7. Data Manipulation Language (DML)

Jezik za rad sa podacima (Data manipulation Language (DML)) služi za umetanje, brisanje i ažuriranje podataka u bazi.

7.1. Umetanje novih redova u tabelu – INSERT sintaksa

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] *tbl_name* [(*col_name*,...)]
VALUES ({*expr* | DEFAULT},...),(...),...
[ON DUPLICATE KEY UPDATE *col_name*=*expr*, ...]

ili:

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] *tbl_name*
SET *col_name*={*expr* | DEFAULT}, ...
[ON DUPLICATE KEY UPDATE *col_name*=*expr*, ...]

ili:

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] *tbl_name* [(*col_name*,...)]
SELECT ...
[ON DUPLICATE KEY UPDATE *col_name*=*expr*, ...]

Komanda INSERT umeće redove u postojeću tabelu. INSERT ... VALUES i INSERT ... SET forme iskaza umeću redove na osnovu eksplicitno specificiranih vrednosti. INSERT ... SELECT forma umeće redove selektovane iz jedne ili više tabela.
tbl_name je naziv tabele u koju se umeću redovi. Kolone za koje iskaz obezbeđuje vrednosti mogu biti specificirane na sledeći način:

- Može se proslediti lista naziva kolona razdvojenih zarezima iza naziva tabele. U ovom slučaju vrednost za svaku imenovanu kolonu u listi mora biti obezbeđena VALUES listom ili SELECT iskazom.
- Ako se ne specificira lista naziva kolona za INSERT ... VALUES ili INSERT … SELECT, vrednosti za svaku kolonu tabele moraju biti obezbeđene VALUES listom ili SELECT iskazom
- SET klauzula omogućava da se izričito zadaju kolone u koje treba umetnuti podatke

Kada se koristi prvi oblik komande INSERT (INSERT ... VALUES) za svaki novi red tabele se mora zadati lista vrednosti koje su poređane istim redosledom kao odgovarajuće ciljne kolone tabele. Ovaj oblik omogućava da se jednom komandom INSERT umetne više novih redova u tabelu (više lista vrednosti u zagradama međusobno razdvojenih zarezima.

INSERT INTO *tbl_name* (*a*,*b*,*c*) VALUES (1,2,3),(4,5,6),(7,8,9);

Prethodni iskaz umeće tri reda u tabelu.
Sledeći iskaz je loš zato što broj vrednosti u listi ne odgovara broju naziva kolona:

INSERT INTO *tbl_name* (*a*,*b*,*c*) VALUES (1,2,3,4,5,6,7,8,9);

Drugi oblik komande INSERT omogućava da se izričito zadaju kolone u koje treba umetnuti podatke. Ovaj oblik omogućava da se unese samo jedan red po komandi, ali ne moraju se zadati vrednosti za sve kolone.

INSERT INTO *firme*
SET *firma* = 1,
naziv_firme = 'BALKAN';
Ovde se dodaje novi red u tabelu *firme*, ali se prosleđuju podaci samo za kolone *firma* i *naziv_firme*.

Primer za treći oblik komande INSERT (INSERT ... SELECT):
INSERT INTO *fakture* (*sifra_firme*)
SELECT *firma* FROM *firme* WHERE *naziv_firme* = 'STIL';
Ovde se dodaje novi red u tabelu *fakture* i prosleđuje se samo vrednost za kolonu *sifra_firme*, a ta vrednost se dobija iz tabele *firme* tako što se selektuje vrednost za kolonu *firma* u redu u kome je vrednost za kolonu *naziv_firme* STIL. Ovo znači dodavanje nove fakture za firmu STIL.

Kolone za koje nisu zadate vrednosti će preuzeti podrazumevane vrednosti (u kolonama u kojima su takve vrednosti definisane) ili vrednost NULL.
Komanda INSERT ima nekoliko neobaveznih odredaba:

- Može se zadati da se komanda INSERT izvršava sa niskim prioritetom (opcija LOW_PRIORITY), ili da se izvršavanje odloži (opcija DELAYED). Obe opcije čine da se umetanje podataka odloži dok više ne bude ni jednog klijenta koji pokušava da učita podatke iz tabele. Razlika između ove dve opcije je u tome što opcija LOW_PRIORITY blokira klijentski program koji umeće podatke, dok opcija DELAYED to ne čini.
- Opcija HIGH PRIORITY zaustavlja ostale istovremene pokušaje umetanja redova.
- Opcija IGNORE je korisna prvenstveno kada se umeće više redova istovremeno. Standardno ponašanje je takvo da ukoliko jedan od redova koje korisnik pokušava da umetne izazove grešku tipa dupliran primarni ključ ili duplirana vrednost u koloni koja prihvata samo jedinstvene vrednosti, dolazi do greške a cela operacija umetanja se poništava. Ako se upotrebi opcija IGNORE greška se zanemaruje, a postupak umetanja se nastavlja sa podacima iz sledećeg reda.
- Može se izričito zadati da kolona treba da preuzme svoju podrazumevanu vrednost ako se umesto vrednosti za kolonu zada opcija DEFAULT.
- Opcija ON DUPLICATE KEY UPDATE pruža elegantno rešenje dupliranog primarnog ključa ili duplirane jedinstvene vrednosti. Iza ove opcije sledi komanda UPDATE koja menja postojeću vrednost primarnog ključa ili postojeću jedinstvenu vrednost u koloni tako da se ona više ne "sudara" sa podacima iz novog reda.

Moguće je specificirati izraz *expr* koji će obezbediti vrednost za kolonu.

INSERT INTO *tbl_name* (*col1*,*col2*) VALUES (15,*col1**2);
Dodaje se jedan red tabeli sa vrednostima za kolonu *col1* 15 i kolonu *col2* 15*2=30.

Prosleđivanjem NULL vrednosti koloni koja je deklarisana kao NOT NULL, vrednost u koloni se podešava na implicitnu podrazumevanu vrednost za tip podataka u toj koloni. Ovo je 0 za numeričke tipove, prazan string (' ') za znakovne tipove i "nulta" vrednost za datumske i vremenske tipove podataka.
Za kolone tipa AUTO_INCREMENT je moguće izričito zadati vrednost ili prepustiti MySQL-u da sam generiše neku vrednost.

7.1.1. Umetanje redova u tabele baze podataka poslovanje

Prilikom unošenja podataka u tabele baze podataka **poslovanje** treba voditi računa o tome da se podaci prvo unose u primarne (roditelj) tabele pa onda u sekundarne (dete) tabele. Podaci se mogu unositi u tabele ovim redosledom: proizvodi, firme, fakture, detalji_fakture.

Unos podataka u tabelu **proizvodi**:

```
INSERT INTO proizvodi VALUES
 (1, 'Cetka', 'kom.'),
 (2, 'Lak', 'lit.'),
 (3, 'Stiropor', 'm2'),
 (4, 'Gips', 'kg'),
 (5, 'Destilovana voda', 'kg'),
 (6, 'Smirgla', 'kom.');
```

maticni_broj	ime	jed_mere
1	Cetka	kom.
2	Lak	lit.
3	Stiropor	m2
4	Gips	kg
5	Destilovana voda	kg
6	Smirgla	kom.

Unos podataka u tabelu **firme**:

```
INSERT INTO firme VALUES
 (1, 'BALKAN', 'Nis', 'Mokranjceva 13', '018-522-854'),
 (2, 'STIL', 'Beograd', 'Takovska 10', '011-562-365'),
 (3, 'KOKOMAX', 'Nis', 'Dusanova 33', '018-352-666'),
 (4, 'HELIO', 'Subotica', 'Nikole Tesle 55', '027-555-125');
```

firma	naziv_firme	mesto	adresa	telefon
1	BALKAN	Nis	Mokranjceva 13	018-522-854
2	STIL	Beograd	Takovska 10	011-562-365
3	KOKOMAX	Nis	Dusanova 33	018-352-666
4	HELIO	Subotica	Nikole Tesle 55	027-555-125

Unos podataka u tabelu **fakture**:

```
INSERT INTO fakture VALUES
 (1, 2, '2006-10-25', '1'),
 (2, 4, '2006-10-28', '1'),
 (3, 1, '2006-10-25', '2'),
 (4, 3, '2006-10-29', '2'),
 (5, 3, '2006-10-25', '1'),
 (6, 1, '2006-11-06', '2'),
 (7, 4, '2006-11-04', '2'),
 (8, 2, '2006-10-23', '1'),
 (9, 1, '2006-11-02', '2'),
 (10, 4, '2006-10-08', '1'),
 (11, 3, '2006-10-15', '1');
```

sifra_fakture	sifra_firme	datum	ulaz_izlaz
1	2	25.10.2006	1
2	4	28.10.2006	1
3	1	25.10.2006	2
4	3	29.10.2006	2
5	3	25.10.2006	1
6	1	6.11.2006	2
7	4	4.11.2006	2
8	2	23.10.2006	1
9	1	2.11.2006	2
10	4	8.10.2006	1
11	3	15.10.2006	1

Unos podataka u tabelu **detalji_fakture**:

```
INSERT INTO detalji_fakture VALUES
 (1, 1, 1, 3, '50.00', '200.00'),
 (2, 2, 1, 5, '50.00', '25.00'),
 (3, 2, 2, 6, '35.00', '60.00'),
 (4, 2, 3, 3, '40.00', '220.00'),
 (5, 3, 1, 1, '20.00', '75.00'),
 (6, 3, 2, 2, '35.00', '300.00'),
 (7, 3, 3, 3, '60.00', '200.00'),
 (8, 3, 4, 4, '50.00', '170.00'),
 (9, 4, 1, 4, '50.00', '190.00'),
 (10, 4, 2, 1, '50.00', '80.00'),
 (11, 5, 1, 5, '120.00', '20.00'),
 (12, 5, 2, 4, '100.00', '150.00'),
 (13, 5, 3, 6, '25.00', '80.00'),
 (14, 6, 1, 4, '75.00', '200.00'),
 (15, 6, 2, 6, '50.00', '85.00'),
 (16, 6, 3, 2, '24.00', '330.00'),
 (17, 6, 4, 3, '70.00', '230.00'),
 (18, 6, 5, 5, '80.00', '30.00'),
 (19, 7, 1, 2, '26.00', '310.00'),
 (20, 8, 1, 6, '45.00', '65.00'),
 (21, 8, 2, 3, '50.00', '250.00'),
 (22, 9, 1, 1, '50.00', '70.00'),
 (23, 9, 2, 5, '25.00', '25.00'),
 (24, 9, 3, 2, '14.00', '320.00'),
 (25, 10, 1, 4, '150.00', '210.00'),
 (26, 10, 2, 5, '200.00', '35.00'),
 (27, 10, 3, 1, '125.00', '90.00'),
 (28, 10, 4, 2, '100.00', '280.00'),
 (29, 10, 5, 6, '60.00', '55.00'),
 (30, 11, 1, 1, '5.00', '75.00');
```

id	faktura	red_br	proizvod	kolicina	dan_cena
1	1	1	3	50	200
2	2	1	5	50	25
3	2	2	6	35	60
4	2	3	3	40	220
5	3	1	1	20	75
6	3	2	2	35	300
7	3	3	3	60	200
8	3	4	4	50	170
9	4	1	4	50	190
10	4	2	1	50	80
11	5	1	5	120	20
12	5	2	4	100	150
13	5	3	6	25	80
14	6	1	4	75	200
15	6	2	6	50	85
16	6	3	2	24	330
17	6	4	3	70	230
18	6	5	5	80	30
19	7	1	2	26	310
20	8	1	6	45	65
21	8	2	3	50	250
22	9	1	1	50	70
23	9	2	5	25	25
24	9	3	2	14	320
25	10	1	4	150	210
26	10	2	5	200	35
27	10	3	1	125	90
28	10	4	2	100	280
29	10	5	6	60	55
30	11	1	1	5	75

7.2. Umetanje novih redova u tabelu – REPLACE sintaksa

Komanda REPLACE deluje slično komandi INSERT, s tom razlikom što ako dođe do dupliranja ključa novi red koji korisnik umeće zamenjuje postojeći red.

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] *tbl_name* [(*col_name*,...)]
VALUES ({*expr* | DEFAULT},...),(...),...

ili:

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] *tbl_name*
SET *col_name*={*expr* | DEFAULT}, ...

ili:

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] *tbl_name* [(*col_name*,...)]
SELECT ...

Može se primetiti da je sintaksa slična sintaksi komande INSERT.

7.3. Brisanje redova iz tabele – DELETE sintaksa

Sintaksa za jednu tabelu:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM *tbl_name*
[WHERE *where_condition*]

[ORDER BY ...]
[LIMIT *row_count*]

Sintaksa za više tabela:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
tbl_name[.*] [, *tbl_name*[.*]] ...
FROM *table_references*
[WHERE *where_condition*]

ili:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM *tbl_name*[.*] [, *tbl_name*[.*]] ...
USING *table_references*
[WHERE *where_condition*]

Kod sintakse za jednu tabelu DELETE iskaz briše redove iz tabele *tbl_name* i vraća broj obrisanih redova. Ako se upotrebi WHERE klauzula brišu se samo redovi koji zadovoljavaju uslov naveden u njoj. Ako se ne koristi WHERE klauzula brišu se svi redovi. Ako je specificirana ORDER BY klauzula redovi se brišu po redosledu koji je specificiran. LIMIT klauzula određuje maksimalan broj redova koji komanda DELETE sme da izbriše. Korisna je u kombinaciji sa klauzulom ORDER BY ili kada se želi da se spreči brisanje prevelikog broja redova. ORDER BY se koristi u kombinaciji sa LIMIT kada se na primer želi da se u tabeli izbriše samo n najstarijih redova.

DELETE FROM *firme* WHERE *mesto* = 'Nis';
Ovde se brišu redovi iz tabele *firme* u kojima je vrednost u koloni *mesto* Nis (drugim rečima brišu se sve firme iz Niša). Ovde se brišu redovi iz tri tabele baze podataka zbog opcije ON DELETE CASCADE za sve strane ključeve u svim tabelama baze!!!

Kod prvog oblika sintakse za više tabela komanda DELETE briše iz svake *tbl_name* tabele redove koji zadovoljavaju navedene uslove. Redovi će biti izbrisani iz tabela navedenih u odredbi DELETE, dok će tabele navedene u odredbi FROM biti pretražene, ali se redovi iz njih neće brisati, osim ako su navedene i u odredbi DELETE.

DELETE *fakture*, *detalji_fakture*
FROM *fakture*, *detalji_fakture*, *firme*
WHERE *fakture*.*sifra_fakture* = *detalji_fakture*.*faktura*
AND *fakture*.*sifra_firme* = *firme*.*firma*
AND *firme*.*naziv_firme* = 'HELIO';
Ovde se brišu sve fakture i detalji tih faktura za firmu HELIO, ali se podaci za tu firmu u tabeli *firme* ne brišu.

Drugi oblik sintakse za više tabela sličan je prvom obliku, s tom razlikom što se u ovom slučaju brišu redovi samo iz tabela navedenih u odredbi FROM dok se tabele referencraju u opciji USING.

DELETE FROM *fakture*, *detalji_fakture*
USING *fakture*, *detalji_fakture*, *firme*
WHERE *fakture*.*sifra_fakture* = *detalji_fakture*.*faktura*
AND *fakture*.*sifra_firme* = *firme*.*firma*

AND *firme*.*naziv_firme* = 'HELIO';

Opšti oblik komande DELETE prihvata i druge neobavezne odredbe:

- Odredbe LOW PRORITY deluje na isti način kao u komandi INSERT
- Odredba QUICK može ubrzati komandu DELETE jer nalaže MySQL-u da odloži neke od poslova održavanja indeksa dok briše podatke iz tabele

7.4. Brisanje svih redova iz tabele – TRUNCATE sintaksa

TRUNCATE [TABLE] *tbl_name*;

Ova komanda je brža od komande DELETE jer radi tako što najpre fizički uklanja celu tabelu, a zatim pravi istu takvu ali praznu. Ovo je brže od brisanja svih redova red po red. Komanda TRUNCATE nije uključena u transakcionu obradu.

7.5. Ažuriranje redova u tabeli – UPDATE sintaksa

Sintaksa za jednu tabelu:

UPDATE [LOW_PRIORITY] [IGNORE] *tbl_name*
SET *col_name1*=*expr1* [, *col_name2*=*expr2* ...]
[WHERE *where_condition*]
[ORDER BY ...]
[LIMIT *row_count*]

Sintaksa za više tabela:
UPDATE [LOW_PRIORITY] [IGNORE] *table_references*
SET *col_name1*=*expr1* [, *col_name2*=*expr2* ...]
[WHERE *where_condition*]

UPDATE *firme*
SET *adresa* = 'Obrenoviceva 10',
telefon = '011-463-375'
WHERE *naziv_firme* = 'STIL';
Ovde se menjaju adresa i telefon za firmu STIL u tabeli *firme*.

Komanda UPDATE je po mnogo čemu slična komandi DELETE.
Može se dodati neobavezna opcija WHERE da bi se ažurirali samo određeni redovi, a ako se izostavi biće ažurirani svi redovi tabele:

UPDATE *user*
SET *password* = 'test';
Ovde se za sve korisnike postavlja vrednost test u koloni *password*.

ALTER TABLE *detalji_fakture*
ADD COLUMN *iznos* DECIMAL(12,2);
UPDATE *detalji_fakture*
SET *iznos* = *kolicina* * *dan_cena*;
Ovde se dodaje kolona *iznos* tabeli *detalji_fakture* i nakon toga se za sve redove te tabele računa iznos kao *kolicina* * *dan_cena*.

Druga navedena verzija komande UPDATE omogućava ažuriranje više tabela jednom komandom. Postupak je sličan brisanju podataka iz više tabela istovremeno. Treba imati u vidu da će biti ažurirane samo one kolone koje se izričito navedu u odredbi SET.

Odredbe LOW PRIORITY i IGNORE deluju na isti način kao u komandi INSERT. Odredbe ORDER BY i LIMIT deluju na isti način kao u komadi DELETE.

7.6. Učitavanje podataka iz baze – SELECT sintaksa

Komanda SELECT ima sledeći opšti oblik:
SELECT *kolone*
FROM *tabele*
[WHERE *uslovi*]
[GROUP BY *grupe*
[HAVING *uslovi_za_grupe*]]
[ORDER BY *kolone_za_sortiranje*]
[LIMIT *broj*];

Ovo nije potpuna sintaksa (potpuna sintaksa će biti objašnjena kasnije), ali ilustruje opšti oblik komande.
Komanda SELECT ima veliki broj neobaveznih odredaba. Ne moraju se uvek navoditi, ali ako se upotrebljavaju, moraju se zadavati redosledom koji je prikazan u opštem obliku komande.

7.6.1. Jednostavni upiti

Primer najjednostavnijeg oblika komande SELECT izgleda ovako:

SELECT * FROM *firme*;

Ako se ovaj upit izvrši sa podacima koji postoje u bazi podataka poslovanje trebalo bi da se dobiju rezultati nalik na sledeće:

firma	naziv_firme	mesto	adresa	telefon
1	BALKAN	Nis	Mokranjceva 13	018-522-854
2	STIL	Beograd	Takovska 10	011-562-365
3	KOKOMAX	Nis	Dusanova 33	018-352-666
4	HELIO	Subotica	Nikole Tesle 55	027-555-125

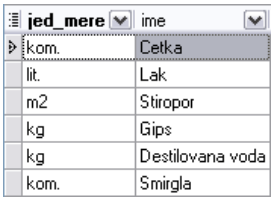
Ovaj upit je učitao sve podatke iz zadate tabele – tj. vrednosti iz svih kolona za sve redove tabele *firme*.
Razume se, suština relacione baze podataka svakako nije u tome da daje sve podatke koji su u nju uneseni, već da omogući pronalaženje određenih podataka.

7.6.2. Učitavanje podataka iz određenih kolona

Znak * u prethodnom primeru upita znači "sve kolone tabele". Umesto zvezdice mogu se zadati samo kolone iz kojih su potrebni podaci. To može biti samo jedna kolona, nekoliko kolona tabele ili čak sve kolone tabele poređane željenim redosledom. Nazive kolona treba zadati u obliku liste vrednosti razdvojenih zarezima.
Sledeći upit učitava samo vrednosti iz kolona *jed_mere* i *ime* za sve redove tabele *proizvodi*:

SELECT *jed_mere*, *ime* FROM *proizvodi*;

Ako se ovaj upit izvrši u bazi podataka poslovanje trebalo bi da se dobiju rezultati nalik na sledeće:



jed_mere	ime
kom.	Cetka
lit.	Lak
m2	Stiropor
kg	Gips
kg	Destilovana voda
kom.	Smirgla

Treba obratiti pažnju da su kolone prikazane redosledom koji je zadat u upitu, a ne redosledom kojim su definisane prilikom kreiranja tabele.

7.6.3. Apsolutna imena baza podataka i tabela

Dodatan oblik notacije koji bi korisnik trebalo da ima u vidu omogućava zadavanje apsolutnih imena baze podataka i tabele s kojom želi da radi. Kolonu *naziv_firme* tabele *firme* je moguće navesti u upitu kao *firme*.*naziv_firme*.

```
SELECT firme.naziv_firme
FROM firme;
```

Trebalo bi da rezultati ovog upita budu nalik na sledeće:

naziv_firme
BALKAN
STIL
KOKOMAX
HELIO

Slično tome, korisnik može izričito zadati koju tabelu u kojoj bazi podataka ima na umu, na primer:

```
SELECT naziv_firme
FROM poslovanje.firme;
```

Trebalo bi da se pomoću ovog upita dobiju isti rezultati kao pomoću prethodnog upita.
U ovom primeru korisnik izričito navodi da želi podatke iz tabele *firme* koja se nalazi u bazi podataka *poslovanje*. Notacija u ovom slučaju je *baza_podataka*.*tabela*.
Ako je potrebno, može se zadati kojoj bazi podataka i tabeli pripada određena kolona. Isti primer može se napisati pomoću sintakse *baza_podataka*.*tabela*.*kolona* u sledećem obliku:

```
SELECT poslovanje.firme.naziv_firme
FROM firme;
```

Ova sintaksa nije naročito korisna za ove jednostavne upite, ali kada se bude prešlo na složenije upite, ona će omogućiti korisniku da na nedvosmislen način zadaje podatke koji su mu potrebni.

7.6.3. Alijasi

Kolonama i izrazima u komandi SELECT mogu se dodeliti drugačija imena, koja će se prikazivati u rezultatima. Na primer, može se upotrebiti sledeći upit:

```
SELECT naziv_firme AS naziv
FROM firme;
```

U ovom upitu kolona *naziv_firme* je preimenovana u *naziv*, ali samo u kontekstu ovog upita. Rezultati izvršavanja ovog upita u bazi podataka poslovanje izgledaju ovako:

naziv
BALKAN
STIL
KOKOMAX
HELIO

Kao što se može videti, sadržaj kolone *naziv_firme* sada je prikazan ispod zaglavlja *naziv*.
Identifikator *naziv* poznat je kao alijas (*alias*).
Navedeni primer alijasa nije naročito koristan. Međutim, prava moć alijasa će biti shvaćena tek kad počnu da se pišu složeniji upiti i upiti u kojima se nešto izračunava.
Alijasi se mogu zadavati i za tabele, kao u sledećem primeru:

```
SELECT f.naziv_firme
FROM firme AS f;
```

Trebalo bi da rezltati ovog upita budu isti kao da je napisan bez upotrebe alijasa. Ovaj način notacije postaće koristan kada se kasnije počnu izvršavati upiti koji obuhvataju više tabela.
U poslednja dva primera rezervisana reč AS nije bila neophodna. Upite je bilo moguće napisati i u sledećem obliku:

```
SELECT naziv_firme naziv
FROM firme;
```

ili:

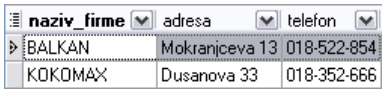
```
SELECT f.naziv_firme
FROM firme f;
```

7.6.4. Upotreba odredbe WHERE za učitavanje samo određenih redova

Učitavanje samo određenih redova je korisno jer često treba da se iz jedne ili više tabela učitaju samo zapisi koji ispunjavaju određene uslove. Ta mogućnost postaje još važnija kada je potrebno učitati samo nekoliko traženih redova iz veoma obimne tabele.
To se može obaviti pomoću odredbe WHERE komande SELECT. Jednostavan primer bio bi sledeći:

```
SELECT naziv_firme, adresa, telefon
FROM firme
WHERE mesto = 'Nis';
```

Kao što se može videti i tekst upita se može rasporediti u više redova zbog čitljivosti.
Rezultati izvršavanja ovog upita u bazi podataka poslovanje izgledaju ovako:



naziv_firme	adresa	telefon
BALKAN	Mokranjceva 13	018-522-854
KOKOMAX	Dusanova 33	018-352-666

U odredbi WHERE bio je zadat uslov zbog kojeg se izdvajaju samo redovi tabele koji ga ispunjavaju – u ovom primeru to su firme iz Niša.
Treba obratiti pažnju da je u upitu bio kombinovan uslov sa kolonama koje su bile potrebne.

U ovom primeru u odredbi WHERE bilo je zadato ispitivanje jednakosti. U SQL-u znak = služi za ispitivanje jednakosti. To je različito od mnogih drugih programskih jezika, u kojima se za tu namenu koristi == ili eq.
Postoji veliki broj funkcija koje se mogu zadati u odredbi WHERE i koje će kasnije biti detaljno objašnjene. Zasad će biti navedeni samo operatori koj se najčešće koriste:

- Jednakost, ili =
- Nejednakost (različitost), koja se piše kao != ili <>
- Sve kombinacije > (veće od), < (manje od), >= (jednako ili veće od) i <= (jednako ili manje od)
- IS NULL ili IS NOT NULL, pomoću kojih se ispituje da li određena vrednost jeste ili nije NULL
- Uobičajeni aritmetički operatori koji se nejčešće koriste u kombinaciji sa operatorima za poređenje *neka_vrednost* > *neka_druga_vrednost* * 10
- Standardni logički operatori AND, OR i NOT, koji se koriste za povezivanje više uslova. Imaju niži prioritet od operatora za poređenje. Primer: *plata* > 30000 AND *plata* < 50000.

Osim operatora u nekim primerima biće korišćene i funkcije. Funkcija COUNT() omogućava prebrojavanje redova koje je upit učitao. Kod funkcija je neophodno da se otvorena zagrada stavi neposredno iza naziva funkcije. Na primer:

SELECT COUNT(*) FROM *firme*;

Rezultati izvršavanja ovog upita u bazi podataka poslovanje izgledaju ovako:

COUNT(*)
4

Ovaj upit prikazuje koliko redova sadrži tabela *firme*.
Standarni redosled prioriteta izračunavanja delova izraza se može menjati tako što se grupišu između zagrada.
Sledeći primer je nešto složeniji upit u kome je zadata odredba WHERE:

SELECT * FROM *fakture*
WHERE *ulaz_izlaz* = 1 AND *datum* > '2006-10-23';

Rezultati izvršavanja ovog upita u bazi podataka poslovanje izgledaju ovako:

sifra_fakture	sifra_firme	datum	ulaz_izlaz
1	2	25.10.2006	1
2	4	28.10.2006	1
5	3	25.10.2006	1

Ovaj upit prikazuje podatke za sve ulazne fakture koje su formirane posle 23. oktobra 2006. godine.
Važno je istaći da u odredbi WHERE nije dozvoljena upotreba alijasa za kolone, već se mora navesti izvorno ime kolone.

7.6.5. Uklanjanje dupliranih vrednosti pomoću opcije DISTINCT

Pomoću rezervisane reči DISTINCT korisnik navodi da u rezultatima upita ne želi da vidi duplirane vrednosti. Izvršavanjem upita:

SELECT *mesto* FROM *firme*;

dobijaju se sledeći rezultati:

mesto
Nis
Beograd
Nis
Subotica

Treba obratiti pažnju da se podatak Nis pojavljuje dva puta. Upit je prikazao sve vrednosti iz kolone *mesto* tabele *firme*.
Upit formulisan na ovaj način:

SELECT DISTINCT *mesto* FROM *firme*;

daje sledeće rezultate:

mesto
Nis
Beograd
Subotica

U ovom primeru duplikati se ne ponavljaju.
U ovom slučaju razlika ne izgleda tako značajna. Razlika bi bila primetnija u tabeli sa velikim brojem podataka koji se ponavljaju.
Upit:

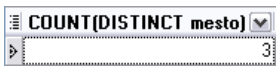
SELECT COUNT(*mesto*) FROM *firme*;

vraća broj vrednosti u koloni *mesto* tabele *firme* ne računajući NULL vrednosti:

COUNT(mesto)
4

Ako se želi videti broj različitih vrednosti u koloni *mesto* tabele *firme* treba napisati sledeći upit:

SELECT COUNT(DISTINCT *mesto*) FROM *firme*;



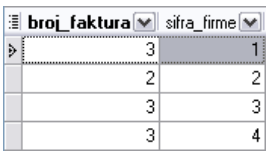
COUNT(DISTINCT mesto)
3

7.6.6. Upotreba odredbe GROUP BY

Ova odredba omogućava grupisanje učitanih redova. Ona je korisna samo kada se upotrebi u kombinaciji sa funkcijama koje deluju na grupe redova (agregatne funkcije: MIN(), MAX(), SUM(), AVG(), COUNT(), …). Upit:

SELECT COUNT(*) AS *broj_faktura*, *sifra_firme*
FROM *fakture*
GROUP BY *sifra_firme*;

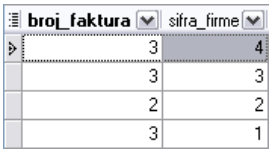
daje sledeće razultate:



broj_faktura	sifra_firme
3	1
2	2
3	3
3	4

Ovaj upit prebrojava fakture po firmama – tj. za svaku firmu utvrđuje broj faktura.
MySQL omogućava da se izabere redosled grupa kojim se prikazuju rezultati. Podrazumevani je rastući redosled. Sledeći upit je isti kao prethodni, ali se rezultati prikazuju opadajućim redosledom:

```
SELECT COUNT(*) AS broj_faktura, sifra_firme
FROM fakture
GROUP BY sifra_firme DESC;
```

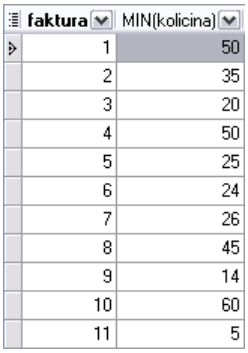


broj_faktura	sifra_firme
3	4
3	3
2	2
3	1

Može se zadati opcija ASC (od *ascending*, rastući redosled), ali pošto se taj redosled podrazumeva nema potrebe da se izričito zadaje.
Izvršavanjem upita:

```
SELECT faktura, MIN(kolicina)
FROM detalji_fakture
GROUP BY faktura;
```

dobijaju se sledeći rezultati:



faktura	MIN(kolicina)
1	50
2	35
3	20
4	50
5	25
6	24
7	26
8	45
9	14
10	60
11	5

Ovaj upit nalazi minimalnu količinu na svakoj fakturi. Funkcija MIN() vraća minimalnu vrednost.

7.6.7. Izdvajanje određenih grupa podataka pomoću opcije HAVING

Odredba GROUP BY kojoj je dodata odredba HAVING deluje na sličan način kao komanda SELECT kojoj je dodata odredba WHERE. Izvršavanjem upita:

```
SELECT faktura, SUM(kolicina * dan_cena) AS iznos
FROM detalji_fakture
GROUP BY faktura
HAVING SUM(kolicina * dan_cena) > 10000;
```

dobijaju se sledeći rezultati:

faktura	iznos
2	12150
3	32500
4	13500
5	19400
6	45670
8	15425
10	81050

Ovaj upit prikazuje sve fakture čiji je ukupan iznos veći od 10000. Iznos za svaku stavku na fakturi je *kolicina* * *dan_cena*. Ukupan iznos na fakturi je suma iznosa za sve stavke, tj. SUM(*kolicina* * *dan_cena*). Funkcija SUM() vrši sumiranje vrednosti.
Treba praviti razliku između odredbi WHERE i HAVING. Odredba WHERE se može upotrebiti u gotovo svakom upitu da bi se zadao uslov koji se odnosi na pojedinačne redove. Odredba HAVING se koristi kada određeni uslov treba da važi za celu grupu. U odredbi WHERE se ne mogu koristiti agregatne funkcije, a u odredbi HAVING mogu.

7.6.8. Sortiranje učitanih rezultata pomoću odredbe ORDER BY

Odredba ORDER BY omogućava sortiranje rezultujućih redova po jednoj ili više kolona. Redosled sortiranja može biti rastući, što se označava sa ASC, ili opadajući, što se označava sa DESC (od *descending*). Izvršavanjem upita:

```
SELECT *
FROM firme
ORDER BY mesto ASC, naziv_firme DESC;
```

dobijaju se sledeći rezultati:

firma	naziv_firme	mesto	adresa	telefon
2	STIL	Beograd	Takovska 10	011-562-365
3	KOKOMAX	Nis	Dusanova 33	018-352-666
1	BALKAN	Nis	Mokranjceva 13	018-522-854
4	HELIO	Subotica	Nikole Tesle 55	027-555-125

Ovaj upit učitava vrednosti iz svih kolona za sve redove tabele *firme*. Rezultati će biti sortirani rastuće po vrednostima u koloni *mesto*, a ako postoje dve ili više firmi iz istog mesta one će biti sortirane opadajuće po nazivu firme.
Ako se za kolonu zada ORDER BY bez opcije ASC ili DESC, podrazumeva se ASC.

7.6.9. Ograničavanje broja redova rezultata pomoću odredbe LIMIT

Odredba LIMIT ograničava broj redova rezultata koje upit daje. Izvršavanjem upita:

```
SELECT *
FROM proizvodi
LIMIT 3;
```

dobijaju se sledeći rezultati:

maticni_broj	ime	jed_mere
1	Cetka	kom.
2	Lak	lit.
3	Stiropor	m2

Ovaj upit daje prva tri reda koji ispunjavaju zadati uslov. U ovom slučaju to su prva tri reda učitana iz tabele *proizvodi*.

Može se za učitavanje zadati i podskup redova drugačiji od prvih n. Ako se pomoću prethodnog upita žele učitati redovi od 4 do 6, to će se uraditi na sledeći način:

```
SELECT *
FROM proizvodi
LIMIT 3, 3;
```

maticni_broj	ime	jed_mere
4	Gips	kg
5	Destilovana voda	kg
6	Smirgla	kom.

Kada se u odredbi LIMIT zadaju dva parametra, prvi je relativni pomak (red od kojeg počinje učitavanje), a drugi je maksimalan broj redova koji se želi učitati. Kada postoji samo jedan parametar on predstavlja maksimalan broj redova koji se želi učitati. Kada se zadaje pomak, on počinje od 0 (za četvrti red je zadat pomak 3, a pomak za prvi red je 0).
Ako se želi da upit vrati redove od pomaka do kraja tabele za vrednost drugog parametra treba navesti neki jako veliki broj.
Odredba LIMIT se najčešće koristi u kombinaciji sa odredbom ORDER BY da bi redosled redova u rezultatima upita imao određeni smisao. Treba imati na umu da bez odredbe ORDER BY, redosled redova rezultata nije predvidljiv.
Ova odredba je naročito korisna u Web ili GUI aplikacijama koje koriste MySQL jer omogućava jednostavan mehanizam podele rezultata na stranice.

7.6.10. Upotreba spojeva u upitima koji obuhvataju više tabela

Može se proanalizirati sledeći upit:

```
SELECT fakture.sifra_fakture, firme.naziv_firme, fakture.datum, fakture.ulaz_izlaz
FROM fakture, firme
WHERE fakture.sifra_firme = firme.firma;
```

U ovom slučaju žele se prikazati podaci sa svih faktura, ali se umesto šifre firme za svaku fakturu želi prikazati naziv firme.
Izvršavanjem ovog upita dobijaju se sledeći rezultati:

sifra_fakture	naziv_firme	datum	ulaz_izlaz
3	BALKAN	25.10.2006	2
6	BALKAN	6.11.2006	2
9	BALKAN	2.11.2006	2
1	STIL	25.10.2006	1
8	STIL	23.10.2006	1
4	KOKOMAX	29.10.2006	2
5	KOKOMAX	25.10.2006	1
11	KOKOMAX	15.10.2006	1
2	HELIO	28.10.2006	1
7	HELIO	4.11.2006	2
10	HELIO	8.10.2006	1

Može se videti da su iza SELECT komande zadate kolone koje postoje u različitim tabelama. Korišćena su apsolutna imena kolona zbog razumljivosti. Da se je desilo da u ove dve tabele postoje kolone sa istim nazivom, a u rezultatima se žele prikazati vrednosti iz obe kolone, apsolutna imena kolona bi bila neophodna. Da bi sve ovo funkcionisalo bilo je neophodno da se iza odredbe FROM navedu nazivi obe tabele.
Najzanimljiviji deo ovog upita je odredba WHERE. Ako se ovaj upit izvrši bez odredbe WHERE u sledećem obliku:

SELECT *fakture*.*sifra_fakture*, *firme*.*naziv_firme*, *fakture*.*datum*, *fakture*.*ulaz_izlaz*
FROM *fakture*, *firme*;

dobiće se sledeći rezultati:

sifra_fakture	naziv_firme	datum	ulaz_izlaz
1	BALKAN	25.10.2006	1
1	STIL	25.10.2006	1
1	KOKOMAX	25.10.2006	1
1	HELIO	25.10.2006	1
2	BALKAN	28.10.2006	1
2	STIL	28.10.2006	1
2	KOKOMAX	28.10.2006	1
2	HELIO	28.10.2006	1
3	BALKAN	25.10.2006	2
3	STIL	25.10.2006	2
3	KOKOMAX	25.10.2006	2
3	HELIO	25.10.2006	2
4	BALKAN	29.10.2006	2
4	STIL	29.10.2006	2
4	KOKOMAX	29.10.2006	2
4	HELIO	29.10.2006	2
5	BALKAN	25.10.2006	1
5	STIL	25.10.2006	1
5	KOKOMAX	25.10.2006	1
5	HELIO	25.10.2006	1
6	BALKAN	6.11.2006	2
6	STIL	6.11.2006	2
6	KOKOMAX	6.11.2006	2
6	HELIO	6.11.2006	2
7	BALKAN	4.11.2006	2
7	STIL	4.11.2006	2
7	KOKOMAX	4.11.2006	2
7	HELIO	4.11.2006	2
8	BALKAN	23.10.2006	1
8	STIL	23.10.2006	1
8	KOKOMAX	23.10.2006	1
8	HELIO	23.10.2006	1
9	BALKAN	2.11.2006	2
9	STIL	2.11.2006	2
9	KOKOMAX	2.11.2006	2
9	HELIO	2.11.2006	2
10	BALKAN	8.10.2006	1
10	STIL	8.10.2006	1
10	KOKOMAX	8.10.2006	1
10	HELIO	8.10.2006	1
11	BALKAN	15.10.2006	1
11	STIL	15.10.2006	1
11	KOKOMAX	15.10.2006	1

Prvi upit, kojem je pridružena odredba WHERE, prikazuje podatke sa svake fakture sa tačnim podatkom za naziv firme, a drugi upit prikazuje sve kombinacije faktura i firmi pri čemu nije moguće utvrditi koji redovi rezultata sadrže tačne podatke, a koji su besmisleni. Ovaj skup rezultata koji se sastoji od svih mogućih kombinacija rezultata iz dve tabele, zove se Dekartov proizvod (*Cartesian product*) dveju tabela.
Sasvim je očigledno da je odredba WHERE ključna za dobijanje željenih rezultata. Kada se u upitu spajaju dve tabele, uslov ili grupa uslova pomoću kojih se povezuju dve tabele zove se **spojni uslov**. U konkretnom slučaju uslov je: *fakture*.*sifra_firme* = *firme*.*firma*, što je veza između tabela koja je definisana preko spoljnih ključeva još prilikom definisanja strukture baze podataka.
Kada se želi da se istovremeno učitaju podaci koji se nalaze u više tabela, moraju se upotrebiti veze između tih tabela.

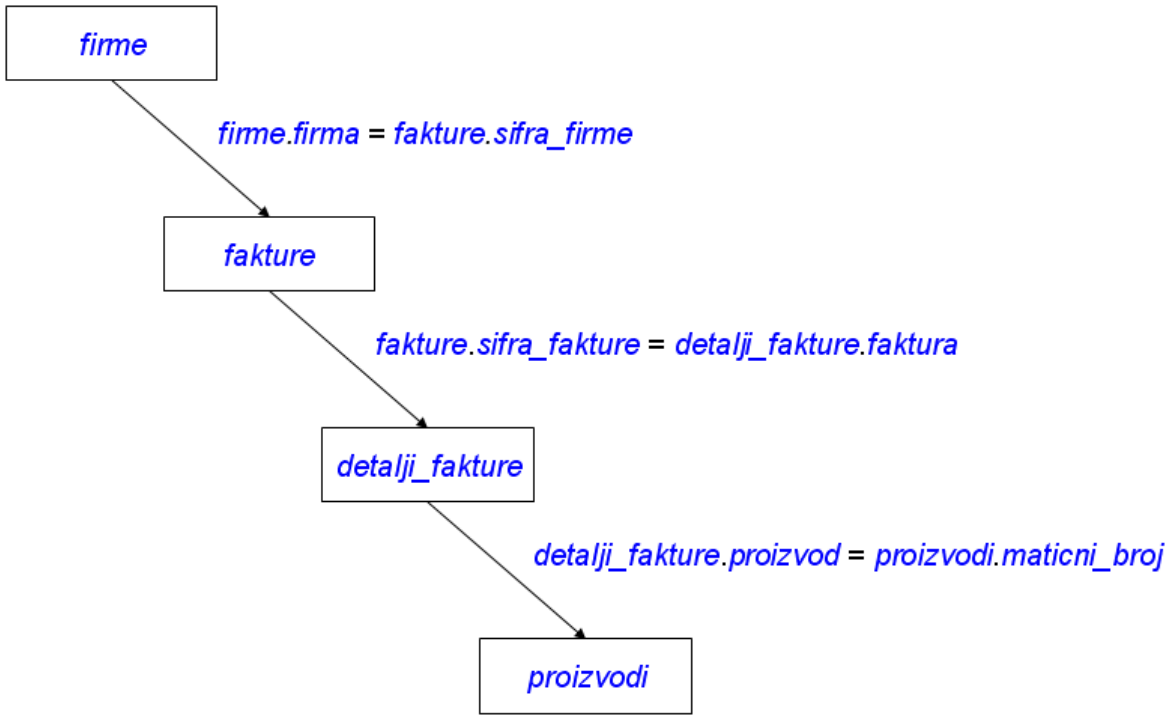
Spajanje više tabela se ne razlikuje od spajanja samo dve tabele.
Recimo da treba pronaći sve proizvode koji su kupljeni od ili prodati firmi KOKOMAX.
Pošto se zna naziv firme, u koloni *firma* tabele *firme* se može naći njena šifra. Pomoću tog podatka mogu se naći šifre svih faktura koje su formirane za tu firmu (kolona *sifra_fakture* tabele *fakture*). Pomoću šifri faktura u tabeli *detalji_fakture* se mogu naći matični brojevi proizvoda za svaku stavku svake fakture (kolona *proizvod* tabele *detalji_fakture*). Na osnovu matičnih brojeva proizvoda moguće je naći nazive proizvoda u tabeli *proizvodi* (kolona *ime* tabele *proizvodi*). Potrebno je samo na kraju prikazati različite proizvode korišćenjem ključne reči DISTINCT.
Upit bi izgledao ovako:

```
SELECT DISTINCT proizvodi.ime AS naziv_proizvoda
FROM firme, fakture, detalji_fakture, proizvodi
WHERE firme.naziv_firme = 'KOKOMAX'
AND firme.firma = fakture.sifra_firme
AND fakture.sifra_fakture = detalji_fakture.faktura
AND detalji_fakture.proizvod = proizvodi.maticni_broj;
```

Izvršavanjem upita dobijaju se sledeći rezultati:

naziv_proizvoda
Gips
Cetka
Destilovana voda
Smirgla

U upitu se vidi da je bilo neophodno navesti sve tabele u putanji koja je sleđena i zadati spojne uslove koji povezuju jednu tabelu sa drugom. U ovom slučaju postoji jedan običan uslov (*firme.naziv_firme* = 'KOKOMAX') i više spojnih uslova. Treba primetiti da su spojene četiri tabele pomoću tri spojna uslova.
Kada se spaja n tabela, u većini slučajeva, biće potrebna po jedna veza između svakog para tabela, što znači da će postojati n-1 spojnih uslova. Spojevi definisani u ovom primeru prikazani su na slici.



Kao što se jedna tabela može spojiti sa drugom, tako se može spojiti i sa samom sobom. Ovakav način spajanja će se koristiti kada se traže veze između redova u istoj tabeli. Recimo da žele da se pronađu sve firme iz istog mesta kao i firma BALKAN. Da bi se došlo do ovih podataka prvo u tabeli *firme* za firmu BALKAN treba pronaći vrednost u koloni *mesto*, a zatim u istoj tabeli treba pronaći i sve ostale firme koje u koloni *mesto* imaju istu vrednost.
Upit bi izgledao ovako:

```
SELECT f2.naziv_firme
FROM firme AS f1, firme AS f2
WHERE f1.naziv_firme = 'BALKAN'
AND f1.mesto = f2.mesto;
```

Izvršavanjem upita dobijaju se sledeći rezultati:

naziv_firme
BALKAN
KOKOMAX

Kao što se vidi, u ovom upitu deklarisana su dva različita alijasa za tabelu *firme*. Time je rečeno MySQL-u da će se raditi kao da postoje dve zasebne tabele, *f1* i *f2*, koje slučajno sadrže iste podatke. Zatim su te tabele bile spojene na isti način kao što bi se to uradilo sa bilo kojim drugim dvema tabelama. Najpre je u tabeli *f1* potražen red u kome je vrednost u koloni *naziv_firme* BALKAN (tj. u kome je ispunjen uslov *f1.naziv_firme* = 'BALKAN'), a zatim su u tabeli *f2* pronađeni redovi koji u koloni *mesto* sadrže istu vrednost kao pronađen red u tabeli *f1* u istoj koloni (*mesto*).
Ovde je samo potrebno zamisliti da se radi sa dve tabele.
Vidi se da se u rezultatima prethodnog upita nalaze sve firme koje su iz istog mesta kao i furma BALKAN, ali se nalazi i firma BALKAN. Upitu se može dodati novi uslov koji će firmu BALKAN isključiti iz skupa rezultata:

```
SELECT f2.naziv_firme
FROM firme AS f1, firme AS f2
WHERE f1.naziv_firme = 'BALKAN'
AND f1.mesto = f2.mesto
AND f2.naziv_firme != 'BALKAN';
```

naziv_firme
KOKOMAX

7.6.11. Vrste spojeva između tabela

Dekartov proizvod se ponekad naziva **punim spojem** (*full join*) ili **unakrsnim spojem** (*cross join*), ali bez obzira na ime sastoji se od svih mogućih kombinacija redova tabela. Kada se tom spoju doda određeni uslov (kao što je bio *fakture.sifra_firme* = *firme.firma*) dobija se nešto što se ponekad naziva **jednakovredni spoj** (*equijoin*), koji ograničava broj redova u skupu rezultata.
Do sada je u odredbi FROM zadavana lista tabela koje su razdvojene zarezima. Time se dobija unakrsni spoj, koji se pretvara u jednakovredni spoj kada mu se doda odredba WHERE. MySQL podržava više oblika sintakse za ovu vrstu spoja.
Izvorni upit je glasio:

```
SELECT fakture.sifra_fakture, firme.naziv_firme, fakture.datum, fakture.ulaz_izlaz
FROM fakture, firme
```

WHERE *fakture*.*sifra_firme* = *firme*.*firma*;

Umesto zareza može se zadati neobavezna rezervisana reč JOIN:

SELECT *fakture*.*sifra_fakture*, *firme*.*naziv_firme*, *fakture*.*datum*, *fakture*.*ulaz_izlaz*
FROM *fakture* JOIN *firme*
WHERE *fakture*.*sifra_firme* = *firme*.*firma*;

Osim reči JOIN može se zadati i CROSS JOIN ili INNER JOIN.
Kada se zada ova vrsta spoja, MySQL pretražuje sve tabele koje su zadate i pokušava da pronađe najefikasniji način spajanja, pri čemu ne spaja tabele obavezno redom koji je naveden. Ako se želi da se naloži MySQL-u da spoji tabele redosledom koji je naveden treba reč JOIN zameniti rečima STRAIGHT_JOIN.
Da bi bile opisane vrste spojeva između tabela, tabeli *firme* biće dodat novi red, pri čemu za novododatu firmu neće postojati podaci za fakture i detalje faktura. Iskaz bi izgledao ovako:

INSERT INTO *firme* VALUES
(5, 'ADRIA', 'Beograd', 'Ustanicka 39', '011-224-299');

Ako se želi da se pronađu nazivi firmi za koje nema formiranih faktura, potrebno je iskoristiti levi spoj, odnosno operator LEFT JOIN, na sledeći nečin:

SELECT *firme*.*naziv_firme*
FROM *firme* LEFT JOIN *fakture*
ON *firme*.*firma* = *fakture*.*sifra_firme*
WHERE *sifra_fakture* IS NULL;

Izvršavanjem upita dobijaju se sledeći rezultati:

Ako se pogleda sadržaj tabela lako se može videti da su rezultati tačni.
Levi spoj radi tako što za sve redove tabele na levoj strani spoja (u ovom primeru to je tabela *firme*) traži odgovarajuće redove u tabeli na desnoj strani spoja. Pronađeni redovi se postavljaju pored leve tabele. Za svaki red iz leve tabele koji nema parnjaka u desnoj tabeli, operator LEFT JOIN dodaje red vrednosti NULL. Redovi iz leve tabele bez parnjaka u desnoj se mogu naći ako se zada uslov da je vrednost ključa u desnoj tabeli NULL. Ako se izvrši sledeći upit:

SELECT *firme*.*naziv_firme*, *firme*.*mesto*, *firme*.*adresa*, *fakture*.*sifra_fakture*, *fakture*.*datum*, *fakture*.*ulaz_izlaz*
FROM *firme* LEFT JOIN *fakture*
ON *firme*.*firma* = *fakture*.*sifra_firme*;

dobijaju se sledeći rezultati:

naziv_firme	mesto	adresa	sifra_fakture	datum	ulaz_izlaz
BALKAN	Nis	Mokranjceva 13	3	25.10.2006	2
BALKAN	Nis	Mokranjceva 13	6	6.11.2006	2
BALKAN	Nis	Mokranjceva 13	9	2.11.2006	2
STIL	Beograd	Takovska 10	1	25.10.2006	1
STIL	Beograd	Takovska 10	8	23.10.2006	1
KOKOMAX	Nis	Dusanova 33	4	29.10.2006	2
KOKOMAX	Nis	Dusanova 33	5	25.10.2006	1
KOKOMAX	Nis	Dusanova 33	11	15.10.2006	1
HELIO	Subotica	Nikole Tesle 55	2	28.10.2006	1
HELIO	Subotica	Nikole Tesle 55	7	4.11.2006	2
HELIO	Subotica	Nikole Tesle 55	10	8.10.2006	1
ADRIA	Beograd	Ustanicka 39	Null	Null	Null

U ovom primeru upotrebljen je operator LEFT JOIN, ali isto tako je mogao da bude upotrebljen i operator RIGHT JOIN, koji deluje na isti način, s tom razlikom što je desna tabela osnova, a nedostajući redovi iz tabele sa leve strane dopunjuju se vrednostima NULL.

7.6.12. Podupiti

Podupit (*subquery*) jeste upit unutar drugog upita, odnosno upit čiji se rezultat koristi u drugom upitu. Ponekad se nazivaju i ugneždenim upitima (*nested queries*). Podupiti ne dodaju novu funkcionalnost, ali su upiti često lakše razumljivi kada se, umesto složenih spojeva između tabela, upotrebe podupiti.
MySQL-u su dodate dve osnovne vrste podupita:

- Podupiti za izvedene tabele
- Podupiti za izraze

Podupiti za izraze zadaju se u odredbi WHERE komande SELECT. Oni se dele na dve podvrste:

- Podupiti čiji je rezultat jedna vrednost ili red
- Podupiti za izraze logičkog tipa

Podupiti za izvedene tabele (*derived table subqueries*) omogućavaju zadavanje upita u odredbi FROM drugog upita. Time se formira privremena tabela koja se dodaje upitu.
Biće razmotren jedan jednostavan upit:

```
SELECT firme.firma, firme.naziv_firme
FROM firme
WHERE mesto = 'Beograd';
```

Ovaj upit učitava šifre firmi i nazive firmi za firme iz Beograda. Izvršavanjem upita dobijaju se sledeći rezultati:

firma	naziv_firme
2	STIL
5	ADRIA

Ovaj upit se može upotrebiti u drugom upitu da bi se dobio drugi koristan rezultat:

```
SELECT fakture.sifra_fakture, beogradske_firme.naziv_firme
FROM (SELECT firme.firma, firme.naziv_firme FROM firme WHERE mesto = 'Beograd')
AS beogradske_firme, fakture
WHERE beogradske_firme.firma = fakture.sifra_firme;
```

Izvršavanjem gornjeg upita dobijaju se sledeći rezultati:

sifra_fakture	naziv_firme
1	STIL
8	STIL

U ovom primeru upotrebljen je podupit (SELECT *firme*.*firma*, *firme*.*naziv_firme* FROM *firme* WHERE *mesto* = 'Beograd') da bi se formirala izvedena tabela čiji redovi sadrže samo kolone *firma* i *naziv_firme*. Privremenoj tabeli dodeljen je alijas ' *beogradske_firme* '. Ta tabela se dalje može pretraživati na isti način kao bilo koja druga tabela. U ovom primeru upotrebljena je da bi se utvrdilo koje fakture su formirane sa firmama iz Beograda.
Što se tiče podupita koji daju jednu vrednost počeće se takođe sa jednostavnim upitom:

SELECT MAX(*kolicina* * *dan_cena*)
FROM *detalji_fakture*;

Izvršavanjem ovog upita dobijaju se sledeći rezultati:

MAX(kolicina * dan_cena)
31500

Rezultat ovog upita je jedna vrednost, koja predstavlja najveći iznos posmatrajući sve stavke na svim fakturama. Agregatna funkcija MAX() pronalazi najveću vrednost iz skupa vrednosti koji joj se prosleđuje. U podupitima koji vraćaju jednu vrednost, često se ova vrsta funkcija koristi za dobijanje međurezultata koji se potom koristi za druge proračune.
Rezultat upita koji vraćaju jednu vrednost jete određena vrednost koja se potom obično poredi sa nekom drugom vrednošću. Sledeći upit koristi prethodni kao podupit:

SELECT *p*.*maticni_broj*, *p*.*ime*
FROM *proizvodi* AS *p*, *detalji_fakture* AS *df*
WHERE *p*.*maticni_broj* = *df*.*proizvod*
AND *df*.*kolicina* * *df*.*dan_cena* = (SELECT MAX(*kolicina* * *dan_cena*) FROM *detalji_fakture*);

U ovom upitu se traži matični broj i ime proizvoda koji odgovara stavci sa najvećim iznosom posmatrajući sve stavke na svim fakturama.
Izvršavanjem gornjeg upita dobijaju se sledeći rezultati:

maticni_broj	ime
4	Gips

Podupiti za izraze logičkog tipa omogućavaju da se u glavnom upitu upotrebi jedna od nekoliko specijalnih funkcija za rad sa rezultatima logičkog tipa. Te funkcije su IN, EXISTS i (grupisano) ALL, ANY i SOME.
Rezervisana reč IN omogućava poređenje sa grupom vrednosti. Može se proanalizirati sledeći upit:

SELECT *naziv_firme*
FROM *firme*
WHERE *firma* NOT IN (SELECT *sifra_firme* FROM *fakture*);

Ovaj upit daje iste rezultate kao razmatrani upit u primeru za operater LEFT JOIN, a to su nazivi firmi za koje nema formiranih faktura. Rezervisana reč IN omogućava da se utvrdi da li se data vrednost nalazi u određenom skupu mogućih vrednosti.

Pomoću operatora IN podaci upita se mogu porediti sa listom navedenih vrednosti. Rezervisana reč EXISTS deluje na malo drugačiji način od rezervisane reči IN. U upitima u kojima je zadato EXISTS, podupit zapravo koristi podatke iz spoljnjeg (glavnog) upita. To se ponekad zove **koreliran** (*corelated*) podupit. Može se proanalizirati sledeći upit:

SELECT *f.naziv_firme*, *f.firma*
FROM *firme f*
WHERE NOT EXISTS (SELECT * FROM *fakture* WHERE *sifra_firme* = *f.firma*);

U ovom primeru se takođe traže nazivi firmi za koje nema formiranih faktura.
Podupit učitava redove u kojima je ispunjen uslov da je vrednost u koloni *sifra_firme* tabele *fakture* jednaka vrednosti u koloni *firme.firma*. Vrednost *f.firma* potiče iz glavnog upita. MySQL za svaki red iz tabele *firme* ispituje rezultate podupita; ako u tom skupu ne postoji ni jedan red, tj. skup je prazan (WHERE NOT EXISTS), podatke o zaposlenom prosleđuje u konačan skup rezultata upita.

Rezervisane reči ALL, ANY i SOME omogućavaju poređenje sa skupom podvrednosti koje vraća podupit.

7.6.13. Opcije komande SELECT

SELECT
[ALL | DISTINCT | DISTINCTROW]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
select_expr, ...
[FROM *table_references*
[WHERE *where_condition*]
[GROUP BY {*col_name* | *expr* | *position*}
[ASC | DESC], ... [WITH ROLLUP]]
[HAVING *where_condition*]
[ORDER BY {*col_name* | *expr* | *position*}
[ASC | DESC], ...]
[LIMIT {[*offset*,] *row_count* | *row_count* OFFSET *offset*}]
[PROCEDURE *procedure_name*(*argument_list*)]
[INTO OUTFILE '*file_name*' *export_options*
| INTO DUMPFILE '*file_name*'
| INTO @*var_name* [, @*var_name*]]
[FOR UPDATE | LOCK IN SHARE MODE]]

table_references:
table_reference [, *table_reference*] ...

table_reference:
table_factor
| *join_table*

table_factor:
tbl_name [[AS] *alias*]
[{USE|IGNORE|FORCE} INDEX (*key_list*)]
| (*table_references*)
| { OJ *table_reference* LEFT OUTER JOIN *table_reference*
ON *conditional_expr* }

join_table:
table_reference [INNER | CROSS] JOIN *table_factor* [*join_condition*]
| *table_reference* STRAIGHT_JOIN *table_factor*
| *table_reference* STRAIGHT_JOIN *table_factor* ON *condition*
| *table_reference* LEFT [OUTER] JOIN *table_reference join_condition*
| *table_reference* NATURAL [LEFT [OUTER]] JOIN *table_factor*
| *table_reference* RIGHT [OUTER] JOIN *table_reference join_condition*
| *table_reference* NATURAL [RIGHT [OUTER]] JOIN *table_factor*

join_condition:
ON *conditional_expr*
| USING (*column_list*)

ALL, DISTINCT i DISTINCTROW opcije definišu da li će biti vraćeni redovi u kojima se vrednosti ponavljaju. Ako nijedna opcija nije navedena podrazumevana opcija je ALL (vraćaju se svi redovi). DISTINCT i DISTINCTROW su sinonimi i nalažu eliminisanje redova sa dupliranim vrednostima.
HIGH_PRIORITY obaveštava MySQL da upit treba da ima prednost nad svim komandama UPDATE koje čekaju pristup tabelama navedenim u upitu.
Odredba STRAIGHT_JOIN na samom početku komande nalaže optimizatoru upita da spoji tabele redosledom koji je korisnik naveo.
SQL_SMALL_RESULT, SQL_BIG_RESULT i SQL_BUFFER_RESULT omogućavaju optimizovanje upita. Pomoću opcija SQL_SMALL_RESULT i SQL_BIG_RESULT obaveštava se MySQL da korisnik očekuje da će se skup rezultata upita sastojati od malog, odnosno velikog broja redova. SQL_BUFFER_RESULT nalaže MySQL-u da skup rezultata smesti u privremenu tabelu. Ova opcija se može iskoristiti kada se zna da će slanje skupa rezultata klijentskom programu potrajati prilično dugo, a želi se da se izbegne da on za to vreme blokira tabele iz kojih podaci treba da se učitaju. Ove opcije su MySQL-ova proširenja ANSI standarda za jezik SQL.
SQL_CACHE i SQL_NO_CACHE nalažu MySQL-u da rezultate smešta, odnosno ne smešta u ostavu (keš).
SQL_CALC_FOUND_ROWS se koristi u odrebi LIMIT; zahteva da MySQL izračuna koliko bi ukupno redova upit vratio kada nebi sadržao odredbu LIMIT. Taj broj redova se zatim može učitati pomoću opcije SELECT FOUND_ROWS().
Opcija PROCEDURE imenuje proceduru koja treba da obradi podatke u rezultujućem setu.
SELECT INTO OUTFILE smešta rezultate komande SELECT u zadati fajl.
SELECT INTO DUMPFILE upisuje samo jedan red u fajl. Ovo je korisno kada je potrebno smestiti BLOB vrdednost u fajl.
Opcije FOR UPDATE i LOCK IN SHARE MODE deluju samo ako mašina za skladištenje zaključava podatke na nivou stranice ili reda.

7.6.14. Još neki primeri upita

Maksimalna količina proizvoda na svakoj fakturi:

```
SELECT faktura, MAX(kolicina)
FROM detalji_fakture
GROUP BY faktura;
```

faktura	MAX(kolicina)
1	50
2	50
3	60
4	50
5	120
6	80
7	26
8	50
9	50
10	200
11	5

Stavka na svakoj fakturi koja ima maksimalan iznos:

```
SELECT faktura, MAX(kolicina * dan_cena)
FROM detalji_fakture
GROUP BY faktura;
```

faktura	MAX(kolicina * dan_cena)
1	10000
2	8800
3	12000
4	9500
5	15000
6	16100
7	8060
8	12500
9	4480
10	31500
11	375

Ukupan broj faktura:

```
SELECT COUNT(sifra_fakture)
FROM fakture;
```

COUNT(sifra_fakture)
11

Prikazivanje svih ulaznih faktura sortiranih rastuće po datumu:

```
SELECT *
FROM fakture
WHERE fakture.ulaz_izlaz = 1
ORDER BY fakture.datum;
```

sifra_fakture	sifra_firme	datum	ulaz_izlaz
10	4	8.10.2006	1
11	3	15.10.2006	1
8	2	23.10.2006	1
1	2	25.10.2006	1
5	3	25.10.2006	1
2	4	28.10.2006	1

Upit koji prikazuje sve stavke na fakturi sa šifrom 2 tako da se vide naziv proizvoda, količina i cena:

```
SELECT proizvodi.ime, detalji_fakture.kolicina, detalji_fakture.dan_cena
FROM proizvodi INNER JOIN detalji_fakture
ON proizvodi.maticni_broj = detalji_fakture.proizvod
WHERE detalji_fakture.faktura = 2;
```

ime	kolicina	dan_cena
Destilovana voda	50	25
Smirgla	35	60
Stiropor	40	220

Sve stavke na fakturama kod kojih je količina veća od 50 ili cena veća od 150:

```
SELECT *
FROM detalji_fakture
WHERE kolicina > 50 OR dan_cena > 150;
```

id	faktura	red_br	proizvod	kolicina	dan_cena
1	1	1	3	50	200
4	2	3	3	40	220
6	3	2	2	35	300
7	3	3	3	60	200
8	3	4	4	50	170
9	4	1	4	50	190
11	5	1	5	120	20
12	5	2	4	100	150
14	6	1	4	75	200
16	6	3	2	24	330
17	6	4	3	70	230
18	6	5	5	80	30
19	7	1	2	26	310
21	8	2	3	50	250
24	9	3	2	14	320
25	10	1	4	150	210
26	10	2	5	200	35
27	10	3	1	125	90
28	10	4	2	100	280
29	10	5	6	60	55