT i eliminacija duplikata

Pretpostavimo da želite da saznate **samo** ID brojeve i imena onih ljudi koji su prodali neki antikvitet. Očigledno, treba nam lista u kojoj je svaki prodavac naveden jedanput – ne interesuje nas koliko antikviteta je neko prodao, već samo činjenica da je neko bio prodavac (ako je važan broj prodatih antikviteta, o tome pročitajte odeljak „Agregatne funkcije“ u nastavku dokumenta). To znači da SQL-u treba kazati da eliminiše duplikate redova prodaje i da svaku osobu prikaže samo jednom. Da biste ovo postigli, koristite ključnu reč DISTINCT.

Najpre nam treba spajanje izjednačavanjem tabele VlasniciAntikviteta da bismo dobili detaljnije podatke o prezimenu i imenu osobe. Međutim, imajte na umu da pošto kolona IDProdavca u tabeli Antikviteti predstavlja strani ključ za tabelu VlasniciAntikviteta, prodavac će biti prikazan samo ukoliko u tabeli VlasniciAntikviteta postoji red sa njegovim ID brojem, imenom i prezimenom. Želimo i da eliminišemo višestruko pojavljivanje broja IDProdavca iz naše liste, pa koristimo DISTINCT ispred kolone iz koje se mogu pojaviti iste vrednosti (međutim, u opštem slučaju nije neophodno uvek pisati DISTINCT ispred imena kolone).

Da bismo pokazali još jednu mogućnost, želimo takođe da lista bude prikazana po abecednom redu prezimena, a zatim imena (u slučaju istih prezimena). Stoga, koristićemo klauzulu ORDER BY:

```
SELECT DISTINCT IDPRODAVCA, PREZIMEVLASNIKA, IMEVLASNIKA
FROM ANTIKVITETI, VLASNICIANTIKVITETA
WHERE IDPRODAVCA = IDVLASNIKA
ORDER BY PREZIMEVLASNIKA, IMEVLASNIKA;
```

U ovom primeru, pošto je svako bio prodavac, dobićemo spisak svih vlasnika po abecednom redu prezimena. Zbog kasnijeg pozivanja na ovu vrsta spajanja (i u slučaju da vas neko o tome pita), smatra se da ovakva vrsta spajanja pripada kategoriji *unutrašnjih spajanja*.

Pseudonimi i IN podupiti

U ovom odeljku govorićemo o pseudonimima, IN podupitima, kao i o tome kako se oni mogu koristiti u našem primeru sa tri tabele. Prvo, razmotrimo sledeći upit koji prikazuje prezimena onih vlasnika koji su poručili nešto i njihovu porudžbinu, pri čemu se daju samo one porudžbine koje se mogu zadovoljiti (to jest, postoji prodavac koji je vlasnik poručenog komada nameštaja):

```
SELECT VL.PREZIMEVLASNIKA Prezime, POR.ŽELJENIKOMAD Poručen komad
FROM PORUDŽBINE POR, VLASNICIANTIKVITETA VL
WHERE POR.IDVLASNIKA = VL.IDVLASNIKA
      AND POR.ŽELJENIKOMAD IN
          (SELECT KOMAD
           FROM ANTIKVITETI);
```

Ovo kao rezultat daje:

```
Prezime      Poručen komad
-----------  -------------
Simonović    sto
Simonović    pisaći sto
Aćimović     stolica
Lazarević    ogledalo
```

Ima nekoliko stvari na koje treba obratiti pažnju u ovom upitu:

1. "Prezime" i "Poručen komad" u redu koji počinje sa SELECT predstavljaju zaglavlja kolona u izveštaju.
2. VL i POR su pseudonimi; to su nova imena za tabele navedene u klauzuli FROM koja se koriste u zapisu sa tačkama kao prefiksi svih imena kolona u upitu. Ovim se eliminiše dvosmislenost, naročito pri spajanju izjednačavanjem u klauzuli WHERE jer obe tabele imaju kolonu pod nazivom IDVlasnika, a zapis sa tačkama ukazuje SQL-u da su u pitanju dve različite kolone IDVlasnika iz dve različite tabele.
3. Obratite pažnju da je tabela Porudžbine navedena prva u klauzuli FROM; ovim se osigurava da se spisak prikazuje prema toj tabeli, a da se druga tabela VlasniciAntikviteta koristi samo za detaljnije podatke (prezime).
4. Najvažnije, AND u klauzuli WHERE izaziva izvršavanje IN podupita ("= ANY" ili "= SOME" su dve ekvivalentna zapisa za IN). Ono što se ovim postiže je izvršavanje podupita koji kao rezultat daje sve posedovane komade nameštaja iz tabele Antikviteti jer nema klauzule WHERE. Zato, da bi se prikazao red iz tabele

Porudžbine, ŽeljeniKomad mora se nalaziti u toj vraćenoj listi komada nameštaja iz tabele Antikviteti, čime se neki antikvitet prikazuje samo ako porudžbina može da se realizuje kupovinom od drugog vlasnika. Ovo može da se shvati na sledeći način: podupit kao rezultat daje *skup* komada nameštaja koji se upoređuje sa svakim željenim komadom iz tabele Porudžbine; IN uslov je tačan samo ako ŽeljeniKomad pripada tom vraćenom skupu iz tabele Antikviteti.

5. Obratite pažnju takođe da se u ovom slučaju desilo da je svaki željeni antikvitet bio raspoloživ, što naravno neće uvek biti slučaj. Pored toga, kada se koriste ključne reči IN, "= ANY" ili "= SOME", one ukazuju na podudarnost redova, a ne kolona. To znači da ne možete navesti više kolona u klauzuli SELECT nekog podupita želeći da uparite kolonu u spoljašnjoj klauzuli WHERE sa jednom od više mogućih vrednosti kolone u podupitu. U podupitu se može navesti samo jedna kolona, a moguća podudarnost proizilazi iz više vrednosti *redova* u toj *jednoj* koloni, a ne obrnuto.

Uh! Dosta je bilo za sada o složenim SELECT upitima. Pređimo sada na druge SQL naredbe.

Pomoćne SQL naredbe

Agregatne funkcije

Razmotrićemo pet važnih *agregatnih funkcija*: SUM, AVG, MAX, MIN i COUNT. One se nazivaju agregatnim jer daju sumarne rezultate nekog upita, a ne spisak svih redova.

- SUM () daje zbir vrednosti date kolone u svim redovima koji zadovoljavaju neki uslov, pri čemu je data kolona numerička.
- AVG () daje prosečnu vrednost date kolone.
- MAX () daje najveću vrednost u datoj koloni.
- MIN () daje najmanju vrednost u datoj koloni.
- COUNT(*) daje broj redova koji zadovoljavaju određen uslov.

Vraćajući se na tabele sa početka ovog dokumenta, razmotrimo tri primera:

```
SELECT SUM(PLATA), AVG(PLATA)
FROM TABELAPRIMANJARADNIKA
```

Ovaj upit prikazuje sumu plata svih zaposlenih u tabeli, kao i prosečnu platu zaposlenih u tabeli.

```
SELECT MIN(PRINADLEŽNOSTI)
FROM TABELAPRIMANJARADNIKA
WHERE POLOŽAJ = 'rukovodilac';
```

Ovaj upit kao rezultat daje najmanju vrednost u koloni Prinadležnosti onih zaposlenih koji su rukovodioci, a to je 12500.

```
SELECT COUNT(*)
FROM TABELAPRIMANJAZAPOSLENIH
WHERE POLOŽAJ = 'činovnik';
```

Ovim upitom saznajemo koliko ima činovnika (3).

Pogledi

U SQL-u, možda (proverite sa administratorom baze podataka) možete da formirate svoje poglede. Ono što pogled omogućava je da rezultate nekog upita dodelite novoj, ličnoj tabeli koju možete koristiti u drugim upitima, pri čemu se u klauzuli FROM novoj tabeli daje ime pogleda. Prilikom pristupa pogledu, izvršava se (u opštem slučaju) upit koji je definisan u naredbi formiranja pogleda, a rezultati tog upita izgledaju isto kao druga tabela u upitu koji ste napisali koji poziva pogled. Na primer, da biste formirali pogled:

```
CREATE VIEW ANTPOGLED AS SELECT ŽELJENIKOMAD FROM PORUDŽBINE;
```

Sada, napišimo upit koji koristi ovaj pogled kao tabelu, pri čemu je tabela predstavlja samo spisak svih željenih komada nameštaja iz tabele Porudžbine:

```
SELECT IDPRODAVCA
FROM ANTIKVITETI, ANTPOGLED
WHERE ŽELJENIKOMAD = KOMADNAMEŠTAJA;
```

Ovaj upit pokazuje sve ID brojeve prodavaca iz tabele Antikviteti, pri čemu komad nameštaja u toj tabeli pripada pogledu AntPogled, a koji predstavlja samo sve željene komade nameštaja u tabeli Porudžbine. Rezultujući spisak se formira prolaskom kroz sve komade nameštaja u tabeli Antikviteti, red po red, dok se ne pronađe isti takav u pogledu AntPogled. Pogledi se mogu koristiti za ograničavanje pristupa bazi podataka, isto kao i za pojednostavljivanje složenih upita, što je ovde slučaj.

Formiranje novih tabela

Sve tabele u bazi podataka moraju se formirati u nekom trenutku. Pogledajmo kako bi na primer formirali tabelu Porudžbine:

```
CREATE TABLE PORUDŽBINE
(IDVLASNIKA INTEGER NOT NULL,
ŽELJENIKOMAD CHAR(40) NOT NULL);
```

Ovom naredbom daje se ime tabeli i opisuje svaka kolona u tabeli. Obratite pažnju da se u ovoj naredbi koriste generički tipovi podataka i da oni mogu biti različiti zavisno od korišćenog sistema za upravljanje bazama podataka. Kao i obično, proverite dokumentaciju svog sistema. Neki opšti tipovi podataka su:

- Char(x) – Kolona znakova, pri čemu broj x označava maksimalan broj dopuštenih znakova (maksimalnu dužinu) u koloni.
- Integer – Kolona celih brojeva, pozitivnih ili negativnih.
- Decimal(x, y) – Kolona decimalnih brojeva, pri čemu x označava maksimalan broj cifara decimalnih brojeva u koloni, a y označava maksimalan broj dopuštenih cifara iza decimalne tačke. Najveći broj tipa (4,2) bio bi 99.99.
- Date – Kolona datuma određenog formata.
- Logical – Kolona koja može da sadrži dve vrednosti: TRUE ili FALSE.

Takođe obratite pažnju da NOT NULL znači da kolona mora da ima neku vrednost u svakom redu. Ako se navede NULL za kolonu, u nekom redu ta kolona može biti prazna.

Menjanje tabela

Dodajmo kolonu tabeli Antikviteti koja treba da sadrži cenu odgovarajućeg komada nameštaja:

```
ALTER TABLE ANTIKVITETI ADD (CENA DECIMAL(8,2) NULL);
```

Podaci za ovu novu kolonu mogu se upisati ili ažurirati na način koji je opisan kasnije.

Dodavanje podataka

Da biste dodali redove tabeli, navedite sledeću naredbu:

```
INSERT INTO ANTIKVITETI VALUES (21, 01, 'otoman', 200.00);
```

Ovim se podaci upisuju u tabelu Antikviteti, kao novi red, kolona po kolona, u predefinisanom redosledu. Promenimo sada redosled i ostavimo kolonu Cena praznom:

```
INSERT INTO ANTIKVITETI (IDKUPCA, IDPRODAVCA, KOMADNAMEŠTAJA)
VALUES (01, 21, 'otoman');
```

Brisanje podataka

Obrišimo ovaj novi red iz baze podataka:

```
DELETE FROM ANTIKVITETI
WHERE KOMADNAMEŠTAJA = 'otoman';
```

Međutim, ako postoji neki drugi red koji sadrži 'otoman', obrisaće se i taj red. Obrišimo sve redove (u ovom slučaju samo jedan) koji sadrži specifične podatke koje smo ranije dodali:

```
DELETE FROM ANTIKVITETI
WHERE KOMADNAMEŠTAJA = 'otoman' AND IDKUPCA = 01 AND IDPRODAVCA = 21;
```

Ažuriranje podataka

Ažurirajmo kolonu Cena u redu koji još ne sadrži podatak o ceni:

```
UPDATE ANTIKVITETI SET CENA = 500.00 WHERE KOMADNAMEŠTAJA = 'stolica';
```

Ovim se cena svih stolica postavlja na 500,00. Kao što se vidi iz ovoga, da bi se ažuriranje ograničilo na određenije redove, mora se koristiti više WHERE uslova povezanih operatorom AND. Isto tako, mogu se navesti vrednosti za više kolona ukoliko se naredbe jednakosti odvoje zarezima.

Dodatne teme

Indeksi

Indeksi omogućavaju sistemu za upravljanje bazama podataka da brže pristupe podacima. *Napomena*: ova mogućnost je nestandardna i nije raspoloživa u svim sistemima. Sistem formira internu strukturu podataka (indeks) koja dovodi do brže selekcije redova, ukoliko se ta selekcija zasniva na indeksiranim kolonama. Indeks ukazuje sistemu gde se u tabeli nalazi određen red ukoliko je data vrednost indeksirane kolone, na sličan način na koji indeks u knjizi govori na kojoj se strani nalazi data reč. Formirajmo indeks za kolonu IDVlasnika u tabeli VlasniciAntikviteta:

```
CREATE INDEX IDVL_IDX ON VLASNICIANTIKVITETA (IDVLASNIKA);
```

Zatim za kolone imena i prezimena:

```
CREATE INDEX IMEIPREZ_IDX ON VLASNICIANTIKVITETA (PREZIMEVLASNIKA,
IMEVLASNIKA);
```

Da bismo uklonili neki indeks, koristimo naredbu DROP:

```
DROP INDEX IDVL_IDX;
```

Usput rečeno, naredbom DROP može se i neka tabela ukloniti iz baze podataka (budite pažljivi! – to znači da se tabela i svi njeni podaci brišu). U drugom primeru, indeks se pravi po dvema kolonama posmatranim kao celina, što može izazvati čudne posledice. Proverite svoj referentni priručnik pre obavljanja ove operacije.

Neki sistemi za upravljanje bazama podataka ne zahtevaju primarne ključeve u tabelama. Drugim rečima, jednoznačnost kolone ne uspostavlja se automatski. To znači da, na primer, ukoliko pokušamo da tabeli VlasniciAntikviteta dodamo novi red koji ima 02 kao vrednost kolone IDVlasnika, neki sistemi će to dopustiti iako ne bi trebalo, pošto ta kolona treba da bude jednoznačna za tabelu (vrednost svakog reda treba da bude različita). Jedan način za rešavanje ovog problema, tj. da bismo primorali sistem da zabrani duplikate, jeste formiranje jednoznačnog indeksa po koloni za koju želimo da bude primarni ključ:

```
CREATE UNIQUE INDEX IDVL_IDX ON VLASNICIANTIKVITETA (IDVLASNIKA);
```

Klauzule GROUP BY i HAVING

Jedan specijalan način upotrebe operatora GROUP BY je povezivanje neke agregatne funkcije sa grupom redova (ovo se naročito odnosi na funkciju COUNT koja broji redove u svakoj grupi). Pretpostavimo najpre ta tabela Antikviteti ima kolonu Cena i da svaki red ima neku vrednost u toj koloni. Želimo da dobijemo cenu najskupljeg antikviteta koji je kupio svaki vlasnik. Zato SQL-u moramo da kažemo da *grupiše* kupovine svakog vlasnika i izdvoji najvišu cenu kupovine:

```
SELECT IDKUPCA, MAX(CENA)
FROM ANTIKVITETI
GROUP BY IDKUPCA;
```

Sledeće, pretpostavimo da želimo da dobijemo najvišu cenu kupovine za antikvitete koji su kupljeni za iznos preko 1000. Za taj slučaj koristimo klauzulu HAVING:

```
SELECT IDKUPCA, MAX(CENA)
FROM ANTIKVITETI
GROUP BY IDKUPCA
HAVING CENA > 1000;
```

Još o podupitima

Još jedna česta upotreba podupita odnosi se na primenu operatora u uslovima klauzule WHERE koji sadrže rezultat upita SELECT nekog podupita. Na primer, prikažimo kupce koji su kupili skup antikvitet (cena antikviteta je za 100 veća od prosečne cene svih kupljenih antikviteta):

```
SELECT IDKUPCA
FROM ANTIKVITETI
WHERE CENA >
      (SELECT AVG(CENA) + 100
      FROM ANTIKVITETI);
```

Podupit izračunava prosečnu cenu plus 100, a zatim se prikazuju ID brojevi kupaca za svaki komad nameštaja koji je koštao iznad tog iznosa. Da bismo eliminisali duplikate, mogli smo da navedemo DISTINCT IDKUPCA.

Dalje, prikažimo prezimena onih vlasnika u tabeli VlasniciAntikviteta *samo* ukoliko su kupili neki antikvitet:

```
SELECT PREZIMEVLASNIKA
FROM VLASNICIANTIKVITETA
WHERE IDVLASNIKA IN
      (SELECT DISTINCT IDKUPCA
      FROM ANTIKVITETI);
```

Podupit kao rezultat daje spisak kupaca, a prezime vlasnika antikviteta prikazuje se ako i samo ako ID broj vlasnika pripada rezultatu podupita (koji se ponekad naziva *lista kandidata*). *Napomena*: u nekim implementacijama mogu se koristiti jednakosti umesto IN, ali zbog razumljivosti pošto je rezultat podupita neki skup, klauzula IN je bolji izbor.

Da bismo pokazali primer sa ažuriranjem podataka, pretpostavimo da osoba koja je kupila sanduk za knjige ima pogrešni ime uneto u bazu podataka – ime treba da bude Jovan:

```
UPDATE VLASNICIANTIKVITETA
SET IMEVLASNIKA = 'Jovan'
WHERE IDVLASNIKA =
      (SELECT IDKUPCA
      FROM ANTIKVITETI
      WHERE KOMADNAMEŠTAJA = 'sanduk za knjige');
```

Podupit najpre pronalazi IDKupca za osobu (ili osobe) koja je kupila sanduk za knjige, a zatim spoljašnji upit ažurira njegovo ime.

Zapamtite sledeće pravilo o podupitima: kada je podupit deo nekog WHERE uslova, klauzula SELECT u tom podupitu mora imati kolone koje se poklapaju po broju i tipu sa onima koje se nalaze u klauzuli WHERE spoljašnjeg upita. Drugim rečima, ako imate "`WHERE ImeKolone = (SELECT...);`", klauzula SELECT mora sadržati samo jednu kolonu koja odgovara koloni ImeKolone u spoljašnjoj klauzuli WHERE *i* njihov tip mora biti odgovarajući (oba tipa su celi brojevi, znakovne niske itd.).

Klauzule EXISTS i ALL

Klauzula EXISTS koristi podupit kao uslov, pri čemu je taj uslov tačan ukoliko se rezultat podupita sastoji od bar jednog reda, a netačan ukoliko rezultat podupita ne daje nijedan red. Ovo je neintuitivna mogućnost koja ima malo jedinstvenih primena. Na primer, ako potencijalni kupac želi da dobije listu vlasnika samo u slučaju da njegova prodavnica drži stolice:

```
SELECT IMEVLASNIKA, PREZIMEVLASNIKA
FROM VLASNICIANTIKVITETA
WHERE EXISTS
      (SELECT *
      FROM ANTIKVITETI
      WHERE KOMADNAMEŠTAJA = 'stolica');
```

Ako postoji stolica kao vrednost u koloni KomadNameštaja tabele VlasniciAntikviteta, rezultat podupita biće jedan red ili više njih, što čini rezultat klauzule EXISTS tačnim, a to izaziva prikazivanje imena i prezimena vlasnika antikviteta. U slučaju da se stolica ne pojavljuje kao vrednost, rezultat spoljašnjeg upita ne bi bio nijedan red.

Klauzula ALL je još jedna neobična mogućnost, pošto se ekvivalentan rezultat obično može dobiti drugim i ponekad jednostavnijim metodima. Razmotrimo primer jednog upita:

```
SELECT IDKUPCA, KOMADNAMEŠTAJA
FROM ANTIKVITETI
WHERE CENA >= ALL
      (SELECT CENA
      FROM ANTIKVITETI);
```

Ovo kao rezultat daje antikvitet sa najvećom cenom (ili više njih ukoliko imaju istu cenu) i njegovog kupca. Podupit kao rezultat daje listu svih cena u tabeli Antikviteti, a spoljašnji upit prolazi kroz svaki red u tabeli Antikviteti i prikazuje ga ako je njegova vrednost u koloni Cena veća ili jednaka od svih (ALL) cena u listi, čime se dobija komad nameštaja sa najvećom cenom. Razlog zašto znak jednakosti „=“ mora da se koristi je zbog toga što će cena najskupljeg antikviteta biti jednaka najvećoj ceni u listi koja uključuje cene svih antikviteta.

Klauzula UNION i spoljašnje spajanje

U nekim slučajevima kada želimo da dobijemo rezultate više upita zajedno, koristimo klauzulu UNION. Da bismo objedinili rezultate dva upita, prikazujući ID brojeve svih kupaca i onih koji su naručili nešto, možemo pisati:

```
SELECT IDKUPCA
FROM ANTIKVITETI
UNION
SELECT IDVLASNIKA
FROM PORUDŽBINE;
```

Obratite pažnju da SQL zahteva da lista kolona u klauzulama SELECT mora biti odgovarajuća, kolona po kolona, prema tipu podataka. U ovom slučaju su kolone IDKupca i IDVlasnika istog tipa (integer). Takođe obratite pažnju da SQL vrši automatsku eliminaciju duplikata prilikom upotrebe klauzule UNION (kao da su rezultati pojedinačnih upita skupovi); u slučaju jednog upita, morate da koristite klauzulu DISTINCT.

Spoljašnje spajanje se koristi u slučaju kada se neki upit spajanja „objedinjuje“ sa redovima koji nisu uključeni u spajanje. Spoljašnje spajanje je naročito korisno kada se navode konstantni tekstualni „indikatori“. Razmotrimo najpre primer:

```
SELECT IDVLASNIKA, 'nalazi se u obema tabelama Porudžbine i Antikviteti'
FROM PORUDŽBINE, ANTIKVITETI
WHERE IDVLASNIKA = IDKUPCA
UNION
SELECT IDKUPCA, 'nalazi se samo u tabeli Antikviteti'
FROM ANTIKVITETI
WHERE IDKUPCA NOT IN
      (SELECT IDVLASNIKA
      FROM PORUDŽBINE);
```

Prvi upit obavlja spajanje da bi prikazao sve vlasnike koji se nalaze u obema tabelama i iza njihovih ID brojeva prikazuje tekst koji na to ukazuje. Klauzula UNION objedinjuje ovu listu sa sledećom listom. Druga lista dobija je tako što se najpre odrede oni ID brojevi kupaca koji se ne nalaze u tabeli Porudžbine, čime se proizvodi lista ID brojeva koji nisu uključeni u rezultat upita spajanja. Zatim se skeniraju svi redovi tabele Antikviteti, a ukoliko IDKupca nije u listi isključenih ID brojeva kupaca, on se prikazuje zajedno sa svojom tekstualnom oznakom. Možda postoji lakši način da se dobije ova lista, ali je teško da se proizvede tekstualni opis svakog reda u spisku.

Ovaj koncept je koristan u slučajevima kada postoji odnos između primarnog ključa i spoljašnjeg ključa, ali vrednost spoljašnjeg ključa je NULL za neke primarne ključeve. Na primer, u jednoj tabeli primarni ključ je prodavac, a u drugoj su to kupci pri čemu su njihovi prodavci navedeni u istom redu. Međutim, ako prodavac nema kupaca, njegovo ime neće se pojaviti u tabeli kupaca. Spoljašnje spajanje se koristi ukoliko treba da se dobije spisak svih prodavaca, zajedno sa njihovim kupcima, bilo da prodavac ima ili nema kupaca – to jest, nijedan kupac se ne prikazuje (logička vrednost NULL) ako prodavac nema kupaca ali se nalazi u tabeli prodavaca. U suprotnom, prodavac se prikazuje sa svim kupcima.

Prilikom spajanja tabela mora se voditi računa o još jednom pravilu koje se tiče vrednosti NULL: redosled tabela u klauzuli FROM je veoma važan. Pravilo glasi da SQL „dodaje“ drugu tabelu prvoj; prva tabela sadrži sve redove sa vrednostima NULL u koloni po kojoj se vrši spajanje; ako i druga tabela u nekom redu ima vrednost NULL u koloni po kojoj se vrši spajanje, podaci iz tog reda se ne spajaju sa podacima reda prve tabele. Ovo je druga česta primena (ako želite da i ti podaci budu uključeni u rezultat) spoljašnjeg spajanja. Koncept nepostojeće vrednosti (vrednosti NULL) je važan, pa ukoliko nije potpuno jasan možda treba da utrošite dodatno vreme za njegovo razumevanje.

Verovatno vam je za sada dosta upita? Pređimo zato na nešto potpuno drugo ...

Embedded SQL – jedan loš primer

```
/* Ovo je primer programa koji koristi dodatak Embedded SQL (Ugrađeni
   SQL). Embedded SQL omogućava programerima da se povežu na bazu
```

```
   podataka i koriste naredbe SQL-a u samom programu, tako da njihovi
   programi mogu da koriste, manipulišu i obrađuju podatke u bazi
   podataka.
   - Ovaj primer C programa (koristeći Embedded SQL) štampa izveštaj.
   - Ovaj program mora da se prekompajlira za SQL naredbe pre regularnog
     kompajliranja.
   - Delovi koji sadrže EXEC SQL su isti (standardizovani), ali okolni
     iskazi jezika C moraju se promeniti, zajedno sa deklaracijama host
     promenljivih, ukoliko koristite drugi jezik.
   - Embedded SQL je različit od sistema do sistema pa, još jednom,
     proverite lokalnu dokumentaciju koja se odnosi naročito na
     deklaracije promenljivih i postupak za prijavljivanje, pri čemu
     mreža, sistem za upravljanje bazama podataka i operativni system
    imaju suštinsku ulogu.
*/
/*****************************************************/
/* OVAJ PROGRAM NE MOŽE SE KOMPAJLIRATI ILI IZVRŠITI */
/* ON SLUŽI SAMO KAO PRIMER                          */
/*****************************************************/
#include <stdio.h>
/* Ovaj deo sadrži deklaracije host promenljivih; to su
   promenljive koje koristi program, ali i one koje SQL koristi za
   upisivanje ili čitanje vrednosti. */
EXEC SQL BEGIN DECLARE SECTION;
  int IDKupca;
  char Ime [100], Prezime [100], Komad[100];
EXEC SQL END DECLARE SECTION;
/* Ovo uključuje promenljivu SQLCA da bi se greške mogle proveriti. */
EXEC SQL INCLUDE SQLCA;
main() {
/* Ovo je mogući način za prijavljivanje na bazu podataka. */
EXEC SQL CONNECT KorisničkoIme/Lozinka;
/* Sledeći deo ispisuje poruku da ste povezani ili, u slučaju greške,
   da je prijavljivanje pogrešno ili nije moguće. */
  if(sqlca.sqlcode) {
    printf(Printer,
           "Greška prilikom povezivanja na server baze podataka.\n");
    exit();
  }
  printf("Uspešno povezivanja na server baze podataka.\n");
/* Ovo deo deklariše "kursor". On se koristi u slučaju da upit kao
   rezultat daje više redova i da neka operacija treba da se obavi nad
   svim redovima. Svi redovi koji se dobijaju kao rezultat upita biće
   korišćeni u izveštaju. Kasnije, koristi se naredba "Fetch" za
   dohvatanje svakog reda, red po red, ali da bi se upit stvarno
   izvršio, koristi se naredba "Open". Naredba "Declare" samo deklariše
   upit. */
EXEC SQL DECLARE KursorKomadaNameštaja CURSOR FOR
  SELECT KOMADNAMEŠTAJA, IDKUPCA
  FROM ANTIKVITETI
  ORDER BY KOMADNAMEŠTAJA;
EXEC SQL OPEN KursorKomadaNameštaja;
/* +-- Ovde može da bude sličan blok za proveru greške --+ */
/* Naredba "Fetch" upisuje "sledeći" red iz rezultata upita u host
   promenljive. Međutim, mora se najpre primeniti tzv. tehnika "prvo
   učitavanje". Kada se dođe do kraja podataka kursora, odgovarajuća
   vrednost u promenljivi sqlcode omogućava nam izlazak iz petlje.
```

```
   Obratite pažnju da se, zbog jednostavnosti, izlazak iz petlje vrši
   ako sqlcode ima bilo koju vrednost različitu od nule, čak i ako je to
   vrednost greške. U opštem slučaju, svaka vrednost mora se posebno
   ispitati. */
EXEC SQL FETCH KursorKomadaNameštaja INTO :Komad, :IDKupca;
  while(!sqlca.sqlcode) {
/* Obrada svakog reda sastoji se u sledećem. Prvo, cena se uvećava za 5
   (provizija posrednika) i uzima ime kupca koje ide u izveštaj. Da bi
   se ovo postiglo, koristimo naredbu Update i Select pre ispisivanja na
   ekran. Ovaj način ažuriranja pretpostavlja da svaki kupac kupuje samo
   jedan komad nameštaja, jer u suprotnom bi se cena uvećavala suviše
   mnogo. U opštem slučaju mora se koristiti jedinstvena identifikacija
   redova pomoću tzv. "RowID" kolone (videti dokumentaciju). Takođe
   obratite pažnju na dvotačku ispred imena host promenljivih koje se
   koriste u SQL naredbama. */
EXEC SQL UPDATE ANTIKVITETI
  SET CENA = CENA + 5
  WHERE KOMADNAMEŠTAJA = :Komad AND IDKUPCA = :IDKupca;
EXEC SQL SELECT IMEVLASNIKA, PREZIMEVLASNIKA
  INTO :Ime, :Prezime
  FROM VLASNICIANTIKVITETA
  WHERE IDKUPCA = :IDKupca;
    printf("%25s %25s %25s", Ime, Prezime, Komad);
    /* Ružan izveštaj – služi samo kao primer! */
/* Učitavanje sledećeg reda. */
EXEC SQL FETCH KursorKomadaNameštaja INTO :Komad, :IDKupca;
  }
/* Zatvaranje kursora, potvrđivanje promena i završetak programa. */
EXEC SQL CLOSE KursorKomadaNameštaja;
EXEC SQL COMMIT RELEASE;
  exit();
}
```

Česta pitanja o SQL-u

1. Zašto ne mogu prosto da tražim prva tri reda? – Jer se u relacionim bazama podataka redovi dodaju u proizvoljnom redosledu tako da redove jedino možete da tražite koristeći odgovarajuće mogućnosti SQL-a, kao npr. ORDER BY i tako dalje.

2. Šta znače skraćenice DDL i DML o kojima ponekad čujem? – DDL je jezik za definisanje podataka (engl. *Data Definition Language*) i odnosi se na naredbu Create Table. DML je jezik za manipulisanje podacima (engl. *Data Manipulation Language*) i odnosi se na naredbe Select, Update, Insert i Delete. Takođe, skraćenica QML predstavlja jezik za manipulaciju upitima (engl. *Query Manipulation Language*) i odnosi se na naredbu Select.

3. Zar tabele baze podataka nisu obične datoteke? – Sistem za upravljanje bazama podataka smešta podatke u datoteke koje se unapred rezervišu (na velikim sistemima), ali se podaci čuvaju u specijalnom formatu i podaci iz jedne tabele mogu biti raspoređeni u nekoliko datoteka. Skup datoteka koji se formira za neku bazu podataka naziva se *prostor za tabele*. U opštem slučaju na malim sistemima,

svi objekti baze podataka (definicije i svi podaci tabela) čuvaju se u jednoj datoteci.

4. Zar tabele baze podataka nisu kao tabele programa za tabelarno izračunavanje? – Ne, iz dva razloga. Prvo, tabela programa za tabelarno izračunavanje može da sadrži podatke u ćeliji, ali ćelija je više od lokacije prostog preseka nekog reda i kolone. Zavisno od programa za tabelarno izračunavanje, ćelija može takođe sadržati formule i instrukcije za formatiranje, što tabele baze podataka ne mogu (za sada). Drugo, ćelije tabele programa za tabelarno izračunavanje često zavise od podataka u drugim ćelijama. U bazi podataka, „ćelije“ su nezavisne, osim što postoji logički odnos između kolona (tako da svaki red tabele opisuje jedan primerak entiteta) i svi redovi tabele su međusobno nezavisni, sa izuzetkom ograničenja primarnog i stranog ključa.

5. Kako mogu tekstualnu datoteku sa podacima da „uvezem“ u bazu podataka? – To ne možete da uradite direktno. Morate da koristite neki uslužni program, kao što je Oracleov SQL*Loader, ili da sami napišete program za punjenje podataka u bazu podataka. Takav program bi jednostavno obrađivao svaki red tekstualne datoteke, deleći ga u kolone, i izvršavao naredbu Insert za upis podataka u bazu podataka.

6. Koje biste Web sajtove i knjige preporučili za dobijanje detaljnijih informacija o SQL-u i bazama podataka? – Pogledajte najpre sajtove na kraju ovog dokumenta. Naročito preporučujemo sledeće: Ask the SQL Pro (o SQL-u), DB Ingredients (više teorijske teme), DBMS Lab/Links (sveobuhvatni spisak veza za akademske sisteme za upravljanje bazama podataka), Access on the Web (o Web pristupu Accessovim bazama podataka), Tutorial Page (spisak drugih kurseva) i miniSQL (detaljne informacije o najpoznatijem besplatnom sistemu za upravljanje bazama podataka).

   Takođe, ukoliko želite da vežbate SQL na interaktivnom sajtu (koristeći Java tehnologije), veoma preporučujemo sajt Frenka Toresa (Frank Torres, torresf@uswest.net) na adresi http://sqlcourse.com i njegov nastavak (takoreći) na adresi http://sqlcourse2.com. Frenk je izvanredno uradio svoj sajt i ako imate noviji Web pretraživač, poseta njegovom sajtu je sigurno vredna truda. Dalje, posetite www.topica.com i pretplatite se da elektronskom poštom dobijate njihove savete o SQL-u (engl. *SQL Tips of the Day*) – izvanredni su. Tom Kvinlen (Tim Quinlan) objašnjava pojmove koje ovde nismo ni dotakli, kao npr. indeksne strukture podataka (B-stabla i B+-stabla) i algoritme spajanja, tako da će napredni korisnici imati dnevni uvid u ove alate za upravljanje podacima.

   Nažalost, na Internetu ne postoji mnogo informacija o SQL-u – lista navedena na kraju dokumenta je prilično sveobuhvatna (i svakako reprezentativna). Što se tiče knjiga, preporučujemo (za početni i srednji nivo čitalaca) "Oracle: The Complete Reference", koju je izdao Oracle, i "Understanding SQL", koju je izdao Sybex i koja govori uopšteno o SQL-u. Takođe, preporučujemo knjige izdavača O'Reilly Publishing i članke Džoa Selkoa (Joe Celko) za napredne korisnike. Da biste dobili informacije o određenim sistemima za upravljanje bazama podataka,

preporučujemo ediciju "Using" izdavača Que i knjige Alison Balter (Alison Balter).

7. Šta je *šema*? – Šema je logički skup tabela kao što je npr. baza podataka o antikvitetima iz našeg primera. Obično se za šemu podrazumeva da je sama baza podataka, mada baza podataka može sadržati više od jedne šeme. Na primer, *zvezdasta šema* je skup tabela u kojem jedna velika, centralna tabela sadrži sve važne podatke, a preko stranih ključeva povezana je sa *dimenzionim* tabelama koje sadrže detalje i koje se mogu koristiti u operacijama spajanja za dobijanje detaljnih izveštaja.

8. Oracle ima specijalnu ključnu reč, Decode, koja omogućava realizaciju uslovnog izvršavanja. Kako to funkcioniše? – Tehnički, Decode omogućava uslovne izlazne rezultate na osnovu vrednosti neke kolone ili funkcije. Sintaksa klauzule Decode je (prema ediciji Oracle: Complete Reference):

   ```
   SELECT ... DECODE (Kolona, Ako1, Tada1, [Ako2, Tada2, ...,] Inače)
   ...
   FROM ... ;
   ```

   `Kolona` je ime kolone, ili funkcija (zasnovana na koloni ili kolonama), a za svaku navedenu vrednost `Ako` dobija se kao rezultat odgovarajuća vrednost `Tada` ukoliko je vrednost `Ako` jednaka vrednosti kolone `Kolona`. Ako to nije slučaj ni za jednu vrednost `Ako`, kao rezultat se dobija vrednost `Inače`. Pogledajmo jedan primer:

   ```
   SELECT DISTINCT City,
   DECODE (City, 'Cincinnati', 'Queen City', 'New York', 'Big Apple',
   'Chicago', 'City of Broad Shoulders', City) AS Nickname
   FROM Cities;
   ```

   Rezultat ovog upita može da bude:

   ```
   City         Nickname
   ------------ ----------------------------
   Boston       Boston
   Cincinnati   Queen City
   Cleveland    Cleveland
   New York     Big Apple
   ```

   Prvi argument City označava ime kolone koja se koristi za testiranje. Drugi, četvrti itd. argumenti su vrednosti za individualne provere jednakosti (datim redosledom) sa svim vrednostima u koloni City. Treći, peti itd. argumenti su odgovarajući rezultat ukoliko je test tačan. Poslednji argument predstavlja rezultat koji se dobija ukoliko nijedan test nije tačan. U našem slučaju, dobija se samo vrednost kolone.

   Savet: Ukoliko ne želite ništa da se dobije kao vrednost, navedite vrednost NULL kao odgovarajuću vrednost. Na primer:

   ```
   SELECT DISTINCT City,
   DECODE (City, 'Cincinnati', 'Queen City', 'New York', 'Big Apple',
   'Chicago', 'City of Broad Shoulders', Null) AS Nickname
   FROM Cities;
   ```

Ako vrednost kolone City nije nijedna od onih koje su navedene, umesto nje kao u prethodnom primeru sada se ne dobija ništa:

```
City         Nickname
------------ ------------------------------
Boston
Cincinnati   Queen City
Cleveland
New York     Big Apple
```

9. Ranije je spomenut referencijalni integritet, ali da li to ima veze sa konceptom kaskadnog ažuriranja i brisanja podataka? – To je teško odgovoriti jer je ovaj koncept različito implementiran u različitim sistemima za upravljanje bazama podataka.

   Na primer, Microsoft SQL Server (verzija 7.0 i ranije verzije) zahteva da napišete „okidače“ (engl. *trigger*) da biste ovo implementirali (posetite sajt Yahoo SQL Club da biste pronašli sajt koji govori o ovoj temi). (Ipak, evo kratke definicije: okidač je neka SQL naredba pohranjena u bazi podataka koja omogućava da se automatski izvrši dati upit [obično je to neki „akcioni“ upit – Delete, Insert, Update] kada se desi određeni događaj u bazi podataka, kao što je npr. ažuriranje kolone). Microsoft Access (verovali ili ne) obavlja ovo ako SQL naredbu definišete na ekranu Relationships, ali će i dalje od vas zahtevati potvrdu. Oracle obavlja ovo automatski, ako definišete određeno „ograničenje“ (videti definiciju dalje u tekstu) za kolonu ključa.

   Zbog toga, ovde ćemo opisati samo koncept. (Podsetite se najpre naše diskusije o primarnim i stranim ključevima.)

   Koncept se sastoji u sledećem: ako se briše/ažurira vrednost primarnog ključa nekog reda, prilikom kaskadnog obavljanja te operacije se brišu/ažuriraju sve odgovarajuće vrednosti stranog ključa u drugim tabelama. (U slučaju operacije brisanja, naravno, brišu se celi redovi.)

   Obrnuto, brisanje/ažuriranje primarnog ključa ukoliko se briše/ažurira strani ključ, ne izvršava se obavezno: ograničenje ili okidač možda nisu definisani, može postojati odnos „jedan-na-više“, ili u samom sistemu za upravljanje bazama podataka ne postoji definisano pravilo za ovaj slučaj. Kao i obično, proverite tehničku dokumentaciju vašeg sistema za upravljanje bazama podataka.

   Na primer, ako definišete kolonu IDKupca kao primarni ključ tabele VlasniciAntikviteta i ako odredite da se u bazi podataka po brisanju primarnog ključa brišu redovi u tabeli Antikviteti prema koloni stranog ključa IDProdavca, tada ako obrišete red tabele VlasniciAntikviteta čiji je IDKupca jednak '01', obrisaće se u tabeli Antikviteti redovi sa komadima nameštaja krevet, orman, kutija za nakit (jer ih je '01' prodao). Naravno, ukoliko bi se vrednost '01' samo promenila u neku drugu, i vrednosti '01' kolone IDProdavca promenile bi se u tu novu vrednost.

10. Pokažite neki primer *spoljašnjeg spajanja.* – Sudeći po pitanjima koje dobijam, ovo je veoma tražen primer pa ću pokazati upit i za Oracle i za Access.

    Uzmimo primer tabele radnika (zbog jednostavnosti, radnicima su dodeljeni brojevi umesto imena):

Ime	Odeljenje
1	10
2	10
3	20
4	30
5	30

    i tabele odeljenja u nekoj radnoj organizaciji:

Odeljenje
10
20
30
40

    Pretpostavimo sada da želimo da spojimo tabele da bismo zajedno dobili sve radnike i sva odeljenja. Za ovo mora da se koristi spoljašnje spajanje koje uključuje „praznog“ radnika u odeljenju 40.

    Ako citiramo knjigu „Oracle 7: the Complete Reference“ o spoljašnjem spajanju, „zamislite da simbol (+), koji mora da sledi odmah iza kolone spajanja tabele, znači da treba dodati ekstra (prazan) red uvek kada nema podudarnosti“, tako da je upit za Oracle (+ ide uz tabelu Radnici čime se dodaje prazan red ako nema podudarnosti):

```
Select R.Ime, O.Odeljenje
From Odeljenje O, Radnici R
Where R.Odeljenje(+) = O.Odeljenje;
```

    Za Access, ovo je (spoljašnje) levo spajanje:

```
SELECT DISTINCTROW Radnici.Ime, Odeljenje.Odeljenje
FROM Odeljenje LEFT JOIN Radnici ON Odeljenje.Odeljenje = Radnici.
Odeljenje;
```

    Dobijeni rezultat je:

Ime	Odeljenje
1	10
2	10
3	20
4	30
5	30
	40

11. Koji su neki opšti saveti da bi se baze podataka i SQL upiti učinili boljim i bržim (*optimizovanim*)?

    - Ako možete, izbegavajte izraze u naredbi SELECT, kao npr. SELECT KolonaA + KolonaB i slično. *Optimizator upita* baze podataka, tj. deo sistema za upravljanje koji određuje najbolji način za izdvajanje podataka iz same baze podataka, izraze obrađuje na način koji obično zahteva više vremena za izdvajanje podataka nego ukoliko se kolone standardno biraju a izrazi programski izračunavaju.
    - Svedite broj kolona u klauzuli GROUP BY na najmanju meru.
    - Ako koristite spajanje, trudite se da kolone spajanja (u obe tabele) budu indeksirane.
    - Kada ste u dilemi, koristite indeks.
    - Ako ne obavljate višestruko brojanje niti neki složen upit, koristite COUNT(*) (broj redova u rezultatu upita) umesto COUNT(Ime_Kolone).

12. Šta je Dekartov proizvod? – Prosto rečeno, to je operacija spajanja bez klauzule WHERE. To daje sve redove prve tabele spojene sa svim redovima druge tabele. Najbolje da pogledamo primer:

```
SELECT *
FROM VlasniciAntikviteta, Porudžbine;
```

    Ovo daje kao rezultat:

VlasniciAntikviteta. IDVlasnika	**VlasniciAntikviteta. PrezimeVlasnika**	**VlasniciAntikviteta. ImeVlasnika**	**Porudžbine. IDVlasnika**	**Porudžbine. ŽeljeniKomad**
01	Jovanović	Branko	02	sto
01	Jovanović	Branko	02	pisaći sto
01	Jovanović	Branko	21	stolica
01	Jovanović	Branko	15	ogledalo
02	Simonović	Boban	02	sto

02	Simonović	Boban	02	pisaći sto
02	Simonović	Boban	21	stolica
02	Simonović	Boban	15	ogledalo
15	Lazarević	Pava	02	sto
15	Lazarević	Pava	02	pisaći sto
15	Lazarević	Pava	21	stolica
15	Lazarević	Pava	15	ogledalo
21	Aćimović	Jelena	02	sto
21	Aćimović	Jelena	02	pisaći sto
21	Aćimović	Jelena	21	stolica
21	Aćimović	Jelena	15	ogledalo
50	Filipović	Sima	02	sto
50	Filipović	Sima	02	pisaći sto
50	Filipović	Sima	21	stolica
50	Filipović	Sima	15	ogledalo

Broj redova rezultata je broj redova prve tabele puta broj redova druge tabele, a ovo se ponekad naziva i unakrsno spajanje.

Ako sada razmislite o tome, možete razumeti kako se dobija rezultat spajanja tabela. Pogledajte rezultat Dekartovog proizvoda, zatim pronađite redove u kojima su vrednosti kolone IDVlasnika jednake, i rezultat je ono što biste dobili u slučaju spajanja izjednačavanjem.

Naravno, stvarno obavljanje spajanja ne izvršava se na ovaj način jer bi to zahtevalo suviše mnogo memorije. Umesto toga, poređenja se obavljaju u ugneždenim petljama, ili poređenjem vrednosti u indeksima, a zatim se određuju redovi rezultata.

13. Šta je *normalizacija*? – Normalizacija je tehnika dizajna baze podataka kojom se na osnovu izvesnih kriterijuma određuje sadržaj tabela (tj. koje kolone treba da obuhvataju tabele i njihova struktura ključa). Osnovna ideja je da se eliminiše nepotrebno dupliranje ne-ključnih podataka u tabelama. O normalizaciji se obično govori u obliku *formi*, a ovde ćemo opisati samo prve tri forme mada se koriste i ostale, složenije forme (četvrta, peta, Boyce-Codd forma).

    Prva normalna forma odnosi se na grupisanje sličnih podataka u odvojene tabele i definisanje primarnog ključa za svaku tabelu.

    Smeštanje podataka u *drugu normalnu formu* sastoji se od premeštanja u druge tabele onih podataka koji su zavisni samo od dela ključa. Na primer, da smo imena vlasnika antikviteta čuvali i u tabeli komada nameštaja, to bi bilo suprotno načelu druge normalne forme jer bi ti podaci bili suvišni. Imena bi bila ponovljena za svaki posedovan komad nameštaja, pa su zbog toga imena izdvojena u zasebnu

tabelu. Imena sama po sebi nemaju nikakve veze sa komadima nameštaja, nego samo sa identitetom kupaca i prodavaca.

Treća normalna forma sastoji se od uklanjanja svih podataka u tabelama koji ne zavise jedino od primarnog ključa. Drugim rečima, treba zadržati samo podatke koji su zavisni od primarnog ključa, a one koji nisu treba premestiti u nove tabele i formirati primarni ključ za njih.

U svakoj formi normalizacije postoji izvesno ponavljanje, pa ukoliko su podaci u *3NF* (skraćenica za 3. normalna forma), oni su već u *1NF* i *2NF*. Sa aspekta dizajna baze podataka to znači da podatke treba organizovati tako da sve kolone koje ne pripadaju primarnom ključu zavise samo od *čitavog primarnog ključa*. Ako pogledamo primer naše baze podataka, videćemo da je način za dobijanje podataka iz te baze podataka onaj koji koristi spajanje tabela po zajedničkim ključnim kolonama.

Druga dva važna pravila kojih se treba pridržavati u dizajnu baze podataka su dobra, konzistentna, logička, puna imena za tabele i kolone, kao i upotreba punih reči u samoj bazi podataka. Ovo drugo pravilo nije ispoštovano u našem primeru baze podataka, pošto se za identifikaciju koriste numerički kodovi. Obično je najbolje ukoliko se može postići da su sami ključevi razumljivi. Na primer, bolji ključ bio bi onaj koji se sastoji od prva četiri slova prezimena i prvog slova imena vlasnika antikviteta, kao npr. JOVAB za Bobana Jovanovića (ili u slučaju sličnih imena i prezimena dodati brojeve na kraju da bi se ključevi razlikovali, na primer JOVAB1, JOVAB2 itd.).

14. Koja je razlika između *upita jednog reda* i *upita više redova* i zašto je važno znati razliku? – Pomenimo najpre ono što je očigledno: upit jednog reda je upit koji kao rezultat daje jedan red, dok je upit više redova onaj koji kao rezultat daje više od jednog reda. Da li je rezultat upita jedan red ili više njih zavisi potpuno od dizajna (ili šeme) tabela baze podataka. Prilikom pisanja upita važno je poznavanje šeme da biste pravilno struktuirali SQL naredbe i dobili željeni rezultat (odnosno, jedan red ili više njih). Na primer, da biste bili sigurni da neki upit nad tabelom VlasniciAntikviteta daje jedan red kao rezultat, koristite uslov jednakosti nad kolonom primarnog ključa IDVlasnika.

    Odmah na um padaju tri razloga zašto je ovo važno. Prvo, dobijanje više redova kada očekujete samo jedan, ili obrnuto, može značiti da je upit pogrešan, da je baza podataka nekompletna, ili prosto da ste saznali nešto novo o podacima. Drugo, ako je u pitanju operacija brisanja ili ažuriranja, treba biti siguran da napisana naredba obavlja operaciju na tačno željeni red (ili redove), jer ćete inače možda obrisati ili ažurirati više redova nego što ste nameravali. Treće, upiti na jeziku Embedded SQL moraju se pažljivo pisati kada je reč o broju redova rezultata. Ako napišete upit jednog reda, za programsku logiku je možda dovoljna samo jedna SQL naredba. Sa druge strane, za upite više redova neophodno je koristiti naredbu FETCH i, sasvim verovatno, naredbu ponavljanja da bi se obradio svaki red rezultata upita.

15. Koji je jednostavan pristup dizajnu baze podataka? – Ovde je data skraćena verzija saveta koje preporučuju Džon Frejm (John Frame, jframe@jframe.com) i Ričard Fridman (Richard Freedman, rfreedm@voicenet.com). O nekim konceptima koji se pominju ponovo se govori u sledećem pitanju.

    Prvo, najpre napravite spisak važnih stvari (entiteti), a u njega uključite i one koje početno niste mislili da su važne. Drugo, povucite granicu između svaka dva entiteta koja su na bilo koji način povezana, osim da dva entiteta ne mogu biti ne povezana bez nekog „pravila“, kao npr. porodice imaju decu, radnici rade u nekom odeljenju i slično. Prema ovome stavite „veze“ u rombove a „entitete“ u kvadrate. Treće, slika sada treba da ima mnogo kvadrata (entiteta) koji su sa drugim entitetima povezani preko rombova (kvadrat koji sadrži entitet sa linijom koja ide do romba koji predstavlja odnos i zatim druga linija od tog romba do drugog entiteta). Četvrto, svakom kvadratu i rombu dodajte oznaku, na primer putnik – avionska kompanija – let. Peto, svakom kvadratu i rombu dodajte njegove atribute (osoba ima ime, račun ima broj), mada neki odnosi ne poseduju atribute (roditelj samo ima dete). Šesto, sve na takvoj slici što ima atribute predstavlja novu tabelu, pri čemu ukoliko su dva entiteta u odnosu koji nema atribute, onda postoji samo strani ključ između tabela. Sedmo, u opštem slučaju treba težiti da neke dve tabele ne ponavljaju podatke. Na primer, ako kupac ima ime i nekoliko adresa, onda će za svaku njegovu adresu biti ponovljeno ime, prezime i drugi lični podaci. Zbog toga, izdvojite ime kupca u jednoj tabeli, a sve njegove adrese u drugoj. Osmo, svaki red treba da bude jedinstven u odnosu na druge – Fridman predlaže broj sa automatskim uvećavanjem kao primarni ključ, tako da se generiše novi, jedinstveni broj za svaki novododati red. Deveto, ključ je bilo koji način kojim se jednoznačno identifikuje red u tabeli, na primer ime i prezime zajedno su dobri kao „kompozitni ključ“.

16. Šta su *odnosi*? – To je još jedno pitanje iz oblasti dizajna baze podataka. Terminom „odnos“ (ili „relacija“) obično se ukazuje na međusobne odnose između primarnih i stranih ključeva tabela. Ovaj koncept je važan jer se njime određuje koje kolone jesu ili nisu primarni ili strani ključevi tabela relacione baze podataka. Možda se čuli za **dijagram odnosa entiteta** (ERD, engl. *Entity-Relationship Diagram*) koji grafički predstavlja tabele šeme baze podataka pri čemu su odgovarajuće kolone tabela povezane linijama. Primer jednog ERD dijagrama možete naći na kraju ovog odeljka. Takođe, posetite neki od sajtova koji su navedeni na kraju da biste dobili detaljnije informacije u vezi sa ovim pitanjem, pošto postoji mnogo načina za crtanje ERD dijagrama. Ali, razmotrimo najpre sve moguće vrste odnosa.

    Odnos jedan-na-jedan znači da postoje kolone primarnog i stranog ključa i da za svaku vrednost primarnog ključa postoji **jedna** vrednost stranog ključa. Na primer, u tabeli TabelaAdresaRadnika dodajmo kolonu IDRadnika.Tada je tabela TabelaAdresaRadnika u relaciji sa tabelom TabelaPrimanjaRadnika preko kolone IDRadnika. Preciznije, svaki zaposleni iz tabele TabelaAdresaRadnika **ima** primanja predstavljena jednim redom u tabeli TabelaPrimanjaRadnika. Iako je ovo iskonstruisan primer, postoji odnos „1-1“. Obratite pažnju na reč „ima“ koja

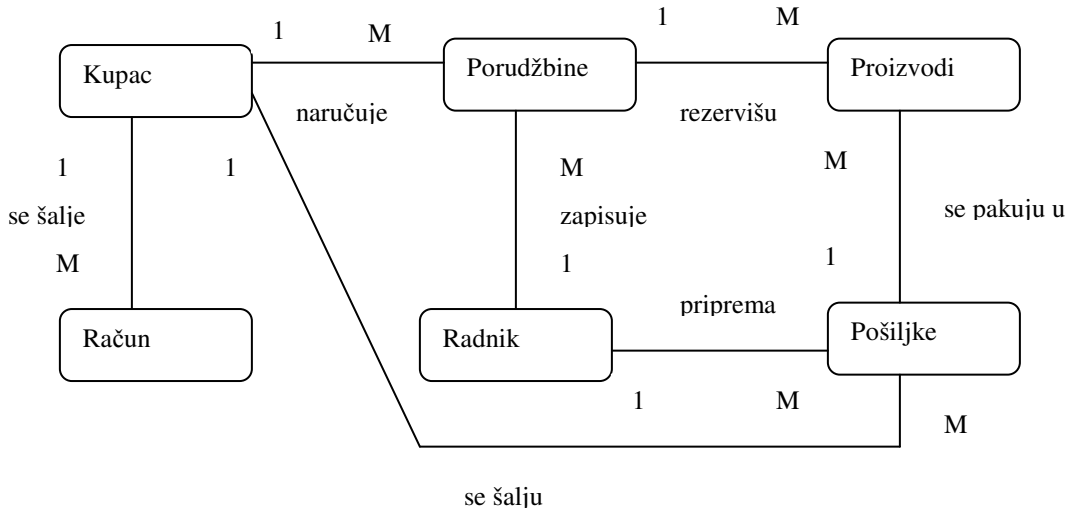
je napisana crnim slovima – pri izražavanju odnosa, važno je taj odnos opisati glagolom.

Druga dva odnosa mogu, ali ne moraju, koristiti logička ograničenja primarnog i stranog ključa. Prvi od njih je *odnos jedan-na-više* („1-M“). To znači da za svaku vrednost neke kolone jedne tabele postoje **jedna ili više** odgovarajućih vrednosti u drugoj tabeli. Pri ovome se mogu definisati ograničenja ključa ili se jednostavno može koristiti neka identifikaciona kolona za uspostavljanje odnosa. Jedan primer bi bio da za svaki IDVlasnika u tabeli VlasniciAntikviteta, postoje jedan ili više (nula je takođe dopuštena) komada nameštaja **kupljenih** u tabeli Antikviteti (glagol: kupiti).

Na kraju, *odnos više-na-više* („M-M“) u opštem slučaju ne uključuje ključeve već identifikacione kolone. Ovaj odnos ukazuje da postoji međusobni odnos dve kolone u različitim tabelama, i da za svaku vrednost u jednoj koloni postoje jedna ili više vrednosti odgovarajuće kolone druge tabele ili, češće, da su dve tabele u odnosu „1-M“ jedna prema drugoj (dva odnosa, po jedan odnos „1-M“ jedne tabele prema drugoj). Jedan (loš) primer ovog češćeg slučaja bila bi baza podataka projekata na kojima rade zaposleni, pri čemu jedna tabela sadrži jedan red za svakog radnika i dodeljeni projekat, a druga tabela sadrži jedan red za svaki projekat sa jednim od pridruženih radnika. Tada, u prvoj tabeli postojaće više redova za svakog radnika, po jedan za svaki dodeljen projekat, i više redova u drugoj tabeli za svaki projekat, po jedan za svakog radnika koji je dodeljen određenom projektu. Između ovih tabela postoji odnos M-M: svaki radnik u prvoj tabeli ima **više** dodeljenih projekata iz druge tabele, a na svakom projektu u drugoj tabeli radi **više** radnika iz prve tabele. Ovo je samo vrh ledenog brega o ovoj temi – detaljnije informacije možete dobiti na nekom od sajtova koji su navedeni na kraju.

Primer dijagrama odnosa entiteta

Pojednostavljena slika aplikacije za praćenje porudžbina



Odnos dva entiteta čita se: pošiljka „se šalje“ kupcu.

Za jednog kupca može biti jedna ili nekoliko („više“) pošiljki.

17. Koje se neke od nestandardnih mogućnosti SQL-a? – Pošto je ovo veoma često pitanje, posvećen mu je ceo sledeći odeljak.

Nestandardni SQL

- Klauzule INTERSECT i MINUS su slične klauzuli UNION, osim što INTERSECT proizvodi redove koji se pojavljuju u rezultatima oba upita, dok MINUS proizvodi redove koji se pojavljuju u rezultatu prvog ali ne i drugog upita.
- Mogućnosti prilikom generisanja izveštaja: klauzula COMPUTE navodi se na kraju upita da bi se rezultat neke agregatne funkcije prikazao na kraju izveštaja, na primer COMPUTE SUM (CENA). Druga opcija je primena logike tačke prekida: definišite tačku prekida kojom se rezultat upita na osnovu neke kolone deli na grupe, kao npr. BREAK ON IDKUPCA. Da biste zatim prikazali neki rezultat iza izveštaja za grupu, koristite COMPUTE SUM (CENA) ON IDKUPCA. Ako ste, na primer, koristili sve tri klauzule (najpre BREAK, zatim COMPUTE ON BREAK i na kraju izračunavanje ukupne sume pomoću COMPUTE), dobili biste izveštaj o antikvitetima po grupama na osnovu njihovog ID broja vlasnika, zatim iza svake grupe ukupnu cenu svih antikviteta određenog vlasnika i na samom kraju sumu svih cena, a sve to uz zaglavlja i kontrolne redove koje prikazuje SQL.
- Pored prethodno navedenih agregatnih funkcija, neki sistemi za upravljanje bazama podataka sadrže dodatne funkcije koje se mogu koristiti u naredbi SELECT, ali pod uslovom da se ove funkcije primenjuju na pojedinačne vrednosti (ne grupe) upita jednog reda (neke znakovne funkcije dopuštaju rezultate koje se sastoje od više redova). Takođe, ove funkcije moraju se koristiti samo sa odgovarajućim tipovima podataka. Evo nekih **matematičkih funkcija**:

ABS(X)	Apsolutna vrednost broja X.
CEIL(X)	X je decimalna vrednost koja se zaokružuje na prvi veći ceo broj.
FLOOR(X)	X je decimalna vrednost koja se zaokružuje na prvi manji ceo broj.
GREATEST(X,Y)	Kao rezultat daje veću od dve navedene vrednosti.
LEAST(X,Y)	Kao rezultat daje manju od dve navedene vrednosti.
MOD(X,Y)	Kao rezultat daje ostatak od celobrojnog deljenja X sa Y.
POWER(X,Y)	Kao rezultat daje broj X dignut na stepen Y.

ROUND(X,Y)	Zaokružuje X na Y decimalnih mesta. Ako Y nije naveden, X se zaokružuje na najbliži ceo broj.
SIGN(X)	Kao rezultat daje minus ako je X < 0, inače plus.
SQRT(X)	Kao rezultat daje kvadratni koren od X.

Znakovne funkcije

LEFT(<string>,X)	Kao rezultat daje X najlevijih znakova stringa.
RIGHT(<string>,X)	Kao rezultat daje X najdesnijih znakova stringa.
UPPER(<string>)	Pretvara sadržaj celog stringa u velika slova.
LOWER(<string>)	Pretvara sadržaj celog stringa u mala slova.
INITCAP(<string>)	Pretvara sadržaj stringa tako da je svaka reč sa početnim velikim slovom.
LENGTH(<string>)	Kao rezultat daje broj znakova u stringu.
<string>||<string>	Spaja dva stringa u jedan string, pri čemu iza prvog odmah sledi drugi string.
LPAD(<string>,X,'*')	Dodaje stringu sa leve strane onoliko znakova * (ili znak naveden unutar apostrofa) koliko je potrebno da bi se dobio string dužine X znakova.
RPAD(<string>,X,'*')	Dodaje stringu sa desne strane onoliko znakova * (ili znak naveden unutar apostrofa) koliko je potrebno da bi se dobio string dužine X znakova.
SUBSTR(<string>,X,Y)	Izdvaja Y znakova iz stringa počevši od pozicije X.
NVL(<kolona>,<vrednost>)	Zamenjuje sve vrednosti NULL u koloni <kolona> vrednostima <vrednost>.

Pregled sintakse naredbi

U ovom delu predstavljen je opšti oblik naredbi o kojima smo prethodno govorili, kao i još nekih važnijih naredbi koje su opisane u nastavku. Zapamtite da sve ove naredbe možda nisu raspoložive na vašem sistemu – obavezno proverite sistemsku dokumentaciju.

`ALTER TABLE` `<IME TABELE> ADD|DROP|MODIFY (SPECIFIKACIJA KOLONE ... videti CREATE TABLE);` – Služi za dodavanje i brisanje kolone ili kolona iz tabele, ili za menjanje specifikacije (tip podataka itd.) postojeće kolone. Ovom naredbom može se menjati i fizička specifikacija tabele (kako se tabela smešta itd.), ali to zavisi od konkretnog sistema za upravljanje bazama podataka. Takođe, ove fizičke specifikacije koriste se prilikom početnog formiranja tabele naredbom CREATE TABLE. Pored toga, jednom naredbom ALTER TABLE može se obaviti samo jedna akcija – dodavanje, brisanje ili modifikacija.

`COMMIT;` – Služi da se promene podataka koje su urađene od poslednje naredbe COMMIT potvrde, odnosno učine trajnim, u bazi podataka (ovo se naziva *transakcija*).

`CREATE [UNIQUE] INDEX` `<IME INDEKSA> ON <IME TABELE> (<LISTA KOLONA>);`

CREATE TABLE <IME TABELE> (<IME KOLONE> <TIP PODATAKA> [(<VELIČINA>)] <OGRANIČENJE KOLONE>, ... druge kolone); (važi i za naredbu ALTER TABLE) – VELIČINA se koristi samo za neke tipove podataka, a ograničenja kolone obuhvataju sledeće opcije (koje automatski proverava sistem za upravljanje bazama podataka i u slučaju greške izdaje odgovarajuću poruku):

1. NULL ili NOT NULL.
2. UNIQUE obezbeđuje da nijedna dva reda nemaju istu vrednost u ovoj koloni.
3. PRIMARY KEY određuje datu kolonu kao primarni ključ (koristi se samo ako se primarni ključ sastoji od jedne kolone, inače se klauzula PRIMARY KEY (kolona, kolona ...) navodi iza definicije poslednje kolone.
4. CHECK omogućava proveravanje nekog uslova prilikom dodavanja ili ažuriranja podataka date kolone, na primer CHECK(CENA 0) obezbeđuje da sistem pre prihvatanja neke vrednosti proverava da li je vrednost kolone Cena veća od nule. Ovo je ponekad implementirano pomoću naredbe CONSTRAINT.
5. DEFAULT upisuje podrazumevanu vrednost u bazu podataka ako se dodaje red bez date vrednosti za odgovarajuću kolonu, na primer PRINADLEŽNOSTI INTEGER DEFAULT = 10000
6. FOREIGN KEY ima slično značenje kao PRIMARY KEY, osim što iza sledi REFERENCES <IME TABELE> (<IME KOLONE>), čime se ukazuje na referencijalni primarni ključ.

CREATE VIEW <IME TABELE> AS <UPIT>;

DELETE FROM <IME TABELE> WHERE <USLOV>;

INSERT INTO <IME TABELE> [(<LISTA KOLONA>)] VALUES (<LISTA VREDNOSTI>);

ROLLBACK; – Služi za poništavanje svih promena u bazi podataka koje su učinjene od trenutka poslednje naredbe COMMIT. Pazite, neki transakcioni sistemi vrše automatsko potvrđivanje tako da naredba ROLLBACK možda nema dejstva.

SELECT [DISTINCT|ALL] <LISTA KOLONA, FUNKCIJA, KONSTANTI ITD.>
FROM <LISTA TABELA ILI POGLEDA>
[WHERE <USLOV(I)>]
[GROUP BY <KOLONE GRUPISANJA>]
[HAVING <USLOV>]
[ORDER BY <KOLONE SORTIRANJA> [ASC|DESC]]; – ASC|DESC određuju da li se sortiranje vrši u rastućem (ASC) ili opadajućem (DESC) redosledu.

UPDATE <IME TABELE>
SET <IME KOLONE> = <VREDNOST>
[WHERE <USLOV>]; – Ako je klauzula WHERE izostavljena, ažuriraju se sve kolone prema klauzuli SET.

Vežbanja

Upiti

Koristeći prethodne tabele iz ovog dokumenta, navedite SQL naredbe za sledeće primere:

1. Prikažite sve porudžbine antikviteta kao i prezimena i imena osoba koja su ih naručila.
2. Prikažite svaku kolonu tabele TabelaPrimanjaRadnika u abecednom redu po koloni Položaj, zatim IDRadnika.
3. Prikažite godišnji budžet za kolonu Prinadležnosti tabele TabelaPrimanjaRadnika.
4. Koristeći operator IN prikažite imena svih vlasnika stolica.
5. Prikažite imena vlasnika antikviteta koji nisu kupili ništa.
6. Prikažite imena onih koji su poručili neki antikvitet, bez duplikata (Savet: obratite pažnju na redosled tabela u klauzuli FROM).
7. Obrišite sve kupovine antikviteta Bobana Simonovića (Savet: Bobanov ID broj je 02).
8. Dodajte kupovinu stolice za ljuljanje za Jelenu Aćimović (Savet: Jelenin ID broj je 21).
9. Formirajte tabelu pod nazivom Radnici sa kolonama IDRadnika (ne brinite oko nula na kraju), Ime i Prezime.
10. (teže pitanje) Prikažite godišnji budžet za kolonu Plata grupisan po položaju radnika iz tabele TabelaPrimanjaRadnika (Savet: pokušajte sa klauzulom GROUP BY).

Baze podataka

11. Koja vrsta odnosa postoji između tabele VlasniciAntikviteta i tabele Antikviteti?
12. Ako nemate primarni ključ u tabeli, koju vrstu kolone treba dodati da bi tabela dobila primarni ključ?
13. Koja funkcija omogućava da vrednosti NULL dobijene izvršavanjem naredbe SELECT zamenite nekom datom vrednošću?
14. Prilikom korišćenja jezika Embedded SQL, šta treba da formirate da biste mogli da obradite red po red rezultata upita koji se sastoji od više redova?
15. Ako su sve kolone svih tabela zavisne samo od vrednosti primarnog ključa u svakoj tabeli, u kojoj se normalnoj formi nalazi baza podataka?
16. Kako se naziva postupak sistema za upravljanje bazom podataka kojim se automatski ažuriraju vrednosti stranih ključeva u drugim tabelama kada se promeni vrednost primarnog ključa?
17. Koji objekat baze podataka omogućava brz pristup podacima koji se čuvaju u redovima neke tabele?
18. Koja se SQL naredba koristi za promenu atributa neke kolone?
19. U naredbi CREATE TABLE, šta znači kada se neka kolona označi kao NOT NULL?
20. Ako želite da postavite upit na osnovu drugih upita a ne tabela, kako moraju da se ovi drugi upiti definišu?

Odgovori

(Pitanja mogu da imaju više ispravnih odgovora.)

1.

```
SELECT VlasniciAntikviteta.PrezimeVlasnika,
VlasniciAntikviteta.ImeVlasnika, Porudžbine.ŽeljeniKomad
FROM VlasniciAntikviteta, Porudžbine
WHERE VlasniciAntikviteta.IDVlasnika= Porudžbine.IDVlasnika;
```

ili

```
SELECT VlasniciAntikviteta.PrezimeVlasnika,
VlasniciAntikviteta.ImeVlasnika, Porudžbine.ŽeljeniKomad
FROM VlasniciAntikviteta RIGHT JOIN Porudžbine ON
VlasniciAntikviteta.IDVlasnika= Porudžbine.IDVlasnika;
```

2.

```
SELECT *
FROM TabelaPrimanjaRadnika
ORDER BY Položaj, IDRadnika;
```

3.

```
SELECT Sum(Prinadležnosti)
FROM TabelaPrimanjaRadnika;
```

4.

```
SELECT PrezimeVlasnika, ImeVlasnika
FROM VlasniciAntikviteta, Antikviteti
WHERE KomadNameštaja IN ('stolica')
AND VlasniciAntikviteta.IDVlasnika= Antikviteti.IDKupca;
```

5.

```
SELECT PrezimeVlasnika, ImeVlasnika
FROM VlasniciAntikviteta
WHERE IDVlasnika NOT IN
(SELECT IDVlasnika
FROM Porudžbine);
```

6.

```
SELECT DISTINCT PrezimeVlasnika, ImeVlasnika
FROM Porudžbine, VlasniciAntikviteta
WHERE VlasniciAntikviteta.IDVlasnika= Porudžbine.IDVlasnika;
```

ili u JOIN notaciji:

```
SELECT DISTINCT VlasniciAntikviteta.PrezimeVlasnika,
VlasniciAntikviteta.ImeVlasnika
FROM VlasniciAntikviteta RIGHT JOIN Porudžbine ON
```

```
VlasniciAntikviteta.IDVlasnika= Porudžbine.IDVlasnika;
```

7.
```
DELETE FROM PORUDŽBINE
WHERE IDVLASNIKA= 02;
```

8.
```
INSERT INTO PORUDŽBINE VALUES (21, 'stolica za ljuljanje');
```

9.
```
CREATE TABLE RADNICI
(IDRadnika INTEGER NOT NULL,
Ime CHAR(40) NOT NULL,
Prezime CHAR(40) NOT NULL);
```

10.
```
SELECT Položaj, Sum(Plata)
FROM TabelaPrimanjaRadnika
GROUP BY Položaj;
```

11.
Jedan-na-više.

12.
Celobrojni identifikacioni broj; najbolji je ID sa automatskim uvećanjem.

13.
NVL.

14.
Kursor.

15.
Treća normalna forma.

16.
Kaskadno ažuriranje.

17.
Indeks.

18.
ALTER TABLE.

19.
Neka vrednost u ovoj koloni mora biti upisana u svakom redu tabele.

20.
Kao pogledi.

Važniji sajtovi o SQL-u, bazama podataka i sličnim temama

SQL Reference Page
Ask the SQL Pro
Programmer's Source
inquiry.com
DB Ingredients
SQL Trainer S/W
Web Authoring
DBMS Lab/Links
SQL FAQ
Query List
SQL Practice Site
SQL Course II
Database Jump Site
Programming Tutorials on the Web
PostgreSQL
Adobe Acrobat
Access on the Web
A Good DB Course
Tutorial Page
Intelligent Enterprise Magazine
miniSQL
SQL for DB2 Book
SQL Server 7
SQL Reference/Examples
SQL Topics
Lee's SQL Tutorial
Oracle/SQL Server Cram Session
Data Warehousing Homepage
MIT SQL for Web Nerds
Online Oracle Documentation

Pripremio Dr. Dejan Živković