1. Šta je MySQL?

MySQL je najpopularniji sistem otvorenog koda (*Open Source*) za upravljanje bazama podataka. Proizvod je kompanije MySQL AB poreklom iz Švedske.

- **MySQL je sistem za upravljanje bazama podataka**. Baza podataka je strukturirana kolekcija podataka. Može biti sve od najjednostavnije liste za kupovinu do kolekcije ogromne količine podataka neke korporacije. Za dodavanje, pristup i obradu podataka koji su smešteni u bazi podataka potreban je sistem za upravljanje bazama podataka kao što je MySQL server.
- **MySQL je sistem za upravljanje relacionim bazama podataka**. U relacionoj bazi podataka se podaci smeštaju u više međusobno povezanih tabela. Ovim se dobija na brzini i fleksibilnosti. SQL deo naziva "MySQL" potiče od "*Structured Query Language*" (strukturirani jezik za upite). SQL je najrasprostranjeniji standardizovani jezik koji se koristi za pristup bazama podataka i definisan je ANSI/ISO SQL standardom.
- **MySQL softver je softver otvorenog koda (*Open Source*)**. Ovo znači da svako može koristiti i modifikovati softver. Svako može preuzeti MySQL softver sa interneta i koristiti ga bez plaćanja. Svaki korisnik može proučiti izvorni kod softvera i izmeniti ga u skladu sa svojim potrebama.
- **MySQL server je veoma brz, pouzdan i jednostavan za korišćenje**. MySQL server je prvobitno bio razvijen radi mnogo bržeg upravljanja velikim bazama podataka u odnosu na postojeća rešenja i uspešno se koristi u visokozahtevnim proizvodnim okruženjima veći broj godina. Iako pod stalnim razvojem, MySQL server danas nudi bogat i koristan skup funkcija. Konektivnost, brzina i sigurnost čine MySQL server jako pogodnim za pristup bazama podataka na internetu.
- **MySQL server radi u klijent/server sistemima**
- **Postoji jako puno softverskih rešenja koja podržavaju rad sa MySQL serverom**

2. Instaliranje MySQL-a

Prilikom instaliranja MySQL-a trebalo bi koristiti nalog koji ima administratorska prava.

2.1. Izbor instalacionog paketa

Za MySQL 5.0 moguće je izabrati jedan od tri instalaciona paketa:

- **The Essentials Package (osnovni paket)**: Ovaj paket sadrži minimalan skup fajlova koji je neophodan da bi se instalirao MySQL, uključujući čarobnjaka za konfigurisanje. Ovaj paket ne uključuje opcione komponente, kao što su ugrađeni (*embedded*) server i paket za testiranje performansi (*benchmark suite*). Ugrađeni server predstavlja ugrađenu biblioteku koja može da radi sa više niti (*multi-threaded*) i koja se može povezati u aplikaciji da bi se dobio manji, brži i za upravljanje lakši samostalan proizvod. Paket za testiranje performansi služi da da korisniku informaciju koje operacije data SQL implementacija izvodi dobro, a koje loše. Na ovaj način se utvrđuje gde su uska grla aplikacije i baze podataka.
- **The Complete Package (kompletan paket)**: Ovaj paket sadrži sve fajlove potrebne za kompletno instaliranje, uključujući čarobnjaka za konfigurisanje. Ovaj paket uključuje opcione komponente, kao što su ugrađeni server i paket za testiranje performansi.
- **The Noninstall Archive**: Ovaj paket sadrži sve fajlove koji se nalaze u prethodnom paketu sa izuzetkom čarobnjaka za konfigurisanje. Ovaj paket ne uključuje automatski alat za instaliranje i mora biti manuelno instaliran i konfigurisan.

Osnovni paket je preporučen za većinu korisnika.

U daljem tekstu biće dat prikaz instaliranja MySQL servera korišćenjem kompletnog paketa. Biće instaliran MySQL Community Server (trenutna stabilna verzija je 5.0.27) koji je besplatan.

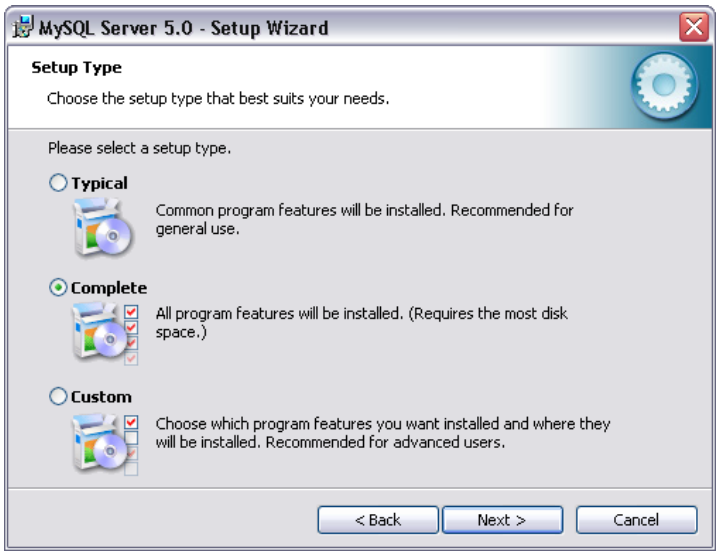
2.2. Preuzimanje i pokretanje MySQL čarobnjaka za instaliranje

MySQL instalacioni paketi se mogu preuzeti sa adrese http://dev.mysql.com/downloads/. Pokretanje čarobnjaka (za kompletan paket) se vrši dvostrukim klikom levim tasterom miša na setup.exe fajl. Pojavljuje se prva stranica čarobnjaka.

Klikom na dugme **Next** prelazi se na sledeću stranicu čarobnjaka na kojoj se bira tip instalacije.

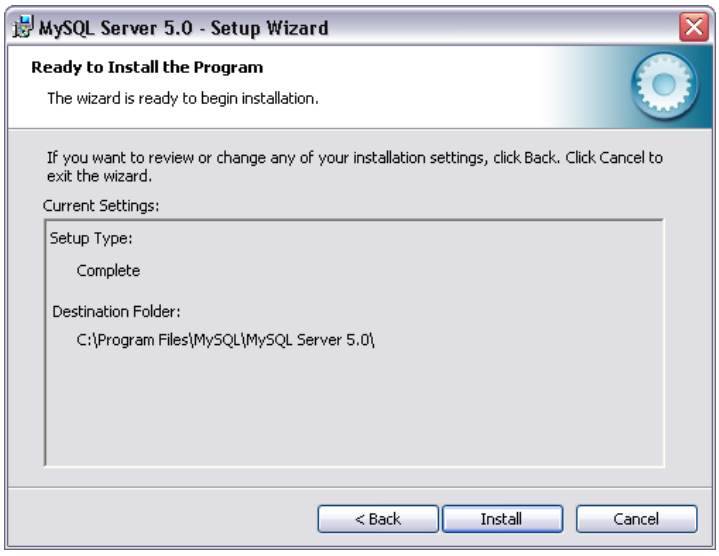
2.3. Izbor tipa instalacije

Moguće je izabrati jedan od tri tipa instalacije: **Typical** (**tipična**), **Complete** (**kompletna**) i **Custom** (**prilagođena**).
Izborom tipične instalacije instalira se MySQL server, mysql klijent za rad sa komandne linije i još neki alati za rad sa komandne linije.
Izborom kompletne instalacije instaliraju se sve komponente koje su uključene u instalacioni paket.
Prilagođeni tip instalacije daje korisniku potpunu kontrolu nad tim koji će paketi biti instalirani i koja će biti lokacija i naziv instalacionog foldera.
Izborom tipične ili kompletne instalacije i klikom na dugme **Next** prelazi se na prozor za potvrdu gde se još jednom mogu proveriti izabrane opcije i otpočeti instalacija.
Potrebno je izabrati opciju **Complete** i kliknuti na dugme **Next**.



2.4. Prozor za potvrdu

Nakon izbora tipa instalacije i eventualnog izbora komponenti koje će biti instalirane (ako je izabran Custom tip instalacije) prelazi se na prozor za potvrdu. Ovde se mogu videti tip instalacije i instalacioni folder.
Ako su podešavanja zadovoljavajuća klikom na dugme **Install** otpočinje instalacija.

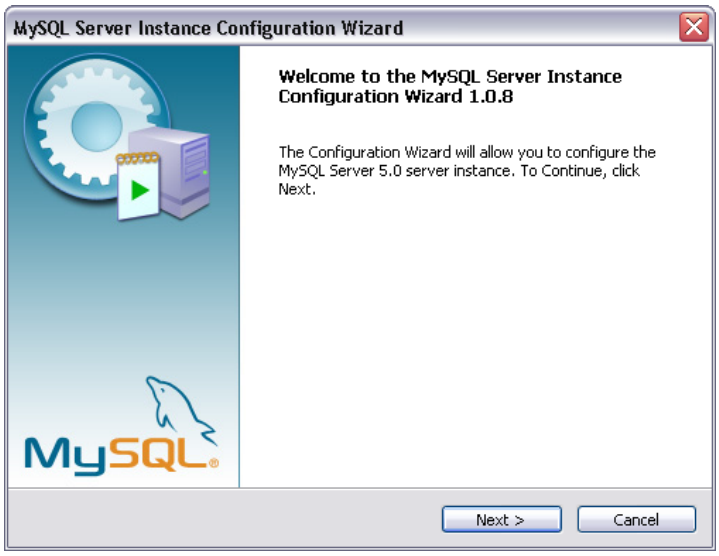


Nakon završetka instalacije pojavljuje se prozor koji nudi mogućnost registracije korisnika na MySQL sajtu. Registracija daje mogućnost potpunog korišćenja MySQL foruma, zajedno sa mogućnošću prijavljivanja bagova i mogućnošću pretplate na MySQL bilten.
Na ovom prozoru treba izabrati opciju **Skip Sign-Up** (preskakanje registracije) i kliknuti na dugme **Next**.

Poslednji prozor čarobnjaka daje obaveštenje o završetku instalacije i daje mogućnost pokretanja MySQL čarobnjaka za konfigurisanje koji se koristi za kreiranje konfiguracionog fajla, instaliranje MySQL servisa i konfigurisanje sigurnosnih podešavanja.
Na ovom prozoru treba zadržati unapred čekiranu opciju **Configure the MySQL Server now** i kliknuti na dugme **Finish**.

2.5. Čarobnjak za konfigurisanje MySQL servera

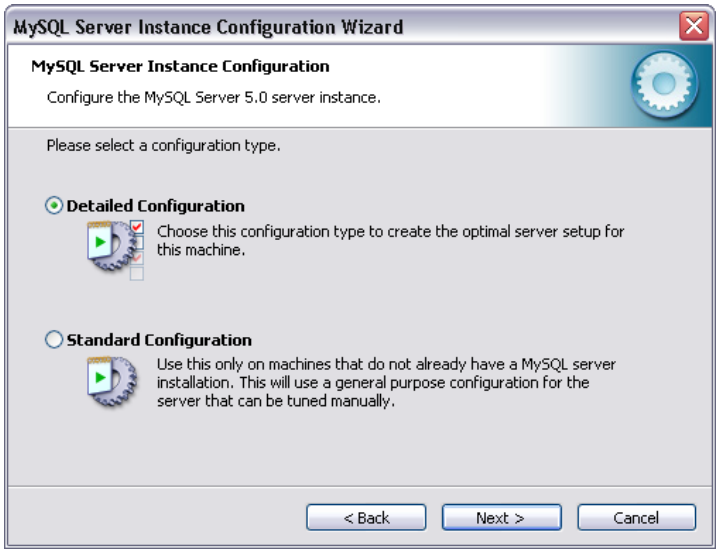
Čarobnjak za konfigurisanje MySQL servera automatizuje proces konfigurisanja servera. On kreira prilagođeni MySQL konfiguracioni fajl (my.ini ili my.cnf) postavljanjem niza pitanja korisniku i primenom niza odgovora na šablon. Ovaj čarobnjak se pokreće automatski kao deo instalacionog procesa. Nakon ovog prvog pokretanja korisnik može ponovo pokrenuti čarobnjaka kada je potrebno da promeni konfiguracione parametre svog servera. my.ini fajl se može modifikovati i otvaranjem u nekom od tekstualnih editora i vršenjem potrebnih izmena. Čarobnjak smešta my.ini fajl u instalacioni folder MySQL servera (C:\Program Files\MySQL\MySQL Server 5.0).



Klikom na dugme **Next** prelazi se na sledeću stranicu čarobnjaka.

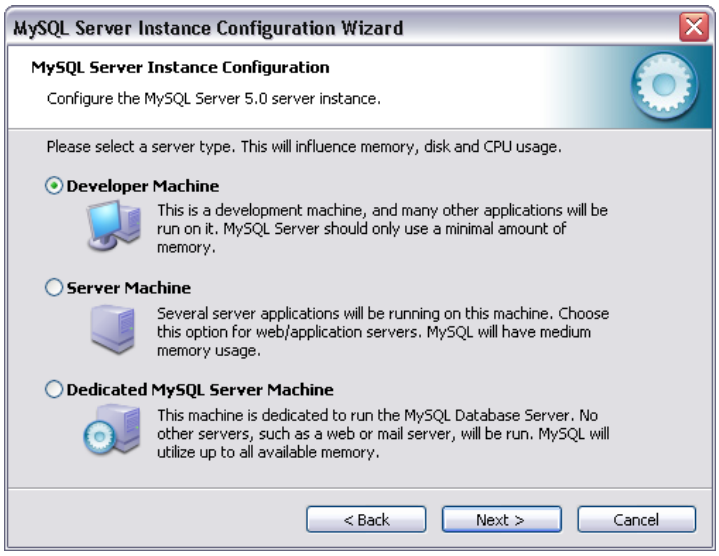
2.6. Izbor tipa konfiguracije

Postoje dva osnovna tipa konfiguracije: Detailed Configuration (detaljna konfiguracija) i Standard Configuration (standardna konfiguracija). Standardna konfiguracija je namenjena za nove korisnike koji žele da započnu rad sa MySQL-om brzo bez potrebe za donošenjem mnogo odluka vezano za konfiguraciju servera. Detaljna konfiguracija je namenjena za napredne korisnike.
Korisnicima koji prvi put koriste MySQL server i žele server koji će biti konfigurisan kao jednokorisnička razvojna mašina, bi trebalo da odgovara standardna konfiguracija. Kod ovog tipa konfiguracije ćarobnjak sve konfiguracione opcije podešava automatski osim opcija vezanih za servis i sigurnost.
Treba izabrati opciju **Detailed Configuration** i kliknuti na dugme **Next**.



2.7. Izbor tipa servera

Postoje tri tipa servera. Tip servera koji korisnik izabere utiče na odluke koje pravi čarobnjak vezano za korišćenje memorije, hard diska i procesora.

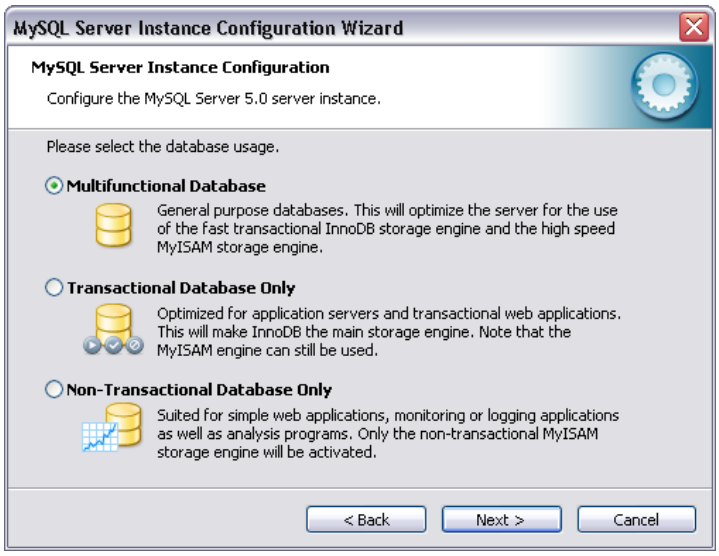


- Developer Machine (razvojna mašina): Ovu opciju treba izabrati za tipičnu desktop radnu stanicu gde je MySQL namenjen samo za lične potrebe. Pretpostavlja se da su mnoge druge aplikacije istovremeno aktivne. MySQL server je konfigurisan da koristi minimum sistemskih resursa.
- Server Machine (serverska mašina): Ovu opciju treba izabrati za serversku mašinu gde je MySQL server aktivan zajedno sa drugim serverskim aplikacijama, kao što su FTP, email i WEB serveri. MySQL server je konfigurisan za osrednje korišćenje sistemskih resursa.
- Dedicated MySQL Server Machine (serverska mašina namenjena samo za MySQL server): Ovu opciju treba izabrati za serversku mašinu na kojoj će biti aktivan samo MySQL server. Pretpostavlja se da druge aplikacije nisu aktivne. MySQL server je konfigurisan da koristi sve rapoložive sistemske resurse.

Treba izabrati opciju **Developer Machine** i kliknuti na dugme **Next**.

2.8. Upotreba baze podataka

Ovaj prozor čarobnjaka omogućava korisniku da odluči koje će mašine za skladištenje koristiti prilikom kreiranja MySQL tabela. Izabrana opcija definiše da li je InnoDB mašina za skladištenje na rapolaganju i koji procenat resursa servera je na rapolaganju za ovu mašinu.



- Multifunctional Database (multifunkcionalna baza podataka): Ova opcija omogućava korišćenje i InnoDB i MyISAM mašine za skladištenje i deli resurse servera jednako između ove dve mašine. Ova opcija je preporučena za korisnike koji koriste obe mašine za skladištenje svakodnevno.
- Transactional Database Only (samo transakciona baza podataka): Ova opcija omogućava korišćenje i InnoDB i MyISAM mašine za skladištenje ali dodeljuje većinu resursa servera InnoDB mašini za skladištenje. Ova opcija je preporučena za korisnike koji skoro isključivo koriste InnoDB mašinu za sladištenje, a minimalno koriste MyISAM mašinu za skladištenje.
- Non-Transactional Database Only (samo netransakciona baza podataka): Ova opcija isključuje potpuno InnoDB mašinu za sladištenje i sve resurse servera dodeljuje MyISAM mašini za skladištenje. Ova opcija je preporučena za korisnike koji ne koriste InnoDB mašinu za sladištenje.

Tipovi tabela se nazivaju i mašine za skladištenje (*storage engine*). To odražava činjenicu da je za upotrebu nekih tipova tabela neophodna značajna količina posebnog programskog koda koji upravlja smeštanjem podataka, indeksiranjem, zaključavanjem podataka i pristupanjem disku. To takođe odražava suštinu baze podataka: skladištenje podataka.

Tabele koje podržavaju transakcije omogućavaju da korisnik zada da je određena grupa upita nedeljiva jedinica obrade – transakcija. Trebalo bi da se cela transakcija obavi do kraja, a ako to nije moguće, baza podataka mora da je poništi (*roll back*), odnosno da se vrati u stanje u kojem je bila pre transakcije.

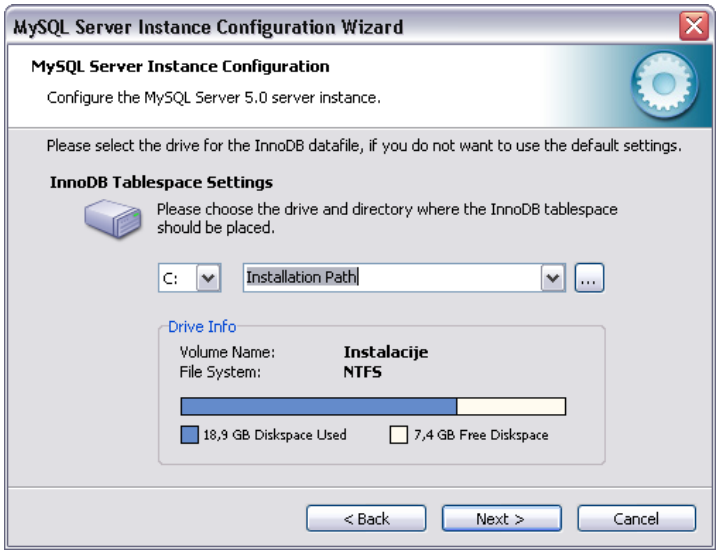
Može se zamisliti baza podataka o prometu na bankovnim računima. Ako se želi da se prebaci 1000 dinara s jednog računa na drugi, potrebna su najmanje dva SQL upita – jedan, koji na prvom računu smanjuje stanje za 1000 dinara i drugi, koji povećava stanje na drugom računu za 1000 dinara. Bila bi prava katastrofa kada bi se zbog nečeg (na primer, nestanka struje) prvi upit izvršio do kraja, ali ne i drugi. U takvim slučajevima bilo bi neuporedivo bolje da se izvrše ili oba upita, ili nijedan, jer baza podataka mora uvek biti u usklađenom stanju.

Treba izabrati opciju **Multifunctional Database** i kliknuti na dugme **Next**.

2.9. Lokacija InnoDB Tablespace fajlova (fajlovi tabelarnog prostora)

Dok MyISAM smešta svaku tabelu u zaseban fajl, InnoDB smešta sve tabele i indekse u tabelarni prostor, što znači da se delovi jedne tabele mogu nalaziti u više fajlova. Neki korisnici žele da lociraju InnoDB tablespace fajlove na neku drugu lokaciju različitu od

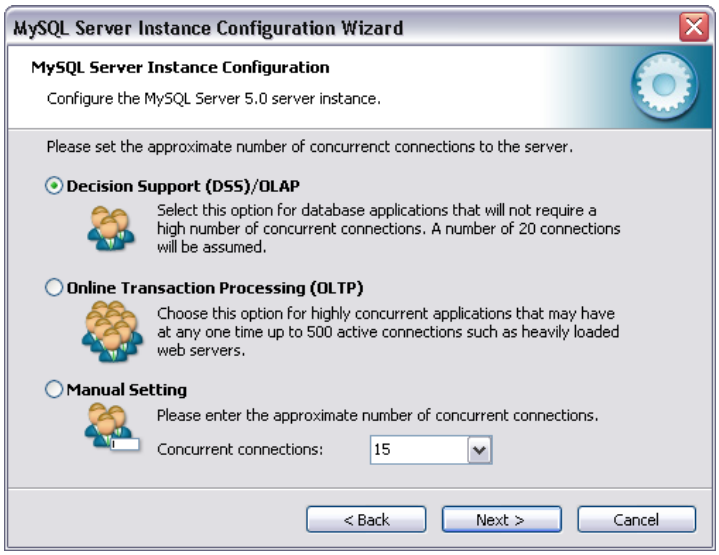
MySQL server data foldera. Smeštanje tablespace fajlova na zasebnu lokaciju može biti poželjno ako sistem ima uređaje za skladištenje podataka većeg kapaciteta ili većih performansi, kao što je RAID sistem za skladištenje.



Da bi se promenila podrazumevana lokacija za InnoDB tablespace fajlove treba izabrati novu particiju ili novi hard disk iz padajuće liste i nakon toga izabrati novi put iz padajuće liste. Ako korisnik želi da sam kreira put treba da klikne na dugme **...**.
U ovom prozoru ne treba ništa menjati. Kliknuti na dugme **Next**.

2.10. Broj istovremenih konekcija

Da bi se izbegle situacije u kojima bi server radio da nedovoljnim resursima, neophodno je ograničiti broj istovremenih konekcija na server koje se mogu uspostaviti. Ovaj prozor omogućava da se izabere očekivano korišćenje servera i u skladu sa tim podesi granica za broj istovremenih konekcija. Granica za broj istovremenih konekcija se može podesiti i manuelno.



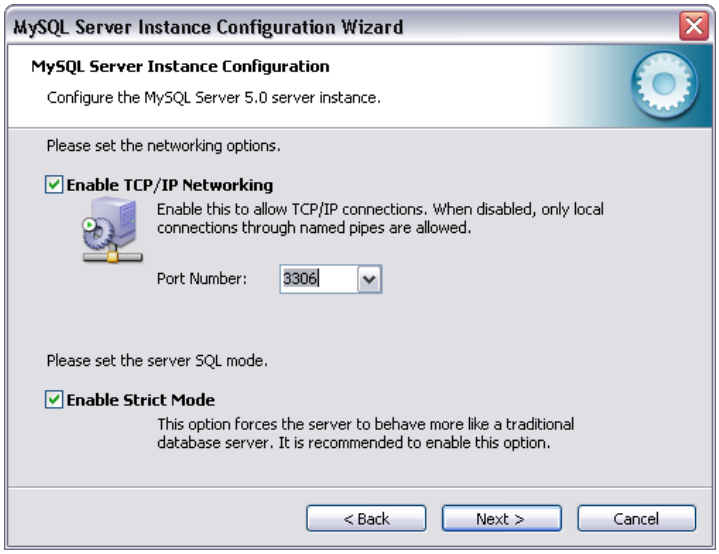
- Decission Support (DSS)/OLAP (podrška odlučivanju): Ovu opciju treba izabrati ako server ne zahteva veliki broj istovremenih konekcija. Maksimalan broj konekcija je podešen na 100, a pretpostavljeno da će biti prosečno 20 istovremenih konekcija.

- Online Transaction Processing (OLTP) (obrada online transakcija): Ovu opciju treba izabrati ako server zahteva veliki broj istovremenih konekcija. Maksimalan broj konekcija je podešen na 500.
- Manual Setting (manuelno podešavanje): Ovu opciju treba izabrati da bi se manuelno podesio maksimalan broj istovremenih konekcija. Treba izabrati maksimalan broj iz padajuće liste ili ga ukucati ako ne postoji u padajućoj listi.

Treba izabrati opciju **Decission Support (DSS)/OLAP** i kliknuti na dugme **Next**.

2.11. Mrežne i Strict Mode opcije

Mrežne opcije omogućavaju da se dozvoli ili zabrani kreiranje TCP/IP konekcija i da se konfiguriše broj porta koji će se koristiti za konektovanje na MySQL server.



Kreiranje TCP/IP konekcija je podrazumevano dozvoljeno. Port 3306 je podrazumevana opcija. Da bi se promenio port potrebno ga je izabrati iz padajuće liste ili ga dokucati ako se ne nalazi u listi.

Strict Mode čini da se server ponaša kao drugi sistemi za upravljanje bazama podataka. Strict Mode kontroliše kako MySQL manipuliše ulaznim vrednostima koje su loše ili nedostaju.

Treba čekirati opciju **Enable TCP/IP Networking** i zadržati port **3306**. Takođe treba čekirati opciju **Enable Strict Mode** i kliknuti na dugme **Next**.

2.12. Character Set opcija

MySQL server podržava različite setove karaktera i moguće je podesiti podrazumevani set karaktera za server koji će biti primenjen na sve tabele, kolone i baze podataka. Set karaktera je skup karaktera i kodiranja za te karaktere. Collation (uparivanje) je skup pravila za poređenje karaktera u okviru seta karaktera.

- Standard Character Set (standarni set karaktera): Izabrati ovu opciju ako se kao podrazumevani set karaktera želi koristiti latin1. Ovaj set karaktera se koristi za engleski i mnoge zapadnoevropske jezike.
- Best Support For Multilingualism (najbolja podrška za veliki broj jezika): Izabrati ovu opciju ako se kao podrazumevani set karaktera želi koristiti utf8. Ovo je Unicode skup karaktera i može smeštati karaktere iz mnogo različitih jezika.
- Manual Selected Default Character Set / Collation (manuelan izbor podrazumevanog seta karaktera / uparivanja): Izabrati ovu opciju ako se želi da se manuelno definiše podrazumevani set karaktera. Željeni set karaktera treba izabrati iz padajuće liste.

Treba izabrati opciju **Best Support For Multilingualism** i kliknuti na dugme **Next**.

2.13. Opcije vezane za servis

Na Windows NT baziranim sistemima MySQL server može biti instaliran kao Windows servis. Kada je instaliran na ovaj način MySQL server može biti startovan automatski prilikom pokretanja sistema i čak restartovan automatski od strane Windows-a u slučaju problema sa servisom.

Čarobnjak instalira MySQL server kao servis podrazumevano i daje naziv MySQL servisu. Ako se ne želi da se MySQL server instalira kao servis treba odčekirati opciju Install As Windows Service. Naziv servisa se može promeniti izborom drugog naziva iz padajuće liste ili ukucavanjem novog naziva ako se željeni ne nalazi u padajućoj listi.

Ako se želi da se MySQL server instalira kao servis ali da se servis ne startuje automatski prilikom pokretanja sistema treba odčekirati opciju Launch the MySQL Server automatically.

Treba čekirati opciju **Install As Windows Service**, zadržati ponuđeno ime servisa i odčekirati opciju **Launch the MySQL Server automatically**. Treba čekirati opciju **Include Bin Directory in Windows PATH**. Sada je moguće pokrenuti bilo koji MySQL izvršni program kucanjem njegovog naziva u interpreteru komandi (*Command Prompt*) iz bilo kog foldera bez potrebe za prosleđivanjem putanje do bin foldera. Kliknuti na dugme **Next**.

2.14. Sigurnosne opcije

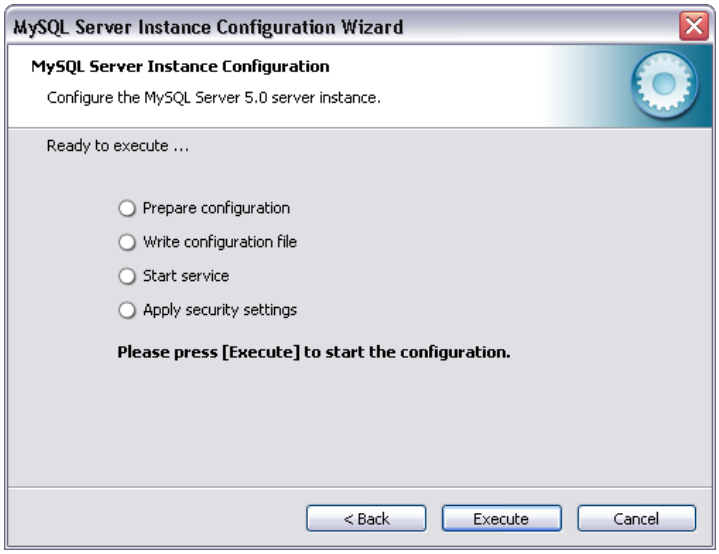
Ovde je neophodno i jako preporučljivo definisati root password za MySQL server i čarobnjak podrazumevano zahteva od korisnika da to uradi. Ako korisnik ne želi da ovo uradi treba da odčekira opciju Modify Security Settings. root je podrazumevani administrativni nalog u MySQL sistemu za upravljanje pravima korisnika.

Da bi se definisao root password treba uneti password u polja New root password i Confirm.
Da bi se sprečilo konektovanje na server preko root naloga sa drugih računara iz mreže treba odčekirati opciju Enable root access from remote machines. Ovo povećava sigurnost root naloga.
Da bi se kreirao anonimni nalog treba čekirati opciju Create An Anonymous Account. Za anonimne naloge nije potrebno zadavanje korisničkog imena i lozinke prilikom

prijavljivanja. Kreiranje anonimnog naloga može umanjiti sigurnost servera i izazvati probleme sa logovanjem i pravima pristupa. Iz ovih razloga nije preporučljivo.
Treba čekirati opciju **Modify Security Settings**, uneti i potvrditi root password (kurs2008) i odčekirati opciju **Enable root access from remote machines**. Kliknuti na dugme **Next**.

2.15. Prozor za potvdu

Poslednji prozor čarobnjaka za konfigurisanje MySQL servera je prozor za potvrdu. Da bi otpočeo proces konfigurisanja treba kliknuti na dugme **Execute**.



Nakon završetka procesa konfigurisanja pojavljuje se prozor sa rezultatima. Klikom na dugme **Finish** izlazi se iz čarobnjaka.

2.16. Raspored foldera nastalih u toku instalacije

Za MySQL 5.0 na Windows-u podrazumevani instalacioni folder je: C:\Program Files\MySQL\MySQL Server 5.0. U instalacionom folderu se nalaze sledeći folderi:

Folder	Sadržaj foldera
bin	MySQL server i klijentski programi
data	Log fajlovi i baze podataka, odnosno podaci
Docs	Uputstvo u .chm formatu
examples	Programi i skriptovi za primer
include	Include (header) fajlovi ili fajlovi zaglavlja (koriste se prilikom kompajliranja)
lib	Biblioteke funkcija koje MySQL koristi
scripts	Skriptovi napisani u jeziku Perl koji obavljaju korisne poslove
share	Fajlovi sa tekstom poruka o greškama koje MySQL šalje
sql-bench	Programi za testiranje MySQL-a

2.17. Pregled izvršnih fajlova

MySQL-ovi izvršni fajlovi nalaze se u folderima bin i scripts. U folderu bin se može naći više fajlova koji u svom nazivu sadrže mysqld (recimo mysqld.exe, mysqld-nt.exe, mysqld-max.exe). To su programi različitih verzija mysql servera. U ovom folderu se nalazi i mysql.exe što je MySQL Monitor. Osim ovih najvažnijih programa mogu se naći i drugi:

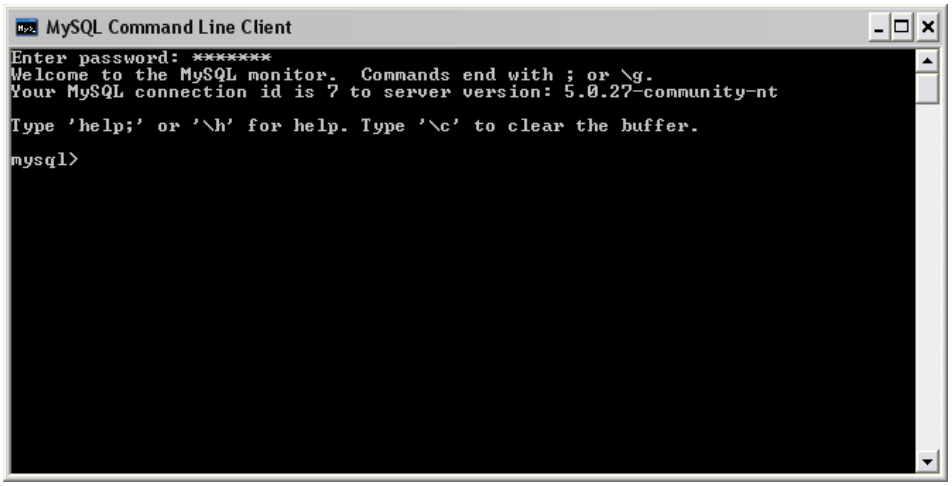
- mysqladmin.exe: Služi za obavljanje administrativnih funkcija

- myisamchk.exe: Služi za proveru i popravljanje oštećenih MyISAM tabela
- mysqldump.exe: Omogućava izradu rezervnih kopija baze podataka
- mysqlbinlog.exe: Služi za čitanje binarnih log fajlova ili dnevnika izmena gde se beleže podaci o svim izmenama na bazi. Beleženje izmena je veoma korisno u slučaju oporavljanja od katastrofalnih grešaka.
- mysqlshow.exe: Služi za prikazivanje podataka o bazama podataka i njihovim tabelama

3. Klijentski programi

3.1. MySQL klijent za rad sa komandne linije

MySQL klijent za rad sa komandne linije (mysql) (MySQL Monitor) služi za interaktivno izvršavanje SQL iskaza. Konfigurisan je da se konektuje na server sa root nalogom, pa se prilikom pokretanja od korisnika zahteva unos lozinke za root nalog ukoliko je definisana. Može se pokrenuti na sledeći način: **start → All Programs → MySQL → MySQL Server 5.0 → MySQL Command Line Client**. Nakon pokretanja neophodno je uneti lozinku za root nalog i kliknuti na taster **Enter**. Korisnik može MySQL i SQL komande upisivati direktno u MySQL Monitor.



Nakon prijave može se videti koje sve baze podataka postoje na serveru korišćenjem komande SHOW:
show databases;
Spisak bi trebalo da sadrži tri baze. Jedna od baza je mysql. To je sistemska baza podataka u kojoj se čuvaju podaci o korisničkim nalozima i njihovim pravima.
Većinu komandi koje korisnik otkuca u MySQL monitoru mora završiti znakom tačka i zarez (;) inače ih MySQL neće izvršiti.
Iz MySQL monitora korisnik se može odjaviti tako što otkuca \q (slovo q potiče od reči quit). Ova komanda se ne završava znakom tačka i zarez. Postoji grupa komandi koje počinju znakom \ (obrnuta kosa crta ili *backslash*). Nijedna od njih se ne završava znakom tačka i zarez. Spisak tih komandi se može dobiti ako se otkuca \h (slovo h potiče od reči *help*).

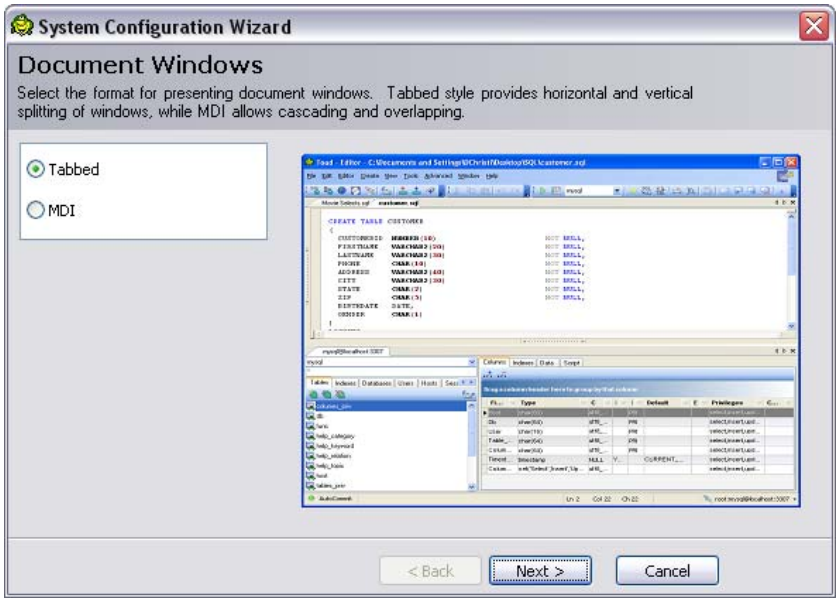
3.2. Klijentski programi sa grafičkim korisničkim okruženjem

Klijentski programi sa grafičkim korisničkim okruženjem služe za razvoj i administraciju baza podataka. Njihova glavna prednost su grafičko korisničko okruženje i moćni alati koji stoje na raspolaganju korisniku. Proizvođač MySQL-a takođe nudi klijentske programe

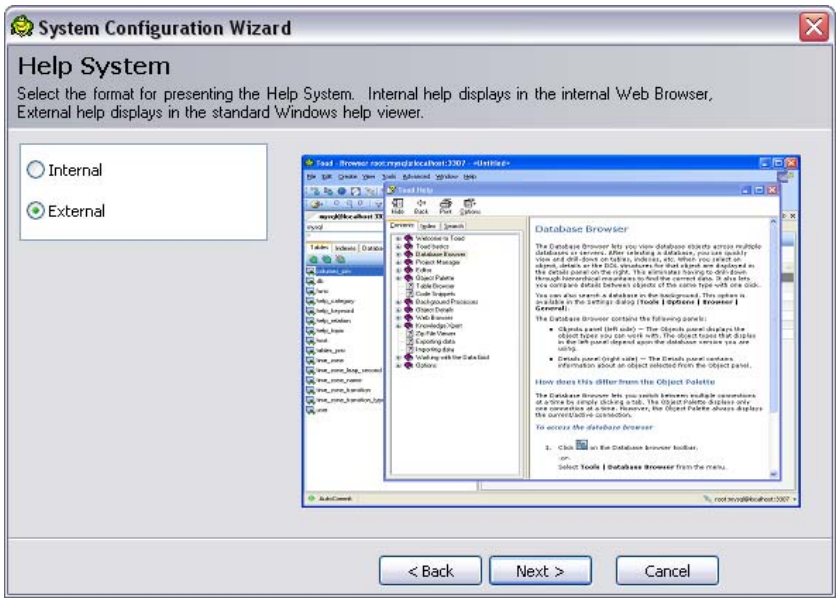
ovog tipa i oni su besplatni. Postoji jako puno klijentskih programa drugih proizvođača koji rade sa MySQL serverom i među njima ima i besplatnih rešenja i onih koja se plaćaju.

3.2.1. Toad for MySQL Freeware

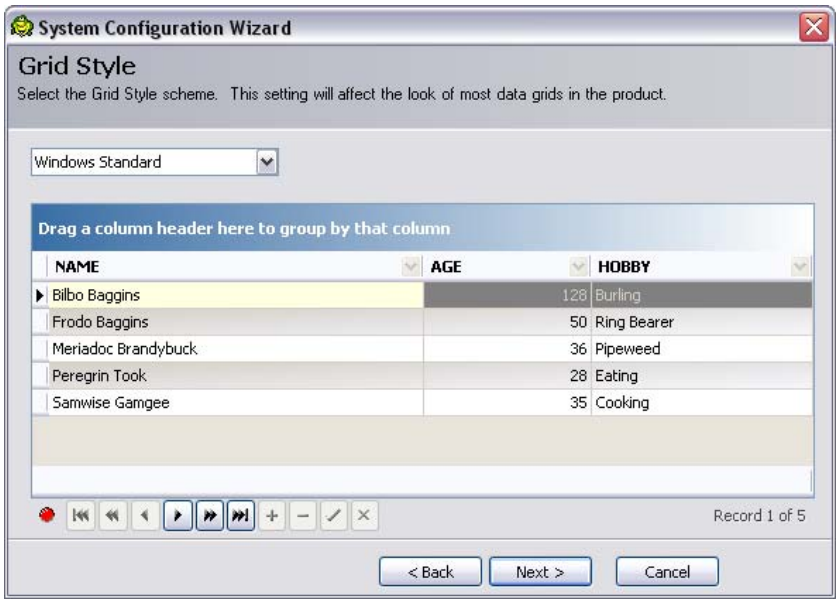
Toad for MySQL Freeware je besplatan klijentski program profesionalnog nivoa. Proizvod je kompanije Quest Software. Da bi ovaj program bio instaliran neophodno je da korisnik ima instaliran Microsoft .NET Framework 2.0 na računaru. Trenutno aktuelna verzija je 2.0.3.795. Nakon instaliranja programa pokreće se čarobnjak za konfigurisanje.
Na prvoj stranici čarobnjaka je moguće izabrati format za prikazivanje prozora dokumenata. Opcija Tabbed daje mogućnost horizontalnog i vertikalnog deljenja prozora, a opcija MDI omogućava kaskadno prikazivanje i preklapanje prozora.



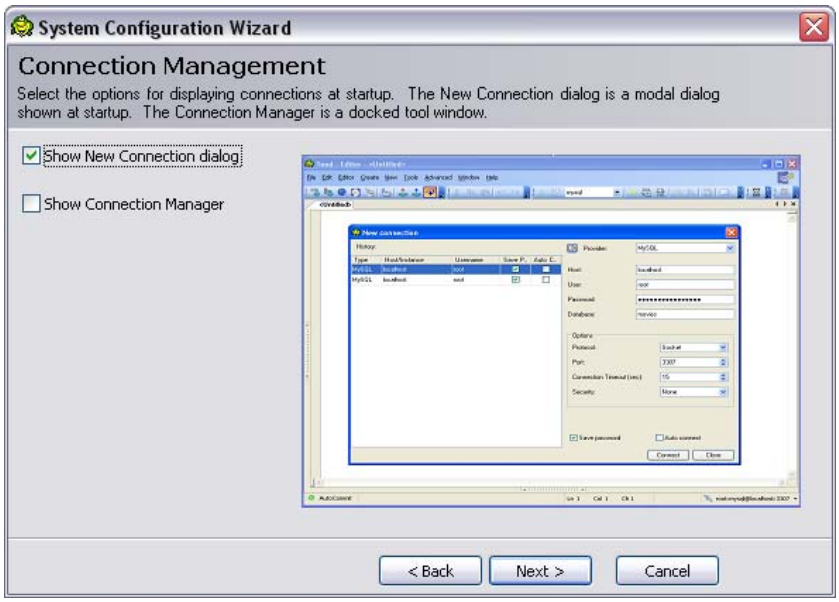
Treba izabrati opciju **Tabbed** i kliknuti na dugme **Next**.
Na sledećoj stranici čarobnjaka je moguće izabrati format za prikazivanje Help-a. Izborom opcije Internal Help se prikazuje u internom Web pretraživaču. Izborom opcije External Help se prikazuje standarno u zasebnom prozoru kao i kod ostalih programa (.chm format).



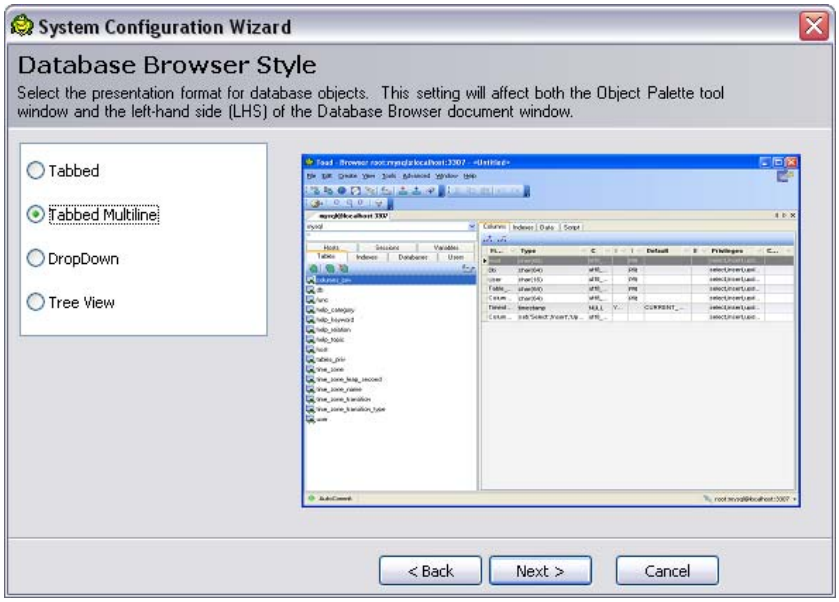
Treba izabrati opciju **External** i kliknuti na dugme **Next**.
Na sledećoj stranici čarobnjaka je moguće izabrati stil tabelarnog prikaza podataka.



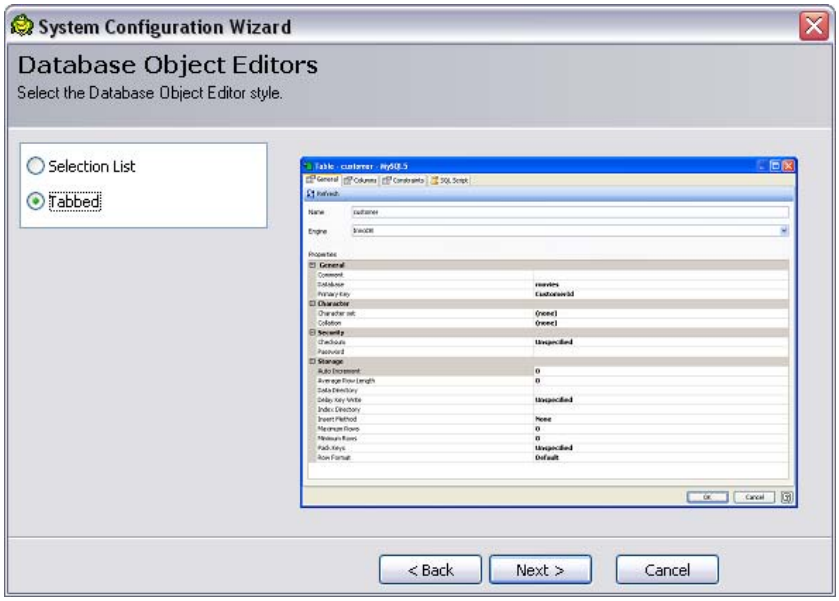
Treba zadržati ponuđenu opciju **Windows Standard** i kliknuti na dugme **Next**.
Na sledećoj stranici čarobnjaka je moguće izabrati opcije za prikaz konekcija prilikom startovanja programa. Izborom opcije Show New Connection dialog prikazivaće se okvir za dijalog New Connection prilikom svakog startovanja programa. Ovaj okvir za dijalog omogućava uspostavljanje nove konekcije sa serverom ili ponovno uspostavljanje neke od prethodno uspostavljanih konekcija. Izborom opcije Show Connection Manager prikazivaće se Connection Manager okno koje će biti usidreno u jednom delu programskog prozora. U ovom oknu je moguće videti sve ranije kreirane ili novokreiranje konekcije kao i to koje su konekcije trenutno aktivne. Preko ovog okna je moguće prekinuti neku od aktivnih konekcija, uspostaviti novu konekciju ili uspostaviti neku od ranije uspostavljanih konekcija. Connection Manager je moguće naknadno otvoriti izborom opcije **Connection Manager** iz menija **View**. Ovde je moguće izabrati obe opcije, samo jednu ili nijednu.



Treba izabrati opciju **Show New Connection dialog** i kliknuti na dugme **Next**.
Na sledećoj stranici čarobnjaka je moguće podesiti format prikaza objekata baze podataka. Izborom opcije Tabbed objekti baze će biti prikazani po karticama. Izborom opcije Tabbed Multiline objekti baze će biti prikazani po karticama u više linija. Izborom opcije Drop Down objekti će biti birani iz padajuće liste. Izborom opcije Tree View objekti će biti prikazani u vidu stabla.

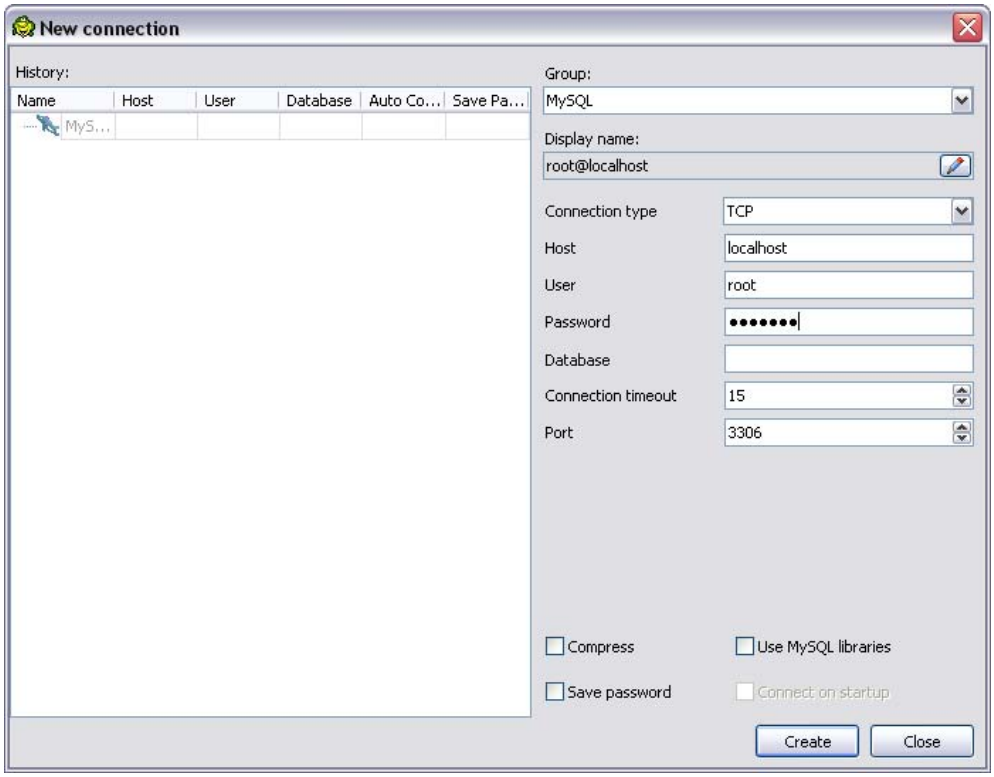


Treba izabrati opciju **Tabbed Multiline** i kliknuti na dugme **Next**.
Na sledećoj stranici čarobnjaka je moguće izabrati stil editora objekata baze podataka. Izborom opcije Selection List svojstva objekta će biti grupisana u listi za izbor sa leve strane prozora. Izborom opcije Tabbed svojstva objekta će biti grupisana na karticama.



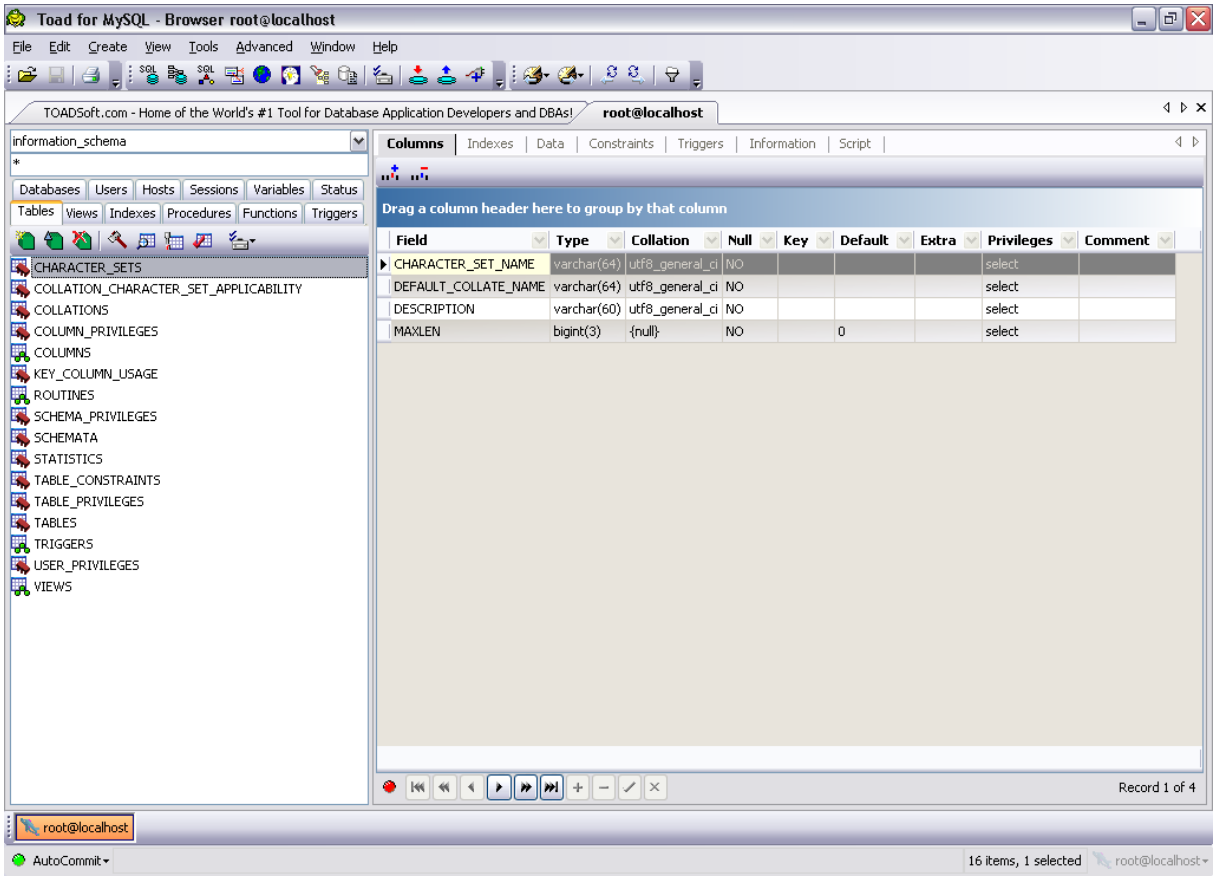
Treba izabrati opciju **Tabbed** i kliknuti na dugme **Next**.
Na sledećoj stranici čarobnjaka je moguće definisati sa fajlovima kojih ekstenzija će program raditi.

Treba zadržati podrazumevane opcije kao na gornjoj slici. Ovo je poslednja stranica čarobnjaka. Klikom na dugme **Finish** izlazi se iz čarobnjaka i primenjuju izabrane opcije. Čarobnjaka je naknadno uvek moguće pokrenuti izborom opcije **Configuration Wizard …** iz menija **Tools**.
Nakon izlaska iz čarobnjaka i nakog svakog sledećeg pokretanja programa pojaviće se okvir za dijalog New Connection.



U padajućoj listi Group treba izabrati grupu u kojoj će biti smeštena nova konekcija. Ako je izabrano MySQL nova konekcija će biti smeštena pod root čvorom. Ako se izabere neka druga grupa nova konekcija će biti smeštena pod podčvorom te grupe. Ovde treba ostaviti unapred izabranu opciju (MySQL).
U polju Display Name se definiše naziv konekcije. Podrazumevani naziv je username@hostname (database) i on se automatski generiše. Korisnik može uneti svoj naziv klikom na dugme . Ovde treba zadržati podrazumevani naziv konekcije.
U padajućoj listi Connection Type treba izabrati protokol koji se koristi prilikom konektovanja. Ovde treba zadržati unapred izabranu opciju (TCP).
U polje Host treba uneti ime računara na kome je instaliran MySQL server na koji se korisnik želi konektovati. Ovde treba zadržati podrazumevanu opciju (localhost) što znači da se MySQL server na koji se korisnik želi konektovati nalazi na njegovom računaru. Ako se korisnik konektuje na MySQL server koji se nalazi na nekom drugom računaru onda je potrebno da u polje Host unese IP adresu tog računara.
U polje User treba uneti korisničko ime koje se koristi prilikom konektovanja. U ovo polje treba uneti korisničko ime root.
U polje Password treba uneti lozinku koja će se koristiti prilikom konektovanja. U ovo polje treba uneti lozinku za root nalog koja je bila definisana prilikom konfigurisanja servera (kurs2008).
U polje Database treba uneti naziv baze podataka sa kojom korisnik želi da radi. Ako se konekcija uspostavlja sa namerom kreiranja nove baze podataka ovo polje treba ostaviti prazno.
U polje Connection Timeout treba uneti vreme u sekundama za koje će program pokušavati da uspostavi konekciju i nakon čijeg isteka će se pokušaj obustaviti. Ovde treba ostaviti unapred izabranu opciju (15).
U polje Port treba uneti broj porta koji će biti korišćen prilikom konektovanja. Ovde treba ostaviti unapred izabranu opciju (3306).

Čekiranjem opcije Save password lozinka će biti sačuvana zajedno sa informacijama vezanim za konekciju, pa neće biti potrebno da se ponovo prosleđuje prilikom sledećih konektovanja. Ovu opciju treba ostaviti odčekiranu.
Klikom na dugme **Create** uspostavlja se konekcija sa MySQL serverom.
Prilikom sledećih konektovanja na server moguće je uspostaviti konekciju koja je nekada bila kreirana njenim izborom u delu History okvira za dijalog New Connection, unosom lozinke i klikom na dugme **Create**.
Nakon uspostavljanja konekcije pojavljuje se programski prozor.



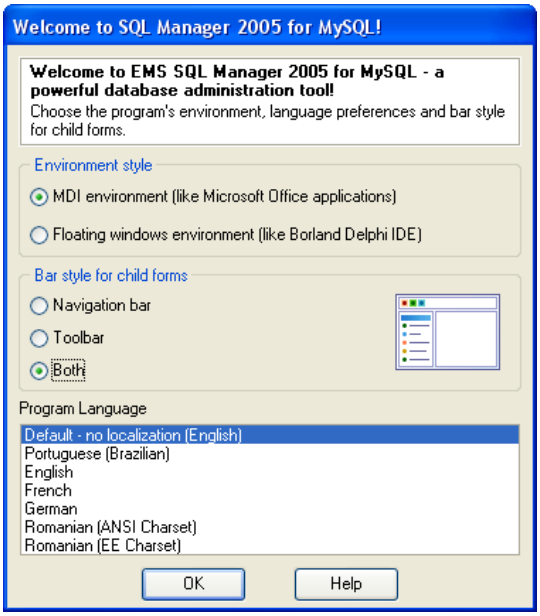
U programskom prozoru je otvoren Database Browser (pretraživač baza). Pomoću ovog alata se mogu pregledati objekti baza podataka iz različitih baza sa jednog ili više servera. Pretraživač baza se sastoji iz dva dela:

- Object panel (prozor sa objektima) (leva strana) - Prikazuje raspoložive tipove objekata
- Details panel (prozor sa detaljima) (desna strana) - Prikazuje detaljne informacije za objekat koji je selektovan u prozoru sa objektima

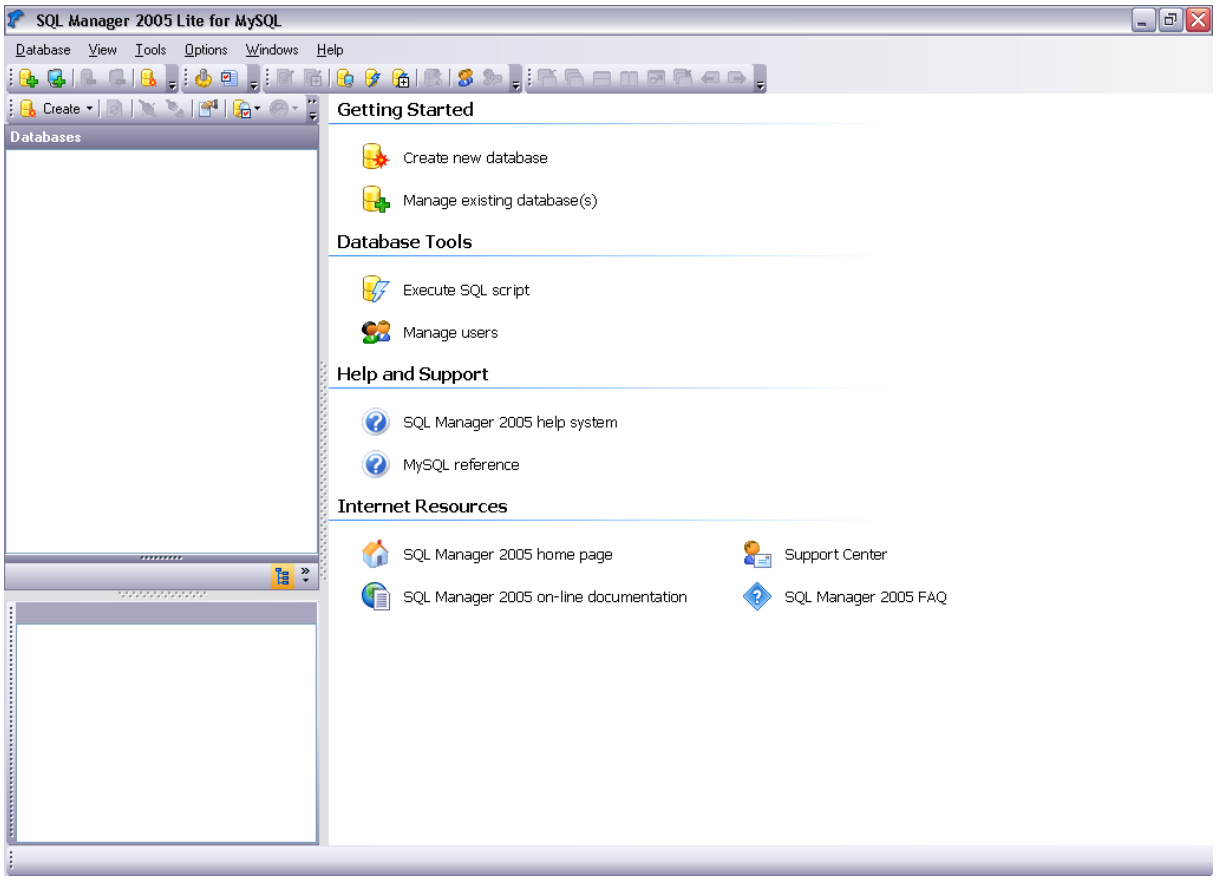
Pretraživač baza se može otvoriti izborom opcije **Database Browser** u meniju **Tools**.
U padajućoj listi Select a schema na vrhu prozora sa objektima se može izabrati baza koja se želi pregledati. Objekti izabrane baze su grupisani po katicama u prozoru sa objektima. Klikom na željenu karticu i izborom željenog objekta mogu se videti detaljne informacije za taj objekat u prozoru sa detaljima koji se nalazi sa desne strane. U donjem delu pretraživača baza se može videti dugme (može ih biti i više) koje označava konekciju koja se koristi trenutno. Ako postoji više uspostavljenih konekcija biće više dugmadi, a dugme koje je pritisnuto označava konekciju koja se trenutno koristi.

3.2.2 SQL Manager 2005 Lite for MySQL

SQL Manager Lite for MySQL je besplatan klijentski program. Proizvod je kompanije EMS (EMS Database Management Solutions). Trenutno aktuelna verzija je 3.7.7.1. Nakon instaliranja programa pojavljuje se pozdravni prozor na kome je potrebno izabrati neke od ponuđenih opcija.



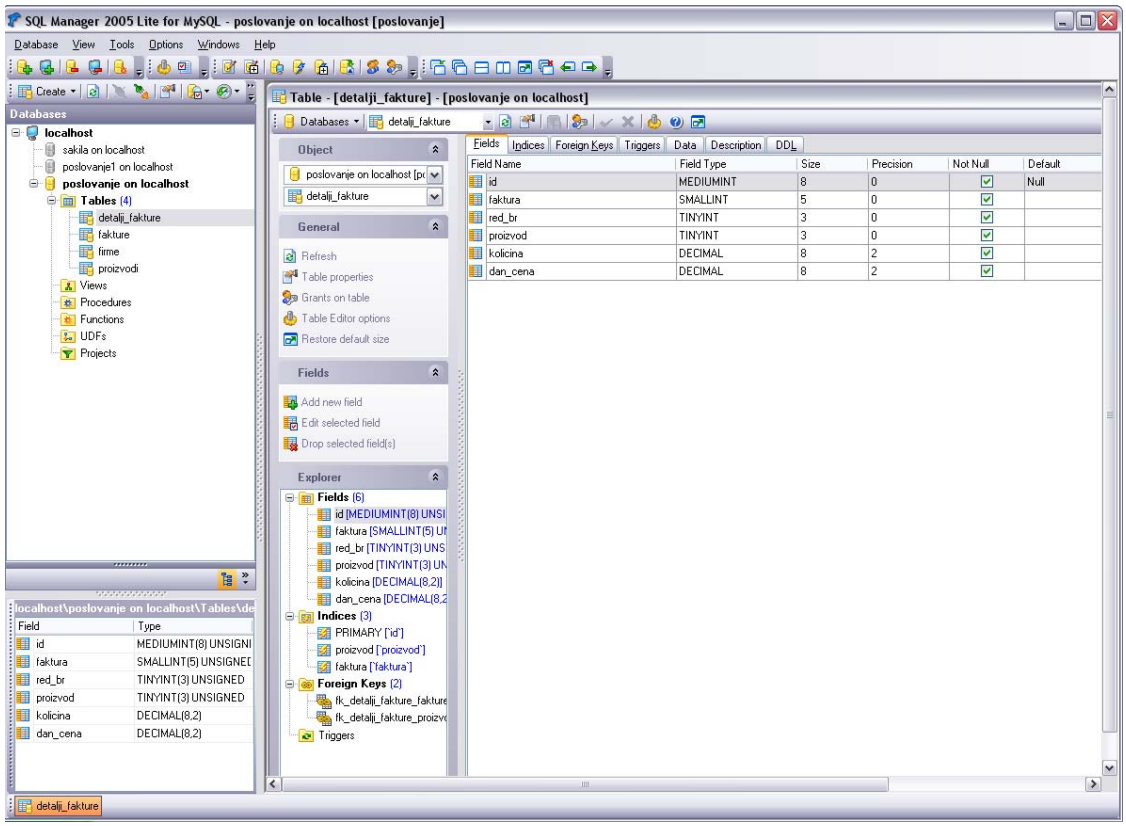
U ovom prozoru je u delu Environment style (stil okruženja) potrebno izabrati opciju **MDI environment (like Microsoft Office applications)**. Ovde se misli na stil korisničkog okruženja programa, odnosno ponašanje glavnog programskog prozora. Izborom navedene opcije definiše se da će okruženje biti nalik okruženju Microsoft Office aplikacija. Druga opcija (Floating windows environment (like Borland Delphi IDE)) daje okruženje sa plivajućim prozorima.
U delu Bar style for child forms definiše se lokacija akcionih dugmadi. Ona se mogu nalaziti u oknu za navigaciju (Navigation bar), na paletama sa alatkama (Toolbar) ili i u oknu za navigaciju i na paletama sa alatkama (Both). Ovde treba izabrati opciju **Both**.
U delu Program Language potrebno je izabrati jezik za interfejs iz liste ponuđenih jezika. Ovde treba izabrati **Default - no localization (English)**.
Klikom na dugme **OK** otvara se programski prozor.



Inicijalno postoje dve mogućnosti: Kreiranje nove baze podataka (**Create new database**) ili korišćenje postojećih baza podataka na serveru (**Manage existing database(s)**).
Izborom opcije **Manage existing database(s)** pokreće se čarobnjak za registrovanje baze podataka. Prva stranica služi za definisanje parametara za konekciju. Ovde treba zadržati ponuđene parametre i uneti lozinku.

Klikom na dugme **Next** prelazi se na sledeću stranicu čarobnjaka. Na ovoj stranici je potrebno izabrati željenu bazu podataka iz padajuće liste **Database name** i nakon toga kliknuti na dugme **Finish**.

U okviru **Databases** sa leve strane prozora treba kliknuti desnim tasterom miša na novokreiranu konekciju i izabrati opciju **Connect to Database**.
Nakon uspostavljene konekcije dvostrukim klikom na bilo koji od objekata baze podataka u okviru Databases otvara se prozor sa desne strane u kome se mogu videti detljne informacije za taj objekat podeljene po karticama.



4. Struktura jezika

Ovde će biti pomenuta pravila za pisanje sledećih elemenata SQL iskaza prilikom korišćenja MySQL-a:

- Literarne vrednosti, kao što su stringovi i brojevi
- Identifikatori, kao što su nazivi baza, tabela i kolona
- Komentari
- Rezervisane reči

4.1. Literarne vrednosti

U literarne vrednosti spadaju stringovi, brojevi, heksadecimalne vrednosti, boolean vrednosti i NULL.

4.1.1 Stringovi

String je niz karaktera ograđen bilo jednostrukim navodnicima ('), bilo dvostrukim navodnicima ("). Primeri:
'string'
"jos jedan string"
U okviru stringa određene sekvence imaju specijalno značenje. Svaka od ovih sekvenci počinje backslash karakterom (\) koji je poznat kao escape karakter. MySQL prepoznaje sledeće escape sekvence:

\'	Jednostruki navodnik (')
\"	Dvostruki navodnik (")
\b	Backspace karakter
\n	Prelazak u novu liniju
\t	Tab karakter
\\	Backslash karakter (\)

Za sve ostale escape sekvence backslash karakter se ignoriše. Na primer, \x se interpretira kao x.
Ove sekvence su osetljive na velika i mala slova. Na primer, \b se interpretira kao backspace, dok se \B tretira kao B.
Postoji više načina uključivanja navodnika unutar stringa:

- Jednostruki navodnik unutar stringa koji je ograničen jednostrukim navodnicima može biti napisan kao ''
- Dvostruki navodnik unutar stringa koji je ograničen dvostrukim navodnicima može biti napisan kao ""
- Korišćenjem escape sekvenci
- Jednostruki navodnik unutar stringa koji je ograničen dvostrukim navodnicima ne zahteva poseban tretman (ne zahteva dupliranje ili pisanje escape sekvence). Isto tako dvostruki navodnik unutar stringa koji je ograničen jednostrukim navodnicima ne zahteva poseban tretman.

4.1.2. Brojevi

Celi brojevi se predstavljaju nizom cifara. Realni brojevi koriste tačku (.) kao decimalni separator. Bilo koji tip brojeva može imati znak + ili znak - na početku radi označavanja pozitivnih ili negativnih vrednosti respektivno.
Primeri ispravno napisanih celih brojeva:
1232

0
-31
Primeri ispravno napisanih realnih brojeva:
128.48
-35.785e+10
120.00
Celi brojevi se mogu koristiti u izračunavanjima sa realnim brojevima i tada se tretiraju kao ekvivalentni realni brojevi.

4.1.3. Boolean vrednosti

Konstantama TRUE i FALSE odgovaraju vrednosti 1 i 0 respektivno. Nazivi konstanti se mogu pisati i velikim i malim slovima.
mysql> SELECT TRUE, true, FALSE, false;
-> 1, 1, 0, 0

4.1.4. NULL vrednosti

NULL vrednost znači „nema podataka“. NULL se može pisati i velikim i malim slovima.
Treba imati na umu da je NULL vrednost različita od vrednosti 0 za numeričke tipove podataka ili praznog stringa za string tipove podataka.

4.2. Nazivi baza, tabela, indeksa, kolona i alias-i

Nazivi baza, tabela, indeksa, kolona i alias-i su identifikatori.
Sledeća tabela definiše maksimalne dužine za svaki tip identifikatora.

Identifikator	Maksimalna dužina
Baza podataka	64
Tabela	64
Kolona	64
Indeks	64
Alias	255

Postoje određena ograničenja po pitanju karaktera koji se mogu pojavljivati u identifikatorima:

- Dozvoljeno je koristiti navodnike u identifikatorima, ali je najbolje to izbegavati
- Nazivi baza, tabela i kolona nebi trebalo da se završavaju space karakterom
- Nazivi baza ne mogu sadržati karaktere / \ . ili karaktere koji nisu dozvoljeni u Windows-u prilikom davanja naziva folderima
- Nazivi tabela ne mogu sadržati karaktere / \ . ili karaktere koji nisu dozvoljeni u Windows-u prilikom davanja naziva fajlovima

Identifikator može biti sa navodnicima ili bez navodnika. Ako je identifikator rezervisana reč ili sadrži specijalne karaktere korisnik mora da piše identifikator sa navodnicima svaki put kada ga koristi. Znak navoda za identifikator je znak backtick (`). (Izuzetak: reč koja ide nakon tačke u kvalifikatoru mora biti identifikator, pa ne mora biti pod navodnicima ako je rezervisana reč). U specijalne karaktere ne spadaju skup alfanumeričkih karaktera trenutnog seta karaktera, _ i $.
mysql> SELECT * FROM `select` WHERE `select`.id > 100;

4.2.1. Kvalifikatori identifikatora

MySQL dozvoljava nazive koji se sastoje od jednog identifikatora ili od više identifikatora. Ako naziv ima više identifikatora neophodno je da oni budu razdvojeni tačkom.
U MySQL kolona se može referencirati na jedan od sledećih načina:

Referenca kolone	Značenje
col_name	Kolona *col_name* iz bilo koje tabele korišćene u iskazu koja ima kolonu sa tim nazivom
tbl_name.col_name	Kolona *col_name* iz tabele *tbl_name* podrazumevane baze podataka
db_name.tbl_name.col_name	Kolona *col_name* iz tabele *tbl_name* baze podataka *db_name*

4.2.2. Osetljivost identifikatora na velika i mala slova

U MySQL-u bazama podataka na serveru odgovaraju folderi u okviru data foldera (C:\Program Files\MySQL\MySQL Server 5.0\data). Svakoj tabeli baze odgovara barem jedan fajl u folderu baze. Prema tome, osetljivost na velika i mala slova kod operativnog sistema određuje osetljivost na velika i mala slova kod naziva baza i tabela. Ovo znači da nazivi baza i tabela nisu osetljivi na velika i mala slova u Windows-u.
Iako nazivi baza i tabela nisu osetljivi na velika i mala slova nebi trebalo na različitim mestima u okviru istog iskaza u nazivu baze ili tabele koristiti različita slova (velika ili mala).
mysql> SELECT * FROM moja_tabela WHERE MOJA_TABELA.kol=1**;**
Nazivi kolona i indeksa i alias-i kolona nisu osetljivi na velika i mala slova. Nazivi okidača su osetljivi na velika i mala slova.
Alias-i tabela nisu osetljivi na velika i mala slova.
Najbolje je usvojiti doslednu konvenciju, kao što je da se prilikom kreiranja i referenciranja baza i tabela koriste nazivi sa malim slovima.

4.3. Sintaksa komentara

MySQL server podržava tri stila za komentare:

- Od karaktera # do kraja linije
- Od sekvence -- do kraja linije. Neophodno je da nakon druge crte postoji barem jedan razmak ili kontrolni karakter (space, tab, nova linija, itd.)
- Od sekvence /* do sekvence */. Ova sintaksa dozvoljava komentar u više redova, pošto početna i krajnja sekvenca ne moraju da budu u istom redu.

Sledi primer koji demonstrira sva tri stila:

```
mysql> SELECT 1+1; # Ovaj komentar se nastavlja do kraja linije
mysql> SELECT 1+1; -- Ovaj komentar se nastavlja do kraja linije
mysql> SELECT 1 /* ovo je komentar u liniji */ + 1;
mysql> SELECT 1+
/*
ovo je
komentar u više linija
*/
1;
```

4.4. Tretman rezervisanih reči u MySQL-u

Kod rezervisanih reči SQL-a ne pravi se razlika između malih i velikih slova. To je standardno u svim sistemima za rad sa bazama podataka.
Dozvoljeno je da se nazivi funkcija koriste kao identifikatori. Na primer, ABS je prihvatljivo kao naziv kolone. Međutim, treba znati da prilikom pozivanja funkcija nije dozvoljeno postojanje razmaka između naziva funkcije i otvorene zagrade koja sledi.

```
mysql> CREATE TABLE abs (val INT);
mysql> CREATE TABLE abs(val INT);
```

U drugom slučaju će se javiti greška u sintaksi pošto iskaz onda pokušava da pozove funkciju ABS().
Većina rezervisanih reči su zabranjene u standardnom SQL-u za nazive kolona i tabela.

5. Konvencije koje će biti korišćene u ovom materijalu

Rezervisane reči u SQL-u nisu osetljive na velika i mala slova. U ovom materijalu će rezervisane reči biti pisane velikim slovima.
U opisu sintaksi uglaste zagrade ([]) ukazuju na opcione reči ili klauzule.
DROP TABLE [IF EXISTS] *tbl_name*;
Kada element sintakse sadrži veći broj alternativa, alternative su razdvojene vertikalnim linijama (|). Kada jedna od alternativa može biti izabrana, alternative su smeštene unutar uglastih zagrada. Kada jedna od alternativa mora biti izabrana, alternative su smeštene unutar vitičastih zagrada ({}).
(...) ukazuje na izostavljanje dela iskaza, sa namerom da se prosledi kraća verzija kompleksnije sintakse. (...) takođe ukazuje da se prethodni element sintakse iskaza može ponavljati.

6. Data Definition Language (DDL)

Jezik za definisanje podataka (*Data Definition Language* (DDL)) omogućava formiranje strukture baze podataka.

6.1. Kreiranje baze podataka – CREATE DATABASE sintaksa

Nakon projektovanja strukture baze podataka, naredni korak je, sasvim logično, izdavanje komande MySQL-u za kreiranje nove baze podataka. To se radi pomoću SQL-ove komande CREATE DATABASE, na sledeći način:

CREATE DATABASE [IF NOT EXISTS] *db_name*;

CREATE DATABASE kreira bazu podataka sa datim nazivom. Ako već postoji baza podataka sa datim nazivom, a nije upotrebljeno IF NOT EXISTS, javiće se greška.
Kreiranje baze podataka poslovanje:
CREATE DATABASE *poslovanje*;
Ako korisnik želi da proveri da li je ova komanda uspešno izvršena može da izda komandu:
SHOW DATABASES;
Trebalo bi da se naziv nove baze podataka pojavi u spisku baza podataka na serveru.
Nazive baza podataka je najbolje pisati malim slovima. U nazivima baza ne treba koristiti srpske karaktere. U nazivima baza ne treba koristiti razmake. Ako se želi efekat razmaka treba koristiti donju crtu (_).
Sada na serveru postoji prazna baza podataka.

6.2. Biranje baze podataka – USE sintaksa

USE *db_name*;

USE *db_name* iskaz kaže MySQL-u da koristi bazu sa datim nazivom kao podrazumevanu (tekuću) bazu za iskaze koji slede. Baza sa datim nazivom ostaje podrazumevana do kraja sesije ili dok se ne prosledi neki drugi USE iskaz.
USE *poslovanje*;
Proglašavanje neke baze podrazumevanom (tekućom) pomoću USE iskaza ne sprečava korisnika da pristupa tabelama iz ostalih baza podataka na serveru.

6.3. Brisanje baze podataka – DROP DATABASE sintaksa

DROP DATABASE [IF EXISTS] *db_name*;

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

6.4. Kreiranje tabele - CREATE TABLE sintaksa

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name*
(*create_definition*,...)
[*table_option* ...]

ili:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name*
[(*create_definition*,...)]
[*table_option* ...]
select_statement

ili:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name*
{ LIKE *old_tbl_name* | (LIKE *old_tbl_name*) }

create_definition:
column_definition
| [CONSTRAINT [*symbol*]] PRIMARY KEY [*index_type*] (*index_col_name*,...)
| {INDEX|KEY} [*index_name*] [*index_type*] (*index_col_name*,...)
| [CONSTRAINT [*symbol*]] UNIQUE [INDEX|KEY]
[*index_name*] [*index_type*] (*index_col_name*,...)
| {FULLTEXT|SPATIAL} [INDEX|KEY] [*index_name*] (*index_col_name*,...)
| [CONSTRAINT [*symbol*]] FOREIGN KEY
[*index_name*] (*index_col_name*,...) [*reference_definition*]
| CHECK (*expr*)

column_definition:
col_name data_type [NOT NULL | NULL] [DEFAULT *default_value*]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT '*string*'] [*reference_definition*]

index_col_name:
col_name [(*length*)] [ASC | DESC]

index_type:
USING {BTREE | HASH}

reference_definition:
REFERENCES *tbl_name* [(*index_col_name*,...)]
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
[ON DELETE *reference_option*]

[ON UPDATE *reference_option*]

reference_option:
RESTRICT | CASCADE | SET NULL | NO ACTION

table_option:
{ENGINE|TYPE} [=] *engine_name*
| AUTO_INCREMENT [=] *value*
| AVG_ROW_LENGTH [=] *value*
| [DEFAULT] CHARACTER SET *charset_name*
| CHECKSUM [=] {0 | 1}
| COLLATE *collation_name*
| COMMENT [=] '*string*'
| CONNECTION [=] '*connect_string*'
| DATA DIRECTORY [=] '*absolute path to directory*'
| DELAY_KEY_WRITE [=] {0 | 1}
| INDEX DIRECTORY [=] '*absolute path to directory*'
| INSERT_METHOD [=] { NO | FIRST | LAST }
| MAX_ROWS [=] *value*
| MIN_ROWS [=] *value*
| PACK_KEYS [=] {0 | 1 | DEFAULT}
| PASSWORD [=] '*string*'
| ROW_FORMAT [=]
{DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
| UNION [=] (*tbl_name*[,*tbl_name*]...)

select_statement:
[IGNORE | REPLACE] [AS] SELECT ...

CREATE TABLE kreira tabelu sa datim nazivom. Tabela se kreira u podrazumevanoj (tekućoj) bazi. Greška se javlja ako tabela postoji, ako ne postoji podrazumevana (tekuća) baza ili ako baza ne postoji.
Naziv tabele može biti specificiran kao *db_name.tbl_name* da bi se kreirala tabela u određenoj bazi podataka. Ovo funkcioniše nezavisno od toga da li postoji podrazumevana (tekuća) baza, pod uslovom da baza sa datim nazivom postoji.
Prilikom kreiranja tabele se može koristiti TEMPORARY ključna reč. Tabela kreirana sa ovom ključnom reči je vidljiva samo u tekućoj konekciji (sesiji) i biva automatski obrisana nakon prekida konekcije.
Ključna reč IF NOT EXISTS sprečava pojavljivanje greške ako tabela sa istim nazivom postoji.
Nazive tabela je najbolje pisati malim slovima. U nazivima tabela ne treba koristiti srpske karaktere. U nazivima tabela ne treba koristiti razmake. Ako se želi efekat razmaka treba koristiti donju crtu (_).
Pomoću odredbe LIKE *old_tbl_name* može se napraviti nova tabela koja ima istu šemu (strukturu) kao neka druga tabela.
U komandi CREATE TABLE deklarišu se unutar zagrada potrebne kolone, njihovi tipovi podataka i druge informacije koje se tiču strukture tabele. Najjednostavnija definicija kolone sastoji se samo od naziva kolone i tipa podataka u koloni.
Komanda CREATE TABLE se može završiti komandom SELECT. SELECT je SQL-ova komanda koja omogućava učitavanje redova iz jedne ili više tabela. Pomoću ove opcije se može napuniti nova tabela podacima koje učitava zadata komanda SELECT.

6.4.1. Tipovi podataka

data_type predstavlja tip podataka prilikom definisanja kolona tabele.
MySQL podržava veliki broj tipova podataka koji se mogu razvrstati u više katgorija: numerički tipovi, tipovi za datum i vreme i znakovni ili tekstualni tipovi.

Numerički tipovi se koriste za skladištenje brojeva.

BIT[(*M*)]
Služi za definisanje binarnog tipa podataka. *M* definiše broj bitova i može biti od 1 do 64. Ukoliko se *M* izostavi podrazumeva se da je 1.

TINYINT[(*M*)] [UNSIGNED] [ZEROFILL]
Veoma mali celi brojevi. SIGNED opseg je od -128 do 127. UNSIGNED opseg je od 0 do 255.
M predstavlja širinu prikazivanja za celobrojne tipove podataka. Širina prikazivanja se koristi za prikazivanje celobrojnih vrednosti čija je širina manja od specificirane širine za kolonu dodavanjem space karaktera (razmaka) sa leve strane. Širina prikazivanja ne ograničava opseg vrednosti koji se može prikazati u koloni. Kada se koristi zajedno sa opcionim atributom ZEROFILL vrši se dodavanje nula (vodeće nule) a ne space karaktera sa leve strane. Na primer, ako je kolona deklarisana sa INT(5) ZEROFILL, vrednost 4 se predstavlja sa 00004.
Svi celobrojni tipovi podataka mogu imati opcioni atribut UNSIGNED. Kada se navede ovaj atribut moguće je unositi samo nenegativne brojeve, a gornja granica dozvoljenog opsega biva povećana, pri čemu veličina opsega ostaje ista.
Ako se za kolonu sa numeričkim tipom podataka navede atribut ZEROFILL, MySQL automatski dodaje UNSIGNED atribut.

BOOL, BOOLEAN
Ovi tipovi su sinonimi za TINYINT(1). false (netačno) odgovara vrednosti 0. true (tačno) odgovara bilo kojoj vrednosti različitoj od nule.

SMALLINT[(*M*)] [UNSIGNED] [ZEROFILL]
Mali celi brojevi. SIGNED opseg je od -32768 do 32767. UNSIGNED opseg je od 0 do 65535.

MEDIUMINT[(*M*)] [UNSIGNED] [ZEROFILL]
Srednje veliki celi brojevi. SIGNED opseg je od -8388608 do 8388607. UNSIGNED opseg je od 0 do 16777215.

INT[(*M*)] [UNSIGNED] [ZEROFILL]
Normalni celi brojevi. SIGNED opseg je od -2147483648 do 2147483647. UNSIGNED opseg je od 0 do 4294967295.

INTEGER[(*M*)] [UNSIGNED] [ZEROFILL]
Ovaj tip je sinonim za INT.

BIGINT[(*M*)] [UNSIGNED] [ZEROFILL]
Veliki celi brojevi. SIGNED opseg je od -9223372036854775808 do 9223372036854775807. UNSIGNED opseg je od 0 do 18446744073709551615.

Aproksimirani numerički tipovi su FLOAT i DOUBLE.

FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]
Mali (jednostruka preciznost) broj sa pokretnim zarezom. Dozvoljene vrednosti su od -3.402823466E+38 do -1.175494351E-38, 0 i od 1.175494351E-38 do 3.402823466E+38.
M je ukupan broj cifara, a *D* je broj cifara iza decimalne tačke. Ako se *M* i *D* izostave vrednosti se smeštaju u skladu sa hardverskim ograničenjima. Broj sa pokretnim zarezom jednostruke preciznosti je tačan do približno 7 decimalnih mesta.
Korišćenje ovog tipa podataka može dovesti do neočekivanih problema pošto se sva izračunavanja u MySQL-u rade sa dvostrukom preciznošću.

DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL]
Normalan (dvostruka preciznost) broj sa pokretnim zarezom. Dozvoljene vrednosti su od 1.7976931348623157E+308 do -2.2250738585072014E-308, 0 i od 2.2250738585072014E-308 do 1.7976931348623157E+308.
M je ukupan broj cifara, a *D* je broj cifara iza decimalne tačke. Ako se *M* i *D* izostave vrednosti se smeštaju u skladu sa hardverskim ograničenjima. Broj sa pokretnim zarezom dvostruke preciznosti je tačan do približno 15 decimalnih mesta.

DOUBLE PRECISION[(*M*,*D*)] [UNSIGNED] [ZEROFILL], REAL[(*M*,*D*)] [UNSIGNED] [ZEROFILL]
Ovi tipovi su sinonimi za DOUBLE.

Tačni numerički tipovi su tip INTEGER i njegove varijante i DECIMAL.

DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]
Broj sa fiksnim zarezom. *M* je ukupan broj cifara (preciznost), a *D* je broj cifara iza decimalne tačke (the scale). Decimalna tačka i (za negativne brojeve) znak – se ne računaju u *M*. Ako je *D* jednako nuli brojevi nemaju decimalnu tačku i razlomljeni deo. Maksimalan ukupan broj cifara (*M*) za ovaj tip podataka je 65. Maksimalan broj cifara iza decimalne tačke (*D*) za ovaj tip podataka je 30. Ako se *D* izostavi, podrazumevana vrednost je 0. Ako se *M* izostavi, podrazumevana vrednost je 10.
Sva osnovna izračunavanja (+, -, *, /) sa ovim tipom podataka se rade sa preciznošću od 65 cifara.
Ovaj tip podataka se obično koristi za rad sa novčanim vrednostima.

DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]
Ovi tipovi su sinonimi za DECIMAL.

DATE
Datum. Podržani opseg je od '1000-01-01' do '9999-12-31'. MySQL prikazuje vrednosti za ovaj tip podataka u formatu 'YYYY-MM-DD'.

DATETIME
Kombinacija datuma i vremena. Podržani opseg je od '1000-01-01 00:00:00' do '9999-12-31 23:59:59'. MySQL prikazuje vrednosti za ovaj tip podataka u formatu 'YYYY-MM-DD HH:MM:SS'.

TIMESTAMP
Ovo je koristan tip podataka za kolone tabele. Ako se u određenom redu ne zada vrednost za kolonu ovog tipa u kolonu se upisuje vreme kada je red dodat tabeli ili kada je poslednji put izmenjen sadržaj reda.

Opseg je od '1970-01-01 00:00:01' UTC do '2038-01-09 03:14:07' UTC. TIMESTAMP vrednosti su smeštene kao broj sekundi proteklih od epohe ('1970-01-01 00:00:00' UTC). TIMESTAMP vrednost se vraća kao string sa formatom 'YYYY-MM-DD HH:MM:SS' i sa širinom prikazivanja fiksiranom na 19 karaktera.

TIME
Vreme. Podržani opseg je od '-838:59:59' do '838:59:59'. MySQL prikazuje vrednosti za ovaj tip podataka u formatu 'HH:MM:SS'.

YEAR[(2|4)]
Godina kao dvocifreni ili četvorocifreni broj. Podrazumevani format je četvorocifreni broj. Kod četvorocifrenog formata podržani opseg je od 1901 do 2155 i 0000. Kod dvocifrenog formata podržani opseg je od 70 do 69, što odgovara godinama od 1970. do 2069. MySQL prikazuje vrednosti za ovaj tip podataka u formatu YYYY.

[NATIONAL] CHAR(*M*) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
String fiksne dužine kome prilikom smeštanja uvek bivaju dodati space karakteri sve do specificirane dužine. *M* predstavlja dužinu, odnosno broj znakova. Opseg za *M* je od 0 do 255 karaktera.
Kada se podatak tipa CHAR upiše u kolonu tabele, on uvek ima dužinu koja je zadata u definiciji kolone. To se postiže dopunjavanjem podatka u koloni razmacima. Ti razmaci se automatski uklanjaju pri učitavanju podatka iz kolone tipa CHAR.
Očigledno je da podaci tipa CHAR zauzimaju više prostora na disku od ekvivalentnih znakovnih vrednosti promenljive dužine. Prednost im je što se podaci brže učitavaju iz tabele čije su sve kolone fiksne dužine (recimo CHAR ili DATE).
Deklaracijama tipova CHAR i VARCHAR može slediti rezervisana reč BINARY, što znači da se pri poređenju znakovnih vrednosti pravi razlika između malih i velikih slova. Podrazumevani način poređenja je da se ta razlika ne pravi.
CHAR je skraćenica za CHARACTER. NATIONAL CHAR (ili odgovarajuća skraćenica NCHAR) je standardni način da se u SQL-u definiše da kolona sa tipom podataka koristi neki unapred definisani skup karaktera.
CHARACTER SET atribut specificira skup karaktera koji se koristi. COLLATE atribut specificira uparivanje za taj skup karaktera. Skup karaktera je skup simbola i šifriranja. Uparivanje je skup pravila za poređenje karaktera u skupu karaktera.
CHARSET je sinonim za CHARACTER SET.

CHAR [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
Ovo je sinonim za CHAR(1).

[NATIONAL] VARCHAR(*M*) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
String varijabilne dužine. *M* predstavlja maksimalnu dužinu podataka. Opseg za *M* je od 0 do 65535 karaktera. VARCHAR je sinonim za CHARACTER VARYING.

BINARY(*M*)
BINARY tip podataka je sličan CHAR tipu podataka, ali smešta binarne stringove umesto nebinarnih stringova sa karakterima.

VARBINARY(*M*)
VARBINARY tip podataka je sličan VARCHAR tipu podataka, ali smešta binarne stringove umesto nebinarnih stringova sa karakterima.

TINYBLOB
BLOB kolona sa maksimalnom dužinom od 255 bajtova. BLOB je veliki binarni objekat koji može sadržati promenljivu količinu podataka. BLOB kolone se tretiraju kao binarni stringovi.

TINYTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
TEXT kolona sa maksimalnom dužinom od 255 karaktera.
TEXT kolone su slične VARCHAR kolonama, a razlika je u tome što za TEXT kolonu nije moguće definisati podrazumevanu (DEFAULT) vrednost.

BLOB[(*M*)]
BLOB je skraćenica od Binary Large Object (veliki binarni objekat). BLOB kolona može da ima maksimalnu dužinu od 65535 bajta.
Za ovaj tip se može dati opciona dužina *M*. Ako se definiše MySQL kreira kolonu koja je najmanji BLOB tip dovoljno veliki da prima vrednosti koje su *M* bajtova dugačke.

TEXT[(*M*)] [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
TEXT kolona sa maksimalnom dužinom od 65535 karaktera.
Za ovaj tip se može dati opciona dužina *M*. Ako se definiše MySQL kreira kolonu koja je najmanji TEXT tip dovoljno veliki da prima vrednosti koje su *M* karaktera dugačke.

MEDIUMBLOB
BLOB kolona sa maksimalnom dužinom od 16777215 bajtova (16 MB).

MEDIUMTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
TEXT kolona sa maksimalnom dužinom od 16777215 karaktera (16 MB).
Omogućava skladištenje tekstualnih podataka dužih od onog što može da stane u tip VARCHAR.

LONGBLOB
BLOB kolona sa maksimalnom dužinom od 4294967295 bajtova (4 GB).

LONGTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
TEXT kolona sa maksimalnom dužinom od 4294967295 karaktera (4GB).
Omogućava skladištenje tekstualnih podataka dužih od onog što može da stane u tip VARCHAR.

ENUM('*value1*','*value2*',...) [CHARACTER SET *charset_name*]
[COLLATE *collation_name*]
Nabrajanje. Ovaj tip podataka omogućava zadavanje liste mogućih vrednosti. Kolona tabele može sadržati samo jednu vrednost, izabranu iz liste vrednosti '*value1*','*value2*',..., NULL ili specijalnu '' greška vrednost. ENUM kolona može imati maksimalno 65535 različitih vrednosti. ENUM vrednosti su interno predstavljene kao celi brojevi.
pol ENUM ('m', 'ž')

SET('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]
Skup. Kolona tabele može sadržati jednu ili više vrednosti, a svaka od njih mora biti izabrana iz liste vrednosti '*value1*','*value2*',... SET kolona može imati maksimalno 64 člana. SET vrednosti su interno predstavljene kao celi brojevi.

6.4.2. Ostali atributi

- Svaka kolona se može deklarisati kao NOT NULL ili NULL, što znači da se ne dozvoljava da kolona sadrži vrednost NULL (opcija NOT NULL) ili da se prihvata vrednost NULL (opcija NULL). Ako nisu specificirani ni NULL ni NOT NULL atribut, kolona se tretira kao da je NULL atribut specificiran.
- Kolone sa celobrojnim tipom podataka mogu imati dodatni atribut AUTO_INCREMENT. Kada se u indeksiranu AUTO_INCREMENT kolonu unese vrednost NULL (preporučeno) ili vrednost 0, vrednost u koloni se podešava na sledeću vrednost u nizu. Tipično to je *value*+1 gde je *value* najveća vrednost u koloni trenutno u tabeli. AUTO_INCREMENT nizovi počinju brojem 1. Može postojati samo jedna AUTO_INCREMENT kolona po tabeli, mora biti indeksirana i ne može imati podrazumevanu (DEFAULT) vrednost. AUTO_INCREMENT funkcioniše ispravno ako sadrži samo pozitivne vrednosti.
- DEFAULT klauzula specificira podrazumevanu vrednost za kolonu. Sa jednim izuzetkom, podrazumevana vrednost mora biti konstanta; ne može biti funkcija ili izraz. Izuzetak je da se može specificirati CURRENT_TIMESTAMP kao podrazumevana vrednost za TIMESTAMP kolonu. Kolone tipa BLOB i TEXT ne mogu imati definisanu podrazumevanu vrednost.
- Komentar za kolonu može biti specificiran pomoću COMMENT opcije i može imati dužinu do 255 karaktera

6.4.3. Ograničenja (Constraints)

Ako se redovno sortiraju podaci u tabeli po jednoj istoj koloni ili kolonama potrebno je napraviti indeks za tu kolonu ili te kolone. Indeks je interna tabela vrednosti koja održava red, odnosno redosled zapisa. Na ovaj način, kada je potrebno da se sortiraju podaci ili kada je potrebno da se određeni podatak pronađe brzo MySQL može da vrši pretraživanje po indeks ključevima po poznatom redosledu, što je bolje od sekvencijalnog pretraživanja po podacima.

Pravljenje indeksa usporava unošenje podataka; svaki novi zapis, obrisani zapis ili promena vrednosti u indeksiranoj koloni zahteva promenu indeksa. Indeksirane kolone treba koristiti samo onda kada su stvarno neophodne.

KEY (ključ) je normalno sinonim za INDEX (indeks) i znači da će zadata kolona (ili kolone) biti indeksirana. Indeksi se koriste da bi se brzo pronašli redovi sa određenom vrednošću u koloni. Atribut PRIMARY KEY se može specificirati i kao KEY kada je dat u definiciji kolone.

UNIQUE (jedinstveni) indeks kreira ograničenje takvo da sve vrednosti u indeksu (odnosno koloni koja je indeksirana) moraju biti različite. Ukoliko korisnik pokuša da doda novi red u tabelu koji ima istu vrednost u koloni za koju je definisan UNIQUE indeks kao neki drugi red javiće se greška. Ovo ograničenje se ne odnosi na NULL vrednosti sem za BDB mašine za smeštanje podataka. Za ostale mašine za smeštanje podataka UNIQUE indeks dozvoljava višestruke NULL vrednosti za kolone koje mogu sadržati NULL vrednosti.

PRIMARY KEY (primarni ključ) je jedinstveni indeks gde sve kolone ključa moraju biti definisane kao NOT NULL. Ako nisu eksplicitno deklarisane kao NOT NULL, MySQL ih deklariše implicitno. Tabela može imati samo jedan primarni ključ. Na ovaj način se postavlja ograničenje nad tabelom. To znači da se upisom novog zapisa u kolonu koja je primarni ključ ne sme upisati vrednost koja već postoji. Za kolonu (ili kolone) koja je primarni ključ se automatski formira indeks.

U kreiranoj tabeli primarni ključ je smešten kao prvi, zatim dolaze svi UNIQUE (jedinstveni) indeksi i na kraju svi nejedinstveni indeksi.

Primani ključ može biti indeks nad više kolona. Međutim, nije moguće kreirati indeks nad više kolona korišćenjem atributa PRIMARY KEY u specifikaciji kolone. Mora se koristiti posebna PRIMARY KEY(*index_col_name, ...*) klauzula.
U MySQL-u naziv primarnog ključa je PRIMARY. Za ostale indekse, ako im se ne dodeli naziv, indeksu se daje naziv prve indeksirane kolone, zajedno sa opcionim sufiksom (_2, _3, ...) da bi bio jedinstven.
Za mašine za skladištenje koje nisu tipa InnoDB je moguće prilikom definisanja kolone koristiti REFERENCES *tbl_name*(*col_name*) koja nema stvarni efekat i služi samo kao podsetnik ili komentar korisniku da se namerava da kolona koja se trenutno definiše ukazuje na kolonu neke druge tabele.
Neke mašine za skladištenje dozvoljavaju definisanje tipa indeksa prilikom kreiranja indeksa.
Specifikacija za *index_col_name* se može završavati sa ASC ili DESC. Ove ključne reči su dozvoljene za buduće ekstenzije za specificiranje smeštanja vrednosti u indeksu po rastućem ili opadajućem redosledu. Trenutno se ove ključne reči ignorišu; vrednosti u indeksu su uvek smeštene po rastućem redosledu.
Moguće je kreirati specijalne FULLTEXT indekse koji se koriste za Full-text (tekstualna) pretraživanja. Samo MyISAM mašina za skladištenje podržava FULLTEXT indekse. Oni mogu biti kreirani samo za CHAR, VARCHAR i TEXT kolone.
Moguće je kreirati SPATIAL indekse za prostorne tipove podataka (dozvoljavaju generisanje, smeštanje i analizu geografskih obeležja). Ovi tipovi podataka su podržani samo u MyISAM tabelama i indeksirane kolone moraju biti deklarisane kao NOT NULL.
InnoDB tabele podržavaju ograničenja tipa strani ključ (FOREIGN KEY). Strani ključ je skup kolona (jedna ili više kolona) iz jedne tabele (sekundarna tabela ili dete tabela) čije vrednosti moraju da se „slažu“, odnosno da odgovaraju vrednostima primarnog ključa u nekoj drugoj tabeli (primarna tabela ili roditelj tabela). Definisanjem stranog ključa se u stvari uspostavlja relacija između tabela. Sintaksa za definisanje stranog ključa kod ovog tipa tabela izgleda ovako:

[CONSTRAINT *symbol*] FOREIGN KEY [*id*] (*index_col_name*, ...)
REFERENCES *tbl_name* (*index_col_name*, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

Prilikom definisanja stranog ključa obe tabele moraju biti InnoDB tabele i ne smeju biti privremene (TEMPORARY) tabele.
InnoDB odbija bilo koju INSERT ili UPDATE operaciju koja pokušava da kreira vrednost za strani ključ u dete tabeli ako ne postoji odgovarajuća vrednost ključa u roditelj tabeli. Akcija koju preduzima InnoDB za bilo koju UPDATE ili DELETE operaciju koja pokušava da promeni ili obriše vrednost ključa u roditelj tabeli koji ima povezane redove u dete tabeli zavisi od referencijalne akcije specificirane korišćenjem ON UPDATE i ON DELETE podklauzula u FOREIGN KEY klauzuli. Kada korisnik pokuša da obriše ili promeni red u roditelj tabeli, a postoje jedan ili više povezanih redova u dete tabeli, InnoDB podržava 5 opcija vezano za akciju koja će biti izvršena:

- CASCADE: Brisanjem ili izmenom reda u roditelj tabeli se automatski brišu ili menjaju povezani redovi u dete tabeli
- SET NULL: Brisanjem ili izmenom reda u roditelj tabeli se vrednost u koloni koja je strani ključ dete tabele za povezane redove postavlja na NULL. Ovo važi samo ako kolone koje su strani ključ nemaju specificiran NOT NULL kvalifikator.
- NO ACTION: U standardnom SQL-u NO ACTION znači da nema akcije u smislu da neće biti moguće obrisati ili promeniti vrednost primarnog ključa u roditelj tabeli ako

postoji povezana vrednost stranog ključa u dete tabeli. InnoDB odbija operaciju brisanja ili izmene za roditelj tabelu.

- RESTRICT: Odbija se operacija brisanja ili izmene za roditelj tabelu
- SET DEFAULT: Ova akcija je prepoznata od strane parsera, ali InnoDB odbija definicije tabela koje sadrže ON DELETE SET DEFAULT ili ON UPDATE SET DEFAULT klauzule

Odgovarajuće kolone koje čine strani i primarni ključ moraju imati slične tipove podataka da bi mogle da budu poređene bez konverzije podataka. Veličina i znak celobrojnih tipova moraju biti isti. Dužine string tipova ne moraju biti iste.

6.4.4. Mašine za skladištenje i tipovi tabela

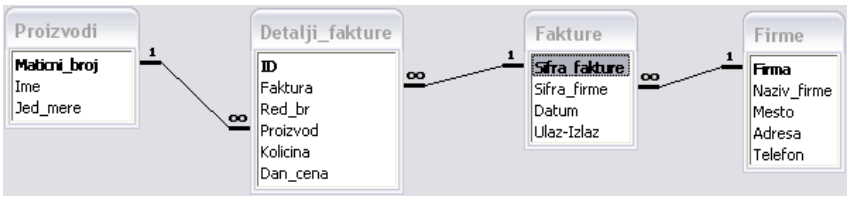
MySQL podržava nekoliko mašina za skladištenje koji rade kao manipulatori za različite tipove tabela. Među ovim mašinama postoje one koje rade sa transakcionim i one koje rade sa netransakcionim tabelama:

- MyISAM upravlja netransakcionim tabelama. Obezbeđuje visoku brzinu smeštanja i preuzimanja podataka kao i mogućnost fulltext pretraživanja.
- MEMORY mašina za skladištenje radi sa tabelama koje su smeštene u memoriji računara i nikada se ne upisuju na disk. Zahvaljujući tome te tabele su veoma brze, ali veličina im je ograničena i podaci iz njih se ne mogu restaurirati ukoliko dođe do kvara sistema. MERGE mašina za skladištenje, takođe poznata i kao MRG_MyISAM mašina omogućava rad sa kolekcijom identičnih MyISAM tabela kao sa jednom tabelom. Pod identičnim tabelama se podrazumeva da tabele imaju identične kolone i indekse. To se može iskoristiti kada operativni sistem ograničava maksimalnu veličinu fajlova – pa zbog toga i tabela. MEMORY i MERGE rade sa netransakcionim tabelama.
- InnoDB mašina za skladištenje radi sa transakcionim tabelama i ima mogućnosti izvršavanja, rollback-a i oporavka prilikom havarije. Ova mašina radi zaključavanje na nivou redova prilikom transakcija. Dizajnirana je za maksimum performansi kada obrađuje velike količine podataka. Ova mašina podržava spoljne ključeve. Ovo je podrazumevana mašina za skladištenje.
- ARCHIVE mašina za skladištenje se koristi za smeštanje velikih količina podataka bez indeksa.

Ovo su samo neke od ukupno 11 mašina za skladištenje.

6.4.5. Kreiranje tabela u bazi podataka poslovanje

Izabran je primer baze podataka **poslovanje** koja služi za praćenje poslovanja jedne firme koja radi sa određenom grupom proizvoda i sarađuje sa određenim brojem drugih firmi.
Baza podataka **poslovanje** ima četiri tabele i to: priozvodi, firme, fakture i detalji_fakture. Na sledećoj slici se mogu videti sve tabele sa nazivima kolona i primarnim ključevima (nazivi kolona ispisani polucrnim slovima), kao i uspostavljene relacije i strani ključevi u odgovarajućim tabelama. Može se videti da su sve uspostavljene relacije tipa jedan-prema više.



Na sledećoj slici je prikazan primer tabela koje bi trebalo da budu povezane preko stranog ključa. Polje Faktura iz tabele Detalji_fakture može uzimati samo one vrednosti koje se pojavljuju u polju Sifra_fakture tabele Fakture.

Detalji_fakture : Table

ID	Faktura	Red_br	Proizvod	Kolicina	Dan_cena
1	1	2	3	50,00	240,00
2	2	1	4	50,00	25,00
3	2	2	2	36,00	11,00
4	2	3	3	50,00	23,00
5	3	1	1	20,00	28,00
6	3	2	2	35,00	35,00
7	3	3	3	50,00	100,00
8	3	4	4	50,00	15,00
9	4	1	4	50,00	23,00
10	4	2	2	50,00	18,00
11	5	1	4	50,00	20,00
12	5	2	4	50,00	500,00
13	5	3	2	25,00	24,00
14	6	1	4	50,00	220,00
15	6	2	4	50,00	840,00
16	7	1	2	26,00	470,00
17	8	1	2	25,00	840,00
18	8	2	3	50,00	1000,00
19	6	3	2	24,00	250,00
20	6	4	3	50,00	230,00
21	6	5	5	50,00	12,00
22	1	3	3	50,00	250,00
23	9	1	3	25,00	26,00
24	9	2	2	14,00	24,00
25	10	1	4	150,00	150,00
26	10	2	5	100,00	100,00
27	10	3	3	125,00	23,00
28	10	4	2	100,00	250,00
29	10	5	3	1,00	1,00
30	10	5	2	5,00	5,00
ber)	0	0	0	0,00	0,00

Fakture : Table

Sifra_fakture	Sifra_firme	Datum	Ulaz-Izlaz
1	3	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	3	6.11.2006	2
7	4	4.11.2006	2
8	2	23.10.2006	1
9	3	2.11.2006	2
10	4	8.10.2006	1
11	3	15.10.2006	1
0	0	20.1.2007	

Na sledećoj slici prikazani su i dve preostale tabele baze podataka **poslovanje** sa unetim podacima.

Proizvodi : Table

Maticni_broj	Ime	Jed_mere
1	Četka	kom.
2	Lak	lit.
3	Stiropor	m2
4	Gips	kg
5	Destilovana voda	kg
6	Šmirgla	kom.
0		

Firme : Table

Firma	Naziv_firme	Mesto	Adresa	Telefon
1	BALKAN	Niš	Mokranjčeva 13	018-522-854
2	STIL	Beograd	Takovska 10	011-562-365
3	KOKOMAX	Niš	Dušanova 33	018-352-666
4	HELIO	Subotica	Nikole Tesle 55	027-555-125
0				

Kreiranje tabele **proizvodi**:

```
CREATE TABLE proizvodi (
  maticni_broj TINYINT(3) UNSIGNED NOT NULL,
  ime VARCHAR(50) NOT NULL,
  jed_mere VARCHAR(20) NOT NULL,
  PRIMARY KEY  (maticni_broj)
);
```

Kreiranje tabele **firme**:

```
CREATE TABLE firme (
  firma TINYINT(3) UNSIGNED NOT NULL,
```

```
  naziv_firme VARCHAR(100) NOT NULL,
  mesto VARCHAR(50) NOT NULL,
  adresa VARCHAR(50) NOT NULL,
  telefon VARCHAR(20) NOT NULL,
  PRIMARY KEY  (firma)
);
```

Kreiranje tabele **fakture**:

```
CREATE TABLE fakture (
  sifra_fakture SMALLINT(5) UNSIGNED NOT NULL,
  sifra_firme TINYINT(3) UNSIGNED NOT NULL,
  datum DATE NOT NULL,
  ulaz_izlaz CHAR(1) NOT NULL,
  PRIMARY KEY  (sifra_fakture),
  KEY k_sifra_firme (sifra_firme),
  CONSTRAINT fk_fakture_firme FOREIGN KEY (sifra_firme) REFERENCES firme (firma)
  ON DELETE CASCADE ON UPDATE CASCADE
);
```

Kreiranje tabele **detalji_fakture**:

```
CREATE TABLE detalji_fakture (
  id MEDIUMINT(8) UNSIGNED NOT NULL AUTO_INCREMENT,
  faktura SMALLINT(5) UNSIGNED NOT NULL,
  red_br TINYINT(3) UNSIGNED NOT NULL,
  proizvod TINYINT(3) UNSIGNED NOT NULL,
  kolicina DECIMAL(8,2) NOT NULL,
  dan_cena DECIMAL(8,2) NOT NULL,
  PRIMARY KEY  (id),
  KEY k_proizvod (proizvod),
  KEY k_faktura (faktura),
  CONSTRAINT fk_detalji_fakture_proizvodi FOREIGN KEY (proizvod) REFERENCES
  proizvodi (maticni_broj) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT fk_detalji_fakture_fakture FOREIGN KEY (faktura) REFERENCES fakture
  (sifra_fakture) ON DELETE CASCADE ON UPDATE CASCADE
);
```

6.5. Kreiranje indeksa – CREATE INDEX sintaksa

Svi indeksi koji su neophodni najčešće će se kreirati automatski prilikom kreiranja tabele. Za svaku kolonu koja se deklariše sa opcijom PRIMARY KEY, KEY, UNIQUE ili INDEX, automatski se formira i indeks. Ako se ustanovi da se koristi veći broj upita koji obuhvataju kolonu za koju nije definisan indeks novi indeks se može dodati pomoću komande CREATE INDEX:

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

index_col_name:
col_name [(length)] [ASC | DESC]

index_type:
USING {BTREE | HASH}
```

Primer:

CREATE INDEX *k_naziv_firme* ON *firme* (*naziv_firme*);
Ovde je bio kreiran indeks za kolonu *naziv_firme* tabele *firme*. Naziv indeksa je *k_naziv_firme*.

Zanimljivo je da se, pre izvršavanja, komanda CREATE INDEX preslikava u komandu ALTER TABLE.
Indeksi definisani nad kolonama tipa CHAR, VARCHAR, BINARY i VARBINARY mogu se ograničiti na prvih nekoliko znakova u polju. To se može uraditi tako što se iza imena indeksirane kolone zada između zagrada broj znakova koji želi da se indeksira:

CREATE INDEX *k_naziv_firme* ON *firme* (*naziv_firme*(5));

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

6.6. Brisanje tabele – DROP TABLE sintaksa

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

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

DROP TABLE *detalji_fakture*; (Ovo je samo primer korišćenja komande. Komandom se briše tabela *detalji_fakture*.)

6.7. Brisanje indeksa – DROP INDEX sintaksa

DROP INDEX *index_name* ON *tbl_name*;

Da bi se izbrisao indeks mora se zadati i naziv tabele kojoj je pridružen.

DROP INDEX *k_naziv_firme* ON *firme*;
Ovde je obrisan prethodno kreirani indeks *k_naziv_firme*.

6.8. Izmena strukture postojeće tabele – ALTER TABLE sintaksa

ALTER [IGNORE] TABLE *tbl_name*
alter_specification [, *alter_specification*] ...

alter_specification:
ADD [COLUMN] *column_definition* [FIRST | AFTER *col_name*]
| ADD [COLUMN] (*column_definition*,...)
| ADD {INDEX|KEY} [*index_name*] [*index_type*] (*index_col_name*,...)
| ADD [CONSTRAINT [*symbol*]]
PRIMARY KEY [*index_type*] (*index_col_name*,...)
| ADD [CONSTRAINT [*symbol*]]
UNIQUE [INDEX|KEY] [*index_name*] [*index_type*] (*index_col_name*,...)
| ADD [FULLTEXT|SPATIAL] [INDEX|KEY] [*index_name*] (*index_col_name*,...)
| ADD [CONSTRAINT [*symbol*]]

FOREIGN KEY [*index_name*] (*index_col_name*,...)
[*reference_definition*]
| ALTER [COLUMN] *col_name* {SET DEFAULT *literal* | DROP DEFAULT}
| CHANGE [COLUMN] *old_col_name column_definition*
[FIRST|AFTER *col_name*]
| MODIFY [COLUMN] *column_definition* [FIRST | AFTER *col_name*]
| DROP [COLUMN] *col_name*
| DROP PRIMARY KEY
| DROP {INDEX|KEY} *index_name*
| DROP FOREIGN KEY *fk_symbol*
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] *new_tbl_name*
| ORDER BY *col_name*
| CONVERT TO CHARACTER SET *charset_name* [COLLATE *collation_name*]
| [DEFAULT] CHARACTER SET *charset_name* [COLLATE *collation_name*]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| *table_option* ...

index_col_name:
col_name [(*length*)] [ASC | DESC]

index_type:
USING {BTREE | HASH}

ALTER TABLE omogućava promenu strukture postojeće tabele. Na primer, moguće je dodati ili izbrisati kolone, kreirati ili uništiti indekse, promeniti tip podataka za postojeće kolone ili preimenovati kolone ili celu tabelu.
Sintaksa za mnoge dozvoljene promene je slična sintaksi za klauzule CREATE TABLE iskaza.
U većini slučajeva ALTER TABLE radi tako što pravi privremenu kopiju originalne tabele. Izmene se vrše na kopiji, a zatim se originalna tabela briše a privremena tabela se preimenuje u originalnu.
Ako se koristi ALTER TABLE *tbl_name* RENAME TO *new_tbl_name* bez dodatnih opcija MySQL jednostavno preimenuje sve fajlove koji odgovaraju tabeli *tbl_name*. U ovom slučaju nema potrebe za pravljenjem privremene tabele.
Opcije CHANGE i MODIFY zapravo su jedna te ista opcija i omogućavaju izmenu definicije kolone ili njenog mesta u tabeli.
Opcija DROP COLUMN briše kolonu iz tabele, dok opcije DROP PRIMARY KEY i DROP INDEX brišu samo indeks pridružen koloni.
Opcija DISABLE KEYS nalaže MySQL-u da ne ažurira sadržaj nejedinstvenih indeksa, ali upotrebljiva je samo za MyISAM tabele. Opcija ENABLE KEYS uključuje ažuriranje indeksa.
Opcija ORDER BY će poređati zadatim redosledom redove tabele na koju je primenjena. Taj redosled neće biti očuvan kasnije kada se budu dodavali ili brisali podaci.
Primeri:

ALTER TABLE *fakture*
ADD COLUMN *primio* VARCHAR(50);
Ovde se dodaje nova kolona sa nazivom *primio* i tipom podataka VARCHAR(50) tabeli *fakture*.

ALTER TABLE *fakture*
DROP COLUMN *primio*;
Ovde se briše prethodno dodata kolona.

ALTER TABLE *detalji_fakture*
MODIFY COLUMN *kolicina* DECIMAL(6,2);
Ovde se menja definicija kolone *kolicina* tabele *detalji_fakture* u smislu promene podešavanja za tip podataka.

ALTER TABLE *detalji_fakture*
MODIFY COLUMN *kolicina* DECIMAL(8,2);
Ovde se ponovo menja definicija kolone *kolicina* tabele *detalji_fakture* u smislu vraćanja na prvobitnu definiciju.

ALTER TABLE *detalji_fakture*
CHANGE COLUMN *dan_cena cena* DECIMAL(10,2);
Ovde se menja definicija kolone *dan_cena* tabele *detalji_fakture* tako što se naziv kolone iz *dan_cena* menja u *cena* i pored svega se menjaju podešavanja za tip podataka (u prvobitnoj definiciji je bilo DECIMAL(8,2)).

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 (server smešta redove koji se umeću u bafer).
- 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.

Ako se prilikom umetanja jednog reda prosledi NULL vrednosti koloni koja je deklarisana kao NOT NULL iskaz neće biti izvršen i javiće se poruka o grešci. Prosleđivanjem NULL vrednosti koloni koja je deklarisana kao NOT NULL prilikom umetanja većeg broja redova ili kada se koristi INSERT … SELECT oblik naredbe 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, 'Četka', 'kom.'),
 (2, 'Lak', 'lit.'),
 (3, 'Stiropor', 'm2'),
 (4, 'Gips', 'kg'),
 (5, 'Destilovana voda', 'kg'),
 (6, 'Šmirgla', 'kom.');
```

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

Unos podataka u tabelu **firme**:

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

firma	naziv_firme	mesto	adresa	telefon
1	BALKAN	Niš	Mokranjčeva 13	018-522-854
2	STIL	Beograd	Takovska 10	011-562-365
3	KOKOMAX	Niš	Dušanova 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	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

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	50.00	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

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* = 'Niš';
Ovde se brišu redovi iz tabele *firme* u kojima je vrednost u koloni *mesto* Niš (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 = 'Obrenovićeva 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	Niš	Mokranjčeva 13	018-522-854
2	STIL	Beograd	Takovska 10	011-562-365
3	KOKOMAX	Niš	Dušanova 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.	Četka
lit.	Lak
m2	Stiropor
kg	Gips
kg	Destilovana voda
kom.	Šmirgla

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

Dodatni 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 jednostavne upite, ali kada se bude prešlo na složenije upite, ona će omogućiti korisniku da na nedvosmislen način definiše kriterijume za podatke koji su mu potrebni.

7.6.3. Alijasi

Kolonama i tabelama 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:

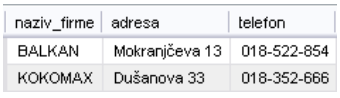
```
SELECT f.naziv_firme
FROM firme f;
```

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 = 'Niš';
```

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	Mokranjčeva 13	018-522-854
KOKOMAX	Dušanova 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 koji 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	2006-10-25	1
2	4	2006-10-28	1
5	3	2006-10-25	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
Niš
Beograd
Niš
Subotica

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

SELECT DISTINCT *mesto* FROM *firme*;

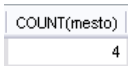
daje sledeće rezultate:

mesto
Niš
Beograd
Subotica

U ovom primeru duplikati se ne pojavljuju.
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*;

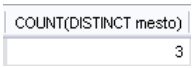
vraća broj vrednosti u koloni *mesto* tabele *firme* ne računajući NULL vrednosti:



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	50.00
2	35.00
3	20.00
4	50.00
5	25.00
6	24.00
7	26.00
8	45.00
9	14.00
10	60.00
11	5.00

Ovaj upit nalazi minimalnu količinu na svakoj fakturi. Funkcija MIN() vraća minimalnu vrednost iz grupe prosleđenih vrednosti.

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	12150.0000
3	32500.0000
4	13500.0000
5	19400.0000
6	45670.0000
8	15425.0000
10	81050.0000

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	Niš	Dušanova 33	018-352-666
1	BALKAN	Niš	Mokranjčeva 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	Četka	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	Šmirgla	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 predvidiv.
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	2006-10-25	2
6	BALKAN	2006-11-06	2
9	BALKAN	2006-11-02	2
1	STIL	2006-10-25	1
8	STIL	2006-10-23	1
4	KOKOMAX	2006-10-29	2
5	KOKOMAX	2006-10-25	1
11	KOKOMAX	2006-10-15	1
2	HELIO	2006-10-28	1
7	HELIO	2006-11-04	2
10	HELIO	2006-10-08	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	2006-10-25	1
1	STIL	2006-10-25	1
1	KOKOMAX	2006-10-25	1
1	HELIO	2006-10-25	1
2	BALKAN	2006-10-28	1
2	STIL	2006-10-28	1
2	KOKOMAX	2006-10-28	1
2	HELIO	2006-10-28	1
3	BALKAN	2006-10-25	2
3	STIL	2006-10-25	2
3	KOKOMAX	2006-10-25	2
3	HELIO	2006-10-25	2
4	BALKAN	2006-10-29	2
4	STIL	2006-10-29	2
4	KOKOMAX	2006-10-29	2
4	HELIO	2006-10-29	2
5	BALKAN	2006-10-25	1
5	STIL	2006-10-25	1
5	KOKOMAX	2006-10-25	1
5	HELIO	2006-10-25	1
6	BALKAN	2006-11-06	2
6	STIL	2006-11-06	2
6	KOKOMAX	2006-11-06	2
6	HELIO	2006-11-06	2
7	BALKAN	2006-11-04	2
7	STIL	2006-11-04	2
7	KOKOMAX	2006-11-04	2
7	HELIO	2006-11-04	2
8	BALKAN	2006-10-23	1
8	STIL	2006-10-23	1
8	KOKOMAX	2006-10-23	1
8	HELIO	2006-10-23	1
9	BALKAN	2006-11-02	2
9	STIL	2006-11-02	2
9	KOKOMAX	2006-11-02	2
9	HELIO	2006-11-02	2
10	BALKAN	2006-10-08	1
10	STIL	2006-10-08	1
10	KOKOMAX	2006-10-08	1
10	HELIO	2006-10-08	1
11	BALKAN	2006-10-15	1
11	STIL	2006-10-15	1
11	KOKOMAX	2006-10-15	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 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

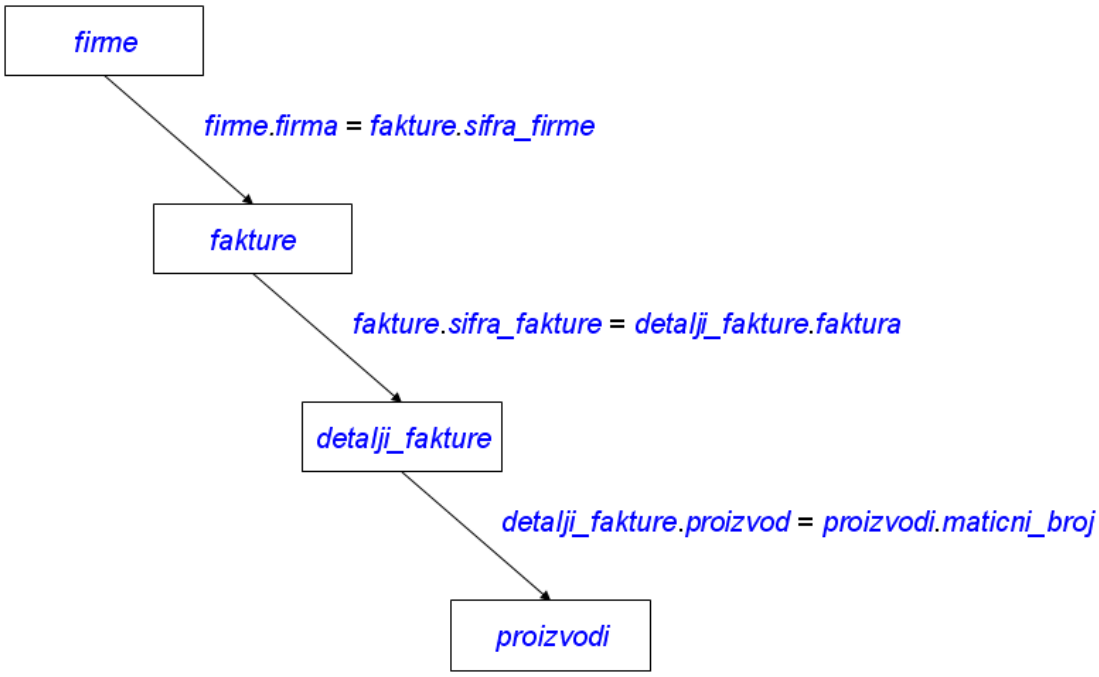
proizvoda za svaku stavku svake fakture (kolona *proizvod* tabele *detalji_fakture*). Na osnovu matičnih brojeva proizvoda moguće je naći nazive proizvoda u tabeli *proizvodi* (kolona *ime* tabele *proizvodi*). Potrebno je samo na kraju prikazati različite proizvode korišćenjem ključne reči DISTINCT.
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
Četka
Destilovana voda
Šmirgla

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 firma 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', 'Ustanička 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:

naziv_firme
ADRIA

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 primarnog 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	Niš	Mokranjčeva 13	3	2006-10-25	2
BALKAN	Niš	Mokranjčeva 13	6	2006-11-06	2
BALKAN	Niš	Mokranjčeva 13	9	2006-11-02	2
STIL	Beograd	Takovska 10	1	2006-10-25	1
STIL	Beograd	Takovska 10	8	2006-10-23	1
KOKOMAX	Niš	Dušanova 33	4	2006-10-29	2
KOKOMAX	Niš	Dušanova 33	5	2006-10-25	1
KOKOMAX	Niš	Dušanova 33	11	2006-10-15	1
HELIO	Subotica	Nikole Tesle 55	2	2006-10-28	1
HELIO	Subotica	Nikole Tesle 55	7	2006-11-04	2
HELIO	Subotica	Nikole Tesle 55	10	2006-10-08	1
ADRIA	Beograd	Ustanička 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)
31500.0000

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 jeste 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*);

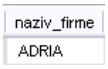
U ovom upitu se traži matični broj i ime proizvoda koji odgovara stavci sa najvećim iznosom posmatrajući sve stavke na svim fakturama.
Izvršavanjem gornjeg upita dobijaju se sledeći rezultati:

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 operator 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.



naziv_firme
ADRIA

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 firmi prosleđuje u konačan skup rezultata upita.

naziv_firme	firma
ADRIA	5

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	50.00
2	50.00
3	60.00
4	50.00
5	120.00
6	80.00
7	26.00
8	50.00
9	50.00
10	200.00
11	5.00

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	10000.0000
2	8800.0000
3	12000.0000
4	9500.0000
5	15000.0000
6	16100.0000
7	8060.0000
8	12500.0000
9	4480.0000
10	31500.0000
11	375.0000

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	2006-10-08	1
11	3	2006-10-15	1
8	2	2006-10-23	1
1	2	2006-10-25	1
5	3	2006-10-25	1
2	4	2006-10-28	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	50.00	25.00
Šmirgla	35.00	60.00
Stiropor	40.00	220.00

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	50.00	200.00
4	2	3	3	40.00	220.00
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
11	5	1	5	120.00	20.00
12	5	2	4	100.00	150.00
14	6	1	4	75.00	200.00
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
21	8	2	3	50.00	250.00
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

8. Upotreba ugrađenih funkcija u komandi SELECT

MySQL sadrži veliki broj ugrađenih operatora i funkcija koji su već bili pominjani. Većina njih se prvenstveno upotrebljava u SELECT i WHERE odredbama. Takođe, postoje i specijalne (agregatne) funkcije koje se koriste za grupu podataka (u odredbi GROUP BY), kao što su COUNT() ili MAX().
Biće razmotren deo najkorisnijih funkcija koje nudi MySQL.
Treba zapamtiti da se u MySQL-u svaki izraz koji sadrži vrednost NULL svodi na vrednost NULL, sem u nekoliko slučajeva koji će biti naglašeni.
U svim MySQL izrazima se mogu zadavati zagrade da bi se upravljalo redosledom kojim se izračunavaju pojedini delovi izraza, na isti način kao i u bilo kom drugom programskom jeziku.

8.1. Operatori

Ranije je napomenuto da se u upitima za selektovanje podataka, u WHERE klauzuli definišu uslovi koje moraju da zadovonjavaju zapisi koji se traže. Sam uslov je logički izraz koji ima logičku vrednost tačno ili netačno (*True* ili *False*). Podaci će biti prikazani samo za one zapise koje zadovoljavaju taj uslov. Sam logički izraz može biti kombinacija manje složenih izraza koji su povezani logičkim operatorima AND, OR i NOT. Oni su samo jedna vrsta operatora o kojima će ovde biti više reči.
Operatori omogućavaju da se dodaju brojevi, upoređuju vrednosti, spajaju stringovi (grupa tekstualnih karaktera) i kreiraju kompleksni relacioni izrazi. Operatori naznačavaju da je potrebno da se određena operacija sprovede nad jednim ili više elemenata. Neki česti primeri operatora su:
=, AND, +, ...
Razlikuju se sledeće vrste operatora:

- Matematički (aritmetički) operatori
- Relacioni operatori (za poređenje vrednosti)
- Logički operatori

8.1.1. Matematički operatori

U matematičke operatore spadaju:

*	Množenje
+	Sabiranje
-	Oduzimanje
/	Deljenje

Matematički operatori se koriste za rad sa brojevima, odnosno sa bilo kojim numeričkim tipom podataka. Broj može biti konstantna vrednost, vrednost promenljive ili sadržaj polja.

*** operator (množenje)**
Jednostavan primer za korišćenje ovog operatora je sračunavanje iznosa za stavke faktura. U bazi poslovanje u tabeli *detalji_fakture* se unose podaci o količini kupljenih proizvoda i današnjoj ceni po proizvodu. Želi se da se izračunamo i iznos za stavke faktura. U ovom slučaju kolona za izračunavanje će sadržati formulu *kolicina* * *dan_cena*. Koristi se tabela *detalji_fakture*.

```
SELECT faktura, red_br, proizvod, kolicina, dan_cena, (kolicina * dan_cena) AS cena
FROM detalji_fakture;
```

faktura	red_br	proizvod	kolicina	dan_cena	cena
1	1	3	50	200	10000
2	1	5	50	25	1250
2	2	6	35	60	2100
2	3	3	40	220	8800
3	1	1	20	75	1500
3	2	2	35	300	10500
3	3	3	60	200	12000
3	4	4	50	170	8500
4	1	4	50	190	9500
4	2	1	50	80	4000
5	1	5	120	20	2400
5	2	4	100	150	15000
5	3	6	25	80	2000
6	1	4	75	200	15000
6	2	6	50	85	4250
6	3	2	24	330	7920
6	4	3	70	230	16100
6	5	5	80	30	2400
7	1	2	26	310	8060
8	1	6	45	65	2925
8	2	3	50	250	12500
9	1	1	50	70	3500
9	2	5	25	25	625
9	3	2	14	320	4480
10	1	4	150	210	31500

+ operator (sabiranje)
Ako želimo da formiramo kolonu koja sabira vrednosti iz drugih kolona, na primer, kolona *cena* i *porez* tada bi se koristio izraz *cena* + *porez*. Ova formula koristi operator za sabiranje radi sabiranja vrednosti iz obe kolone i prikazivanja rezultata u koloni koji sadrži formulu.

- operator (oduzimanje)
Primer korišćenja ovog operatora je oduzimanje popusta od konačnog iznosa na fakturi. Formula bi bila:
bruto_iznos – (*bruto_iznos* * *popust*)
Iako zagrade nisu matematički operatori, one igraju važnu ulogu u radu sa operatorima, jer određuju redosled izvršavanja operacija.

/ operator (deljenje)
Ovaj operator se može koristiti za deljenje dva broja i prikazivanje rezultata na željenom mestu. Na primer, može se pretpostaviti da je 212 ljudi dobilo premiju na lutriji od 1000000 dinara. Formula za izračunavanje pojedinačnog dobitka će biti 1000000/212, što daje 4716.98 dinara.

8.1.2. Relacioni operatori – operatori za poređenje

Postoji šest osnovnih relacionih operatora koji su poznati i kao operatori poređenja. Oni porede dve vrednosti ili izraza. Relacioni operatori su:

=	jednako
< >, !=	nije jednako
<	manje
< =	manje ili jednako
>	veće
> =	veće ili jednako

Izrazi koji su formirani korišćenjem relacionih operatora vraćaju ili YES (tačno) ili NO (netačno) ili NULL (nepoznato/nema vrednosti).
Ako bilo koja strana izraza koji koriste ove operatore ima NULL vrednost, rezultat će uvek biti NULL vrednost.

= operator (jednako)
Ovaj operator vraća logičku vrednost tačno ako su dva izraza koji se porede ista.
ulaz_izlaz = '1' vraća vrednost tačno ako je faktura ulazna (vrednost 1), odnosno netačno ako se radi o izlaznim fakturama. Dakle, ako treba prikazati samo ulazne fakture, koristi se sledeći upit:

```
SELECT fakture.sifra_fakture, fakture.sifra_firme, fakture.datum, fakture.ulaz_izlaz
FROM fakture
WHERE fakture.ulaz_izlaz = '1';
```

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
8	2	23.10.2006	1
10	4	8.10.2006	1
11	3	15.10.2006	1

U većini slučajeva, pri poređenju vrednosti znakovnog tipa u MySQL-u se ne pravi razlika između malih i velikih slova. Ako se želi da se znakovne vrednosti porede tako da se ipak pravi razlika između malih i velikih slova, treba ispred jedne od njih dodati prefiks BINARY.
Primer pretrage po mestu firme koja ne pravi razliku izmedju malih i velikih slova:

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

naziv_firme	mesto
BALKAN	Nis
KOKOMAX	Nis

Primer pretrage po mestu firme koja pravi razliku izmedju malih i velikih slova:

```
SELECT firme.naziv_firme, firme.mesto
FROM firme
WHERE firme.mesto = BINARY 'nis';
```

Ovakvim upitom neće se vratiti ni jedna vrednost jer ni jedna firma nema naziv mesta nis, već Nis.

<> (nije jednako)
Ovaj operator je suprotnost operatora jednako. Kriterijum *ulaz_izlaz* <> '1' vraća vrednost tačno ako se radi o fakturi koja nije ulazna, dakle vraća izlazne fakture kojim odgovara vrednost 2 u toj koloni. Za ovaj operator mogu se koristiti oznake <> i !=.

< (manje)
Ovaj operator vraća vrednost tačno ako je leva strana izraza manja od desne.

<= (manje ili jednako)
Ovaj operator vraća vrednost tačno ako je leva strana izraza manja ili jednaka u odnosu na desnu.

> (veće)
Ovaj operator je suprotan od operatora manje. Ovaj operator vraća vrednost tačno ako je leva strana izraza veća od desne:

```
SELECT firme.firma
FROM firme
WHERE firme.firma > 3;
```

firma
4
5

>= (veće ili jednako)
Ovaj operator vraća vrednost tačno ako je leva strana izraza veća ili jednaka u odnosu na desnu.

8.1.3. Predikati

Postoje i drugi operatori za poređenje koji se zovu predikati:

BETWEEN ... AND	(između dve vrednosti)
IN	(u skupu vrednost)
IS NULL	(vrednost u polju ne postoji)

BETWEEN ... AND operator
Ovaj operator se koristi da se utvrdi da li se vrednost nalazi u definisanom opsegu vrednosti. Sintaksa je:
<Izraz > BETWEEN <vrednost 1> AND <vrednost 2>
Ako je vrednost između navedenih vrednosti rezultat je tačno, a u suprotnom je rezultat netačno.
Sledeći upit prikazuje samo one stavke faktura kod kojih je iznos između 500 i 1500:

```
SELECT faktura, red_br, proizvod, kolicina, dan_cena, (kolicina * dan_cena) AS cena
FROM detalji_fakture
WHERE (kolicina * dan_cena) BETWEEN 500 AND 1500;
```

faktura	red_br	proizvod	kolicina	dan_cena	cena
2	1	5	50	25	1250
3	1	1	20	75	1500
9	2	5	25	25	625

IN operator
Ovaj operator se koristi da bi se odredilo da li se vrednost poklapa sa nekom vrednošću iz liste vrednosti. Sintaksa je:
<Izraz> IN <vrednost 1, vrednost 2, vrednost 3, …>
Ako je vrednost nađena u listi rezultat je tačno, a u suprotnom je rezultat netačno.
Primer: Napraviti upit koji nalazi sve firme koje su iz Niša ili iz Beograda.

```
SELECT firme.naziv_firme, firme.mesto
FROM firme
WHERE firme.mesto IN ('Nis', 'Beograd');
```

naziv_firme	mesto
BALKAN	Nis
STIL	Beograd
KOKOMAX	Nis
ADRIA	Beograd

IS NULL
Koristi se da bi se odredilo da li objekat sadrži nešto u sebi, tj da li n ima vrednost NULL. Sintaksa glasi:
n IS NULL
Primer: firma koja nema ni jednu fakturu:

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

8.1.4. Logički operatori

Postoji više logičkih operatora. Oni se koriste za postavljanje uslova u izrazima. Logički operatori se koriste da se naprave kompleksni izrazi sa više uslova. Kao i relacioni operatori uvek vraćaju vrednosti tačno, netačno ili NULL. Logički operatori su:

AND ili &&	logičko **i**
OR ili ||	logičko uključivo **ili**
XOR	logičko isključivo **ili**
NOT ili !	logičko **ne**

AND operator (logičko i)
Ovaj operator vraća vrednost tačno samo ako su oba uslova tačna. Sintaksa je:
<Izraz 1> AND <Izraz 2>
ime = 'Milan' AND *prezime* = 'Kostić' je tačno samo ako su oba uslova tačna.
Sledeći upit naći će firmue koje su iz Niša i čija je adresa Dušanova 33.

```
SELECT firme.naziv_firme
FROM firme
WHERE firme.mesto = 'Nis' AND firme.adresa = 'Dusanova 33';
```

naziv_firme
KOKOMAX

Pri tome treba obratiti pažnju na sledeće:
True AND True = True
False AND bilo šta = false
Svi ostali izrazi se svode na NULL.

OR operator (logičko uključivo ili)
Ovaj operator vraća vrednost tačno ako je bar jedan od izraza tačan. Sintaksa je:
<Izraz 1> OR <Izraz 2>
Donjim upitom biće pronađeni svi proizvodi čiji je naziv četka ili je jedinica mere kilogram:

SELECT *proizvodi.ime*, *proizvodi.jed_mere*
FROM *proizvodi*
WHERE *proizvodi.ime* = 'Cetka' OR *proizvodi.jed_mere* = 'kg';

ime	jed_mere
Cetka	kom.
Gips	kg
Destilovana voda	kg

Pri tome, treba voditi računa o sledećem:
True OR bilo šta = True
NULL OR False = NULL
NULL OR NULL = NULL
False OR False = False

NOT operator (logičko ne)
Ovaj operator se koristi za negaciju. Operator vraća vrednost tačno ako uslov nije tačan. Ovaj operator menja logički rezultat izraza. Za NULL vraća NULL. Sintaksa glasi:
NOT <Izraz>
NOT *dan_cena* >= 1000
vraća tačno ako je cena manja od 1000, odnosno prikazuje samo one redove kod kojih današnja cena nije veća ili jednaka od 1000.

XOR operator (logičko isključivo ili)
Vraća vrednost tačno jedino u slučajevima da jedan od izraza ima vrednost tačno ili oba izraza imaju vrednost tačno. Ukoliko jedan od izraza ima vrednost NULL, vraća vrednost NULL.

8.2. Funkcije za upravljanje tokom komandi

Među ovim funkcijama su najkorisnije IF i CASE, koje deluju slično kao ekvivalentni iskazi u većini programskih jezika.

8.2.1. Funkcija IF

Sintaksa za ovu funkciju je:
IF(e1, e2, e3)
Ako izraz e1 ima vrednost tačno, funkcija IF vraća rezultat e2; u suprotnom, rezultat funkcije je e3.
Na primer, ako se hoće da se umesto 1 i 2 u koloni *ulaz_izlaz* za fakture vide vrednosti 'ulazna' i 'izlazna' respektivno, upit će izgledati ovako:

SELECT *fakture.sifra_fakture*, IF(*fakture.ulaz_izlaz* = '1', 'ulazna', 'izlazna') AS *tip*
FROM *fakture*;

sifra_fakture	tip
1	ulazna
2	ulazna
3	izlazna
4	izlazna
5	ulazna
6	izlazna
7	izlazna
8	ulazna
9	izlazna
10	ulazna
11	ulazna

8.2.2. Funkcija CASE

Sintaksa za ovu funkciju je:
CASE
WHEN [condition] THEN result
  [WHEN [condition] THEN result ...]
   [ELSE result]
END

Treba napraviti upit koji u zavisnosti od zastarelosti fakture vraća određeni komentar, i to ako je faktura starija od 1. januara 2006. godine, onda će biti vođena kao arhivirana, ako je starija od 1. novembra 2006. godine, onda će biti vođena kao kao stara, a ako je njen datum nakon 1. novembra 2006. godine onda će biti vođena kao kao aktuelna.

SELECT *sifra_fakture*,
CASE
WHEN *datum* < '2006-01-01' THEN 'arhivirana'
WHEN *datum* < '2006-11-01' THEN 'stara'
ELSE 'aktuelna'
END AS *komentar*
FROM *fakture*;

sifra_fakture	komentar
1	stara
2	stara
3	stara
4	stara
5	stara
6	aktuelna
7	aktuelna
8	stara
9	aktuelna
10	stara
11	stara

8.3. Funkcije za obradu znakovnih vrednosti - String operatori

MySQL-ove funkcije za rad sa znakovnim vrednostima se dele u dve kategorije: funkcije za obradu znakovnih vrednosti i funkcije za poređenje znakovnih vrednosti.

8.3.1. Funkcije za obradu znakovnih vrednosti

Najznačajnija među ovim funkcijama je:
CONCAT(*s1*, *s2*, ...) koja spaja znakovne vrednosti.

SELECT CONCAT(*adresa*, ', ', *mesto*) AS *lokacija*
FROM *firme*;

lokacija
Mokranjceva 13, Nis
Takovska 10, Beograd
Dusanova 33, Nis
Nikole Tesle 55, Subotica
Ustanicka 39, Beograd

REPLACE(*izvor*, *tekst*, *zamena*) vraća niz znakova *izvor* u kojem je podniz *tekst* zamenjen nizom *zamena* na svim mestima gde se pojavljuje.
Sledećim upitom je u koloni *mesto* string Nis zamenjen stringom NIS.

SELECT *naziv_firme*,
REPLACE(*mesto*, 'Nis', 'NIS') AS *mesto*
FROM *firme*;

naziv_firme	mesto
BALKAN	NIS
STIL	Beograd
KOKOMAX	NIS
HELIO	Subotica
ADRIA	Beograd

Pored ovih, postoji veliki broj drugih funkcija za rad sa znakovnim vrednostima. Biće navedene samo neke od njih:
LOWER(*s*) i UPPER(*s*) - Pretvara niz znakova u mala odnosno velika slova
LOAD_FILE(*naziv_fajla*) - Vraća sadržaj fajla *naziv_fajla* kao jedan niz znakova
TRIM(*s*) - Uklanja početne i završne *space* karaktere iz niza znakova *s*
LTRIM(*s*) - Uklanja samo leve *space* karaktere
RTRIM(*s*) - Uklanja samo desne *space* karaktere

8.3.2. Funkcije za poređenje znakovnih vrednosti

Osim operatora jednakosti, postoji i više drugih funkcija za poređenje dve znakovne vrednosti:
LIKE - Poređenje sa džokerima
RLIKE - Poređenje sa regularnim izrazima
STRCMP - Poređenje znakovnih vrednosti
MATCH - Tekstualno pretraživanje
Napomena: Tekstualno pretraživanje (MATCH) je moguće samo u MyISAM tabelama i ovde neće biti posebno obrađivano.

LIKE (isto kao ...)
Ovaj operator kao i njegova varijanta NOT LIKE se koriste za upoređivanje dva stringa. Ovaj operator dozvoljava upotrebu „džokera". On određuje da li se objekti sa leve i desne strane izraza poklapaju ili ne. Rezultat može biti tačno, netačno ili NULL. NULL se dobija kao rezultujuća vrednost ukoliko je bilo koji od objekata sa leve ili desne strane NULL. Sintaksa je:
<objekat> LIKE <uzorak>
Ovaj operator pruža mogućnost za poređenje stringova korišćenjem džokera. Stavljanjem specijalnog karaktera iza ili ispred dela podatka koji nam je poznat dolazimo da podataka:

% - Poklapanje sa grupom znakova bilo koje dužine
_ - Zamenjuje jedan (bilo koji) karakter (A do Z, od 0 do 9)

RLIKE omogućava poređenje znakovnih vrednosti sa regularnim izrazima. Regularan izraz je šablon koji opisuje opšti oblik znakovne vrednosti uz uslove koji se opisuju posebnom notacijom. Uslovi se mogu odnositi na višestrukost ponavljanja određenog stringa.
Sledeći upiti pokazuju razliku između upotrebe predhodna dva operatora.

SELECT * FROM *firme* WHERE *naziv_firme* RLIKE 'il';

firma	naziv_firme	mesto	adresa	telefon
2	STIL	Beograd	Takovska 10	011-562-365

SELECT * FROM *firme* WHERE *naziv_firme* LIKE 'il';

firma	naziv_firme	mesto	adresa	telefon

Nema ni jednog rezultata.

Sledeći upit izdvaja sve fakture koje su fakturisane u oktobru 2006. godine:

SELECT * FROM *fakture* WHERE *datum* RLIKE '2006-10';

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
8	2	23.10.2006	1
10	4	8.10.2006	1
11	3	15.10.2006	1

Korišćenjem džokera, možemo napraviti i upit koji vraća sve firme koje u imenu mesta imaju slovo i:

SELECT * FROM *firme* WHERE *mesto* LIKE '%i%';

firma	naziv_firme	mesto	adresa	telefon
1	BALKAN	Nis	Mokranjceva 13	018-522-854
3	KOKOMAX	Nis	Dusanova 33	018-352-666
4	HELIO	Subotica	Nikole Tesle 55	027-555-125

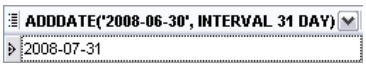
8.4. Numeričke funkcije

ABS(*n*) - Vraća apsolutnu vrednost broja *n*
CEILING(*n*) - Vraća vrednost *n* zaokruženu naviše na najbliži ceo broj
FLOOR(*n*) - Vraća vrednost *n* zaokruženu naniže na najbliži ceo broj
MOD(*n*,*m*) - Deli *n* sa *m* i vraća celobrojni ostatak deljenja
DIV(*n*,*m*) - Deli *n* sa *m* i vraća celobrojni rezultat
POWER(*n*,*m*) - Vraća *n* podignuto na stepen *m*
RAND(*n*) - Vraća slučajno generisan broj u opsegu od 0 do 1. Parametar *n* nije obavezan, ali se njime inicijalizuje algoritam za generisanje pseudoslučajnih brojeva.
ROUND(*n*,[*d*]) - Vraća *n* zaokruženo oa najbližu celobrojnu vrednost. Ako se zada parametar *d*, vrednost se zaokružuje na *n* decimalnih mesta
SQRT(*n*) - Vraća kvadratni koren od *n*

8.5. Funkcije za rad sa datumom i vremenom

U MySQL-u postoji mnogo različitih funkcija koje omogućavaju rad sa datumom i vremenom. Ovde će biti razmotrene samo neke koje se najčešće koriste:
ADDDATE(*datum*,INTERVAL *n tip*) i
SUBDATE(*datum*, INTERVAL *n tip*)
Ove funkcije služe za sabiranje i oduzimanje datuma, odnosno vremenskih intervala. Vrednosti koja je zadata parametrom *datum* dodaje se, odnosno od nje se oduzima period koji je zadat pomoću rezervisane reči INTERVAL. Pri tome se mora zadati i količina *n* i *tip* te količine.
Sam parametar *tip* može imati vrednost SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, itd.
Primer1:
Napraviti upit koji će datum povećati za 31 dan.

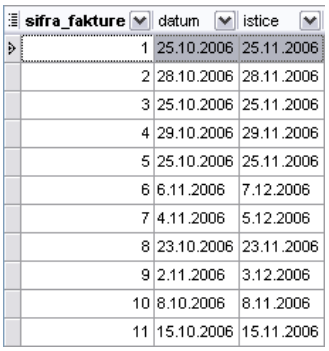
SELECT ADDDATE('2008-06-30', INTERVAL 31 DAY);



ADDDATE('2008-06-30', INTERVAL 31 DAY)
2008-07-31

Primer 2 :
Napraviti upit koji prikazuje datum kada ističe rok za plaćanje fakture po datumu fakturisanja ako je rok 31 dan.

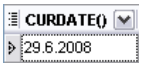
SELECT *fakture.sifra_fakture*, *fakture.datum*,
ADDDATE(*fakture.datum*, INTERVAL *31 DAY*) AS *istice*
FROM *fakture*;



sifra_fakture	datum	istice
1	25.10.2006	25.11.2006
2	28.10.2006	28.11.2006
3	25.10.2006	25.11.2006
4	29.10.2006	29.11.2006
5	25.10.2006	25.11.2006
6	6.11.2006	7.12.2006
7	4.11.2006	5.12.2006
8	23.10.2006	23.11.2006
9	2.11.2006	3.12.2006
10	8.10.2006	8.11.2006
11	15.10.2006	15.11.2006

CURDATE() - Vraća trenutni datum

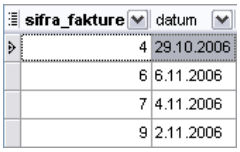
SELECT CURDATE();



CURDATE()
29.6.2008

Primer 3
Napraviti upit koji izdvaja sve fakture koje su u odnosu na trenutni datum nisu starije od tri meseca.

SELECT *fakture.sifra_fakture*, *fakture.datum*
FROM *fakture*
WHERE SUBDATE(CURDATE(),INTERVAL *3 MONTH*) <= *fakture.datum*;



sifra_fakture	datum
4	29.10.2006
6	6.11.2006
7	4.11.2006
9	2.11.2006

Funkcije:
DAYNAME()
DAYOFMONTH()
DAYOFWEEK()
DAYOFYEAR()
Vraćaju ime ili redni broj dana u nedelji, mesecu ili godini posmatranog datuma. Funkcija NOW() vraća trenutni datum sa vremenom.

Ove funkcije se mogu probati izvršavanjem sledećih upita:
SELECT DAYNAME(CURDATE());
SELECT DAYOFMONTH('2007-01-01');
SELECT DAYOFWEEK(CURDATE());
SELECT DAYOFYEAR(CURDATE());
SELECT NOW();

8.6. Funkcije za konverziju podataka

Postoje dve funkcije za konverziju tipova podataka:
CAST(*izraz* AS *tip*) i CONVERT(*izraz*, *tip*) koje daju isti rezultat. Ove funkcije omogućavaju konverziju tipova podataka, na primer signed integer u char. Parametar *tip* može imati vrednost BINARY, CHAR, DATE, DATETIME, DECIMAL, SIGNED(INTEGER), TIME i UNSIGNED(INTEGER). Većina konverzija tipova podataka odvija se u MySQL-u automatski kad zatreba. Ako se, na primer, funkciji za rad sa vrednostima znakovnog tipa prosledi broj, on će automatski biti pretvoren u niz znakova.

8.7. Agregatne funkcije

Upiti se obično koriste za pronalaženje svih zapisa koji zadovoljavaju neke kriterijume. Osim izbora pojednih vrednosti podataka koji su upisani u bazu podataka, ponekad je potrebno dobiti integralne podatke o celom skupu vrednosti: ukupan broj, zbir, srednju vrednost, pronalaženje najmanje i najveće vrednosti i slično. Za dobijanje takvih podataka iz skupova pojedinačnih vrednosti se koriste agregatne fukncije. Agregatne fukcije se izvršavaju nad jednom kolonom i vraćaju jednu vrednost za određenu grupu podataka ili kolonu. U MySQL-u su podržane sledeće agregatne funkcije:

Funkcija	**Izračunava**
SUM	Zbir vrednosti u koloni
AVG	Prosek vrednosti u koloni
COUNT	Broj vrednosti u koloni, ne računajući NULL vrednosti
MIN	Najmanju vrednost u koloni
MAX	Najveću vrednost u koloni
STD	Standardnu devijaciju vrednosti u koloni

COUNT(*expr*)
Vraća ukupan broj ne-NULL vrednosti dobijenih odgovarajućim Select iskazom. Za razliku od njega, COUNT(*) prebrojava sve redove, bez obzira da li neki sadrži i NULL vrednosti.
Primer 1: Koliko ima ukupno stavki na svim fakturama ?

SELECT COUNT(*detalji_fakture.red_br*)
FROM *detalji_fakture*;

Primer 2: Kada je knjižena zadnja faktura?

SELECT MAX(*fakture.datum*)
FROM *fakture*;

MAX(fakture.datum)
6.11.2006

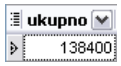
Primer 3: Ukupna količina novca na svim fakturama.

```
SELECT SUM(kolicina * dan_cena) AS ukupno
FROM detalji_fakture;
```

ukupno
246735

Dok je ukupna količina novca na svim *ulaznim* fakturama:

```
SELECT Sum(kolicina * dan_cena) AS ukupno
FROM  fakture, detalji_fakture
WHERE fakture.sifra_fakture = detalji_fakture.faktura
AND fakture.ulaz_izlaz = '1';
```



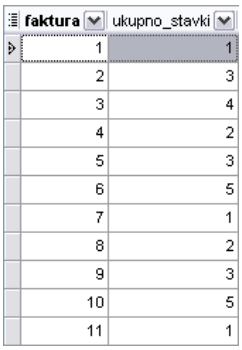
ukupno
138400

GROUP BY klauzula

Ovo je još jedan od neobaveznih delova SELECT naredbe. Najčešće se koristi u upitima na kojima se bazira neki izveštaj, a u cilju grupisanja rezultata upita u manje grupe, pri čemu se za grupu prikazuje određeni zbirni rezultat. Ovaj zbirni rezultat se odnosi na neka izračunavanja. Kako se agregatne funkcije primenjuju na odeđene skupove podataka, može se iskoristiti GROUP BY naredbu radi grupisanja.

Primer 4: Ukupan broj stavki na svakoj fakturi.

```
SELECT detalji_fakture.faktura, COUNT(detalji_fakture.red_br) AS ukupno_stavki
FROM detalji_fakture
GROUP BY detalji_fakture.faktura;
```



faktura	ukupno_stavki
1	1
2	3
3	4
4	2
5	3
6	5
7	1
8	2
9	3
10	5
11	1

Primer 5: Ukupna suma novca na svakoj od faktura.

```
SELECT firme.naziv_firme, detalji_fakture.faktura, SUM(kolicina * dan_cena) AS iznos
FROM detalji_fakture, firme, fakture
WHERE firme.firma = fakture.sifra_firme
AND fakture.sifra_fakture=detalji_fakture.faktura
GROUP BY detalji_fakture.faktura;
```

naziv_firme	faktura	iznos
STIL	1	10000
HELIO	2	12150
BALKAN	3	32500
KOKOMAX	4	13500
KOKOMAX	5	19400
BALKAN	6	45670
HELIO	7	8060
STIL	8	15425
BALKAN	9	8605
HELIO	10	81050
KOKOMAX	11	375

Primer 7 : Ukupan broj faktura po firmama.

SELECT *firme*.*naziv_firme*, *fakture*.*sifra_firme*,
COUNT(*fakture*.*sifra_fakture*) AS *ukupno_faktura*
FROM *firme*, *fakture*
WHERE *firme*.*firma* = *fakture*.*sifra_firme*
GROUP BY *fakture*.*sifra_firme*;

naziv_firme	sifra_firme	ukupno_faktura
BALKAN	1	3
STIL	2	2
KOKOMAX	3	3
HELIO	4	3

Primer 8: Izdvojiti sve fakture na kojima je ukupno (po svim stavkama) faktuturisana suma veća od 2000 dinara.

SELECT *detalji_fakture*.*faktura*,
SUM(*detalji_fakture*.*kolicina* * *detalji_fakture*.*dan_cena*) AS *suma*
FROM *detalji_fakture*
GROUP BY *detalji_fakture*.*faktura*
HAVING SUM(*detalji_fakture*.*kolicina* * *detalji_fakture*.*dan_cena*) > 2000;

faktura	suma
1	10000
2	12150
3	32500
4	13500
5	19400
6	45670
7	8060
8	15425
9	8605
10	81050

8.8. Ostale funkcije

U MySQL-u postoji niz drugih funkcija koje se mogu koristiti, na primer, za šifrovanje.
AES_ENCRYPT(*str*, *key_str*)
AES_DECRYPT(*crypt_str*, *key_str*)

SELECT AES_ENCRYPT('tajna', 'password') AS *sifra*;

sifra

SELECT AES_DECRYPT(AES_ENCRYPT('tajna', 'password'), 'password') AS *dekodirana*;

MD5(*s*) - Vraća 128 bitni heš ulaznog niza znakova *s* u MD5 formatu. Ovaj oblik se preporučuje za skladištenje lozinki u bazu podataka.
LAST_INSERT_ID() vraća poslednju vrednost AUTO_INCREMENT-a koja je bila automatski generisana. Korisna je kada se u tabelu dodaje novi red i potreban je njegov identifikator da bi bio upisan kao vrednost spoljašnjeg ključa u drugi tabelu.

9. Pogledi (VIEW)

Pogled je objekt koji se ponaša kao tabela, međutim on sam ne sadrži podatke. To je sacuvani upit koji se svaki put mora da pokrene da bi prikazao određene podatke.Pogledi se koriste kada želimo da se:

1. Izdvoji skup redova
2. Izdvoje određene kolone tabele
3. Spoje povezani redovi iz više tabela

Pogledi, uključujući i poglede koji dozvoljavaju ažuriranje, su implementirani u MySQL Server 5.0.

Sintaksa za kreiranje novog ili zamenu (REPLACE) postojećeg pogleda je:

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

Sam *select_statement* je SELECT iskaz koji omogućava definisanje pogleda. Njime se selektuju podaci iz baznih tabela ili iz drugih pogleda. Da bi se izvršilo kreiranje pogleda potrebno je da korisnik ima priviligiju za kreiranje, dok za korišćenje pogleda korisnik treba da ima privilegiju selektovanja podataka iz kolona koje se pojavljuju u SELECT iskazu. Sami pogledi su objekti baze koji se u njoj i čuvaju. Pogledi i bazne tabele treba da imaju različite nazive. Sami pogledi, kao i tabele moraju da imaju jedinstvene nazive kolona. Podrazumevano, nazivi kolona koji se koriste u SELECT iskazu postaju nazivi kolona pogleda, sem ako im se ne dodele drugi nazivi pomoću aliasa. Ukoliko se želi da se eksplicitno definišu nazivi za kolone u pogledu, oni se mogu navesti u *column_list* klauzuli, pri čemu se razdvajaju zarezima. Broj naziva kolona u ovoj listi mora da se poklapa sa brojem kolona koje se izdvajaju SELECT iskazom. Osim kolona baznih tabela, u okviru SELECT iskaza mogu se pojaviti izrazi koji koriste funkcije, operatore, itd. Pogledi imaju sledeća ograničenja:

- SELECT iskaz ne može sadržati podupit u FROM klauzuli
- SELECT iskaz ne može da referencira promenljivu ili parametre procedura
- Nazivi tabela koji se navode u definiciji pogleda moraju unapred da postoje
- Pogledu se ne može dodeliti triger

DEFINER i SQL SECURITY klauzule određuju koji će se koncept sigurnosti sprovoditi pri proveri privilegija za dati pogled. Podrazumevana vrednost za DEFINER je korisnik koji je kreirao pogled. SQL SECURITY određuje koji će se MySQL nalog koristiti kada se proveravaju privilegije koje korisnik ima pri korišćenju pogleda. Opciona klauzula ALGORITHM može imati jednu od tri vrednosti: MERGE, TEMPTABLE, ili UNDEFINED. Ovaj algoritam se odnosi na to kako MySQL procesuira poglede. Ukoliko se koriste agregatene funkcije, izbor će biti UNDEFINED. Neki pogledi mogu ažurirati podatke u baznim tabelama, odnosno mogu se koristiti u odredbama kao što su UPDATE, DELETE ili INSERT, ali samo ako postoji 1-prema-1 veza sa kolonama u baznoj tabeli. Pogledi koji u okviru SELECT iskaza sadrže:

- Agregatne funkcije (SUM(), MIN(), MAX(), COUNT(), itd.)
- DISTINCT
- GROUP BY
- HAVING
- UNION ili UNION ALL
- Podupite
- Spojeve (JOIN)
- Podupit u WHERE klauzuli koji se poziva na tabelu iz FROM klauzule
- ALGORITHM = TEMPTABLE (korišćenje privremenih tabela)

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

Sintaksa za brisanje pogleda glasi:

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

Sintaksa za izmenu pogleda glasi:

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

Primer 1: Kreirati pogled koji prikazuje šifru fakture, odgovarajuću firmu, datum i tip fakture:

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

SELECT * FROM *v_fakture*;

faktura	firma	datum	tip
3	BALKAN	25.10.2006	izlazna
6	BALKAN	6.11.2006	izlazna
9	BALKAN	2.11.2006	izlazna
1	STIL	25.10.2006	ulazna
8	STIL	23.10.2006	ulazna
4	KOKOMAX	29.10.2006	izlazna
5	KOKOMAX	25.10.2006	ulazna
11	KOKOMAX	15.10.2006	ulazna
2	HELIO	28.10.2006	ulazna
7	HELIO	4.11.2006	izlazna
10	HELIO	8.10.2006	ulazna

Primer 2: Kreirati pogled koji prikazuje šifru fakture, naziv firme, datum, tip fakture (ulazna ili izlazna), kao i sumu stavki na fakturi.

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

SELECT * FROM suma_na_fakturi;
```

faktura	naziv_firme	datum	tip	iznos
1	STIL	25.10.2006	ulazna	10000
2	HELIO	28.10.2006	ulazna	12150
3	BALKAN	25.10.2006	izlazna	32500
4	KOKOMAX	29.10.2006	izlazna	13500
5	KOKOMAX	25.10.2006	ulazna	19400
6	BALKAN	6.11.2006	izlazna	45670
7	HELIO	4.11.2006	izlazna	8060
8	STIL	23.10.2006	ulazna	15425
9	BALKAN	2.11.2006	izlazna	8605
10	HELIO	8.10.2006	ulazna	81050
11	KOKOMAX	15.10.2006	ulazna	375

Primer 3: Naći firmu za koju je fakturisana faktura sa najvećim iznosom. U ovom primeru će biti iskorišćen već postojeći pogled *suma_na_fakturi*:

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

SELECT * FROM najveca_faktura;
```

naziv_firme	faktura	iznos
HELIO	10	81050

Pogledi mogu unutar SELECT iskaza pozivati kako funkije tako i uskladištene procedure o čemu će kasnije biti više reči.

Primer 4: Prikazati fakture koje umesto matičnog broja proizvoda prikazuju njegovo ime, zajedno sa opisom tipa, datumom, današnjom cenom, količinom i iznosom.

```
CREATE VIEW fakture_proizvodi AS
SELECT detalji_fakture.faktura AS faktura, proizvodi.ime AS ime,
detalji_fakture.kolicina AS kolicina, detalji_fakture.dan_cena AS dan_cena,
detalji_fakture.kolicina * detalji_fakture.dan_cena AS iznos
FROM proizvodi JOIN detalji_fakture
ON proizvodi.maticni_broj = detalji_fakture.proizvod
ORDER BY detalji_fakture.faktura;

SELECT * FROM fakture_proizvodi;
```

faktura	ime	kolicina	dan_cena	iznos
1	Stiropor	50	200	10000
2	Stiropor	40	220	8800
2	Destilovana voda	50	25	1250
2	Smirgla	35	60	2100
3	Lak	35	300	10500
3	Cetka	20	75	1500
3	Stiropor	60	200	12000
3	Gips	50	170	8500
4	Cetka	50	80	4000
4	Gips	50	190	9500
5	Gips	100	150	15000
5	Smirgla	25	80	2000
5	Destilovana voda	120	20	2400
6	Stiropor	70	230	16100
6	Destilovana voda	80	30	2400
6	Lak	24	330	7920
6	Gips	75	200	15000
6	Smirgla	50	85	4250
7	Lak	26	310	8060
8	Stiropor	50	250	12500
8	Smirgla	45	65	2925
9	Cetka	50	70	3500
9	Lak	14	320	4480
9	Destilovana voda	25	25	625
10	Smirgla	60	55	3300

10. Aktivne baze podataka i transakcije

Aktivni sistemi baza podatka uvode sistem pravila koji se koristi kao univerzalni mehanizam za kontrolu integriteta baze podataka, kontrolu sigurnosti u smislu prava pristupa, generisanje izvedenih podataka, prikupljanje statističkih podataka praćenje funkcionisanja baze i vođenje denevnika kao i mnoge druge karakteristike i funkcije baza podataka. Ova oblast se intenzivno razvijala krajem osamdesetih i početkom devedesetih godina. Danas se koncepti aktivnih baza sve više uključuju u komercijalne sisteme i postali su standardizovani. Specijalna vrsta pravila, trigeri, uključeni su i u SQL : 1999 standard . Za razliku od konvencionalnih odnosno pasivnih sistema za upravljanje bazama podataka, aktivni sistemi imaju mogućnost da automatski izvršavaju određene operacije kao odgovor na određeni događaj. U pasivnim sistemima se podaci kreiraju, menjaju, brišu i prikazuju kao odgovor na operacije koje direktno pokreću korisnici ili se pozivaju u toku izvršavanja aplikativnih programa.
Aktivni sistemi baziraju se na ideji da određene događaje i uslove treba opisati (tzv. situacije) a potom ih prepoznati u bazi podataka, aplikacijama i okruženju, da bi se potom automatski pokrenula određena akcija (tzv reakcija) koja predstavlja niz operacija nad bazom podataka. Ova pravila i akcije imaju zajedničko ime Event-Condition-Action ili ECA. Event je događaj kome odgovara ključna reč ON. Condition je uslov koji je opisan odgovarajućim iskazom a odgovara mu ključna reč IF dok akciji odgovara ključna reč DO.

Pored softvera za upravljanje bazom podataka, aktivni sistem ima i dve specifične komponente: detektor događaja i mehanizam za izvršavanje pravila. Događaj može da se prepozna u bazi podataka, u aplikaciji ili može biti određen vremenskim trenutkom. Nakon prepoznavanja događaja aktivira se mehanizam za izvršavanje pravila. U bazi pravila pronalaze se pravila koja se odnose na identifikovani događaj i proverava se uslov. Ukoliko je uslov zadovoljen izvršava se specificirana akcija. Pod akcijom podrazumevamo bilo koji program koji može da uključuje operacije nad bazom podataka i da generiše nove događaje.Svako pravilo se odnosi na određeni događaj. Nakon prepoznavanja događaja vrši se provera određenih uslova i tek onda se pristupa izvršavanju definisane akcije ako je uslov zadovoljen.Samo se izuzetno i retko definišu pravila koja se izvršavaju bezuslovno.

Da bi se definisala akcija prvo se moraju tačno odrediti događaji. Razlikujemo dve vrste događaja : primitivni i složeni događaji.

Pod primitivnim događajima podrazumevaju se :

- Ažuriranje podataka
- Prikaz podataka
- Vreme, u smislu vremenskog događaja koji može biti apsolutan u smislu dostizanja nekog unapred definisanog vremenskog trenutka, poput 28.Okt 2005 u 20:00 ili periodičan (svakog dana u 08:00)
- Aplikativno definisan događaj, koji određuje sama aplikacija

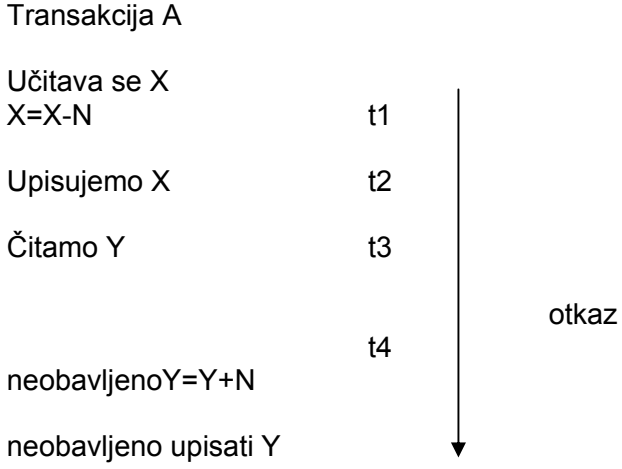
Akcije se uglavnom pokreću nakon detektovanja određenih događaja (AFTER) ali je moguće da se akcije pokreću i neposredno pre detektovanja događaja (BEFORE). Za razliku od primitivnih, složeni događaji se formiraju kombinovanjem primitivnih I predhodno definisanih složenih događaja Oni se mogu kombinovati logičkm operatorima (AND; NOT; OR), pomoću sekvenci, odnosno kada se dva ili više događaja pojave u određenom definisanom redosledu ili vremenskim kompozicijama odnosno kombinovanjem vremenskih i događaja koji nisu vremenski (na primer pokrenuti pravilo 3 sekunde nakon događaja D).
Da bi se pravilo formiralo, potrbno je definisati i uslov koji se proverava nakon pojave događaja. Uslov može biti upit nad bazom ili izvršavanje neke aplikativne procedure. Iza ključne reči IF nalazi logički izraz koja ima vrednost tačno (i tada se akcija izvršava) ili netačno (i tada se akcija ne izvršava).

Sa tačke gledišta korisnika stanje baze podataka se menja menja samo po okončanju logičke jedinice posla. Logičku jedinicu posla definišemo kao najmanji deo složenog posla koji ima neko realno značenje. Ona može da bude predstavljena sa više operacija nad bazom. Svako operacija nad bazom može da menja njeno stanje, ali je od značaja samo ukupna promena koju izaziva logička jedinica posla. Pod transakcijom podrazumevamo niz operacija nad bazom podataka koja odgovara jednoj logičkoj jedinici posla u realnom sistemu.

U jednom trenutku vremena nad bazom se izvršava više transakcija. Takođe se i izvršenje jednog istog programa, poput podizanja ili ulaganja novca u banci sa više šaltera, može odvijati konkurentno.

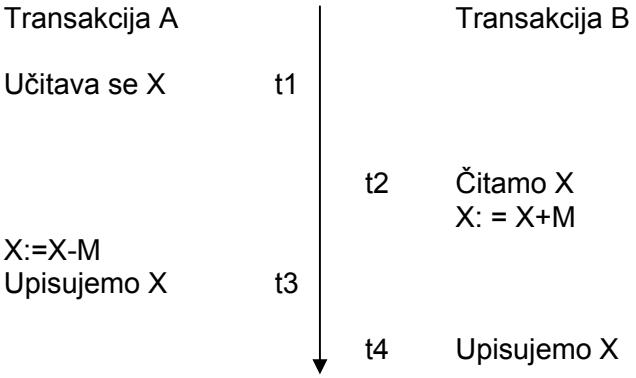
Do kakvih problema može da dovede otkaz sistema ili nekontrolisano konkurentno izvršenje transakcija?

Otkaz sistema u toku obrade transakcije:



Na gornjoj slici prikazano je odvijanje transakcije za prenos novca (N dinara) komitenta banke sa računa X na račun Y. Ako u trenutku t4 dođe do otkaza sistema transakcija se neće u potpunosti obaviti i doći će do narušavanja integriteta baze podataka – novac će se skinuti sa računa X ali se neće preneti na račun Y.

Gubljenje rezultata ažuriranja: Recima da se izvršavaju dve transakcije u vremenu. Transakcija A podiže a transakcija B ulaže novac na isti račun. Ažuriranje koje transakcija obavlja u trenutku t3 je izgubljeno, odnosno prebrisano ažuriranjem koje je izvršila transakcija B u trenutku t4.



Nekorektna analiza podataka:

Ovim nazivom se definiše skup problema do kojih dolazi kada za vreme nekog sračunavanja koje se obavlja u jednoj transakciji druga promeni vrednost argumenta koji je prva već obradila. Recimo da jedna transakcija računa sumu koristeći kao sabirke neke podatke iz baze, dok druga ažurira vrednosti tog istog skupa podataka. Moguće je da se u sumu uzme neka vrednost koja je u međuvremenu promenjena, čime bi se dobila nekorektna vrednost sume.

Postoje i drugi problemi do kojih dolazi pri otkazu sistema i konkurentnoj obradi transakcija.

Da bi se izbegle ivakve nekorektnosti, od transakcije se zahtevaju četiri osobine:

• Atomičnost, pri čemu se zahteva da se prilikom transakcije ili uspešno obave sve poeracije nad bazom ili nijedna. Kada se recimo vrši ažuriranje skupa i kada se ta atomska operacija prekine, neophodno je da budu poništene sve promene koje je ona do tada izazvala.

• Konzistentnost-pre i posle transakcije stanje baze podataka mora da bude konzistentno

• Izolacija, pri čemu se zahteva da istovremeno obavljanje dve transakcije dovodi do istog rezultata kao i kada bi se one izvršile jedna za drugom Po pravilu, transakcija ne čini promene vidljivim drugim transakcijama pre nego što se ona ne završi i potvrde se promene u bazi podataka. Da bi se povećala konkurentna obrada, dozvoljavaju se različiti nivoi izolovanosti transakcije.

• Trajnost, koja pdrazumeva da efekti transakcije ne mogu biti izgubljeni čak ni ako se desi otkaz neposredno po izavršetku transakcije.

Ove osobine se jednim imenom zovu ACID osobine transakcije.

Transakcija se ne može prekinuti, suspendovati ili odložiti a da se pri tome ne pojavi opasnost po integritet podataka. Da bi se gornje osobine obezbedile,te da bi se osigurao

integritet, transakcija se mora završiti bez grešaka ili se ne može uopšte izvršiti (sve ili ništa).Skup naredbi koje predstavljajku jednu transakciju počinje instrukcijom BEGINTRANSACTION a završava se ili instrukcijom COMMIT (ako su sve instrukcije u transakciji uspešno obavljene, ovom instrukcijom se potvrđuju sve promene nastale u bazi podataka) ili instrukcijom ROLLBACK , kojom se poništavaju promene u bazi ukoliko nisu sve instrukcije uspešno izvršene).Rollback se može inicirati od strane SUBP i kada iz bilo kog razloga dođe do neplaniranog završetka transakcije.

```
BEGIN TRANSACTION
....
COMMIT TRANSACTION
```

11. Trigeri (okidači)

Trigeri su imenovani objekti baze koji su povezani sa nekom tabelom i aktiviraju se kada se desi određeni događaj (Insert, Update i Delete) na toj tabeli. Trigeri se automatski pozivaju od strane MySQL-a (ne koristi se Execute naredba). Sami trigeri nemaju ulazne ni izlazne parametre. Triger se poziva prilikom svakog izvršenja određene naredbe odnosno svaki put kada se desi određeni događaj.

Sintaksa za kreiranje trigera je

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_stmt
```

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

Triger_time se odnosi na vreme okidanja, odnosno određuje da li će se triger aktivirati pre ili posle uslova koja ga aktivira.

Trigger_event je jedan od sledećih događaja

- INSERT: triger se aktivira svaki put kada se novi zapis insertuje u tabelu
- UPDATE: triger se aktivira svaki put kada se zapis ažurira
- DELETE : triger se aktivira svaki put kada se briše zapis iz tabele. Međutim,naredba DROP TABLE ne aktivira triger.

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

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

Modifikovanje i brisanje trigera

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

Za brisanje trigera koristi se:

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

Pogledajmo primer sledećeg trigera napisanog iz komandne linije, koji u sebi ima blok BEGIN...END u okviru koga su naredbe koje se završavaju sa; Stoga se na početku definiše

novi delimiter kao //. Sam triger treba da pri ažuriranju podataka proveri novu vrednost i modifikuje ih tako da budu u rasponu od 0 do 100.Ovakav triger mora biti BEFORE triger jer vrednosti moraju da se provere pre nego što se upišu u tabelu:

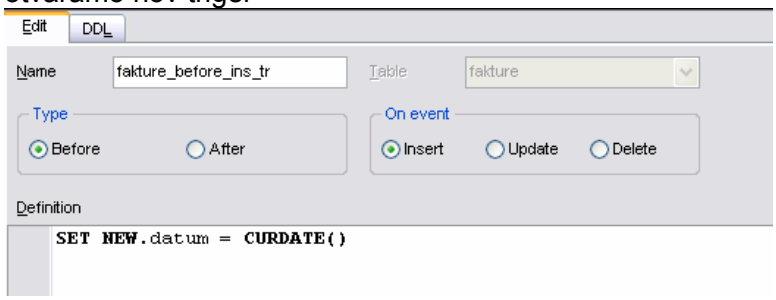
```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
-> FOR EACH ROW
-> BEGIN
-> IF NEW.amount < 0 THEN
-> SET NEW.amount = 0;
-> ELSEIF NEW.amount > 100 THEN
-> SET NEW.amount = 100;
-> END IF;
-> END;//
mysql> delimiter ;
```

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

Da bi napravili određene korisne trigere u našoj bazi poslovanje , najpre napravite kopiju postojeće baze poslovanje, jer ćemo menjati neke tabele i kolone a takođe ćemo menjati I postojeće podatke. Novu bazu zovimo poslovanje2.
Zašto ćemo napraviti izmene? Zato što nam postojanje stranih ključeva, ukoliko nije postavljeno na opciju CASCADE ne dozvoljava da napravimo trigere koji se odnose na kaskadno ažuriranje i/ili brisanje podataka.

Napravite sledeće izmene u bazi poslovanje 2:
1. U tabeli firme dozvolite da kolona telefon ima Null vrednost, a potom izbrišite sve brojeve telefona u ovoj tabeli (nije obavezno brisanje).
2. U tabeli fakture uklonite strani ključ nad kolonom sifra_firme.
3. U tabeli fakture dozvolite da kolona datum bude Null.

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



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

Na kartici DDL možete videti kako izgleda definicija celog trigera

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

Iskompajlirajte triger, nakon čega se on pojavljuje kao objekat baze.
Da bi videli kako radi, ubacićemo nov zapis u tabelu firme i u tabelu fakture.
Najpre u tabelu firme unesite firmu sa sifrom 20, ime PROBNA, adresa nepoznata a mesto

iz Niša.

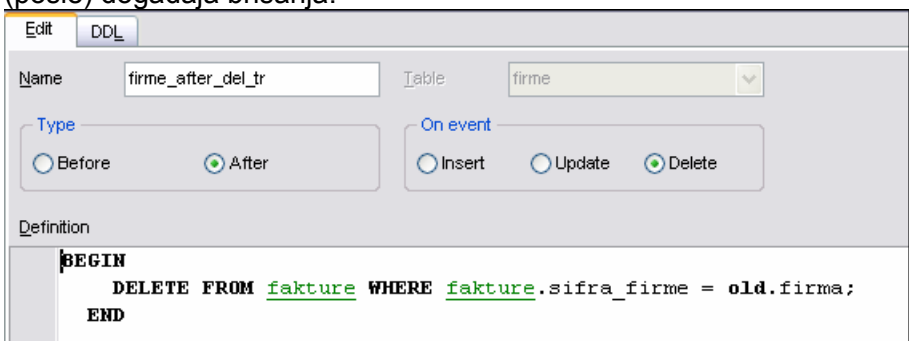
INSERT INTO *firme (firma, naziv_firme, mesto, adresa)*
VALUES (*20, 'PROBNA', 'Nis', 'nepoznata'*);

U tabelu fakture unesite podatak o jednoj fakturi koja se odnosi na unetu firmu, na primer

INSERT INTO *fakture (sifra_fakture,sifra_firme,datum,ulaz_izlaz)*
VALUES (*30, 20, ' 2006-10-23', '1'*);

Upišimo podatak a potom pritisnite taster post edit – štikla(ili refresh) i videćete kako se datum promenio u današnji datum.Sličnu akciju smo mogli proizvesti postavljanjem difoltne vrednosti nad kolonom datum, međutim ima razlike.Ovde je upisan današnji datum bez obzira što smo mi uneli *2006-10-23*. Ovo ima smisla ukoliko postoji pravilo kojim se reguliše da se datum unosa fakture mora poklapati sa datumom na fakturi.

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



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

Sada obrišimo firmu sa šifrom 20 (to je firma PROBNA, koji smo maločas uneli i za koju smo uneli i jednu fakturu).

DELETE FROM *firme* WHERE *firma=20*;

U tabeli firme više ne postoji PROBNA

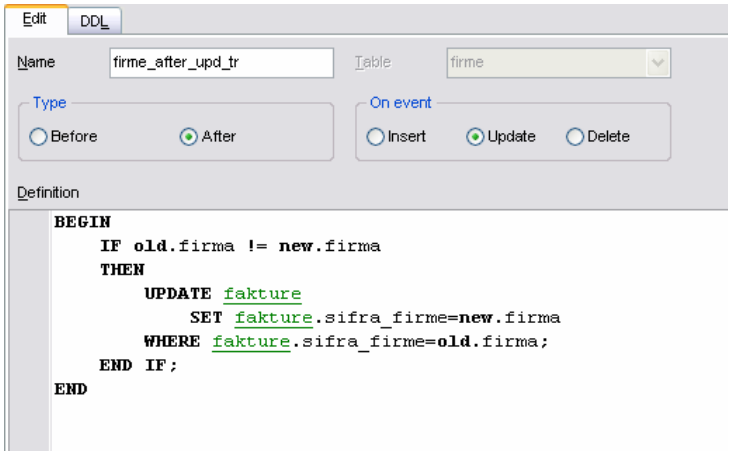
firma	naziv_firme	mesto	adresa
2	STIL	Beograd	Takovska 10
3	KOKOMAX	Nis	Dusanova 33
4	HELIO	Subotica	Nikole Tesle 55
5	BALKAN	Nis	Mokranjceva 13

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

sifra_fakture	sifra_firme	datum	ulaz_izlaz
1	2	26/10/2006	1
2	4	28/10/2006	1
3	5	25/10/2006	2
4	3	29/10/2006	2
5	3	25/10/2006	1
6	5	06/11/2006	2
7	4	04/11/2006	2
8	2	23/10/2006	1
9	5	02/11/2006	2
10	4	08/10/2006	1
11	3	15/10/2006	1
20	1	30/01/2007	1

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

```
delimiter //
CREATE TRIGGER `firme_after_upd_tr` AFTER UPDATE ON `firme`
FOR EACH ROW
BEGIN
IF old.firma != new.firma
THEN
UPDATE fakture
SET fakture.sifra_firme=new.firma
WHERE fakture.sifra_firme=old.firma;
END IF;
END;//
```



Probajte izmenu određene šifre firme i pogledajte izmene u tabeli fakture.

12. Uskladištene procedure (Stored Procedures)

Uskladištena procedura je skup SQL iskaza koji su kompajlirani i sačuvani u trenutku njenog kreiranja. Veoma su moćne i preko njih mogu da se izvršavaju sve operacije iz DDL-a i DML-a kao, na primer, kreiranje tabele, izvršavanje UPDATE iskaza nad više tabela, umetanje, brisanje podataka ali i postavljanje vrednosti (SET) kao i prihvatanje transakcije (COMMIT) ili vraćanje baze u predhodno stanje (ROLLBACK) ...
U okviru tela procedure ne mogu se koristiti sledeće naredbe: CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, CREATE FUNCTION, DROP FUNCTION, CREATE TRIGGER, DROP TRIGGER.
Generalno, uskladištene procedure rade kao i procedure u programskim jezicima. Uskladištena procedura je imenovani objekat baze podataka i čuva se na strani servera gde se i izvršava, a klijentu se prosleđuju samo rezultati. Prilikom davanja privilegija, dovoljno je dati privilegiju za pokretanje procedure; nije potrebno davati posebna ovlašćenja za pojedinačne tabele koje se koriste u okviru nje. Sama procedura može da vrati paramtere, *result set*, kod i da kreira kursore. Takođe može da sadrži ulazne parametre, lokalne promenljive (varijable), numeričke operacije i operacije nad karakterima, operacije dodeljivanja, SQL operacije i logiku za kontrolu toka izvršavanja.

SQL procedura se kreira CREATE PROCEDURE iskazom, a sa komandne linije se poziva sa CALL *naziv_procedure* iskazom. Uvek se kreira u tekućoj bazi. Kada se definiše procedura, naziv procedure sledi iza službenih reči CREATE PROCEDURE, a potom slede parametri procedure.

```
CREATE PROCEDURE procedure1              /* name */
(IN parameter1 INTEGER)                  /* parameters */
BEGIN                                    /* start of block */
DECLARE variable1 CHAR(10);              /* variables */
IF parameter1 = 17 THEN                  /* start of IF */
SET variable1 = 'birds';                 /* assignment */
ELSE
SET variable1 = 'beasts';                /* assignment */
END IF;                                  /* end of IF */
INSERT INTO table1 VALUES (variable1);   /* statement */
END                                      /* end of block */
```

U narednim primerima biće korišćena krajnje jednostavna baza *db5*:

```
CREATE DATABASE db5;
CREATE TABLE t (s1 INT);
INSERT INTO t VALUES (5);
```

Biće kreirana najjednostavnija moguća procedura:

```
CREATE PROCEDURE p1 ()
SELECT * FROM t;
```

(Ukoliko se koristi MySQL Comand Line Client neophodno je na kraju iskaza koristiti delimiter //, i naravno, prethodno ga definisati.)

Procedura se poziva na sledeći način:

```
CALL p1();
```

Procedura se menja kao i svaki drugi objekat baze pomoću komande ALTER PROCEDURE. Procedura se briše pomoću opcije DROP PROCEDURE.

Pogledajmo sledeći primer gde u okviru baze db5 kreiramo proceduru p2 :

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t;

CALL p2();
```

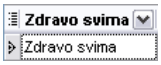
Informacija o jeziku (SQL) je tu zbog kompatibilnosti sa drugim sistemima, na primer DB2 sistemom. Odredba (NOT) DETERMINISTIC znači da procedura za jedne te iste ulazne podatke uvek vraća isti skup vrednosti. SQL SECURITY DEFINER daje instrukciju serveru da u trenutku pozivanja procedure proveri privilegije koje ima kreator procedure (INVOKER ispituje privilegije koje ima korisnik koji poziva proceduru). Umesto gornje definicije moglo je samo da stoji:

```
CREATE PROCEDURE p2 ()
SELECT CURRENT_DATE, RAND() FROM t;
```

Rezultat bi bio isti.

Primer 1: Napisati proceduru koja vraća „Zdravo svima“.

```
CREATE PROCEDURE p3 ()
SELECT 'Zdravo svima';

CALL p3();
```



12.1. Parametri uskladištene procedure

Svaka procedura može imati svoje parametre. Parametri mogu biti ulazni (IN) izlazni (OUT) ili ulazno/izlazni (INOUT). Ključna reč IN je opciona i ako se ne navede parametri su podrazumevano ulazni. Parametri procedure se navode nakon naziva procedure na sledeći način: VRSTA *naziv_parametra* TIP_PODATAKA. Ukoliko ima više parametara oni su razdvojeni zarezom.

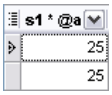
```
CREATE PROCEDURE p5 (p INT)
SET @x = p;

CALL p5(12345);
SELECT @x;
```

U gornjem primeru, *p* je ulazni parametar koji je u proceduri dodeljen promenljivoj na nivou sesije *x*. Ona potom biva selektovana (nakon poziva procedure i prosleđivanja vrednosti parametru) i vidi se da je vraćena vrednost ista kao i vrednost ulaznog parametra.

12.2. Blokovi naredbi

Gde god je dozvoljeno da se jedan SQL iskaz upotrebi, može se koristiti i blok iskaza ograničen sa BEGIN i END ključnim rečima. U okviru njega se skup od više iskaza tretira kao jedan iskaz. Svaki iskaz u okviru bloka mora biti završen tačkom i zarezom (;). U narednom primeru u okviru tela procedure postoji nekoliko odvojenih iskaza koje se izvršavaju kao jedan.

```
CREATE PROCEDURE p7 ()
BEGIN
SET @a = 5;
SET @b = 5;
INSERT INTO t VALUES (@a);
SELECT s1 * @a FROM t WHERE s1 >= @b;
END;

CALL p7();
```



Može se primetiti da su korišćene **globalne promenljive**, tj. promenljive sesije koje se označavaju sa @*a*, @*b*, ... Ove promenljive se dalje mogu koristiti gde se želi unutar sesije i vrednost će im biti jednaka poslednjoj vrednosti koju su primile.
Osim toga, procedura može da sadrži lokalne promenljive. Deklaracija lokalne promenljive se vrši pomoću iskaza DECLARE *naziv_promenljive* TIP_PODATAKA. Ove promenljive se koriste samo unutar procedure.

```
CREATE PROCEDURE p9 ()
BEGIN
DECLARE a INT;
DECLARE b INT;
SET a = 5;
SET b = 5;
INSERT INTO t VALUES (a);
SELECT s1 * a FROM t WHERE s1 >= b;
END;

CALL p9();
```

Za dodeljivanje vrednosti parametru ili promenljivoj koristi se SET naredba.
Kako izlaznom, tako i ulaznom parametru se može dodeliti vrednost, pri čemu dodeljivanje vrednosti izlaznom parametru nema efekta na taj podatak van procedure.

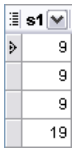
12.3. Korišćenje uslova IF-THEN-ELSE (Uslovno izvršenje naredbe)

U uskladištenoj proceduri mouće je koristiti IF ... ELSE naredbu slično kao sto se koristi u bilo kom programskom jeziku. Moguće je ugnjezditi više IF ... ELSE naredbi:

```
CREATE PROCEDURE p12 (IN parametar1 INT)
BEGIN
DECLARE promenljiva1 INT;
SET promenljiva1 = parametar1 + 1;
IF promenljiva1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF parametar1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END;

CALL p12(1);

SELECT * FROM t;
```

s1
7
7
7

```
CALL p12(-1);

SELECT * FROM t;
```



s1
9
9
9
19

Ovo je primer uskladištene procedure koja sadrži dve IF klauzule.
Slučaj CASE klauzule unutar procedure:

```
CREATE PROCEDURE p13 (IN parametar1 INT)
BEGIN
DECLARE promenljiva1 INT;
SET promenljiva1 = parametar1 + 1;
CASE promenljiva1
WHEN 0 THEN INSERT INTO t VALUES (17);
WHEN 1 THEN INSERT INTO t VALUES (18);
ELSE INSERT INTO t VALUES (19);
END CASE;
END;

CALL p13(-1);
```

```
SELECT * FROM t;
```

s1
9
9
9
19
17

12.4. Petlje

LOOP ... END LOOP, GOTO
REPEAT ... END REPEAT
WHILE ... END WHILE

WHILE petlje se koriste da bi se omogućilo izvršavanje jedne ili više naredbi više puta. Posle službene reči WHILE sledi uslov pod kojim se izvršava petlja, a zatim telo petlje. Pre svakog izvršenja tela petlje, ispituje se uslov i ukoliko je on tačan izvrašava se telo petlje, a ako nije izvršenje se prenosi na prvu naredbu posle petlje.

```
CREATE PROCEDURE p14 ()
BEGIN
DECLARE v INT;
SET v = 0;
WHILE v < 5 DO INSERT INTO t VALUES (v);
        SET v = v + 1;
END WHILE;
END;

CALL p14();

SELECT * FROM t;
```

s1
9
9
9
19
17
0
1
2
3
4

REPEAT ... END REPEAT - Ova petlja radi isto što i predhodna samo što se u okviru nje uslov ispituje nakon što se izvrši određena akcija.

```
CREATE PROCEDURE p15 ()
BEGIN
DECLARE v INT;
SET v = 0;
REPEAT
        INSERT INTO t VALUES (v);
        SET v = v + 1;
        UNTIL v >= 5
END REPEAT;
END;
```

LOOP ... END LOOP – Izlaz iz petlje se obezbeđuje preko LEAVE klauzule.

```
CREATE PROCEDURE p16 ()
BEGIN
DECLARE v INT;
SET v = 0;
loop_label: LOOP
        INSERT INTO t VALUES (v);
        SET v = v + 1;
                IF v >= 5 THEN
                LEAVE loop_label;
                END IF;
        END LOOP;
END;
```

Izlaz iz petlje se ispituje preko IF klauzule.
ITERATE – Ovaj iskaz se može pojaviti samo unutar LOOP, REPEAT i WHILE iskaza. ITERATE znači „proći ponovo kroz petlju“.

```
BEGIN
DECLARE v INT;
SET v = 0;
loop_label: LOOP
        IF v = 3 THEN
                SET v = v + 1;
                ITERATE loop_label;
        END IF;
        INSERT INTO t VALUES (v);
        SET v = v + 1;
        IF v >= 5 THEN
                LEAVE loop_label;
        END IF;
END LOOP;
END;
```

12.5. Beleženje grešaka

Biće kreirane još dve tabele - *t2* i *t3* - gde će biti beležene greške.

```
CREATE TABLE t2
(s1 INT, PRIMARY KEY (s1));

CREATE TABLE t3
(s1 INT, KEY (s1), FOREIGN KEY (s1) REFERENCES t2 (s1));

INSERT INTO t3 VALUES (5);
```

ERROR - Cannot add or update a child row: a foreign key constraint fails

Vidi se da zbog postojanja stranog ključa nije dozvoljeno narušavanje referencijalnog integriteta. Međutim, MyISAM tabele ne podržavaju strani ključ pa će biti napravljena procedura koja će voditi računa o integritetu a sa druge strane postoji tabela u koju će biti beležene greške.

```
CREATE TABLE error_log
(error_message CHAR(80));

CREATE PROCEDURE p22 (parametar1 INT)
BEGIN
DECLARE EXIT HANDLER FOR 1216
INSERT INTO error_log VALUES (CONCAT('Vreme: ', NOW(), '. Referncijalni integritet
narusen za vrednost ', parametar1));
INSERT INTO t3 VALUES (parametar1);
END;
```

12.6. Kursori

Za upite koji vraćaju više redova potrebno je eksplicitno definisati kursor za pojedinačnu obradu redova.

```
CREATE PROCEDURE p25 (OUT povratna_vrednost INT)
BEGIN
DECLARE a,b INT;
DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET b = 1;
OPEN cur_1;
REPEAT FETCH cur_1 INTO a;
UNTIL b = 1 END REPEAT;
CLOSE cur_1;
SET povratna_vrednost = a;
END;
```

13. Uskladištene funkcije

Uskladištene funkcije imaju istu sintaksu kao i uskladištene procedure, osim što uskladištene funkcije vraćaju neku vrednost. Sledi primer funkcije kojom se izračunava faktorijel broja *n*:

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

U gornjoj funkciji se dobija vrednost za *n* faktorijel. Ova funkcija se može naknadno iskoristiti na sledeći način:

INSERT INTO *t* VALUES (*factorial*(10));

SELECT *s1*, *factorial*(*s1*) FROM *t*;

UPDATE *t* SET *s1* = *factorial*(*s1*)
WHERE *factorial*(*s1*) < 5;

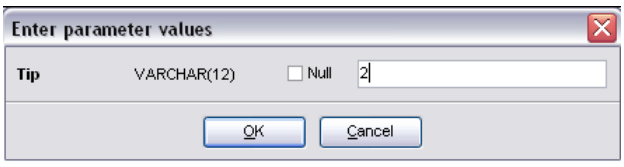
Za razliku od procedura, funkcije imaju mnogo više ograničenja u smislu šta se može naći u okviru tela definicije funkcije. Naime, ne sme se koristiti ništa od dalje navedenog:

ALTER 'CACHE INDEX' CALL COMMIT CREATE DELETE DROP 'FLUSH PRIVILEGES' GRANT INSERT KILL LOCK OPTIMIZE REPAIR REPLACE REVOKE ROLLBACK SAVEPOINT 'SELECT FROM *table*' 'SET *system variable*' 'SET TRANSACTION' SHOW 'START TRANSACTION' TRUNCATE UPDATE

Na dalje će u bazi podataka poslovanje biti definisane nekoliko procedura i funkcija. Za početak, biće napravljena procedura koja vraća sve ulazne ili sve izlazne fakture u zavisnosti od ulaznog parametra. Taj ulazni parameter je promenljiva *Tip*, koja može da primi vrednost 1 ili 2.

CREATE PROCEDURE *fak_proc* (IN *Tip* VARCHAR(12))
NOT DETERMINISTIC
SQL SECURITY DEFINER
SELECT *fakture.sifra_fakture*, *fakture.datum*, *fakture.ulaz_izlaz*
FROM *fakture*, *firme*
WHERE *fakture.sifra_firme* = *firme.firma*
AND *fakture.ulaz_izlaz* = *Tip*;

Po kreiranju procedure, pokrećemo njeno izvršenje. Ako ppozivamo proceduru il komandne linije MySQL ovog klijenta, moraćemo da pored imena procedure navedemo i vrednost ulaznog parametra Tip (u zagradi).
CALL *fak_proc*(2);
Ukoliko to činimo prekoEMS SQL Managera, biramo opciju execute.Pojaviće se prozor u koji treba uneti vrednost ulazne promenljive, a nakon toga možemo da vidimo i rezultujući skup.



sifra_fakture	datum	ulaz_izlaz
3	25.10.2006	2
6	6.11.2006	2
9	2.11.2006	2
4	29.10.2006	2
7	4.11.2006	2

Sledeća procedura radi isto što i predhodna, samo što se osim fakture želi da se izabere i firma za koju će biti prikazane sve ulazne, odnosno sve izlazne fakture. Zato mora da se doda i tabela *firme*, koju treba spojiti sa ostalim tabelama.

```
CREATE PROCEDURE vrsta_faktura
(IN Tip VARCHAR(12), IN Ime VARCHAR(15))
NOT DETERMINISTIC
SQL SECURITY DEFINER
SELECT firme.naziv_firme, fakture.sifra_fakture,
fakture.datum, fakture.ulaz_izlaz
FROM fakture, firme
WHERE fakture.sifra_firme = firme.firma
AND fakture.ulaz_izlaz = Tip
AND firme.naziv_firme = Ime;

CALL vrsta_faktura(2, 'BALKAN');
```

naziv_firme	sifra_fakture	datum	ulaz_izlaz
BALKAN	3	25.10.2006	2
BALKAN	6	6.11.2006	2
BALKAN	9	2.11.2006	2

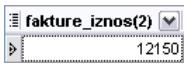
Sledeći korak biće procedura koja će sem malopređašnih podataka pružiti informaciju o ukupnoj količini novca koja je fakturisana na svakoj od faktura.

```
CREATE PROCEDURE vrsta_faktura_1
(IN Tip VARCHAR(12), IN Ime VARCHAR(15))
NOT DETERMINISTIC
SQL SECURITY DEFINER
SELECT firme.naziv_firme, fakture.sifra_fakture,
fakture.datum, fakture.ulaz_izlaz,
SUM(detalji_fakture.kolicina * detalji_fakture.dan_cena) AS iznos
FROM fakture, firme, detalji_fakture
WHERE fakture.sifra_firme = firme.firma
AND fakture.sifra_fakture = detalji_fakture.faktura
AND fakture.ulaz_izlaz = Tip
AND firme.naziv_firme = Ime
GROUP BY fakture.sifra_fakture;

CALL vrsta_faktura_1(2, 'BALKAN');
```

naziv_firme	sifra_fakture	datum	ulaz_izlaz	iznos
BALKAN	3	25.10.2006	2	32500
BALKAN	6	6.11.2006	2	45670
BALKAN	9	2.11.2006	2	8605

Kao što se može primetiti, već puno puta je, što u okvitu upita, što u okviru pogleda i drugih objekata baze, bila računata suma iznosa po stavkama fakture koja se dobija množenjem količine i cene određenog proizvoda. Zato je to odlično mesto gde treba napraviti funkciju. Na dalje, umesto da se svaki put ukucava agregatna funkcija, moći će da se pozove ovako kreirana funkcija koja će za određenu fakturu vratiti iznos koji je na njoj fakturisan.

```
CREATE FUNCTION fakture_iznos (faktura_sifra INTEGER(10))
RETURNS DECIMAL(20,2)
NOT DETERMINISTIC
SQL SECURITY DEFINER
BEGIN
DECLARE f_vrednost_fakture DECIMAL(20,2);
DECLARE EXIT HANDLER FOR NOT FOUND RETURN NULL;
SELECT SUM(kolicina * dan_cena) INTO f_vrednost_fakture
FROM detalji_fakture, fakture
WHERE fakture.sifra_fakture = detalji_fakture.faktura
AND fakture.sifra_fakture = faktura_sifra;
RETURN f_vrednost_fakture;
END;

SELECT fakture_iznos(2);
```



1. zadatak: Iskoristite funkciju fakture_iznos u okviru predhodne procedure.

2. zadatak Napravite funkciju koja ispituje da li je faktura ulazna ili izlazna i ukoliko je ulazna množi vrednost fakture sa (-1) , a ako je izlazna ne menja je. Pomoću ove funkcije napraviti proceduru koja računa dobit za uneti period.