Emir Mešković, Dario Osmanović

OSNOVI SQL-a ZA RELACIJSKE BAZE PODATAKA SA PRIMJERIMA U MySQL-u

Tuzla, 2018. god.

Predgovor

Sadržaj knjige prati nastavni plan i program auditornih i laboratorijskih vježbi predmeta Baze podataka, studijskog programa Elektrotehnika i računarstvo na Fakultetu elektrotehnike u Tuzli, ali je namijenjena i svim ostalim studentima koji u okviru svojih studijskih programa pohađaju nastavu iz predmeta koji se bave uvodom u područje obrade podataka i osnovama baza podataka, prvenstveno relacijskih baza podataka kao još uvijek dominantnih načina organizacije i obrade podataka u području poslovnih i web aplikacija. Osim toga namijenjena je i svima koji se na bilo koji način nastoje baviti obradom podataka i pitanjima koja se odnose na pohranu, dohvat i ažuriranje velike količine podataka.

Cilj knjige je upoznati čitatelja sa osnovama relacijskih baza podataka i SQL upitnog jezika za relacijske baze podataka čijim savladavanjem će se steći dovoljno znanja kako bi se mogla koristiti bilo koja relacijska baza podataka, odnosno relacijski sistem za upravljanje bazama podataka. U tu svrhu, u knjizi su opisane osnovne osobine SQL-a i detaljno predstavljena sintaksa osnovnih SQL naredbi te se kroz brojne primjere upita i naredbi nad konkretnim oglednim modelom baze podataka iz prakse omogućava čitatelju da ovlada upitnim jezikom koji čini osnovu svakog relacijskog sistema za upravljanje bazom podataka.

Fokus u knjizi je na stadardnom SQL-u, ali je za ilustraciju primjera ipak morao biti odabran konkretan RSUBP te je izbor pao na MySQL kao najšire korišteni server baze podataka u području razvoja desktop i web aplikacija koji se može izvršavati na svim popularnim operativnim sistemima. Treba naglasiti da se MySQL u knjizi koristi samo kao konkretan server za implementaciju i izvršavanje SQL naredbi i cilj nije upoznavanje sa svim mogućnostima i načinom korištenja MySQL-a, ali su sve razlike u odnosu na standardni SQL u knjizi jasno naznačene te će na taj način čitatelj ovladati i osnovnim znanjima za rukovanje ovim konkretnim RSUBP-om.

Knjiga ne obuhvata sve osobine i mogućnosti SQL upitnog jezika, ali predstavlja polaznu osnovu za daljna razmatranja koja se odnose implementaciju sigurnosnih pravila i konkuretnog pristupa u relacijskim bazama podataka, definiranje i rad sa okidačima (*trigger*) i pohranjenim procedurama, kao i drugim "naprednim" karakteristikama u SQL-u. Nadamo se da će knjiga najprije pomoći studentima u razumijevanju i savladavanju osnovama rada sa relacijskim bazama podataka, a u konačnici i polaganju kolegija u okviru kojih se obrađuje ova tematika

Autori

Sadržaj

POGLAVLJE 1

UVOD

Kada je 1971. godine E. F. Codd postavio osnovne koncepte relacijskog modela podataka između ostalog je iznio tvrdnju da: "... usvajanje relacijskog modela podataka ... dopušta razvoj univerzalnog jezika podataka baziranog na primjeni relacijske algebre." Međutim, tek nakon tri godine (1974) po prvi puta je predstavljen jedan strukturni jezik za pretraživanje podataka pod nazivom SEQUEL i to u članku autora D. D. Chamberlaina i R. F. Boycea. Isti autori godinu dana kasnije (1975) predstavljaju jezik SQUARE koji je koristio matematičke izraze, za razliku od engleskih termina koje je koristio SEQUEL. Ova dva koncepta, relacijski potpuna prema kriteriju koji je postavio E. F. Codd, korištena su kao osnov za razvoj prvog prototipa sistema za upravljanje bazama podataka (SUBP) pod nazivom System/R u laboratorijama IBM korporacije te pridruženog jezika baze podataka SEQUEL2 (Structured English Query Language). Nedugo zatim ovaj jezik mijenja naziv u SQL (Structured Query Language).

Relational Software Inc. je 1979. godine predstavio prvi relacijski sistem za upravljanje bazama podataka (RSUBP) pod nazivom Oracle koji se izvodio na miniračunarima uz upotrebu SQL-a kao upitnog jezika. Proizvod je postao tako popularan da je kompletna kompanija promijenila naziv u Oracle. Nedugo zatim, 1982. godine IBM predstavlja svoj prvi komercijalni SQL-bazirani RSUBP pod nazivom SQL/Data System ili SQL/DS, a tri godine kasnije (1985) se pojavljuje novo izdanje pod nazivom Database 2 System ili DB2.

S obzirom na činjenicu da je SQL vrlo brzo postao standardni jezik administratorima baza podataka (DBA) za pristup i upravljanje bazama podataka, kako bi se unificirale najbolje SQL prakse, American National Standards Institute (ANSI) je najprije 1986. godine inicijalno formalizirao SQL-87, da bi u godinama koje slijede bili kreirani specifični standardi za upitni jezik baza podataka.

1989: ANSI je objavio prvi skup standarda za upitne jezike baza podataka pod nazivom SQL-89 ili FIPS 127-1.

1992: ANSI je objavio revidirane standarde ANSI/ISO SQL-92 ili SQL2 koji su bili striktniji u odnosu na SQL-89 te u kojima su dodane neke nove osobine. Ovim standardima je predstavljen nivo usklađenosti kojima se navodi u kojoj mjeri dijalekti zadovoljavaju ANSI standarde.

1999: ANSI objavljuje SQL3, ili ANSI/ISO SQL: 1999, sa novim osobinama kao što su podrška za objekte. Zamjenjeni su nivoi usklađenosti sa osnovnim specifikacijama, kao i dodatne specifikacije za još devet paketa.

2003: ANSI objavljuje SQL: 2003 u kojem su predstavljene standardizirane sekvence, osobine koje se odnose na XML te kolone identiteta.

2006: ANSI objavljuje SQL: 2006 koji definiše kako koristiti SQL sa XML-om te omogućuje aplikacijama integraciju XQuery u postojeći SQL kod.

2008: ANSI objavljuje SQL: 2008 koji predstavlja INSTEAD OF okidače te TRUNCATE iskaz.

2011: ANSI objavljuje SQL: 2011 ili ISO/IEC 9075:2011, sedmu reviziju ISO (1987) i ANSI (1986) standarda za SQL upitni jezik baza podataka koji je podijeljen u devet dijelova.

Svaki proizvođač RSUBP-a nastoji što je moguće u većoj mjeri slijediti navedene standarde što se tiče SQL-a, što za posljedicu ima da su različite implementacije SQL-a namijenjene različitim RSUBP-ima vrlo slične. Međutim, svaka od implementacija nastoji SQL obogatiti različitim dodatnim opcijama, što s druge strane za posljedicu ima još uvijek vlastiti jedinstveni dijalekt za korištenje SQL-a koji uključuje određena proširenja ili dopune standarda. Činjenica je da zbog svoje složenosti niti jedan RSUBP ne podržava standarde u potpunosti te se u cilju omogućavanja kvantificiranja stepena usaglašenosti sa standardom SUBP-i klasicifiraju u tri kategorije: Entry SQL, Intermediate SQL i Full SQL. U knjizi je zastupljen Intermediate SQL koji pokriva mogućnosti standarda koji su najznačajniji u primjeni u danas raspoloživim komercijalnim proizvodima, ali budući da se u primjerima koristi konkretan SUBP (MySQL), bilo je potrebno sve slučajeve gdje se pojavljuju nestandardna proširenja jasno naznačiti.

Notacija

S obzirom na činjenicu da je SQL jezik slobodnog formata naredbi, što znači da naredbu možete napisati u jednoj liniji koda ili svaku riječ u zasebnoj liniji, te da osim toga nije osjetljiv na razliku između malih i velikih slova (*case insensitive*), na samom početku definiraćemo pravila notacije koja će biti korištena u svim primjerima u knjizi što se tiče SQL-a. Koliko god je to moguće, sintaksa SQL naredbe je napisana na najjednostavniji mogući način uvažavajući najčešće korištenu konvenciju u praksi. To prije svega podrazumijeva da će sve SQL ključne riječi, kao i sve ugrađene SQL funkcije, biti pisane velikim podebljanim slovima, dok će sav "izmjenjivi" dio naredbe (poput identifikatora, naziva kreiranih procedura i funkcija i sl.) biti pisan malim ukošenim slovima. Tako npr. naredbom:

```
CREATE DATABASE imeBazePodataka
```

se predstavlja opšti oblik ili sintaksa naredbe za kreiranje baze podataka u ovom slučaju, dok se u konkretnoj naredbi umjesto identifikatora *imeBazePodataka* koristi naziv neke stvarne baze podataka i piše u obliku:

```
CREATE DATABASE mojaBaza
```

Prilikom predstavljanja sintakse u uglatim zagradama će biti navedene opcije ili dijelovi pojedine naredbe koji nisu obavezni za navesti. Npr. prilikom kreiranja baze podataka (i

većine drugih objekata) u MySQL-u proizvoljno se može navesti opcija `IF NOT EXISTS`, što se u sintaksi navodi na sljedeći način:

```
MySQL CREATE DATABASE [IF NOT EXISTS] imeBazePodataka
```

Osim toga, u vitičastim zagradama se odvojene uspravnom crtom (|) navode obavezne opcije koje su međusobno isključive (što znači da se samo jedna smije navesti). Npr. U MySQL-u shema je sinonim za bazu podataka te je prilikom kreiranja baze podataka obavezno navesti jednu od ove dvije međusobno isključive ključne riječi:

```
MySQL CREATE { DATABASE | SCHEMA } [IF NOT EXISTS] imeBazePodataka
```

Ukoliko se neka opcija ili dio SQL naredbe može ponavljati više puta nakon nje se dodaju zarez i tri tačkice u uglatim zagradama, npr.

```
MySQL CREATE { DATABASE | SCHEMA } [IF NOT EXISTS] imeBazePodataka
    [create_specification [, ...]]
```

S obirom na gore navedenu činjenicu da je SQL case insensitive jezik slobodnog formata naredbi, sljedeće dvije naredbe su identične, odnosno ispravno napisane:

```
SELECT pbr, nazMjesto
  FROM mjesto
  WHERE pbr = 75000
```

```
select    Pbr          ,
      NAZMJESTO   From        mJESTo
Where
            PBR
      = 75000
```

Očigledno je desna naredba u prethodnom primjeru izrazito nepregledno napisana, te je uz prethodno predstavljenu notaciju u vezi s dijelovima naredbe koji se pišu malim i velikim slovima opće prihvaćena konvencija da se svaki segment naredbe (*Clause*) piše u zasebnom redu. U tom smislu lijeva naredba u prethodnom primjeru je napisana u skladu s tom preporukom te će i svi ostali primjeri SQL naredbi u nastavku biti napisani na isti način.

POGLAVLJE 2

SINTAKSA I OSNOVE SQL-a

Kod većine programskih jezika naredbu je obavezno završiti interpukcijskim znakom tačka-zarez (;) (poput C-a ili Jave) ili kraj linije ujedno označava i kraj naredbe (poput Visual Basic-a). Kako se SQL naredba može protezati preko više redova, nameće se zaključak da se na kraj naredbe mora postaviti određeni separator. Međutim, ovo u osnovi zavisi od načina na koji se naredba izvršava. Ukoliko se koristi interaktivni alat za izvođenje SQL naredbi (poput MySQL Workbench-a), naredbu je obavezno završiti tačkom-zarez (;) ako se želi izvesti više naredbi istovremeno jedna iza druge. Ukoliko se izvršava samo jedna naredba onda se ona ne mora završiti znakom (;). Kada se SQL naredbe uključuju u druge programske jezike (*Embedded SQL*), naredbe se završavaju na način svojstven tom programskom jeziku.

Većina programskih jezika, pa tako i SQL, ima mogućnost navođenja komentara u jednom ili više redova. Jednolinijski komentar počinje sa dva uzastopna znaka minus (--), proteže se do kraja linije i definiran je standardom što znači da ga podržavaju svi RSUBP-i:

```
SELECT * FROM mjesto; -- ovo je komentar
```

Osim ovog načina komentarisanja, u većini slučajeva (pa tako i u MySQL-u) podržani su i višelinijski komentari na način na koji se koriste u C programskom jeziku. Kombinacija znakova /* označava početak komentara, a kombinacija znakova */ kraj komentara.

```
    SELECT * FROM mjesto;
MySQL /* ovo je komentar koji se proteže
         preko više linija koda */
```

Prilikom postavljanja SQL naredbi koriste se identifikatori, odnosno imena objekata u SQL-u. Osnovni objekti koji se koriste u SQL-u su:

- baza podataka (*Database*)
- relacija, tabela (*Table*)
- atribut, kolona (*Column*)
- pogled ili virtuelna tabela (*View*)
- indeks (*Index*)
- pohranjena procedura (*Stored procedure*)
- pravilo integriteta (*Constraint*)
- okidač (*Trigger*)

Prilikom davanja imena objektima (identifikatora) u bazi podataka potrebno je poštovati određena pravila koja zavise i od odgovarajućeg RSUBP-a:

- ime objekta se formira od slova (ne smije sadržavati dijakritičke znakove poput č, ć, đ, ž, š), cifri i znaka donja crtica (_). U MySQL-u identifikator može sadržavati i znak dolar ($).
- Ime objekta mora početi sa slovom. U MySQL-u identifikator može početi i sa donjom crticom ili cifrom, ali se ne smije sastojati samo od cifri.
- Ime objekta ne može biti rezervisana ključna riječ. Npr. `BEGIN` i `END` jesu ključne rječi ali nisu rezervisane od sistema te se mogu koristiti kao identifikatori. U MySQL-u i rezervisane ključne riječi se mogu koristiti kao identifikatori, ali se moraju navesti pod ukošenim navodnicima (`) ili dvostrukim navodnicima (") ako je omogućen ANSI_QUOTES SQL način.
- Dužina identifikatora zavisi od implementacije sistema i sam standard je ne definira. Za MySQL većina identifikatora (osim *Alias*-a) ne može biti duža od 64 karaktera.

Na osnovu navedenih pravila možemo navesti primjere:

Ispravno formirani identifikatori: `knjiga` `posudbe2015god` `2015godina`

Neispravno formirani identifikatori: `2015.godina` `primjerak-knjige` `#invBroj`

Jedinstvena konvencija za određivanje identifikatora objekata u SQL-u (relacija, atributa, itd.) ne postoji i svakako nije definisana standardom. Međutim, u osnovi bi trebalo birati identifikatore koji imaju određeno značenje i opisuju objekat na koji se odnose te odabranu konvenciju koristiti u svim slučajevima (puni naziv ili skraćenice, više riječi razdvojene donjim crticom ili velikim slovom za svaku riječ, itd.).

Tipovi podataka

Osnovna namjena baza podataka je pohrana i upravljanje podacima koji se sa stanovišta baze podataka posmatraju samo kao jedna od vrijednosti određenog tipa podataka. SQL podržava tri vrste tipova podataka: predefinisani tipovi podataka, konstruisani tipovi i korisnički-definisani tipovi podataka. Predefinisani tipovi podataka se ponekad nazivaju i "ugrađeni tipovi", mada taj naziv nije internacionalni standard. Osim osnovnih predefinisanih tipova podataka koji su utvrđeni SQL standardom, postoje i izvedeni ugrađeni tipovi podataka implementirani u različitim komercijalnim RSUBP-ima. Ovdje će biti opisani samo najčešće korišteni predefinisani tipovi podataka, a svaki put kada se opisuje tip podataka koji je podržan u MySQL-u a nije definiran standardom biće posebno naglašeno. Konstruisani tipovi podataka poput `ARRAY`, `REF` i `ROW`, te korisnički definisani tipovi nisu predmet razmatranja relacijskih baza podataka te kao takvi nisu obrađeni ni u ovoj knjizi, kao ni tipovi definisani standardom a koji nisu podržani u MySQL-u poput `INTERVAL`.

Znakovni tipovi podataka

Znakovni tipovi podataka služe kako bi se pohranili nizovi znakova ili stringovi karaktera (*character string*) kako je uobičajen naziv u literaturi. Nizovi znakova mogu sadržavati alfanumeričke (slova i cifre) i specijalne karaktere (*non-printing characters*) pod kojim se podrazumijevaju svi kontrolni znakovi koji se kao takvi ne mogu odštampati na štampaču (npr. '\n' ili Carriage Return, tj. novi red pri štampanju). Tipovi podataka koji se mogu koristiti kako bi se mogli pohraniti stringovi karaktera u SQL-u su:

CHAR - Niz karaktera (string) fiksne dužine. Nenegativan cijeli broj koji određuje maksimalnu dužinu stringa se navodi kao argument u zagradama, npr. `CHAR(25)`. Vrijednosti ovog tipa se navode pod jednostrukim navodnicima, a maksimalna dužina stringa ovisi o RSUBP-u (do 255 znakova u MySQL-u) i nije definisana standardom. Ako je stvarna dužina unesenog stringa manja od maksimalne, SUBP dodaje prazne znakove do maksimalne dužine.

VARCHAR - Niz znakova (string) varijabilne dužine pri čemu se također mora definisati maksimalna dužina stringa (do 65.535 znakova u MySQL-u). Koliko će memorije zauzeti podatak ovog tipa zavisi od dužini stvarno upisanog niza znakova. Osim upisanog niza znakova u memoriju se pohranjuje i prefiks veličine 1 ili 2 bajta (zavisno da li se pohranjuje 255 ili više znakova) koji indicira stvarnu dužinu upisanog niza znakova.

NCHAR - Isto kao i `CHAR` tip podatka s tim da može sadržavati standardizovane višebajtne karaktere ili Unicode karaktere.

NVARCHAR - U odnosu na `VARCHAR` je isto što i `NCHAR` u odnosu na `CHAR`.

MySQL **TEXT** - MySQL tip podatka za *Character Large Object* ili `CLOB` koji predstavlja kolekciju karakter podataka u SUBP-u, obično pohranjenih na odvojenoj lokaciji koja se referencira iz same relacije.

BLOB - *Binary Large Object* koji može sadržavati varijabilnu količinu podataka koja se tretira kao binarni string (string bajta) te nema pridružen skup karaktera i način usporedbe (*collation*) te se poređenje i sortiranje temelji na numeričkim vrijednostima pohranjenih bajta u kolonama.

Boolean tip podatka

BOOLEAN - ili skraćeno `BOOL` tip podatka služi za pohranu i poređenje dvije vrijednosti istinitosti - istina (*true*) i laž (*false*).

Numerički tipovi podataka

Numerički tipovi podataka se koriste za pohranu brojčanih vrijednosti. U osnovi se mogu podijeliti u tri skupine: cjelobrojni tipovi (tačne vrijednosti), tipovi sa nepomičnom tačkom (tačne vrijednosti) i tipovi sa pomičnom tačkom (približne vrijednosti).

Cjelobrojni tipovi

INTEGER - Služi za pohranu cijelog broja, a alternativni naziv je INT. Minimalna i maksimalna vrijednost zavisi od SUBP-a, a najčešće se pohranjuje u 4 bajta.

SMALLINT - Kao i INTEGER služi za pohranu cijelog broja, ali može sadržavati manji opseg vrijednosti što zavisi od SUBP-a, a najčešće se pohranjuje u 2 bajta.

BIGINT - Kao i INTEGER služi za pohranu cijelog broja, ali može sadržavati veći opseg vrijednosti što zavisi od SUBP-a, a najčešće se pohranjuje u 8 bajta.

MySQL **SERIAL** - Služi za generisanje niza cijelih brojeva a u MySQL-u je *alias* za BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE. Više riječi o ovom tipu podataka i načinu generisanja i pohrane vrijednosti biće kasnije.

Tipovi sa nepomičnom tačkom

DECIMAL(p, s) - Služi za pohranu decimalnog broja pri čemu se specificira ukupan broj cifara koji decimalni broj sadrži (*precision*) i broj cifara koji se nalazi nakon decimalne tačke (*scale*). Npr. DECIMAL(9,3) predstavlja decimalan broj sa ukupno 9 cifara od kojih se 3 nalaze nakon decimalne tačke. Maksimalna preciznost odnosno broj cifara zavisi od SUBP-a (65 u MySQL-u).

NUMERIC(p, s) - Koristi se na isti način kao i DECIMAL.

Tipovi sa pomičnom tačkom

FLOAT(p) - služi za pohranu približne (aproksimativne) vrijednosti do preciznosti mantise p. Određivanje preciznosti je opciono i u MySQL-u se koristi samo kako bi se odredila veličina pohrane. Preciznost od 0 do 23 rezultira 4-bajtnom FLOAT kolonom jednostruke preciznosti, a preciznost od 24 do 53 rezultira 8-bajtnom DOUBLE kolonom dvostruke preciznosti.

DOUBLE - Sinonim za DOUBLE PRECISION iz SQL standarda

REAL - Sinonim za DOUBLE PRECISION iz SQL standarda, osim u slučaju kada je aktiviran REAL_AS_FLOAT SQL način.

Tipovi podataka za datum i vrijeme

SQL standard definira veliki broj mogućnosti za podržavanje podataka za prikaz datuma i vremena.

DATE - Služi za predstavljanje datumskih vrijednosti (bez podataka o vremenu) i u MySQL-u se vrijednosti ovog tipa dohvataju i prikazuju u formatu 'YYYY-MM-DD' te imaju opseg od '1000-01-01' do '9999-12-31'

TIME - Služi za predstavljanje vrijednosti o vremenu (bez podataka o datumu) i u MySQL-u se vrijednosti ovog tipa dohvataju i prikazuju u formatu 'HH:MM:SS[.fraction]' (ili

'HHH:MM:SS[.fraction]' za veće vrijednosti sata) te imaju opseg od '-838:59:59:[.000000]' do '838:59:59[.000000]'. Ukoliko se navode skraćene vrijednosti korištenjem dvotačke (npr. '12:25') MySQL ih interpretira kao vrijeme u toku dana ('12:25:00' a ne '00:12:25'). Ukoliko se navode skraćene vrijednosti bez dvotačke (npr. 1225) MySQL ih interpretira po dvije cifre s desna na lijevo ('00:12:25' a ne '12:25:00'). Ukoliko se unesu vrijednosti za vrijeme koje nisu validne one se pretvaraju u '00:00:00'. S obzirom da je ovo validna vrijednost, ukoliko u tabeli postoji pohranjena vrijednost '00:00:00' ne može se sa sigurnošću tvrditi da li je to validna vrijednost ili nije bila validna pa je izvršeno pretvaranje.

MySQL **DATETIME** - Služi da bi se predstavile i pohranile vrijednosti koje sadrže i datum i vrijeme. U MySQL-u se vrijednosti ovog tipa dohvataju i prikazuju u formatu 'YYYY-MM-DD HH:MM:SS[.fraction]' te imaju opseg od '1000-01-01 00:00:00[.000000]' do '9999-12-31 23:59:59[.999999]'

TIMESTAMP - Također služi da bi se predstavile i pohranile vrijednosti koje sadrže i datum i vrijeme. U MySQL-u se vrijednosti ovog tipa dohvataju i prikazuju u istom formatu kao i DATETIME te imaju opseg od '1970-01-01 00:00:01[.000000]' UTC do '2038-01-19 03:14:07[.999999]' UTC.

MySQL prilikom pohrane konvertuje TIMESTAMP vrijednosti iz trenutne vrmenske zone (koja se može postaviti na nivou konekcije, a podrazumjevano je vrijeme poslužitelja) u UTC, i obratno prilikom dohvata vrijednosti (što nije slučaj za DATETIME tip podatka). Ukoliko je uključen striktni način rada u MySQL-u, DATE, DATETIME i TIMESTAMP vrijednosti koje nisu validne se konvertuju u "nulte" vrijednosti odgovarajućeg tipa ('0000-00-00' ili '0000-00-00 00:00:00') i generiše se upozorenje. Ukoliko je uključen striktni način rada, vrijednosti koje nisu validne se ne dozvoljavaju i generiše se greška. Datumi koji sadrže dvije cifre za godinu nisu određeni jer nije poznato o kojem vijeku se radi. MySQL interpretira ovakve vrijednosti na način da se opseg godina od 00 do 69 konvertuje u opseg godina od 2000 do 2069, dok se opseg godina od 70 do 99 konvertuje u opseg godina od 1970 do 1999.

SQL literali

Termin literal u SQL-u se koristi da bi se označila konstantna vrijednost podatka. U skladu sa predstavljenim tipovima podataka SQL definira četiri tipa literala za numeričke, stringove karaktera, datumske ili vremenske i Boolean vrijednosti. Primjeri ispravnih literala u SQL-u su:

Numerički: 150, -200, 366.55, -250.05, 4E2, 5E-3

Stringovi karaktera (navode se pod jednostrukim navodnicima (' ')): 'BiH', '2017', 'Baze podataka', 'Dec 31, 2017'

Datum i vrijeme: '2000-12-31', '22:15:35', '2017-05-14 16:22:17.333'

Boolean: TRUE, FALSE

Naredbe za kreiranje i brisanje baze podataka

U SQL-u naredba za kreiranje baze podataka je `CREATE DATABASE`, ali ona nije definisana ANSI standardom. Međutim, većina RSUBP-a podržava `CREATE DATABASE` naredbu uz određene varijacije, pa je MySQL varijanta već predstavljena prilikom defnisanja notacije korištene u knjizi:

MySQL
```
CREATE { DATABASE | SCHEMA } [IF NOT EXISTS] imeBazePodataka
    [create_specification [, ...]]
```

Korisnik koji obavi ovu naredbu postaje vlasnik kreirane baze podataka. Na jednom MySQL serveru ne mogu postojati dvije baze podataka pod istim nazivom. Ukoliko neki korisnik pokuša kreirati bazu podataka sa istim imenom a da pri tome nije naveo opciju `IF NOT EXISTS` dobiće poruku da se dogodila greška.

Prilikom kreiranja baze podataka mogu se navesti i karakteristike baze podataka u `create_specification` dijelu naredbe koje se pohranjuju u `db.opt` fajl u direktoriju baze podataka:

MySQL
```
create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

`CHARACTER SET` dio naredbe specificira podrazumijevani skup karaktera baze podataka, dok `COLLATE` specificira podrazumjevani način poređenja u bazi podataka (*collation*). Npr.

```
CREATE DATABASE bibliotekaFET
CHARACTER SET = 'utf8' COLLATE = 'utf8_croatian_ci';
```

Ukoliko korisnik ima dozvolu za pristup bazi podataka na određenom serveru, to može uraditi naredbom:

MySQL
```
USE imeBazePodataka
```

Sve SQL naredbe koje slijede će se odnositi na spojenu bazu podataka. Korisnik ostaje spojen na bazu podataka do kraja sesije ili dok ne izvede novu `USE` naredbu. Međutim, spajanjem na jednu bazu podataka korisnika ne sprječava da koristi podatke i iz drugih baza podataka na MySQL serveru za koje ima dozvole pristupa, ali u SQL naredbama mora navoditi ime baze podataka prije naziva objekta kojem pristupa. Npr. ukoliko se korisnik naredbom:

```
USE bibliotekaFET
```

spojio na bazu podataka, može koristiti i relacije iz baze podataka `nastavaFET` na istom serveru na sljedeći način:

```
SELECT brIndex, ocjena
FROM stud INNER JOIN nastavaFET.ispit
ON stud.mbrStud = nastavaFET.ispit.mbrStud
```

Za brisanje objekata (relacija, pogleda, indeksa, itd.) iz baza podataka koristi se SQL naredba `DROP`. Ukoliko se želi obrisati kompletna baza podataka sa svim objektima koje sadrži, komplementarno `CREATE DATABASE` naredbi može se koristiti naredba

MySQL
```
DROP { DATABASE | SCHEMA } [IF EXISTS] imeBazePodataka
```

Ova naredba će ukloniti i iz direktorija baze podataka sve fajlove i direktorije koje MySQL kreira tokom normalnog izvođenja operacija kao i `db.opt` fajl, npr.

```
DROP DATABASE bibliotekaFET
```

Većina postojećih SUBP-a nude mogućnost modifikovanja opštih karakteristika postojeće baze podataka ili fajlova pridruženih bazi podataka korištenjem naredbe `ALTER DATABASE`. Šta je sve moguće izmijeniti i na koji način kada je riječ o postojećim bazama podataka zavisi od kojem konkretnom SUBP-u se radi, a u MySQL-u ova naredba ima sljedeću sintaksu:

MySQL
```
ALTER { DATABASE | SCHEMA } [imeBazePodataka]
    alter_specification [,...]
```

`alter_specification` dio naredbe podrazumijeva iste karakteristike koje se mogu navesti i prilikom kreiranja baze podataka sa `CREATE DATABASE` naredbom, a koje su pohranjene u `db.opt` fajl u direktoriju baze podataka. Kao što se može primijetiti, *`imeBazePodataka`* je opciono i ukoliko se izostavi naredba se odnosi na podrazumijevanu (default) bazu podaaka na serveru. Drugi oblik `ALTER DATABASE` naredbe:

MySQL
```
ALTER { DATABASE | SCHEMA } imeBazePodataka
    UPGRADE DATA DIRECTORY NAME
```

se koristi prilikom nadogradnje sa verzija koje su starije od MySQL 5.1 i služi za ažuriranje naziva direktorija pridruženog bazi podataka kako bi se koristilo kodiranje implementirano u MySQL 5.1 za preslikavanje naziva baza podataka u nazive direktorija baza podataka.

Naredbe za kreiranje i brisanje relacija

Relacije ili tabele (*table*) su osnovne jedinice organizacije i pohrane podataka u relacijskim bazama podataka, pa tako i u SQL-u kao njihovom pridruženom jeziku. Pojednostavljeno, relacija se organizuje u redove i kolone kao dvodimenzionalna tablica u kojoj redovi predstavljaju n-torke ili zapise, a kolone atribute ili svojstva. Svaki zapis u relaciji predstavlja jedan entitet iz realnog svijeta, dok relacija predstavlja skup sličnih entiteta kojima se posmatraju (evidentiraju) ista svojstva. Npr. svako mjesto se može opisati poštanskim brojem, nazivom i kantonom kojem pripada koji se mogu predstaviti atributima `pbr`, `nazMjesto` i `sifKanton` u relaciji `mjesto`. U tom slučaju, svaki zapis u relaciji `mjesto` će predstavljati konkretno mjesto (entitet) iz realnog svijeta:

pbr	nazMjesto	sifKanton
75000	Tuzla	3
72000	Zenica	4
76120	Brčko	12
78000	Banja Luka	11

Struktura svake relacije u bazi podataka definira se pomoću SQL naredbe `CREATE TABLE`. Sintaksa ove naredbe implementirana u konkretnim SUBP-ima ponekad odstupa od standarda, a često se dodaju i mnoge druge opcije. Međutim, osnovni oblik naredbe je zajednički bez obzira na implementaciju i prati SQL standard:

```
CREATE [TEMPORARY] TABLE imeRelacije (
   column_definition [, ...]
   [table_constraint [, ...]]
)

column_definition:
  column_name data_type [NOT NULL] [DEFAULT default_value]
[column_constraint [, ...]]
```

Kao što se vidi iz sintakse `CREATE TABLE` naredbe, pri kreiranju relacije korisnik mora definirati bar jedan atribut (`column_definition`), a za svaki atribut koji se definira obavezno je specificirati naziv atributa (`column_name`) i tip podatka (`data_type`). Opciono se mogu navesti `NOT NULL` i `DEFAULT` kvalifikatori. `NOT NULL` kvalifikator definira da se prilikom unosa zapisa za dati atribut mora navesti određena vrijednost, odnosno da atribut ne može poprimiti `NULL` vrijednost. Koncept `NULL` vrijednosti će biti opisan nešto kasnije i detaljno razmatran u posebnom poglavlju o `NULL` vrijednostima. `DEFAULT` kvalifikator omogućava da se za atribut definira podrazumijevana vrijednost koja će biti pohranjena ukoliko se prilikom unosa zapisa ista eksplicitno ne navede. U MySQL-u to mora biti konstantna vrijednost (literal) i ne smije biti funkcija ili izraz. Izuzetak je `CURRENT_TIMESTAMP` kao podrazumijevana vrijednost za atribut `DATETIME` ili `TIMESTAMP` tipa podatka.

Svaka relacija ima svoj ključ, a to je atribut ili skup atributa čije vrijednosti u relaciji moraju biti jedinstvene, odnosno u relaciji se ne mogu pojaviti dva zapisa koja će imati jednaku vrijednost ključa. U relaciji može postojati više ključeva, ali se samo jedan od njih proglašava primarnim ključem. Na osnovu ovoga može se zaključiti da vrijednost primarnog ključa jedinstveno određuje svaki zapis u relaciji, odnosno sve ostale vrijednosti atributa za svaki zapis u relaciji. U SQL-u se jedinstvenost vrijednosti osigurava ograničenjem `UNIQUE`, dok se primarni ključ definira pomoću naznake `PRIMARY KEY`. Razlika je u tome što `PRIMARY KEY` neće dozvoliti pojavu `NULL` vrijednosti za atribut koji je sastavni dio primarnog ključa. Oba ograničenja se mogu definirati uz definiciju atributa na koji se odnose (`column_constraint`) ili zasebno na kraju `CREATE TABLE` naredbe, nakon definicije svih atributa relacije (`table_constraint`).

Osim primarnog, relacija može sadržavati i strani ključ koji predstavlja atribut ili skup atributa koji referiraju na primarni ključ neke druge relacije. Strani ključ u relaciji se

definira pomoću naznake `FOREIGN KEY` koja se također može navesti uz definiciju atributa ili na kraju `CREATE TABLE` naredbe. O načinu definiranja navedenih i ostalih ograničenja u SQL-u kao sastavnog dijela `CREATE TABLE` naredbe detaljno će biti riječi u zasebnom poglavlju posvećenom integritetu podataka.

Naredba za kreiranje relacije mjesto iz naprijed navedenog primjera sa definiranim pojedinim ograničenjima za atribute bi imala sljedeći oblik:

```
CREATE TABLE mjesto(
    pbr           INTEGER                 PRIMARY KEY
  , nazMjesto     NCHAR(30)   NOT NULL    UNIQUE
  , sifKanton     SMALLINT    NOT NULL
  , FOREIGN KEY (sifKanton) REFERENCES kanton(sifKanton)
)
```

Ukoliko se prilikom kreiranja relacije navede ključna riječ `TEMPORARY` kreira se privremena relacija koja ima sve karakteristike trajne relacije, osim što se vidi samo u sesiji u kojoj je kreirana te će biti uništena kada se ta sesija okonča. Ovo praktično znači da u jednoj bazi podataka dvije različite sesije mogu koristiti iste nazive za privremene relacije a da ne dođe do konflikta. Privremena relacija se najčešće koristi u slučajevima kada je nad bazom podataka potrebno izvesti složenu operaciju koja se ne može izvesti u jednom koraku, nego se izvodi u više koraka u kojima se međurezultati smještaju u privremenu relaciju.

Kao i za sve druge objekte u SQL-u, i za brisanje trajne ili privremene relacije se koristi naredba `DROP`. Opšti oblik ove naredbe kojom se bespovratno uništavaju i podaci i struktura relacije ili privrmene relacije je:

MySQL **`DROP [TEMPORARY] TABLE [IF EXISTS]`** *`imeRelacije`*

Kada se jednom obriše relacija sa `DROP` naredbom, ta operacija ne može biti poništena te se mora biti vrlo pažljiv prilikom njenog izvođenja i po potrebi prije brisanja kreirati backup relaciju. Ukoliko je u relaciji definirano `FORIGN KEY` ograničenje, potrebno ga je izbrisati prije nego se izvrši naredba za brisanje relacije. Kada se relacija izbriše, sva ostala ograničenja i okidači (*trigger*) koji se odnose na obrisanu relaciju će također biti automatski obrisani.

Često se u praksi javlja potreba za izmjenom strukture postojeće relacije koja može podrazumijevati dodavanje novih atributa ili brisanje postojećih, kreiranje novih i brisanje postojećih indeksa, promjena tipa podataka atributa, preimenovanje atributa ili same relacije, itd. Osim strukture relacije mogu se promijeniti i njene karakteristike poput načina pohrane ili komentar. Iako je ove izmjene moguće napraviti u praksi bi ih trebalo izbjegavati što je više moguće jer mogu dovesti do gubitka informacija, narušavanja strukture baze podataka, potrebe za izmjenom SQL naredbi ili koda aplikacije koja koristi bazu podataka, itd. Naravno, kako ne postoji savršeno dizajniran sistem uprkos najbolje provedenom planiranju i dizajniranju, ponekad je neophodno izmijeniti strukture relacija te se u tu svrhu koristi `ALTER TABLE` naredba sa sljedećom sintaksom:

MySQL
```
ALTER [IGNORE] TABLE imeRelacije
    [alter_specification [,...]]
```

S obzirom na činjenicu da je moguće napraviti veliki broj izmjena nad relacijom od kojih je samo mali broj naprijed naveden, `alter_specification` podrazumijeva veliki broj opcija koje se mogu navesti. Detaljna specifikacija svih opcija nije predmet razmatranja u ovoj knjizi te ćemo navesti samo jedan ograničeni broj najčešće koristenih opcija od kojih će neke biti zastupljene i u primjerima SQL naredbi u poglavljima koji slijede.

```
alter_specification:
    ADD [COLUMN] col_name column_definition [FIRST | AFTER col_name]
  | ADD {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
    [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
    [index_option] ...
  | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index_name] [index_type]
    (index_col_name,...) [index_option] ...
  | 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 new_col_name column_definition
    [FIRST|AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP {INDEX|KEY} index_name
  | DROP FOREIGN KEY fk_symbol
```

Npr. ukoliko želimo promijeniti tip podatka i veličinu za atribut `nazMjesto` u relaciji `mjesto` koristićemo naredbu:

```
ALTER TABLE mjesto MODIFY nazMjesto VARCHAR(50)
```

Osnovne SQL naredbe za manipulaciju podacima

Osnovne operacije koje korisnik može obavljati nad podacima u bazi podataka podrazumijevaju unos podataka, brisanje, izmjenu i pretraživanje postojećih podataka. Ovdje ćemo samo navesti četiri osnovne naredbe u SQL-u koje omogućavaju izvođenje ovih operacija, bez detaljnog razmatranja i analiziranja njihove sintakse, jer će svaka od naredbi biti detaljno objašnjena u poglavljima koja slijede.

a) Dohvat (selekcija, pretraživanje) podataka

Za pregled ili pretraživanje podataka u bazi podataka najčešće se koristi termin upit (*query*) nad bazom podataka po čemu je SQL i dobio naziv upitni jezik (*query language*). Za postavljanje upita nad bazom podataka koristi se `SELECT` naredba.

***Primjer**: Dohvatiti sve nazive mjesta čiji su poštanski brojevi veći od 70000 i nalaze se u kantonu sa šifrom 4.*

```
SELECT nazMjesto FROM mjesto
   WHERE pbr > 70000 AND sifKanton = 4
```

Relacija (`mjesto`) iz koje se dohvataju podaci odnosno zapisi se navodi u `FROM` dijelu `SELECT` naredbe. U listi za selekciju (*SELECT Clause*) navode se svojstva (atributi) entiteta koji se žele vidjeti u izlaznoj listi (`nazMjesto`). U `WHERE` dijelu naredbe se navodi predikat (`pbr > 70000 AND sifKanton = 4`) koji svaki zapis treba zadovoljiti da bi se pojavio u izlaznoj listi, odnosno rezultatu upita. Ukoliko se `WHERE` dio naredbe izostavi dohvataju se svi zapisi koji postoje u relaciji `mjesto`.

b) Unos podataka

Za unos jednog zapisa (n-torke, reda) u relaciju koristi se `INSERT` naredba.

***Primjer**: U relaciju mjesto unijeti podatke o novom mjestu čiji je poštanski broj 88000, naziv mjesta Mostar i pripada kantonu sa šifrom 7.*

```
INSERT INTO mjesto VALUES (88000, 'Mostar', 7)
```

Ovo je najjednostavniji oblik `INSERT` naredbe u kojem se u `INTO` dijelu naredbe navodi naziv relacije u koju se unosi zapis (`mjesto`), a u `VALUES` dijelu naredbe unutar zagrada vrijednosti atributa onim redoslijedom kojim su navedeni u `CREATE TABLE` naredbi prilikom kreiranja relacije.

c) Ažuriranje podataka

Za ažuriranje podataka koristi se `UPDATE` naredba.

***Primjer**: Mjestu sa poštanskim brojem 75000 promijeniti naziv u Grad Tuzla.*

```
UPDATE mjesto SET nazMjesto = 'Grad Tuzla'
   WHERE pbr = 75000
```

Nakon ključne riječi `UPDATE` navodi se naziv relacije (`mjesto`) u kojoj će se ažurirati (izmijeniti) podaci. U `SET` dijelu naredbe se navodi atribut koji se ažurira i nov vrijednosti koja će se dodijeliti tom atributu (`nazMjesto = 'Grad Tuzla'`). `WHERE` dio naredbe kao i u slučaju `SELECT` naredbe služi za navođenje predikata kako bi se odredili zapisi koji će biti ažurirani (samo zapisi čija je prijednost atributa `pbr` jednaka 75000).

d) Brisanje podataka

Za brisanje zapisa u relaciji koristi se `DELETE` naredba.

***Primjer:** Obrisati sve podatke o mjestima čiji naziv počinje slovom S.*

```
DELETE FROM mjesto
   WHERE nazMjesto LIKE 'S%'
```

Naredbom se uklanja kompletan zapis iz relacije te `FROM` i `WHERE` dijelovi naredbe imaju isto značenje kao u slučaju `SELECT` naredbe.

Naredbe za prenos podataka u/iz datoteke operativnog sistema

Kada je potrebno izvršiti prenos (transfer) podataka iz jedne baze podataka u drugu na različitim serverima, najčešće se obavlja na način da se podaci eksportuju u datoteku na disku u odgovarajućem formatu, a potom iz datoteke učitavaju u odredišnu relaciju. Format podataka u datotekama je najćešće takav da jedan zapis u relaciji predstvalja jedan red u datoteci, a da su vrijednosti atributa razdvojene specijalnim karakterom (delimiter) čija se vrijednost ne može naći u samim podacima, npr.

```
75000#Tuzla#3
72000#Zenica#4
76120#Brčko#12
```

Iako većina interaktivnih alata za rad sa bazama podataka (pa tako i MySQL Workbench) imaju mogućnost jednostavnog izbora odgovarajućih opcija iz traka sa alatima ili menija, za izvođenje ovih operacija mogu se koristiti i odgovarajuće SQL naredbe. Prenos podataka iz relacije u datoteku na disku u ASCII formatu u MySQL-u se obavlja naredbom `SELECT` na kraju koje se navodi opcija `INTO OUTFILE`

MySQL
```
SELECT ... INTO OUTFILE 'imeFajla'
    [CHARACTER SET charset_name] export_options
```

Ovaj oblik `SELECT` naredbe rezultat upita umjesto na ekranu upisuje u fajl na serveru te korisnik koji izvodi naredbu mora imati `FILE` dozvolu. *`imeFajla`* predstavlja apsolutni put do fajla koji ne smije postojati, što između ostalog ima za svrhu da se spriječi uništavanje fajlova poput `/etc/passwd` i tabela baze podataka. Vrijednosti atributa se upisuju konvertovane u skup karaktera specificiran sa `CHARACTER SET` opcijom, a ako se izostavi koristi se `binary` skup karaktera. Sintaksa `export_options` je sljedeća:

```
export_options:
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
```

`FIELDS` i `LINES` opcije su opcione ali ukoliko se obje navedu `FIELDS` mora prethoditi `LINES`. Svaka od `FIELDS` podopcija (`TERMINATED BY`, `ENCLOSED BY` i `ESCAPED BY`) su također opcione ali se mora navesti bar jedna. Ukoliko se `FIELDS` ili `LINES` opcije ne navedu podrazumijeva se sljedeće:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

Drugim riječima, `SELECT ... INTO OUTFILE` naredba podrazumijevano podatke zapisuje na sljedeći način:

- Ubaci tab između polja (vrijednosti atributa)
- Ne zatvaraj polja (vrijednosti atributa) bilo kakvim navodnicima
- Koristi \ kao escape karakter za tab, novi red ili znak \ koji se pojavi u poljima (vrijednosti atributa)
- Na kraju linije ubaci oznaku za novi red

Dakle, ukoliko želimo podatke iz relacije `mjesto` u navedenom ASCII obliku upisati u fajl `mjesto.csv` koji se nalazi u direktoriju `temp`, koristićemo sljedeću naredbu:

```
SELECT * FROM mjesto
INTO OUTFILE '/temp/mjesto.csv'
FIELDS TERMINATED BY '#'
LINES TERMINATED BY '\n';
```

Za prenos podataka iz datoteke u ASCII formatu u relaciju u bazi podataka, kao komplement naredbi `SELECT ... INTO OUTFILE` u MySQL-u se koristi naredba `LOAD DATA INFILE` čiji je oblik:

MySQL
```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [fields_statement]
    [lines_statement]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Sintaksa za `fields_statement` i `line_statement` je identična kao u slučaju `SELECT ... INTO OUTFILE` naredbe i ima komplementarno značenje. Ukoliko se koristi `LOW PRIORITY` opcija, izvršavanje naredbe se odgađa sve dok svi korisnici ne prestanu sa čitanjem podataka iz relacije, dok opcija `CONCURRENT` dozvoljava unos podataka u skladu sa konkuretnim pristupom. `LOCAL` ključna riječ definira očekivanu lokaciju fajla te ako se ne navede mora biti lociran na serveru na kojem se nalazi baza podataka. `REPLACE` i `IGNORE` ključne riječi kontrolišu unos zapisa koji dupliciraju postojeće zapise u pogledu `UNIQUE`

ograničenja. `IGNORE` *`number`* `LINES` opcija se može koristiti za ignorisanje redova na početku fajla.

Ukoliko se na kraju naredbe ne navede lista atributa, očekuje se da zapisi iz ulaznog fajla sadrže vrijednosti za sve atribute relacije. Ukoliko se žele unositi vrijednosti za samo određene atribute navodi se lista tih atributa. `SET` opcija se može koristiti da se za atribute definiraju vrijednosti koje ne postoje u ulaznom fajlu ili se prethodno trebaju izmijeniti određenom operacijom.

Na osnovu navedenog, da bismo iz postojećeg fajla `mjesto.csv` podatke učitali u relaciju `mjesto` koristićemo sljedeću naredbu:

```
LOAD DATA INFILE '/temp/mjesto.csv' INTO TABLE mjesto
FIELDS TERMINATED BY '#'
LINES TERMINATED BY '\n';
```

POGLAVLJE 3

SELECT NAREDBA ZA POSTAVLJANJE UPITA NAD BAZOM PODATAKA

Kao što je već navedeno u prethodnom poglavlju, SQL `SELECT` naredba se koristi za dohvat podataka iz relacija (trajnih ili prvremenih) i pogleda (*view*). Osim toga, `SELECT` naredba se koristi i kao sastavni dio drugih operacija nad bazama podataka poput prenosa podataka u datoteku na disku, unosa podataka u relaciju, kreiranja privremenih relacija i pogleda, kao podupit u drugim upitima i SQL naredbama, itd. SQL `SELECT` naredba implementira operaciju selekcije relacijske algebre te njen rezultat ima strukturu relacije, odnosno atribute i njihove vrijednosti te se kao takav može koristiti i kao argument druge `SELECT` naredbe pri ugnježđivanju. Osnovna sintaksa `SELECT` naredbe je:

```
SELECT [ALL | DISTINCT ] select_expr [, ...]
    FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name MySQL| expr MySQL| position} [, ...]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | alias | position} [ASC | DESC] [, ...]]
```

Kako i u praksi pristup i dohvat tačno određenih podataka i informacija od interesa može biti izuzetno težak i komplikovan zbog uslova i kriterija koje je potrebno ispitati, tako i sama struktura `SELECT` naredbe, a i pojedinih njenih segmenata, može biti prilično složena, te zavređuje posebno razmatranje. Zbog toga će svaki pojedini segment `SELECT` naredbe biti postepeno opisan u podpoglavljima koji slijede.

Jednostavni SELECT upiti

Svaka `SELECT` naredba obavezno mora sadržavati listu za selekciju ili `SELECT` segment naredbe (*SELECT Clause*) i listu naziva objekata iz kojih se dohvataju podaci ili `FROM` segment naredbe (*FROM Clause*). U listi za selekciju navode se atributi čije se vrijednosti žele dohvatiti ili izrazi čiji se rezultati trebaju pojaviti u rezultatu, a opciono se mogu

navesti kvalifikatori `ALL` ili `DISTINCT`. U `FROM` segmentu naredbe se navodi ime relacije iz koje se dohvataju podaci.

***Primjer**: Dohvatiti nazive, skraćenice i broj ECTS kredita za sve predmete koji se nalaze u relaciji `pred`.*

```
SELECT nazPred, skrPred, brojECTS FROM pred
```

Ovo je primjer najjednostavnijeg oblika `SELECT` naredbe u kojem se u listi za selekciju navode nazivi atributa čije se vrijednosti dohvataju, a u `FROM` segmentu naredbe naziv samo jedne relacije kojoj atributi pripadaju. Lista za selekciju u opštem slučaju ima sljedeći oblik:

```
select_expr:
    expression | [{table_name | view_name}.]col_name [[AS] alias]
  | [{table_name | view_name}.]*
```

Kao što se može vidjeti iz opšteg oblika liste za selekciju (*SELECT Clause*), osim naziva atributa mogu se koristiti i izrazi (*expression*) koji se mogu sastojati od konstanti, naziva atributa, aritmetičkih operatora i različitih funkcija te mogu biti vrlo složeni. Zbog toga ćemo im nešto kasnije posvetiti posebnu pažnju. Prilikom navođenja naziva atributa ispred se uvijek može navesti naziv relacije kojoj atribut pripada i tačka. U `SELECT` naredbama u kojima se koristi više relacija postoji mogućnost da različite relacije sadrže atribute sa istim nazivom te ukoliko se takav atribut upotrijebi u listi za selekciju onda on nije jednoznačno određen jer server neće znati iz koje od relacija da dohvati njegove vrijednosti. U takvim slučajevima navođenje naziva relacije ispred naziva atributa je obavezno. Npr. u sljedećoj naredbi navođenje naziva relacije nije potrebno, ali isto tako nije ni pogrešno (neće dovesti do greške):

```
SELECT mjesto.nazMjesto FROM mjesto
```

U listi za selekciju se osim naziva atributa i izraza može koristiti i iznaka (*) ili `table_name.*` koja se koristi da zamijeni sve atribute relacije navedene u `FROM` dijelu naredbe odnosno relacije `table_name`. U ovom slučaju se skraćuje pisanje SQL naredbi jer nije potrebno navoditi nazive svih atributa u slučajevima kada je potrebno dohvatiti vrijednosti svih atributa. Tako npr. sljedeće tri SELECT naredbe su ekvivalentne i imaće isti rezultat:

```
SELECT * FROM mjesto
```

```
SELECT mjesto.* FROM mjesto
```

```
SELECT pbr, nazMjesto, sifKanton FROM mjesto
```

Uz svaki izraz ili naziv atributa u listi za selekciju se može navesti zamjenski ili korelacijski naziv, odnosno *alias* naziv, koji će se u izlaznom rezultatu pojaviti kao naziv kolone. Ukoliko se *alias* naziv ne navede za naziv kolone se koristi naziv atributa kako je definisan prilikom kreiranja relacije, a za izraze termin koji ovisi od interaktivnog alata za izvođenje naredbi (najčešće *Expression*). Uloga *alias* naziva je isključivo vizuelne prirode kao "naziv atributa" u rezultatu, te se ne može koristiti niti u jednom dijelu `SELECT` naredbe

osim u `ORDER BY` dijelu. Ključna riječ `AS` se može a i ne mora navesti prilikom dodjeljivanja *alias* naziva.

```
SELECT pbr POŠTANSKI_BROJ, nazMjesto NAZIV_MJESTA, sifKanton ŠIFRA_KANTONA
FROM mjesto
```

Ukoliko se u listi za selekciju upotrijebi kvalifikator `DISTINCT` tada se iz rezultata eliminišu svi višestruki zapisi, odnosno u rezultatu se pojavljuje samo jedna n-torka iz skupa n-torki koje imaju iste vrijednosti svake od kolona navedenih u listi za selekciju. Npr. ukoliko se žele dohvatiti poštanski brojevi mjesta stanovanja članova biblioteke tako da se svaki poštanski broj u rezultatu pojavi samo jednom (jer je poznato da više članova može stanovati u istom mjestu), koristi se sljedeći upit:

```
SELECT DISTINCT pbrStan FROM clan
```

Prethodna naredba predstavlja primjer implementacije operacije projekcije relacijske algebre u SQL-u, konkretno projekcije relacije `clan` po atributu `pbrStan`, tj. $\pi_{\text{pbrStan}}(\text{clan})$.

Ukoliko se kvalifikator `DISTINCT` izostavi u rezultatu se pojavljuju vrijednosti atributa `pbrStan` za svaku n-troku relacije `clan`, što znači da se vrijednost jednog poštanskog broja stanovanja pojavi onoliko puta u rezultatu koliko ima članova koji stanuju u tom mjestu. Isti rezultat se dobije upotrebom kvalifikatora `ALL` te on ima suprotno značenje od `DISTINCT` i može se izostaviti u naredbi.

Izrazi

Izrazi se mogu koristiti na više mjesta u SQL naredbama, poput `ORDER BY` ili `HAVING` dijela `SELECT` naredbe, u listi za selekciju ili `WHERE` dijelu `SELECT` naredbe, `UPDATE` ili `DELETE` naredbi ili u `SET` iskazima. Izrazi se pišu korištenjem literala ili konstanti, naziva atributa, `NULL` vrijednosti, ugrađenih, pohranjenih ili korisnički definisanih funkcija i operatora.

Operatori

Operatori u SQL-u se izražavaju specijalnim karakterima kao i u drugim programskim jezicima te ključnim riječima. Npr. *plus* operator se predstavlja znakom (+) dok operator koji provjerava null vrijednosti se predstavlja ključnim riječima `IS NULL` ili `IS NOT NULL`. Postoji nekoliko tipova operatora koji se mogu koristiti u SQL-u, a navedeni su u Tabeli 1.

Poseban operator su dvije okomite crte (*pipe sign*) navedene bez razmaka (||) koje predstavljaju operator za konkatenaciju (spajanje) niza znakova. U MySQL-u ovaj operator podrazumijevano služi kao logičko OR te ukoliko se želi koristiti za konkatenaciju stringova mora se uključiti opcija `PIPES_AS_CONCAT`.

Redoslijed obavljanja operacija, odnosno prioritet operatora, je uobičajen kao u drugim programskim jezicima, a izrazi se mogu grupisati korištenjem zagrada. U tabeli 2 je data lista prioriteta operatora počevši od operatora sa najvišim prioritetom (u prvom redu) do

operatora sa najnižim prioritetom (u posljednjem redu) dok su operatori istog prioriteta navedeni u istom redu.

Tabela 1 Operatori u SQL-u

SQL operatori	Opis
Aritmetički operatori	Izvode aritmetičke operacije koje uključuju numeričke operande. Aritmetički operatori su sabiranje (+), oduzimanje (-), množenje (*), dijeljenje (/) i operator ostatka cjelobrojnog dijeljenja (% ili MySQL MOD).
Operatori poređenja (relacijski)	Matematički simboli koji se koriste za poređenje dvije vrijednosti. Rezultat poređenja može biti `true`, `false` ili `uknown`. Relacijski operatori su <, <=, =, > i >= i imaju isto značenje kao i u drugim programskim jezicima. Operator nejednakosti je <> (MySQL !=).
Operator dodjeljivanja	U SQL-u operator dodjeljivanja (=) se koristi za dodjeljivanje vrijednosti sa desne strane varijabli ili atributu tabele na lijevoj strani. U MySQL-u se kao operator dodjeljivanja koristi i :=
Operatori sa bitima	Izvode manipulaciju nad bitima između dva cjelobrojna izraza iz kategorije cjelobrojnih tipova podataka. Operatori sa bitima su bitsko AND (&), bitsko OR (|) i bitsko ekskluzivno OR ili XOR (^). U MySQL-u se koriste i operatori sa bitima lijevi pomak (<<) i desni pomak(>>).
Logički operatori	Služe za povezivanje više uslova poređenja i kao rezultat daju `true` ili `false`. Logički operatori su `AND`, `OR`, `NOT`, `IN`, `BETWEEN`, `ANY`, `ALL`, `SOME`, `EXISTS` i `LIKE` (MySQL `RLIKE`). Više riječi o svakom od logičkih operatora će biti u nastavku.
Unarni operatori	Izvode operacije nad jednim izrazom koji je bilo kojeg tipa iz kategorije numeričkih tipova podataka. Unarni operator (+) znači da je numerička vrijednost pozitivna, (-) znači da je negativna, a (~) je bitsko NOT ili invertovanje bita i koristi se samo nad cjelobrojnim tipovima podataka.

Tabela 2 Prioritet operatora u SQL-u

!
- (unarni minus), ~
^
*, /, %
-, +
<<, >>
&
|
= (poređenje), >=, >, <=, <, <>, !=, LIKE (MySQL RLIKE), IN
BETWEEN
NOT
AND
XOR
OR
= (dodjeljivanje) (MySQL :=)

Za operatore koji su istog nivoa prioriteta a pojave se u jednom izrazu, izračunavanje se izvodi s lijeva na desno, sa izuzetkom dodjeljivanja koje se izvodi s desna na lijevo. Ukoliko se operator koristi sa operandima različitog tipa podataka, obavlja se konverzija tipova kako bi operandi bili kompatibilni. Neke konverzije se obavljaju implicitno (poput

konverzije brojeva u stringove i obrnuto) dok se eksplicitna konverzija u MySQL-u može izvršiti korištenjem `CAST()` funkcije.

Funkcije

U SQL-u postoji veliki broj ugrađenih funkcija koje se mogu koristiti u izrazima, a mogu se podijeliti u nekoliko kategorija:

Aritmetičke funkcije - izvode matematičke operacije najčešće na osnovi ulaznih vrijednosti koje se proslijede kao argumenti, te kao rezultat operacije vraćaju numeričku vrijednost. Aritmetičke funkcije se mogu dodatno podijeliti na algebarske (poput `ABS`, `CEIL`, `FLOOR`, `MOD`, `POWER`, `SQRT`, itd.), eksponencijalne i logaritamske (poput `EXP`, `LOG`, `LN`, `LOG2`, `LOG10`, itd.), trigonometrijske (poput `SIN`, `COS`, `TAN`, `ASIN`, `ACOS`, `ATAN`, `ATAN2`, itd.) i funkcije za pretvaranje između različitih brojnih sistema.

Datumske i vremenske funkcije - koriste se za manipulaciju datumskim i vremenskim vrijednostima. U bazama podataka vrijednosti ovih tipova podataka su vrlo česte te se ove funkcije vrlo često koriste u upitima. U Tabeli 3 su navedne neke od najčešće korištenih funkcija u MySQL iz ove kategorije.

Tabela 3 Najčešće korištene funkcije u MySQL-u

Funkcija	Opis
`ADDDATE(), DATE_ADD()`	Dodaje vremenske vrijednosti (intervale) na datum
`CURDATE(), CURRENT_DATE`	Vraća trenutni datum
`CURTIME(), CURRENT_TIME`	Vraća trenutno vrijeme
`DATE()`	Izdvaja datum iz `date` ili `datetime` izraza
`SUBDATE(), DATE_SUB()`	Oduzima vremensku vrijednost (interval) od datuma
`DATEDIFF()`	Oduzima dva datuma
`DAY(), DAOFMONTH()`	Vraća dan u mjesecu (1-31)
`DAYNAME()`	Vraća naziv dana u sedmici
`DAYOFWEEK()`	Vraća indeks dana u sedmici za `date` izraz (1 = nedjelja, ..., 7 = subota)
`WEEKDAY()`	Isto kao i `DAYOFWEEK()` ali 0 = ponedjeljak, ..., 6 = nedjelja
`DAYOFYEAR()`	Vraća dan u godini (1-366)
`EXTRACT()`	Izdavaja dio datuma
`HOUR()`	Vraća sate
`NOW(), LOCALTIME`	Vraća trenutni datum i vrijeme
`MAKEDATE()`	Kreira datum od godine i dana u godini
`MAKETIME()`	Kreira vrijeme od sata, minuta i sekundi
`MINUTE()`	Vraća minute
`MONTH()`	Vraća mjesec proslijeđenog datuma
`SECOND()`	Vraća sekunde (0-59)
`STR_TO_DATE()`	Konvertuje string u datum
`SYSDATE()`	Vraća vrijeme u kojem se funkcija izvršava
`TIME()`	Izdvaja vrijeme iz proslijeđenog izraza
`TIMEDIFF()`	Oduzima vremena
`WEEK()`	Vraća broj sedmice
`YEAR()`	Vraća godinu

Funkcije za rad sa stringovima ili karakterima - služe za manipulaciju karakterima ili nizovima karaktera te postoji veliki skup takvih funkcija koji ovisi od konkretnog SUBP-a, npr. `LOWER`, `UPPER`, `TRIM`, `CONCAT`, `SUBSTRING`, `STRCMP`, `LENGTH`, itd. Većina ovih funkcija imaju istu namjenu i način upotrebe kao i funkcije sa istim ili sličnim nazivima u drugim programskim jezicima.

Agregatne funkcije - izračunavaju jednu vrijednost na osnovu vrijednosti iz jednog ili više zapisa te će biti detaljnije razmatrane u posebnom poglavlju.

Funkcije za kontrolu toka - posebna kategorija u koju u MySQL-u spadaju funkcije `CASE`, `IF`, `IFNULL` i `NULLIF`. `CASE` funkcija se može koristiti u dva oblika:

```
CASE value WHEN [compare_value] THEN result [WHEN ...] [ELSE result] END
```

```
CASE WHEN [condition] THEN result [WHEN ...] [ELSE result] END
```

Prvi oblik vraća rezultat (`result`) gdje je `value = compare_value`. Drugi oblik vraća rezultat (`result`) za prvi uslov pod `WHEN` koji je istinit (*true*). Ukoliko ne postoji podudaranje niti sa jednim `WHEN`, vraća se rezultat nakon `ELSE` ili `NULL` ukoliko se `ELSE` ne navede.

`IF``(`*`expr1, expr2, expr3`*`)` vraća *`expr2`* ukoliko je *`expr1`* istinit (*true*); inače vraća *`expr3`*.

`IFNULL``(`*`expr1, expr2`*`)` vraća *`expr1`* ukoliko *`expr1`* nije `NULL`; inače vraća *`expr2`*.

`NULLIF``(`*`expr1, expr2`*`)` vraća `NULL` ako je `expr1 = expr2` istinito (*true*); inače vraća *`expr1`*.

WHERE segment SELECT naredbe

Nakon što smo u prethodnom poglavlju detaljno predstavili izraze i način njihovog kreiranja, u nastavku možemo predstaviti na koji način se izrazi mogu ugraditi u `WHERE` dio `SELECT` naredbe. U `SELECT` naredbi, `WHERE` segment je opcioni i ukoliko se izostavi dohvataju se vrijednosti atributa navedenih u listi za selekciju iz svih zapisa u relacijama navedenih u `FROM` dijelu naredbe. U `WHERE` dijelu naredbe se definiše uslov dohvata (logički izraz) koji se može sastojati od više uslova međusobno povezanih logičkim operatorima. Uslovi od kojih se sastoji uslov dohvata mogu biti uslovi poređenja ili uslovi sa podupitima koji će biti obrađeni nešto kasnije. Što se tiče uslova poređenja koji se mogu koristiti u uslovima dohvata u `WHERE` dijelu naredbe, mogu se podijeliti u 5 različitih oblika:

1. ***poređenje*** - poređenje dva izraza (*expression*) sa nekim od relacijskih operatora

Primjer*: Ispisati šifru člana i inventarni broj knjige, za sve posudbe koje nisu vraćene na vrijeme.*

```
SELECT sifClan, invBroj FROM posudba
   WHERE datumDo > datumVrac
```

2. ***filtriranje opsega*** - ispitivanje da li je izraz unutar (ili izvan) intervala omeđenog sa druga dva izraza korištenjem operatora `BETWEEN ... AND ...` (ili `NOT BETWEEN ... AND ...`)

***Primjer**: Ispisati šifre i nazive knjiga izdatih u proteklih 10 godina*

```
SELECT sifKnjiga, naslov FROM knjiga
   WHERE godIzdanja BETWEEN YEAR(CURRENT_DATE) - 10 AND YEAR(CURRENT_DATE)
```

***Primjer**: Ispisati skraćenicu, naziv i broj ECTS kredita za predmete koji imaju više od 5 ili manje od 3 ECTS kredita*

```
SELECT skrPred, nazPred, brojECTS FROM pred
   WHERE brojECTS NOT BETWEEN 3 AND 5
```

3. ***filtriranje liste*** - ispitivanje da li je izraz element nekog skupa (ili nije element skupa) koji može sadržavati literale korištenjem operatora `IN` (ili `NOT IN`)

***Primjer**: Ispisati šifru, ime i prezime članova koji su rođeni u jednom od mjesta sa poštanskim brojevima 71000, 76000 ili 88000.*

```
SELECT sifClan, imeClan, prezClan FROM clan
   WHERE pbrRod IN (71000, 76000, 88000)
```

***Primjer**: Ispisati poštanske brojeve i nazive mjesta koji ne pripadaju kantonima sa šifrom 2, 4, 6 ili 8*

```
SELECT pbr, nazMjesto FROM mjesto
  WHERE sifKanton NOT IN (2, 4, 6, 8)
```

4. ***NULL testiranje*** - ispitivanje da li je vrijednost atributa `NULL` (ili nije `NULL`) korištenjem operatora `IS NULL` (ili `IS NOT NUL`)

***Primjer**: Ispisati sve podatke o posudbama kojima je istekao rok a još nisu vraćene (nije im unesen `datumVrac`)*

```
SELECT * FROM posudba
   WHERE datumDo < CURRENT_DATE
   AND datumVrac IS NULL
```

***Primjer**: Ispisati ime, prezime i datum rođenja članova za koje je poznat podataka o jedinstvenom matičnom broju.*

```
SELECT imeClan, prezClan, datRod FROM clan
   WHERE jmbgClan IS NOT NULL
```

5. ***Podudaranje sa uzorkom znakova*** - ispitivanje zadovoljava li ili ne zadovoljava vrijednost atributa zadati uzorak (*pattern*) korištenjem operatora `LIKE` i `RLIKE` (ili `NOT LIKE` i `NOT RLIKE`)

Prilikom korištenja `LIKE` operatora mogu se koristiti sljedeći *wildcard* znakovi:

- `%` - zamjenjujebilo koju kombinaciju karaktera, uključujući i nula karaktera
- `_` - zamjenjuje tačno jedan karakter
- `\` - *escape* karakter, ukida specijalno značenje *wildcard* znakova. U MySQL-u se može koristiti `ESCAPE` dio kako bi se definisao drugi *escape* karakter.

***Primjer**: Ispis naslova i godina izdanja svih knjiga koje u naslovu sadrže riječ 'zbirka'*

```
SELECT naslov, godIzdanja FROM knjiga
   WHERE naslov LIKE '%zbirka%'
```

`LIKE` je standardni SQL operator za poređenje stringova sa zadatim uzorkom znakova. Međutim, ukoliko je potrebno koristiti poređenje više uzoraka znakova ili složenijih uzoraka znakova, `LIKE` operator pokazuje određene nedostatke i takvi upiti korištenjem samo `LIKE` operatora mogu biti izuzetno komplikovani ili jednostavno nepraktični za pisati. Zbog toga većina SUBP-a obezbjeđuje dodatne operatore za poređenje stringova sa zadatim uzorkom znakova korištenjem regularnih izraza (*regular expressions*). U slučaju MySQL-a radi se o operatorima `RLIKE` i `REGEXP` koji su sinonimi jedan drugome. Specijalni karakteri i konstrukcije koje mogu biti korištene u MySQL-u za operacije nad regularnim izrazima korištenjem `RLIKE` operatora su:

- `^`- označava početak stringa, npr. `'^ab'` je string koji počinje sa `ab`
- $ - označava kraj stringa, npr. `'ić$'` je string koji završava na `ić`
- `.` - mijenja bilo koji karakter
- `a*` - odgovara bilo kojoj sekvenci od nula ili više `a` karaktera
- `a+` - odgovara bilo kojoj sekvenci od jednog ili više `a` karaktera
- `a?` - odgovara nula ili jednom `a` karakteru
- `ab|cde` - odgovara sekvencama `ab` ili `cde`
- `(abc)*` - odgovara jednoj ili više sekvenci `abc`
- `{n}`, `{m,n}` - opštiji način zapisivanja višestrukog pojavljivanja prethodno navedenog uzorka. `a{n}` odgovara tačno `n` pojavljivanja od `a`, `a{n,}` odgovara `n` ili više pojavljivanja od `a`, dok `a{m,n}` odgovara `m` do `n` pojavljivanja od `a`
- `[a-dX]`,`[^a-dX]` - odgovara bilo kojem karakteru koji je (ili koji nije ako se navede `^`) `a`, `b`, `c`, `d` ili `X`. Karakter (-) služi da se definira opseg karaktera pa tako `[0-9]` zamjenjuje bilo koju cifru

***Primjer**: Ispisati ime, prezime, datum rođenja i jedinstveni matični broj članova čije ime počinje samoglasnikom a prezime im se ne završava na `'ić'`.*

```
SELECT imeClan, prezClan, datRod, jmbgClan FROM CLAN
  WHERE imeClan RLIKE '^[AEIOU]'
  AND prezClan NOT RLIKE 'ić$'
```

Sortiranje rezultata

Rezultat `SELECT` naredbe se može sortirati prema atributima ili izrazima iz liste za selekciju korištenjem `ORDER BY` dijela naredbe čiji je opšti oblik:

```
ORDER BY {col_name | expr | alias | position} [ASC | DESC] [, ...]
```

U `ORDER BY` dijelu se navode atributi ili izrazi prema kojima se vrši sortiranje te opcionalno uz svaki atribut ili izraz i smjer sortiranja. Ako se navede ključna riječ `ASC` (*ascending*) znači da će se sortiranje po atributu ili izrazu uz koji je navedena izvršiti uzlazno (od manje vrijednosti ka većoj), dok će se sortiranje po atributu ili izrazu uz koji je navedena ključna riječ `DESC` (*descending*) izvršiti silazno (od veće vrijednosti ka manjoj). S obzirom da je navođenje smjera sortiranja opciono, ukoliko se ove ključne riječi izostave sortiranje će se po svakom atributu ili izrazu obaviti uzlazno.

Primjer: *Ispisati imena, prezimena i datume rođenja članova biblioteke rođenih nakon 1992. godine od najmlađeg prema najstarijem. Ako su članovi rođeni isti dan sortirati ih po prezimenu i imenu.*

```
SELECT imeClan, prezClan, datRod FROM clan
  WHERE YEAR(datRod) > 1992
  ORDER BY datRod DESC, prezClan ASC, imeClan ASC
```

Kao što se može vidjeti iz opšteg oblika `ORDER BY` dijela, umjesto naziva atributa se mogu koristiti i zamjenski nazivi atributa (*alias*) i redni brojevi atributa kako su navedeni u listi za selekciju čime se skraćuje pisanje upita, te bi se u prethodnom upitu `ORDER BY` dio mogao napisati i ovako:

```
ORDER BY 3 DESC, 2 ASC, 1 ASC
```

Ili još kraće ukoliko se za uzlozno sortiranje izostavi ključna riječ `ASC`:

```
ORDER BY 3 DESC, 2, 1
```

Sortiranje rezultata upita se može obaviti samo prema atributima ili izrazima koji su navedeni u selekciju. Upotreba drugih atributa ili izraza u `ORDER BY` dijelu nije dozvoljena.

Primjeri upita za vježbu

Q1: *Ispis šifri, skraćenica, naziva i broja ECTS kredita za sve predmete.*

```
SELECT sifPred, skrPred, nazPred, brojECTS FROM pred
```

Q2: *Ispis svih godina u kojima je izdana bar jedna knjiga (svaka godina se u listi treba pojaviti samo jednom, bez obzira koliko knjiga je izdano te godine).*

```
SELECT DISTINCT godIzdanja FROM knjiga
```

Q3: *Ispis svih inventarnih brojeva i stanja primjeraka za knjigu sa šifrom 1234 koji su trenutno u upotrebi.*

```
SELECT invBroj, stanje FROM primjerak
   WHERE sifKnjiga = 1234 AND koristenje
```

Q4: *Ispis naslova i procjene knjige koja se izračunava tako da se godina izdanja podijeli sa vrijednošću knjige koja je prethodno uvećana za 0.5. Ispisati samo one knjige čija je procjena između 200 i 500.*

```
SELECT naslov, godIzdanja / (vrijednost + 0.5) AS procjena FROM knjiga
   WHERE godIzdanja / (vrijednost + 0.5) BETWEEN 200 AND 500
```

Q5: Ispisati naziv predmeta (samo prvih 25 karaktera) te skraćenicu i broj ECTS kredita spojene crticom (npr. 'MAT1-6') za sve predmete koji pripadaju organizacionoj jedinici sa šifrom 1001 i imaju više od 2 ECTS kredita.

```
SELECT SUBSTRING(nazPred, 1, 25), CONCAT(skrPred, '-', brojECTS) FROM pred
   WHERE sifOrgjed = 1001 AND brojECTS > 2
```

Q6: *Ispisati ime i prezime člana, datum rođenja i broj dana koji je protekao od rođenja za svakog člana koji je rođen bilo koje nedjelje u maju mjesecu bilo koje godine. Ispis sortirati od najmlađeg ka najstarijem članu, a članove koji su rođeni isti dan poredati po prezimenu i imenu*

```
SELECT imeClan, prezClan, datRod, DATEDIFF(CURRENT_DATE, datRod) FROM clan
   WHERE WEEKDAY(datRod) = 6 AND MONTH(datRod) = 5
   ORDER BY datRod DESC, prezClan, imeClan
```

Q7: *Ispisati inventarni broj, šifru člana, datum posudbe i datum na koji se navršilo tačno 3 godine od posudbe primjerka. Ispisati samo posudbe za koje je taj dan već prošao.*

```
SELECT invBroj, sifKnjiga, datumPos, ADDDATE(datumPos, INTERVAL 3 YEAR)
   FROM posudba
   WHERE ADDDATE(datumPos, INTERVAL 3 YEAR) < CURRENT_DATE
```

Q8: *Ispisati ime i prezime člana, datum rođenja i broj dana koliko je imao na dan osnivanja biblioteke ako je biblioteka osnovana 01.03.1998. godine. Ispisati samo podatke za članove koji su tada bili mlađi od 18 godina, a ispis sortirati prema prezimenu i imenu člana.*

```
SELECT imeClan, prezClan, datRod,
       STR_TO_DATE('1,3,1998', '%d,%m,%Y') - datRod
   FROM clan
   WHERE DATE_ADD(datRod, INTERVAL 18 YEAR) >
         STR_TO_DATE('1,3,1998', '%d,%m,%Y')
   ORDER BY 2, 1
```

Q9: *Ispisati šifre organizacionih jedinica u opadajućem redoslijedukojima pripadaju predmeti čiji naziv počinje slovom E.*

```
SELECT DISTINCT sifOrgjed FROM pred
   WHERE nazPred LIKE 'E%'
   ORDER BY 1 DESC
```

Q10: *Ispisati sve podatke o knjigama izdate prije više od 30 godina čiji naslovi počinju slovom K ili slovom M ili slovom N ili slovom T. Upit najprije napisati upotrebom operatora LIKE, a potom upotrebom operatora RLIKE.*

```
SELECT * FROM knjiga
   WHERE (naslov LIKE 'K%' OR naslov LIKE 'M%' OR naslov LIKE 'N%'
          OR naslov LIKE 'T%')
   AND godIzdanja < YEAR(CURRENT_DATE) - 30
```

```
SELECT * FROM knjiga
   WHERE naslov RLIKE '^[KMNT]'
   AND godIzdanja < YEAR(CURRENT_DATE) - 30
```

Q11: *Ispisati sve podatke o predmetima čiji naziv počinje a ne završava samoglasnikom.*

```
SELECT * FROM pred
   WHERE nazPred RLIKE '^[AEIOU]' AND nazPred RLIKE '[^aeiou]$'
```

Q12: *Ispisati naslove i godine izdanja knjiga koje u svojim naslovima sadrže riječ 'baza' ili 'java'. Koristiti operator `RLIKE`.*

```
SELECT naslov, godIzdanja FROM knjiga
   WHERE naslov RLIKE 'baza|java'
```

Q13: *Ispisati poštanske brojeve i nazive mjesta za sva mjesta kojima se u nazivu bar dva puta pojavljuje kombinacija 'ar'.*

```
SELECT pbr, nazMjesto FROM mjesto
   WHERE nazMjesto RLIKE '(ar){2,}'
```

***Q14**: Ispisati sve podatke o autorima čije ime ne počinje slovima od A do F, a prezime završava sa 'ić'.*

```
SELECT * FROM autor
   WHERE imaAutor RLIKE '^[^A-F]' AND prezAutor RLIKE 'ić$'
```

Agragatne funkcije

U poglavlju o SQL funkcijama već je naglašeno da se agregatne funkcije primjenjuju nad skupovima vrijednosti, odnosno da kao rezultat daju jednu vrijednost na osnovi vrijednosti atributa ili izraza iz jedne ili više n-troki. Za sve agregatne funkcije bi se mogla predstaviti opšta sintaksa:

```
aggregate_function([ALL | DISTINCT] expression)
```

pri čemu se za *`aggregate_function`* može koristiti jedna od sljedećih agregatnih funkcija:

`COUNT` vraća broj n-troki u kojima vrijednost atributa ili izraza nije `NULL`. Poseban oblik u odnosu na ostale agregatne funkcije je `COUNT(*)` koji vraća broj n-torki koje odgovaraju uslovu dohvata bez obzira da li su u njima sadržane `NULL` vrijednosti.

SUM vraća sumu vrijednosti atributa ili izraza.

AVG vraća prosječnu vrijednost atributa ili izraza.

MAX vraća najveću odnosno maksimalnu vrijednost atributa ili izraza.

MIN vraća najmanju odnosno minimalnu vrijednost atributa ili izraza.

Ukoliko nema n-torki koji odgovaraju uslovu dohvata svaka od navedenih agregatnih funkcija vraća `NULL` osim `COUNT` funkcije koja vraća 0. Kvalifikator `DISTINCT` ima istu namjenu kao u listi za selekciju te ukoliko se navede agragatne funkcije prilikom izračunavanja u obzir uzimaju samo različite vrijednosti atributa ili izraza.

Osim navedenih agragatnih funkcija postoje i standardne SQL agregatne funkcije za izračunavanje standardne devijacije i variance za vrijednost atributa ili izraza (`STDEV_POP` i `VAR_POP` u MySQL-u), te dodatne agregatne funkcije u MySQL-u koje izvode operacije nad bitima (`BIT_AND`, `BIT_OR` i `BIT_XOR`) i spajanje svih string vrijednosti atributa ili izraza koje nisu `NULL` (`GROUP_CONCAT`).

Agragatne funkcije se u upitima mogu koristiti na dva načina:

- Agregatne funkcije se primjenjuju na sve selektovane n-torke, odnosno n-torke koje zadovoljavaju uslov selekcije (jednostavniji slučaj). Kao rezultat u ovom slučaju se dobija samo jedna n-torka koja ima onoliko kolona (vrijednosti) koliko je agregatnih funkcija navedeno u listi za selekciju

- Selektovane n-torke se na osnovu određenog kriterija podijele u grupe, a potom se agregatne funkcije primjenjuju na svaku od kreiranih grupa (složeniji slučaj). U ovom slučaju se kao rezultat dobija onoliko n-torki koliko je formirano različitih grupa, a svaka n-torka ponovo ima onoliko vrijednosti koliko je agregatnih funkcija navedeno u listi za selekciju. Ovaj složeniji slučaj će biti posebno razmatran u poglavlju o grupisanju rezultata.

***Primjer**: Ispisati broj posudbi obavljenih u toku tekućeg mjeseca.*

```
SELECT COUNT(*) FROM posudba
   WHERE MONTH(datumPos) = MONTH(CURRENT_DATE)
   AND YEAR(datumPos) = YEAR(CURRENT_DATE)
```

***Primjer**: Ispisati u koliko različitih mjesta su rođeni članovi koji stanuju u mjestu sa poštanskim brojem 75000.*

```
SELECT COUNT(DISTINCT pbrRod) FROM clan
   WHERE pbrStan = 75000
```

***Primjer**: Izračunati broj nastavnika i njihov prosječni koeficijent za platu.*

```
SELECT COUNT(sifNast), AVG(koef) FROM nast
```

***Primjer**: Kolika je razlika u godinama između najstarije i najmlađe izdate knjige.*

```
SELECT MAX(godIzdanja) - MIN(godIzdanja) FROM knjiga
```

Prilikom korištenja agregatnih funkcija vrlo bitno je na umu imati sljedeća ograničenja:

1. Ukoliko se agregatne funkcije koriste u jednostavnijem obliku, odnosno ne koristi se grupisanje za kreiranje grupa n-torki, u listi za selekciju se pored agregatnih funkcija ne smiju pojavljivati drugi atributi i izrazi.

Npr. sljedeći upit:

```
SELECT sifOrgjed, AVG(koef) FROM nast
```

nije ispravan jer agregatna funkcija `AVG` izračunava samo jednu prosječnu vrijednost koeficijenta za platu na osnovu vrijednosti atributa `koef` u svim n-torkama relacije `nast`, dok u relaciji postoji više različitih vrijednosti atributa `sifOrgjed`.

Napomena: U MySQL-u gornji upit neće proizvesti grešku i kao rezultat će vratiti jednu n-torku na način da uz vrijednost agregatne funkcije ispiše vrijednost atributa `sifOrgjed` koju prvu dohvati iz relacije. Međutim, takav rezultat nema smisla te ovakve upite ne treba izvoditi ni u kojem slučaju.

2. Agregatne funkcije se zasebno ne mogu koristiti u WHERE dijelu SELECT naredbe, odnosno u uslovima dohvata.

Npr. sljedeći upit, kojim se pokušavaju dohvatiti podaci o nastavnicima čiji je koeficijent za platu veći od prosječnog koeficijenta za platu svih nastavnika:

```
SELECT * FROM nast
   WHERE koef > AVG(koef)
```

nije ispravan, jer se agregatna funkcija na ovaj način ne može koristiti u `WHERE` dijelu `SELECT` naredbe. Ispravno rješenje ovakvih upita kojima se pokušavaju porediti vrijednosti dobijene primjenom agregatnih funkcija biće predstavljeno u posebnom poglavlju o upotrebi podupita.

Primjeri upita za vježbu

Q1: *Ispisati koliko različitih jedinica pripadaju predmeti koji imaju 5 ili 6 ECTS kredita, a naziv im sadrži riječ 'tehnike'.*

```
SELECT COUNT(DISTINCT sifOrgjed) FROM pred
   WHERE brojECTS IN (5, 6) AND nazPred LIKE '%tehnike%'
```

Q2: *Ispisati prosječnu vrijednost knjiga koje su izdate prije 2000. godine a naslov im počinje samoglasnikom.*

```
SELECT AVG(vrijednost) FROM knjiga
   WHERE godIzdanja < 2000 AND naslov RLIKE '^[AEIOU]'
```

Q3: *Ispisati datume rođenja najmlađeg i najstarijeg člana koji stanuju u mjestu sa poštanskim brojem 75000.*

```
SELECT MAX(datRod), MIN(datRod) FROM clan
   WHERE pbrStan = 75000
```

Q4: *Ispisati broj dana koji je protekao između prve i posljednje obavljene posudbe u prethodnoj godini.*

```
SELECT DATEDIFF(MAX(datumPos), MIN(datumPos)) FROM posudba
   WHERE YEAR(datumPos) = YEAR(CURRENT_DATE) - 1
```

Q5: *Ispisati prosječnu starost u godinama za sve knjige čiji naslovi počinju slovima R do Z i završavaju samoglasnikom.*

```
SELECT AVG(YEAR(CURRENT_DATE) - godIzdanja) FROM knjiga
   WHERE naslov RLIKE '^[R-Z]' AND naslov RLIKE '[aeiou]$'
```

Q6: *Ispisati koliko puta je zadužen i koliko ukupno dana je trajalo zaduženje primjerka sa inventarnim brojem 273 u prvih šest mjeseci tekuće godine.*

```
SELECT COUNT(*), SUM(DATEDIFF(datumVrac, datumPos)) FROM posudba
   WHERE invBroj = 273 AND YEAR(datumPos) = YEAR(CURRENT_DATE)
   AND MONTH(datumPos) BETWEEN 1 AND 6
```

Q7: *Ispisati koliko dana je najduže trajala neka posudba primjerka koji nije vraćen na vrijeme u proteklih 5 godina.*

```
SELECT MAX(DATEDIFF(datumVrac, datumPos)) FROM posudba
   WHERE datumVrac > datumDo
   AND YEAR(datumPos) >= YEAR(CURRENT_DATE) - 5
```

Q8: *Ispisati prosjek godina starosti članova biblioteke koji stanuju u jednom od mjesta sa poštanskim brojevima 75260, 75270 i 75300 i prezime im sadrži bar dva slova 'm'.*

```
SELECT AVG(YEAR(CURRENT_DATE) - YEAR(datRod)) FROM clan
   WHERE pbrStan IN (75260, 75270, 75300)
   AND prezClan RLIKE 'm{2, }'
```

Spajanje relacija

U svakom do sada navedenom primjeru `SELECT` naredbe korišteni su podaci iz jedne tabele, odnosno u njenom `FROM` dijelu je navedena samo jedna relacija. Međutim, ukoliko dohvatamo podatke o mjestima, umjesto šifre kantona puno značajnije bi bilo ispisati njegov naziv ili ako dohvatamo podatke o posudbama knjiga, ime i prezime člana i naziv knjige imaju puno više smisla od šifre i inventarnog broja. Da bi ovakav dohvat podataka bio moguć u `FROM` dijelu `SELECT` naredbe potrebno je navesti više od jedne relacije, odnosno svaku od relacija u kojoj su pohranjeni podaci koji se žele dohvatiti. U praksi je upravo u većini situacija slučaj da se u upitima koriste podaci iz više relacija te je potrebno izvršiti spajanje relacija.

Spajanje relacija u SQL-u se može podijeliti u dvije skupine - EQUI JOIN i NON EQUI JOIN.

- SQL EQUI JOIN ili spajanje sa izjednačavanjem je primjer spajanja relacija u kojem se u uslovu spajanja kao operator poređenja koristi znak jednakosti (=). Najčešći primjer ove vrste spajanja relacija je prirodno spajanje, a u opštem slučaju se EQUI JOIN može podijeliti na dva tipa - unutarnje (*inner*) i vanjsko (*outer*) spajanje.

- SQL NON EQUI JOIN ili spajanje uz uslov je primjer spajanja relacija u kojem se u uslovu spajanja koristi bilo koji drugi operator poređenja poput <, >=, >, >=

Opšti oblik sintakse `FROM` dijela `SELECT` naredbe (`table_references`) u pojednostavljenom obliku:

```
table_references:
    table1 join_type table2 [ON (join_condition)]

join_type:
    [INNER | CROSS] JOIN
  | STRAIGHT_JOIN
  | {LEFT | RIGHT} [OUTER] JOIN
  | NATURAL [{LEFT | RIGHT} [OUTER]] JOIN

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

Dekartov proizvod relacija

Neka imamo sljedeće relacije:

MJESTO

pbr	nazMjesto	sifKanton
75000	Tuzla	3
75300	Lukavac	3
76120	Brčko	12

KANTON

sifKanton	nazKanton
3	Tuzlanski kanton
12	Brčko distrikt

Sljedeći upit:

```
SELECT * FROM mjesto, kanton
```

kao rezultat daje:

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
75000	Tuzla	3	12	Brčko distrikt
75300	Lukavac	3	3	Tuzlanski kanton
75300	Lukavac	3	12	Brčko distrikt
76120	Brčko	12	3	Tuzlanski kanton
76120	Brčko	12	12	Brčko distrikt

Prethodna `SELECT` naredba je primjer za operaciju relacijske algebre Dekartov proizvod, `mjesto` × `kanton`. Znak * koji je naveden u listi za selekciju zamjenjuje sve atribute relacija koje su navedene u `FROM` dijelu naredbe, a kako je izostavljen kriterij po kojem će se zapisi iz relacije mjesto "upariti" sa zapisima iz relacije kanton (uslov spajanja), doći će do spajanja

svake n-torke iz prve relacije sa svakom n-torkom druge relacije. Osim toga, nije definiran niti uslov dohvata u `WHERE` dijelu naredbe pa se pojavljuju sve na taj način spojene n-torke.

Operacija Dekartovog proizvoda ima vrlo rijetku ili nikakvu primjenu u praksi i najčešće se pojavljuje kao greška programera, kao u slučaju kada se zaboravi navesti uslov spajanja relacija. U takvom slučaju dolazi do bespotrebnog trošenja računarskih resursa jer za relacije koje recimo imaju po 10000 zapisa, rezultat ovakvog upita bi sadržavao 10^8 zapisa koji je pri tome još sasvim beskoristan.

Dekartov proizvod relacija se osim gore navedenom naredbom dobija i upotrebom bilo kojeg tipa spajanja `JOIN`, `INNER JOIN` ili `CROSS JOIN` bez navođenja uslova spajanja relacija. U MySQL-u su ova tri tipa spajanja sintaksno jednaka i mogu zamijeniti jedan drugog, dok u standardnom SQL-u nisu i `CROSS JOIN` se koristi bez `ON` dijela (bez navođenja uslova dohvata).

Prirodno spajanje relacija

Kako je prethodno istaknuto, prirodno spajanje je najčešći oblik spajanja relacija u upitima. Prema definiciji, prirodno spajanje se obavlja na temelju jednakih vrijednosti istoimenih atributa. U suštini, najčešće se radi o spajanju relacija izjedančavanjem vrijednosti atributa koji u jednoj relaciji čini primarni ključ, dok u drugoj relaciji predstavlja strani ključ koji referencira na prvu relaciju. Da bi prirodno spajanje relacija odgovaralo strogoj definiciji prirodnog spajanja, osim navođenja uslova spajanja, u listi za selekciju oni atributi koji se pojavljuju u obje relacije navode se samo jednom.

Ukoliko želimo obaviti operaciju prirodnog spajanja relacija `mjesto` i `kanton` (mjesto kanton), koristićemo sljedeću naredbu:

```
SELECT mjesto.*, nazKanton
  FROM mjesto INNER JOIN kanton
  ON mjesto.sifKanton = kanton.sifKanton
```

i kao rezultat bi dobili:

pbr	nazMjesto	sifKanton	nazKanton
75000	Tuzla	3	Tuzlanski kanton
75300	Lukavac	3	Tuzlanski kanton
76120	Brčko	12	Brčko distrikt

Isti rezultat bi dobili ukoliko bi koristili `JOIN` tip spajanja. Prirodno spajanje relacija u MySQL-u možemo dobiti i sljedećim naredbama:

```
SELECT * FROM mjesto INNER JOIN kanton USING (sifKanton)
```

```
SELECT * FROM mjesto NATURAL JOIN kanton
```

Kao što možemo vidjeti kod ova dva načina spajanja MySQL odmah primjenjuje definiciju prirodnog spajanja relacija te iako se u listi za selekciju navodi znak *, u rezultatu se atribut koji postoji u obje relacije pojavljuje samo jednom. U odnosu na `INNER JOIN` i `JOIN` tip spajanja u rezultatu se najprije navodi zajednički atribut, a potom ostali atributi

prirodno spojenih relacija. Međutim, ova dva načina prirodnog spajanja relacija se mogu primijeniti jedino u slučaju kada relacije koje se spajaju sadrže istoimene atribute.

***Primjer**: Ispisati imena i prezimena članova biblioteke te poštanske brojeve i nazive njihovih mjesta rođenja.*

```
SELECT imeClan, prezClan, pbrRod, nazMjesto
  FROM clan INNER JOIN mjesto USING (pbrRod)
```

```
SELECT imeClan, prezClan, pbrRod, nazMjesto
  FROM clan NATURAL  JOIN mjesto
```

Niti jedan od prethodna dva upita u MySQL-u nije sintaksno ispravan jer relacije `clan` i `mjesto` ne sadrže istoimeni atribut iako se mogu prirodno spojiti preko atributa `pbrRod` koji je strani ključ u relaciji `clan` koji referira na atribut `pbr` kao primarni ključ relacije `mjesto`. Rješenje se dobija upitom:

```
SELECT imeClan, prezClan, pbrRod, nazMjesto
  FROM clan INNER JOIN mjesto ON pbrRod = pbr
```

Prirodno spajanje relacija može se dobiti i tipom spajanja sa zarezom (,) pri čemu se uslov spajanja navodi u `WHERE` dijelu `SELECT` naredbe:

```
SELECT imeClan, prezClan, pbrRod, nazMjesto
  FROM clan, mjesto
  WHERE clan.pbrRod = mjesto.pbr
```

Međutim, ovakav način spajanja nije preporučljivo koristiti jer smanjuje čitljivost i razumjevanje napisanih upita s obzirom da se u WHERE dijelu naredbe osim uslova dohvata navode i uslovi spajanja relacija. Zbog toga, ukoliko je u upitu potrebno obaviti spajanje relacija, to se izvodi u FROM dijelu SELECT naredbe izborom odgovarajućeg tipa spajanja i navođenjem uslova spajanja po potrebi, dok se u WHERE dijelu naredbe isključivo navode uslovi za dohvat n-torki. Sljedećim upitom ilustrira se mogućnost spajanja više relacija uz dodavanje uslova dohvata.

***Primjer**: Ispis poštanskog broja, naziva mjesta i naziva kantona svih mjesta koja se nalaze u kantonima čiji nazivi počinju slovom 'T'.*

```
SELECT pbr, nazMjesto, nazKanton
  FROM mjesto INNER JOIN kanton
  ON mjesto.sifKanton = kanton.sifKanton
  WHERE nazKanton LIKE 'T%'
```

pbr	nazMjesto	nazKanton
75000	Tuzla	Tuzlanski kanton
75300	Lukavac	Tuzlanski kanton

Spajanja relacija između kojih postoji "paralelna" veza

U relaciji `clan` nalaze se atributi koji opisuju mjesto stanovanja i mjesto rođenja člana:

ČLAN

sifClan	imeClan	prezClan	pbrRod	pbrStan	datRod	jmbgClan
1001	Mensur	Kasumović	75000	75300		
1002	Student	Studentić	76120	75000		
1003	Marina	Pejić	75000	75000		

Ukoliko se žele dohvatiti podaci o nazivima mjesta stanovanja svih članova biblioteke upit se jednostavno postavlja na način kako je prethodno opisano kod spajanja relacija:

```
SELECT imeClan, prezClan, nazMjesto
  FROM clan INNER JOIN mjesto ON pbrStan = pbr
```

Kao što se može primijetiti nazivi atributa preko kojih se vrši spajanje relacija su jednoznačno određeni te se u ovom upitu nisu morali kvalificirati imenima relacija kojima pripadaju. Međutim, ukoliko se žele uz nazive mjesta stanovanja ujedno ispisati i nazivi mjesta rođenja članova, upit postaje nešto složeniji. Ukoliko se primijeni do sada usvojeno znanje o spajanju relacija moglo bi se pretpostaviti da sljedeći upit predstavlja rješenje:

```
SELECT imeClan, prezClan, nazMjesto nazMjestoStan, nazMjesto nazMjestoRod
  FROM clan INNER JOIN mjesto
  ON pbrStan = pbr AND pbrRod = pbr
```

Međutim, prethodni upit je neispravno postavljen i rezultat koji se dobija ovim upitom ne vrijedi jer će se u rezultatu pojaviti samo oni članovi biblioteke kojima su mjesto rođenja i mjesto stanovanja jednaki (za jednu vrijednost poštanskog broja n-torke iz relacije `mjesto` gornji uslov spajanja će biti ispunjen samo ako je ta vrijednost jednaka i poštanskom broju rođenja i poštanskom broju stanovanja n-torke iz relacije `clan`). Kao moguće rješenje nameće se dvostruko spajanje sa relacijom `mjesto`, prvi put preko poštanskog broja rođenja, a drugi put preko poštanskog broja stanovanja:

```
SELECT imeClan, prezClan, nazMjesto nazMjestoStan, nazMjesto nazMjestoRod
  FROM clan INNER JOIN mjesto ON pbrStan = pbr
  INNER JOIN mjesto ON pbrRod = pbr
```

Međutim, prethodno postvljen upit sintaksno nije ispravan jer naziv relacije `mjesto` u upitu nije jednoznačno određen. Rješenje bi bilo očigledno da u bazi podataka postoje dvije relacije sa podacima o mjestima rođenja i mjestima stanovanja (npr. `mjestoRod` i `mjestoStan`), ali kako one ne postoje očito je da jedna relacija `mjesto` u istom upitu mora preuzeti njihove uloge, odnosno jednom se pojaviti u ulozi mjesta stanovanja, a drugi put u ulozi mjesta rođenja. Kako bi se jednoj relaciji u istom upitu dodijelile dvije uloge uvode se alternativni ili zamjenski nazivi relacija (*alias*) i to njihovim navođenjem iza originalnog naziva relacije slično kao sa *alias* nazivima atributa u listi za selekciju. Međutim, za razliku od alternativnih naziva atributa koji se ne mogu koristiti u upitima (osim u `ORDER BY` dijelu), alternativni nazivi relacija se moraju koristiti u ostatku upita te na ovaj način se

"prividno" stvaraju dvije nove relacije `mjestoRod` i `mjestoStan` koje se mogu u upitu koristiti na isti način kao da se radi o "postojećim" relacijama u bazi podataka.

```
SELECT imeClan, prezClan, mjestoStan.nazMjesto nazMjestoStan,
       mjestoRod.nazMjesto nazMjestoRod
  FROM clan INNER JOIN mjesto mjestoStan ON pbrStan = mjestoStan.pbr
  INNER JOIN mjesto mjestoRod ON pbrRod = mjestoRod.pbr
```

imeClan	prezClan	nazMjestoStan	nazMjestoRod
Mensur	Kasumović	Lukavac	Tuzla
Student	Studentić	Tuzla	Brčko
Marina	Pejić	Tuzla	Tuzla

Spajanje relacije kod koje postoji "refleksivna" veza

Kada se n-torke jedne relacije vežu sa drugim n-torkama iz iste relacije (na način da atribut koji je strani ključ u relaciji referencira na primarni ključ iste relacije), tada u relaciji postoji refleksivna veza. Npr. kada bi se model baze podataka fakultetske biblioteke proširio na način da obuhvata podatke za cjelokupan univerzitet jedno od proširenja bi sasvim sigurno bilo da se prikaže hijerarhijska organizacija univerziteta, odnosno koja organizaciona jedinica je direktno nadređena određenoj organizacionoj jedinici. Ovo bi se postiglo na način da se svaka organizaciona jedinica iz relacije `orgjed` povezuje sa svojom nadređenom organizacionom jedinicom preko atributa `sifNadOrgjed` koji bi bio strani ključ koji referencira na atribut `sifOrgjed` kao primarni ključ iste relacije.

ORGJED

sifOrgjed	nazOrgjed	sifNadOrgjed
101	Univerzitet u Tuzli	NULL
102	Fakultet elektrotehnike	101
103	Prirodno-matematički fakultet	101
104	Računarstvo i informatika	102
105	Automatika i robotika	102
106	Telekomunikacije	102
107	Matematika	103
108	Fizika	103

U slučaju da se želi napisati upit kojim bismo htjeli za svaku organizacionu jedinicu prikazati njezinu nadređenu organizacionu jedinicu ili sve njezine podređene organizacione jedinice, očigledno bi se moralo obaviti spajanje relacije `orgjed` "same sa sobom". Kako se naziv jedne relacije u `FROM` dijelu `SELECT` naredbe može pojaviti samo jednom, problem je sličan i rješava se na sličan način kao u slučaju postojanja paralelne veze opisanom u prethodnom potpoglavlju. Relacija `orgjed` će se u upitu pojaviti dva puta, jednom u svojoj originalnoj ulozi organizacione jedinice, a drugi put u ulozi njezine nadređene (ili podređene) organizacione jedinice. Zbog toga se ponovo mora koristiti alternativni naziv (*alias*) relacije, s tim da se relacija uvijek pojavljuje u svojoj originalnoj ulozi te se preporučuje korištenje *aliasa* samo za drugu ulogu koju relacija ima u upitu.

Primjer: *Ispis šifre i naziva organizacione jedinice te naziva njezine direktno nadređene organizacione jedinice*

```
SELECT orgjed.sifOrgjed, orgjed.nazOrgjed, nadOrgjed.nazOrgjed
nazNadredjena
  FROM orgjed INNER JOIN orgjed nadOrgjed
  ON orgjed.sifNadOrgjed = nadOrgjed.sifOrgjed
```

sifOrgjed	nazOrgjed	nazNadredjena
102	Fakultet elektrotehnike	Univerzitet u Tuzli
103	Prirodno-matematički fakultet	Univerzitet u Tuzli
104	Računarstvo i informatika	Fakultet elektrotehnike
105	Automatika i robotika	Fakultet elektrotehnike
106	Telekomunikacije	Fakultet elektrotehnike
107	Matematika	Prirodno-matematički fakultet
108	Fizika	Prirodno-matematički fakultet

One organizacione jedinice koje nemaju nadređenu organizacionu jedinicu (*Univerzitet u Tuzli*) nisu se pojavile u rezultatu iz prostog razloga jer u relaciji ne postoji n-torka sa kojom bi se mogle spojiti (vrijednost stranog ključa te n-torke je `NULL`, a u relaciji primarni ključ ne može imati vrijednost `NULL`).

Primjer: *Ispis šifre i naziva organizacione jedinice te naziva svih njezinih direktno podređenih organizacionih jedinica*

```
SELECT orgjed.sifOrgjed, orgjed.nazOrgjed, podOrgjed.nazOrgjed
nazPodredjena
  FROM orgjed INNER JOIN orgjed podOrgjed
  ON orgjed.sifOrgjed = podOrgjed.sifNadOrgjed
```

sifOrgjed	nazOrgjed	nazPodredjena
101	Univerzitet u Tuzli	Fakultet elektrotehnike
101	Univerzitet u Tuzli	Prirodno-matematički fakultet
102	Fakultet elektrotehnike	Računarstvo i informatika
102	Fakultet elektrotehnike	Automatika i robotika
103	Fakultet elektrotehnike	Telekomunikacije
103	Prirodno-matematički fakultet	Matematika
103	Prirodno-matematički fakultet	Fizika

One organizacione jedinice koje nemaju podređenih organizacionih jedinica (odsjeci ili usmjerenja na fakultetima) nisu se pojavile u rezultatu iz prostog razloga jer u relaciji ne postoji n-torka sa kojom bi se mogle spojiti (vrijednosti primarnog ključa takvih n-torki se ne pojavljuju kao vrijednosti stranog ključa niti u jednoj n-torci relacije).

Treba naglasiti da se aliasi relacija ne koriste samo u slučajevima kada u modelu baze podataka postoje očigledne paralelne i refleksivne veze između relacija, nego uopšte u svim slučajevima kada postoji potreba da relacija u upitu ima dvije uloge. Npr. ukoliko bi se uz nazive mjesta rođenja i mjesta stanovanja željeli ispisati i nazivi kantona kojima mjesta pripadaju sljedeći upit ne bi bio ispravno postavljen:

```
SELECT imeClan, prezClan, mjestoStan.nazMjesto nazMjestoStan,
       nazKanton nazKantonStan, mjestoRod.nazMjesto nazMjestoRod,
       nazKanton nazKantonRod
  FROM clan INNER JOIN mjesto mjestoStan ON pbrStan = mjestoStan.pbr
  INNER JOIN mjesto mjestoRod ON pbrRod = mjestoRod.pbr
  INNER JOIN kanton ON (mjestoStan.sifKanton = kanton.sifKanton
                        AND mjestoRod.sifKanton = kanton.sifKanton)
```

jer bi se u rezultatu pojavili samo oni članovi kojima se mjesto rođenja i mjesto stanovanja nalaze u istom kantonu. Iako u modelu baze podataka ne postoji paralelna veza između relacija `mjesto` i `kanton`, zbog potrebe da se relacija `mjesto` koristi u dvije uloge u upitu posljedično i relacija `kanton` u istom upitu mora imati dvije uloge, jednom kao kanton u kojem se nalazi mjesto rođenja a drugi put kao kanton u kojem se nalazi mjesto stanovanja. Ispravno postavljen upit u ovom slučaju bi izgledao ovako:

```
SELECT imeClan, prezClan, mjestoStan.nazMjesto nazMjestoStan,
       kantonStan.nazKanton nazKantonStan, mjestoRod.nazMjesto
       nazMjestoRod, kantonRod.nazKanton nazKantonRod
  FROM clan INNER JOIN mjesto mjestoStan ON pbrStan = mjestoStan.pbr
  INNER JOIN mjesto mjestoRod ON pbrRod = mjestoRod.pbr
  INNER JOIN kanton kantonStan ON mjestoStan.sifKanton=kantonStan.sifKanton
  INNER JOIN kanton kantonRod ON mjestoRod.sifKanton = kantonRod.sifKanton
```

NON EQUI JOIN

Kao što je prethodno naglašeno, SQL NON EQUI JOIN ili spajanje uz uslov je primjer spajanja relacija u kojem se u uslovu spajanja koristi bilo koji operator poređenja izuzev znaka jednakosti (=). Ovakvo spajanje relacija se obavlja na sličan način kao i prirodno spajanje relacija odnosno uopšte spajanje relacija koje koristi operator izjednačavanja. Neka su date sljedeće dvije relacije:

RACUN

brRacun	stanje
1340002570025	500,00
1340002570128	340,00
1650001370056	180,00

NARUDZBA

sifNarudzba	iznos
N-107/17	347,50
N-108/17	254,70
N-109/17	733,10

Primjer: *Ispisati račune i uplate narudžbi koje je moguće izvršiti s tih računa (s obzirom na stanje računa i iznosa narudžbe koji je potrebno uplatiti).*

```
SELECT * FROM racun INNER JOIN narudzba
  ON stanje >= iznos
```

brRacun	stanje	sifNarudzba	iznos
1340002570025	500,00	N-107/17	347,50
1340002570025	500,00	N-108/17	245,70
1340002570128	340,00	N-108/17	254,70

U rezultatu se ne pojavljuje račun broj 1650001370056 jer na njemu nema dovoljno sredstava da bi se mogla platiti bilo koja narudžba, niti narudžba sa šifrom N-109/17 jer je iznos preveliki da bi se mogao platiti sa bilo kojeg računa. Gornja naredba je primjer operacije spajanja uz uslov `racun` $\triangleright\!\!\triangleleft_{\theta}$ `narudzba`.

Primjeri upita za vježbu

Q1: *Ispisati naziv, skraćenicu, broj ECTS kredita i naziv organizacione jedinice svih predmeta čiji naziv počinju sa riječju 'Osnovi'.*

```
SELECT nazPred, skrPred, brojECTS, nazOrgjed FROM pred
  INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
    WHERE nazPred LIKE 'Osnovi%'
```

Q2: *Ispisati naslove knjiga, ime i prezime autora i koji po redu potpisuje knjigu za sve knjige koje su izdate nakon 1990. godine a ime i prezime autora im počinje istim slovom.*

```
SELECT naslov, imeAutor, prezAutor, rbrAutor FROM knjiga
  INNER JOIN napisao ON knjiga.sifKnjiga = napisao.sifKnjiga
  INNER JOIN autor ON autor.sifAutor = napisao.sifAutor
    WHERE SUBSTRING(imeAutor, 1, 1) = SUBSTRING(prezAutor, 1, 1)
    AND godIzdanja > 1990
```

Q3: *Ispisati koliko različitih knjiga se koristi kao obavezna literatura na predmetima koji pripadaju organizacionoj jedinici pod nazivom 'Telekomunikacije'.*

```
SELECT COUNT(DISTINCT sifKnjiga) FROM pred
  INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
  INNER JOIN literatura ON pred.sifPred = literatura.sifPred
    WHERE nazOrgjed = 'Telekomunikacije' AND obavezna
```

Q4: *Ispisati imena i prezimena studenata koji su posuđivali knjige u prvom kvartalu prošle godine. Svaki student se u listi treba pojaviti samo jednom.*

```
SELECT DISTINCT imeClan, prezClan FROM stud
  INNER JOIN clan ON stud.sifClan = clan.sifClan
  INNER JOIN posudba ON clan.sifClan = posudba.sifClan
    WHERE MONTH(datPosudba) IN (1, 2, 3)
    AND YEAR(datPosudba) = YEAR(CURRENT_DATE) - 1
```

Q5: *Ispisati koliko posudbi su u toku tekuće godine obavili studenti koji su rođeni u Tuzlanskom kantonu.*

```
SELECT COUNT(*) FROM posudba
  INNER JOIN clan ON posudba.sifClan = clan.sifClan
  INNER JOIN stud ON clan.sifClan = stud.sifClan
  INNER JOIN mjesto ON clan.pbrRod = mjesto.pbr
  INNER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
    WHERE nazKanton = 'Tuzlanski kanton'
```

Q6: *Ispisati koliko je članova posuđivalo knjige autora sa inicijalima J.S. u proteklih 5 godina od današnjeg dana.*

```
SELECT DISTINCT(sifClan) FROM posudba
  INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj
  INNER JOIN knjiga ON primjerak.sifKnjiga = knjiga.sifKnjiga
  INNER JOIN napisao ON knjiga.sifKnjiga = napisao.sifKnjiga
  INNER JOIN autor ON napisao.sifAutor = autor.sifAutor
    WHERE imeAutor LIKE 'J%' AND prezAutor LIKE 'S%'
    AND datPosudba >= DATE_ADD(CURRENT_DATE, INTERVAL -5 YEAR)
```

Q7: *Ispisati prosječnu starost u godinama primjeraka knjiga koje se koriste kao prva po važnosti literatura na predmetima koji pripadaju organizacionoj jedinici Automatika i robotika.*

```
SELECT AVG(YEAR(CURRENT_DATE) - YEAR(datNabavka)) FROM primjerak
  INNER JOIN knjiga ON primjerak.sifKnjiga = knjiga.sifKnjiga
  INNER JOIN literatura ON knjiga.sifKnjiga = literatura.sifKnjiga
  INNER JOIN pred ON literatura.sifPred = pred.sifPred
  INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
    WHERE nazOrgjed = 'Automatika i robotika'
    AND rbrVaznost = 1
```

Q8: *Ispisati naziv predmeta i naziv predmeta koji mu je preduslov za one predmete kojima predmet koji je preduslov ne pripada istoj organizacionoj jedinici.*

```
SELECT pred.nazPred, uslov.nazPred FROM pred
  INNER JOIN preduslov ON pred.sifPred = preduslov.sifPred
  INNER JOIN pred uslov ON preduslov.sifPredUslov = uslov.sifPred
    WHERE pred.sifOrgjed <> uslov.sifOrgjed
```

Q9: *Za sve predmete koji pripadaju organizacionoj jedinici Računarstvo i informatika ispisati naziv predmeta, broj ECTS kredita, nazive i broj ECTS kredita predmeta koji su im preduslov. Ispisivati samo preduslovne predmete koji imaju preko 5 ECTS kredita.*

```
SELECT pred.nazPred, pred.brojECTS, uslov.nazPred, uslov.brojECTS FROM pred
  INNER JOIN preduslov ON pred.sifPred = preduslov.sifPred
  INNER JOIN pred uslov ON preduslov.sifPredUslov = uslov.sifPred
  INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
    WHERE nazOrgjed = 'Računarstvo i informatika'
    AND uslov.brojECTS > 5
```

Q10: *Ispisati ime i prezime, naziv mjesta rođenja i naziv kantona rođenja za studente koji stanuju u Brčko distriktu.*

```
SELECT imeClan, prezClan, mjestoRod.nazMjesto, kantonRod.nazKanton
  FROM stud INNER JOIN clan ON stud.sifClan = clan.sifClan
  INNER JOIN mjesto mjestoRod ON clan.pbrRod = mjestoRod.pbr
  INNER JOIN kanton kantonRod ON mjestoRod.sifKanton = kantonRod.sifKanton
  INNER JOIN mjesto mjestoStan ON clan.pbrStan = mjestoStan.pbr
```

```
  INNER JOIN kanton kantonStan ON mjestoStan.sifKanton =
kantonStan.sifKanton
    WHERE kantonStan.nazKanton = 'Brčko distrikt'
```

Q11: *Ispisati naslove knjiga koje su posuđivali članovi rođeni u Brčko distriktu koji su trenutno nastanjeni u Tuzlanskom kantonu.*

```
SELECT DISTINCT naslov FROM knjiga
  INNER JOIN primjerak ON knjiga.sifKnjiga = primjerak.sifKnjiga
  INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
  INNER JOIN clan ON posudba.sifClan = clan.sifClan
  INNER JOIN mjesto mjestoRod ON clan.pbrRod = mjestoRod.pbr
  INNER JOIN kanton kantonRod ON mjestoRod.sifKanton = kantonRod.sifKanton
  INNER JOIN mjesto mjestoStan ON clan.pbrStan = mjestoStan.pbr
  INNER JOIN kanton kantonStan ON mjestoStan.sifKanton =
kantonStan.sifKanton
    WHERE kantonRod.nazKanton = 'Brčko distrikt'
    AND kantonStan.nazKanton = 'Tuzlanski kanton'
```

Q12: *Ispisati naslove knjiga koje se koriste kao obavezna literatura na usmjerenju Telekomunikacije a koje su u tekućoj godini posuđivali nastavnici zaposleni na organizacionoj jedinici Računarstvo i informatika. Svaki takav naslov se u listi treba pojaviti samo jednom.*

```
SELECT DISTINCT naslov FROM knjiga
  INNER JOIN primjerak ON knjiga.sifKnjiga = primjerak.sifKnjiga
  INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
  INNER JOIN nastavnik ON posudba.sifClan = nastavnik.sifClan
  INNER JOIN orgjed orgjedN ON nastavnik.sifOrgjed = orgjedN.sifOrgjed
  INNER JOIN literatura ON knjiga.sifKnjiga = literatura.sifKnjiga
  INNER JOIN pred ON literatura.sifPred = pred.sifPred
  INNER JOIN orgjed orgjedP ON pred.sifOrgjed = orgjedP.sifOrgjed
    WHERE orgjedN.nazOrgjed = 'Računarstvo i informatika'
    AND orgjedP.nazOrgjed = 'Telekomunikacije'
    AND obavezna AND YEAR(datPosudba) = YEAR(CURRENT_DATE)
```

Upotreba podupita u uslovima dohvata

Kako je već istaknuto, uslovi od kojih se sastoji uslov dohvata osim uslova poređenja mogu biti i uslovi sa podupitima. Podupit (*subquery*) je `SELECT` naredba koja je upotrebom okruglih zagrada ugniježdena u `WHERE` dijelu neke druge `SELECT`, `UPDATE` ili `DELETE` naredbe. Ukoliko neki upit u svom `WHERE` dijelu sadrži uslov sa podupitom, tada se takav upit naziva vanjski upit (*outer statement*).

Postoje tri načina na koja se podupit može koristiti u `WHERE` dijelu vanjskog upita:

1. Poređenje vrijednosti izraza iz vanjskog upita sa rezultatom podupita
2. Ispitivanje da li je vrijednost nekog izraza iz vanjskog upita sadržana u rezultatu podupita

3. Ispitivanje da li se kao rezultat podupitapojavljuje barem jedna n-torka

Poređenje vrijednosti izraza iz vanjskog upita sa rezultatom podupita

Opšti oblik uslova ovakvog načina korištenja podupita u `WHERE` dijelu vanjskog upita je sljedeći:

```
expression relational_operator [ ALL | ANY | SOME ] ( subquery )
```

Relacijski operator (`relational_operator`) koji se koristi za poređenje vrijednosti izraza (`expression`) sa rezultatom podupita može biti bilo koji relacijski operator (<, >, =, itd.). Za jednu n-torku vanjskog upita izračunava se jedna vrijednost izraza te zbog toga podupit koji se koristi na ovakav način mora vratiti tačno jednu kolonu, tj. lista za selekciju `SELECT` naredbe koja se koristi kao podupit (*subquery*) smije sadržavati samo jedan atribut ili izraz. Ukoliko ovo nije slučaj upit će završiti sa pogreškom. Osim toga, u podupitu se ne smije koristiti `ORDER BY` dio `SELECT` naredbe.

Poznato je da se kao rezultat `SELECT` naredbe, pa tako i one koja se koristi kao podupit, može pojaviti jedna n-torka, više n-torki ili niti jedna n-torka. Zato se poređenje izraza iz uslova vanjskog upita vrši sa svakom od vrijednosti koja se pojavi u rezultatu podupita te se n-torka iz vanjskog upita pojavljuje u rezultatu ukoliko je to poređenje zadovoljeno za jednu ili više n-torki iz rezultata podupita u zavisnosti od toga da li se koristi ključna riječ `ALL`, `ANY` ili `SOME`.

Ukoliko se prilikom poređenja koristi ključna riječ `ALL`, da bi se n-torka iz vanjskog upita pojavila u rezultatu tada uslov poređenja izraza sa svakom od n-torki koje su rezultat podupita mora biti zadovoljen (*true*). Ukoliko upit ne vrati niti jednu n-torku tada se uslov poređenja uvijek evaluira kao istinit (*true*).

Primjer: *Ispisati podatke o nastavnicima koji su stariji od svih studenata.*

```
SELECT * FROM nastavnik
  INNER JOIN clan ON nastavnik.sifClan = clan.sifClan
    WHERE datRod < ALL (SELECT datRod FROM stud
                          INNER JOIN clan ON stud.sifClan = clan.sifClan)
```

Da bi se nastavnik pojavio u rezultatu upita njegov datum rođenja mora biti manji od datuma rođenja svih studenata (datuma rođenja koji se pojave u rezultatu podupita). Dakle, možemo smatrati da se ovaj upit obavlja u sljedećim koracima:

1. Izvrši se podupit koji kao rezultat daje skup vrijednosti svih datuma rođenja studenata
2. Uzima se prva n-torka iz vanjskog upita te se vrijednost atributa `datRod` iz te n-torke poredi sa svakom vrijednšću iz skupa datuma rođenja koji je rezultat podupita
3. Ako je vrijednost `datRod` iz n-torke vanjskog upita manja od svake vrijednosti datuma rođenja sa kojim je poređena, prva n-torka iz vanjskog upita se pojavljuje u rezultatu
4. Koraci 2 i 3 se ponavljaju za sljedeću n-torku vanjskog upita

Na osnovu gore opisanog postupka možemo primijetiti da se `SELECT` naredba koja se koristi kao podupit u uslovu dohvata izvrši samo jednom te se n-torke iz vanjskog upita porede sa skupom vrijednosti koji se dobije sa tim izvršavanjem.

Ukoliko se prilikom poređenja koristi ključna riječ `ANY` (`SOME` je sinonim), da bi se n-torka iz vanjskog upita pojavila u rezultatu tada je dovoljno da uslov poređenja izraza bude zadovoljen (*true*) sa barem jednom n-torkom iz rezultata podupita. Ukoliko upit ne vrati niti jednu n-torku tada se uslov poređenja uvijek evaluira kao lažan (*false*).

Primjer: *Ispisati podatke o nastavnicima koji su mlađi od bar jednog studenta.*

```
SELECT * FROM nastavnik
  INNER JOIN clan ON nastavnik.sifClan = clan.sifClan
    WHERE datRod > ANY (SELECT datRod FROM stud
                          INNER JOIN clan ON stud.sifClan = clan.sifClan)
```

Ukoliko će podupit sa sigurnošću vratiti samo jednu n-torku, s obzirom da se u listi za selekciju podupita nalazi samo jedan atribut ili izraz, tada je rezultat podupita tačno jedna vrijednost (*single-valued subquery*). U takvim slučajevima se ključne riječi `ALL`, `ANY` i `SOME` u uslovima poređenja mogu ispustiti jer vrijednost podupita se može posmatrati kao rezultat funkcije, a uslov poređenja sa upitom kao poređenjeizraza sa vrijednošću koju vrati neka funkcija. Međutim, ukoliko bi se kod takvog definisanja uslova dohvata dogodilo da podupit vrati više od jedne n-torke, onda bi upit završio sa pogreškom. Ukoliko podupit ne bi vratio niti jednu n-torku tada bi se uslov poređenja evaluirao kao lažan (*false*).

Primjer: *Ispisati naslove i godine izdanja knjiga izdatih nakon knjige 'SQL i relacijski model podataka' autora Ratka Vujnovića.*

```
SELECT naslov, godIzdanja FROM knjiga
WHERE godIzdanja >
      (SELECT godIzdanja FROM knjiga
         INNER JOIN napisao ON knjiga.sifKnjiga = napisao.sifKnjiga
         INNER JOIN autor ON napisao.sifAutor = autor.sifAutor
           WHERE imeAutor = 'Ratko' AND prezAutor = 'Vujnović')
```

Poznato je da agregatne funkcije izračunavaju tačno jednu vrijednost na osnovu vrijednosti iz više n-torki te se upravo one najčešće koriste u listi za selekciju u podupitima. Na ovaj način se zapravo korištenjem podupita prevazilazi ograničenje da se agregatne funkcije ne mogu koristiti u uslovima dohvata te je moguće napisati upite koje ranije bez upotrebe podupita nismo bili u mogućnosti.

Primjer: *Ispisati podatke o nastavnicima čiji je koeficijent za platu veći od prosječnog koeficijenta za platu svih nastavnika.*

```
SELECT * FROM nastavnik
  WHERE koef > (SELECT AVG(koef) FROM nastavnik)
```

Primjer: *Ispisati naslove knjiga koje su prve posuđene u biblioteci (može ih biti više jer se ne evidentira vrijeme nego samo datum posudbe).*

```
SELECT DISTINCT naslov FROM knjiga
  INNER JOIN primjerak ON knjiga.sifKnjiga = primjerak.sifknjiga
  INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
    WHERE datPosudba = (SELECT MIN(datPosudba) FROM posudba)
```

MySQL uz podupite koji vraćaju tačno jednu vrijednost (*single-valued subquery*) podržava i varijantu upita koja vraća jedan red (*row subquery*) sa više kolona čiji se rezultat može koristiti u uslovu dohvata sa operatorima poređenja. Opšti oblik uslova ovakvog načina korištenja podupita koji vraća jedan red u `WHERE` dijelu vanjskog upita je sljedeći:

MySQL
```
[ROW](expr1, expr2 [, ...]) relational_operator ( subquery )
```

Podupit (`subquery`) korišten na ovaj način mora vratiti tačno jedan red te u listi za selekciju sadržavati isti broj izraza koji se nalazi na lijevoj strani uslova poređenja. U suprotnom upit završava sa pogreškom.

Primjer: *Ispisati ime, prezime i datum rođenja nastavnika koji imaju isto ime i prezime kao i autor sa šifrom 506.*

MySQL
```
SELECT imeClan, prezClan, datRod FROM clan
  INNER JOIN nast ON clan.sifClan = nast.sifClan
    WHERE (imeClan, prezClan) = (SELECT imeAutor, prezAutor FROM autor
                                   WHERE sifAutor = 506)
```

Prethodni upit je primjer skraćenog pisanja sljedećeg upita u MySQL-u:

```
SELECT imeClan, prezClan, datRod FROM clan
  INNER JOIN nast ON clan.sifClan = nast.sifClan
    WHERE imeClan = (SELECT imeAutor FROM autor
                       WHERE sifAutor = 506)
    AND prezClan = (SELECT prezAutor FROM autor
                      WHERE sifAutor = 506)
```

Ispitivanje da li je vrijednost izraza iz vanjskog upita sadržana u rezultatu podupita

Opšti oblik uslova ovakvog načina korištenja podupita u `WHERE` dijelu vanjskog upita je sljedeći:

```
expression [ NOT ] IN ( subquery )
```

Kao i u prethodno opisanom načinu ugradnje podupita, i u ovom slučaju za jednu n-torku vanjskog upita izračunava se jedna vrijednost izraza (`expression`). Zbog toga podupit koji se koristi na ovakav način također mora vratiti tačno jednu kolonu, tj. lista za selekciju `SELECT` naredbe koja se koristi kao podupit (`subquery`) smije sadržavati samo jedan atribut ili izraz. Ukoliko ovo nije slučaj upit će završiti sa pogreškom, a osim toga u podupitu se ne smije koristiti ni `ORDER BY` dio `SELECT` naredbe.

U ovom načinu upotrebe podupita u uslovu dohvata, da bi se n-torka iz vanjskog upita pojavila u rezultatu, odnosno da bi zadovoljila uslov dohvata:

- vrijednost izraza (`expression`) mora biti sadržana u skupu vrijednosti dobivenih kao rezultat podupita - ako se **ne koristi** modifikator `NOT`
- vrijednost izraza (`expression`) ne smije biti sadržana u skupu vrijednosti dobivenih kao rezultat podupita - ako se **koristi** modifikator `NOT`

Ova dva načina korištenja modifikatora `IN` i `NOT IN` su u suštini ekvivalentna poređenju izraza sa rezultatom podupita korištenjem operatora jednakosti (=) i kvalifikatora `ANY` (ili `SOME`) i operatora nejednakosti (<>) i kvalifikatora `ALL`, respektivno.

Primjer: *Ispis imena i prezimena nastavnika koji nisu obavili niti jednu posudbu u biblioteci.*

```
SELECT imeClan, prezClan FROM nastavnik
  INNER JOIN clan ON nastavnik.sifClan = clan.sifClan
    WHERE nastavnik.sifClan NOT IN (SELECT sifClan FROM posudba)
```

Primjer: *Ispisati ime, prezime i naziv mjesta stanovanja nastavnika koji stanuju u mjestu u kojem je rođen bar jedan student.*

```
SELECT imeClan, prezClan, nazMjesto FROM nastavnik
  INNER JOIN clan ON nastavnik.sifClan = clan.sifClan
  INNER JOIN mjesto ON clan.pbrStan = mjesto.pbr
    WHERE pbrStan IN (SELECT pbrRod FROM stud
                        INNER JOIN clan ON stud.sifClan = clan.sifClan)
```

Kao i u prethodnom primjeru ugradnje podupita koji vraćaju jednu n-torku (*row subquery*) u uslov dohvata, i ovom slučaju MySQL podržava proširenje korištenja podupita u kombinaciji sa `[NOT] IN` kvalifikatorom. Opšti oblik uslova ovakvog načina korištenja podupita u `WHERE` dijelu vanjskog upita je sljedeći:

MySQL
```
[ROW](expr1, expr2 [, ...]) [ NOT ] IN ( subquery )
```

Ovakav način korištenja podupita u MySQL-u služi da bi se odredilo da li su (ili nisu) n-torke vanjskog upita sadržane u rezultatu podupita.

Primjer: *Ispisati sve podatke o mjestima koji se koriste i kao mjesto rođenja i kao mjesto stanovanja članova biblioteke.*

MySQL
```
SELECT DISTINCT mjesto.* FROM mjesto
  INNER JOIN clan ON mjesto.pbr = clan.pbrStan
    WHERE (pbr, nazMjesto, sifKanton) IN
      (SELECT DISTINCT mjesto.* FROM mjesto
         INNER JOIN clan ON mjesto.pbr = clan.pbrRod)
```

Ispitivanje da li se kao rezultat podupita pojavljuje barem jedna n-torka

Opšti oblik uslova ovakvog načina korištenja podupita u WHERE dijelu vanjskog upita je sljedeći:

```
[ NOT ] EXISTS ( subquery )
```

Za razliku od prethodno dva opisana načina ugradnje podupita, u ovom slučaju u uslovu dohvata nema izračunavanja vrijednost izraza (expression). Zbog toga podupit koji se koristi na ovakav način može vratiti proizvoljan broj kolona, tj. lista za selekciju `SELECT` naredbe koja se koristi kao podupit (`subquery`) može sadržavati proizvoljan broj atributa ili izraza. S obzirom da broj atributa ili izraza u listi za selekciju podupita ne utiče na rezultat, najčešće se upotrebljava znak *.

U ovom načinu upotrebe podupita u uslovu dohvata, da bi se n-torka iz vanjskog upita pojavila u rezultatu:

- podupit mora vratiti barem jednu n-torku - ukoliko se **ne koristi** modifikator `NOT`
- podupit ne smije vratiti niti jednu n-torku - ukoliko se **koristi** modifikator `NOT`

Primjer: *Ispis imena, prezimena i koeficijenta za platu svih nastavnika ukoliko u ovom mjesecu nije obavljena niti jedna posudba knjige.*

```
SELECT imeClan, prezClan, koef FROM nastavnik
  INNER JOIN clan ON nastavnik.sifClan = clan.sifClan
    WHERE NOT EXISTS (SELECT * FROM posudba
                        WHERE YEAR(datPosudba) = YEAR(CURRENT_DATE)
                        AND MONTH(datPosudba) = MONTH(CURRENT_DATE))
```

Međutim, veoma je upitna svrha postavljanja ovakvog ili sličnog upita u kojima je pojavljivanje n-troki u rezultatu vanjskog upita uslovljeno postojanjem n-torki iz nekih drugih relacija sa kojima nije uspostavljen nikakav odnos. Za prethodni primjer mnogo više smisla bi imao zahtjev da se ispišu podaci o nastavnicima koji nisu obavili niti jednu posudbu u ovom mjesecu. To bi značilo da bi u podupitu trebalo uspostaviti odnos sa n-torkom iz vanjskog upita koja se trenutno dohvata kako bi se podupitom mogle dohvatiti posudbe samo tog nastavnika. Zbog toga se ovakav način ugradnje podupita u uslov dohvata najčešće koristi u koreliranim podupitima koji su objašnjeni u nastavku.

Korelirani podupiti

Svi podupiti koji su napisani u prethodnim primjerima bez obzira na način ugradnje u uslov dohvata su nekorelirani podupiti (*uncorrelated subquery*) jer za svoje izvršavanje ne zahtjevaju korištenje vrijednosti iz relacija vanjskog upita. Za razliku od ovih podupita, ukoliko je za izvršavanje podupita neophodna vrijednost iz relacija vanjskog upita, takav podupit se naziva korelirani podupit (*correlated subquery*).

Primjer: *Ispisati naslove i godine izdanja knjiga koje imaju više primjeraka od starosti knjige.*

```
SELECT naslov, godIzdanja FROM knjiga
  WHERE YEAR(CURRENT-DATE) - godIzdanja <
    (SELECT COUNT(*) FROM primjerak
       WHERE primjerak.sifKnjiga = knjiga.sifKnjiga)
```

Podupit korišten u prethodnom upitu je koreliran jer je za njegovo izvršavanje potrebno koristiti vrijednost šifre knjige iz relacije vanjskog upita (`knjiga.sifKnjiga`). Dakle, možemo smatrati da se ovaj upit koji koristi korelirani podupit obavlja u sljedećim koracima:

1. Uzima se prva n-torka iz vanjskog upita (iz relacije `knjiga`) te se vrijednost atributa `sifKnjiga` iz te n-torke koristi da bi se izvršio podupit koji kao rezultat daje broj primjeraka za knjigu sa tom šifrom
2. Ako je vrijednost izračunata podupitom (broj primjeraka knjige) veća od starosti te knjige (razlika između trenutne godine i godine izdanja), prva n-torka iz vanjskog upita se pojavljuje u rezultatu
3. Koraci 1 i 2 se ponavljaju za svaku n-torku iz vanjskog upita (relacije `knjiga`)

Ukoliko se uporedi način obavljanja upita sa koreliranim podupitom i upita sa nekoreliranim podupitom, može se zaključiti da se nekorelirani podupit izvršava samo jednom te se njegov rezultat koristi za poređenje sa n-trokama vanjskog upita, dok se korelirani upit izvršava jednom za svaku n-troku iz vanjskog upita s obzirom da koristi vrijednosti iz te n-troke za izračunavanje rezultata. Kao i nekorelirani podupiti, i korelirani podupiti se u uslove dohvata vanjskog upita mogu ugrađivati na sva tri prethodno opisana načina.

Primjer: *Ispis imena, prezimena i koeficijenta za platu svih nastavnika koji u ovom mjesecu nisu obavili niti jednu posudbu knjige.*

```
SELECT imeClan, prezClan, koef FROM nastavnik
  INNER JOIN clan ON nastavnik.sifClan = clan.sifClan
    WHERE NOT EXISTS (SELECT * FROM posudba
                        WHERE posudba.sifClan = clan.sifClan
                        AND YEAR(datPosudba) = YEAR(CURRENT_DATE)
                        AND MONTH(datPosudba) = MONTH(CURRENT_DATE))
```

Primjer: *Ispisati šifru, naziv i broj ECTS kredita predmeta koji nisu preduslov niti jednom predmetu iz organizacione jedinice kojoj pripadaju.*

```
SELECT sifPred, nazPred, brojECTS FROM pred
  WHERE sifPred NOT IN
    (SELECT sifPredUslov FROM preduslov
       INNER JOIN pred ostali ON preduslov.sifPred = ostali.sifPred
         WHERE ostali.sifOrgjed = pred.sifOrgjed)
```

Primijetimo da u prethodnom upitu atribut `pred.sifOrgjed` nije nigdje naveden u vanjskom upitu ali se svakako može koristiti za izračunavanje podupita s obzirom da se mogu koristiti svi atributi koji postoje u n-torkama vanjskog upita.

Kombinovanje uslova s podupitima

S obzirom da se kao podupit može koristiti bilo koja `SELECT` naredba (uz ograničenje da ne smije sadržavati `ORDER BY` dio), upit može sadržavati podupit koji u svom uslovu dohvata koristi podupit, koji opet također može sadržavati podupit, itd. U tom slučaju se radi o ugniježdenim podupitima.

Primjer: *Ispisati članove koji niti jednom nisu zadržali primjerak knjige duže od prosječnog broja dana zaduženja svih posudbi obavljenih u tekućoj godini.*

```
SELECT * FROM clan
  WHERE sifClan NOT IN (SELECT sifClan FROM posudba
  WHERE DATEDIFF(datVracanje, datPosudba) >
    (SELECT AVG(DATEDIFF(datVracanje. datPosudba)) FROM posudba
       WHERE YEAR(datPosudba) = YEAR(CURRENT_DATE)))
```

S obzirom da je uslov sa podupitom samo jedna od vrsta uslova oni se u uslovu dohvata mogu kombinovati međusobno, a i sa drugim uslovima korištenjem logičkih operatora.

Primjer: *Ispisati naslove knjiga koje je posudio bar jedan nastavnik ali ih nije posudio niti jedan student.*

```
SELECT naslov FROM knjiga
  WHERE sifKnjiga IN
    (SELECT sifknjiga FROM primjerak
       INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
       INNER JOIN nastavnik ON posudba.sifClan = nastavnik.sifClan)
  AND sifKnjiga NOT IN
    (SELECT sifknjiga FROM primjerak
       INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
       INNER JOIN stud ON posudba.sifClan = stud.sifClan)
```

Primjer: *Ispisati naslove i godine izdanja svih knjiga koje se ne koriste kao obavezna literatura niti na jednom predmetu a izdate su nakon 2000. godine.*

```
SELECT naslov, godIzdanja FROM knjiga
  WHERE sifKnjiga NOT IN
    (SELECT sifKnjiga FROM literatura
       WHERE obavezna)
  AND godIzdanja > 2000
```

Dodatna razmatranja o podupitima

U svim prethodnim primjerima podupiti su korišteni u uslovima dohvata (`WHERE` dijelu) `SELECT` naredbe. Međutim, podupiti se također mogu koristiti i u drugim SQL naredbama (`INSERT`, `UPDATE`, `DELETE`) o čemu će više biti riječi u nastavku, kao i u `FROM` dijelu SQL naredbi što nije predmet razmatranja u ovoj knjizi.

Prilikom upotrebe podupita treba voditi računa o sljedećem:

- Podupit se mora navesti u okruglim zagradama
- Podupit se u uslovima dohvata koristi na desnoj strani operatora poređenja
- Podupiti ne mogu interno manipulisati svojim rezultatom tako da ne mogu sadržavati `ORDER BY` dio
- Ukoliko podupit vrati `NULL` vrijednost vanjskom upitu, vanjski upit neće vratiti niti jednu n-torku kada se koriste određeni operatori poređenja (o čemu će više riječi biti u dijelu koji se bavi `NULL` vrijednostima)

Iz svega do sada navedenog se može zaključiti da rješenje jednog upita u SQL-u nije jedinstveno i da se za dobijanje istog rezultata može napisati više `SELECT` naredbi koje nisu identične. Npr. ukoliko se želi ispisati ime, prezime i datume rođenja članova koji su rođeni u Sarajevskom kantonu, upotrebom spajanja relacija upit bi bio:

```
SELECT imeClan, prezClan datRod FROM clan
  INNER JOIN mjesto ON clan.pbrRod = mjesto.pbr
  INNER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
    WHERE nazKanton = 'Sarajevski kanton'
```

Međutim, na osnovu naučenog o podpitima isti upit bi se korištenjem podupita mogao napisati na sljedeći način:

```
SELECT imeClan, prezClan datRod FROM clan
  WHERE pbrRod IN
    (SELECT pbr FROM mjesto
       WHERE sifKanton IN (SELECT sifKanton FROM kanton
                           WHERE nazKanton = 'Sarajevski kanton')
    )
```

Na osnovu naprijed navedenog mogao bi se izvući zaključak da se spajanje relacija u upitima može zamijeniti upotrebom podupita. Međutim, podupite nikada ne treba koristiti bez razloga a posebno ne kao zamjenu za spajanje relacija iz više razloga:

- Spajanje relacija je mnogo brže od korištenja podupita i vrlo su rijetki obrnuti slučajevi
- Kada se koristi spajanje relacija SUBP kreira plan izvršavanja upita kojim se može predvidjeti koji podaci će biti učitani i koliko vremena će trebati za njihovu obradu što kao rezultat ima uštedu vremena s obzirom da SUBP bira najoptimalniji plan izvršavanja. Nasuprot tome, kod upotrebe podupita nema prethodnog preračunavanja te se izvršavaju svi upiti i dohvataju svi podaci tih upita kako bi se obavila obrada.
- Kod korištenja spajanja relacija se prvo provjeravaju uslovi a potom šalju u relacije i prikazuju dok se kod podupita interno stvaraju privremene relacije pa potom provjeravaju uslovi.
- Kod spajanja relacija očigledno postoji veza između dvije ili više relacija koji se međusobno povezuju dok podupit podrazumjeva upit u nekom drugom upitu, nema potrebe za bilo kakvim povezivanjem te jednostavno radi sa kolonama i uslovima.

Određivanje unije, presjeka i razlike relacija

Da bi se mogli odrediti unija, presjek ili razlika relacija, one moraju biti unijski kompatibilne, odnosno moraju imati isti broj atributa i odgovarajući (korespodentni) atributi moraju biti definisani nad istim domenama. Neka postoje sljedeće dvije unijski kompatibilne relacije:

PREDMET

oznFak	oznPred	nazPred
FET	MAT1	Matematika I
FET	AKT	Aktuatori
FET	BP	Baze podataka
PMF	MAT1	Mat. analiza I
PMF	FIZ1	Opća fizika I

PREDUSLOV

oznFak	oznPred	nazPred
FET	MAT1	Matematika I
FET	FIZ1	Fizika I
FET	BP	Baze podataka
PMF	MAT1	Mat. analiza I

Primarni ključ u obje relacije je (`oznFak`, `oznPred`).

Operacija unije relacija (`predmet ∪ preduslov`) kao rezultat daje sve n-torke iz relacije `predmet` i sve n-torke iz relacije `preduslov` uz eliminaciju duplih n-torki. Da bi se obavila operacija unije potrebno je spojiti rezultate dvije `SELECT` naredbe što se može obaviti upotrebom `UNION [ALL | DISTINCT]` operatora.

```
SELECT oznFak, oznPred, nazPred FROM predmet
UNION
SELECT oznFak, oznPred, nazPred FROM preduslov
```

S obzirom da je dovoljno da dvije relacije budu unijski kompatibilne da bi se obavila operacija unije, za slučaj da relacije nemaju iste nazive atributa u rezultatu će biti prikazani nazivi iz liste za selekciju prve `SELECT` naredbe. `UNION` operator podrazumijevano eliminiše duple n-torke koje se pojavljuju u obje relacije tako da se `DISTINCT` kvalifikator može izostaviti. Ukoliko se navede kvalifikator `ALL` iz rezultata ne bi bile eliminisane duple n-torke što znači da se ne ubavlja operacija unije relacija nego jednostavno "spajanje" skupova n-torki koji su rezultat `SELECT` naredbi.

`UNION` operator se može koristiti na više od dvije `SELECT` naredbe te se upotreba `ALL` i `DISTINCT` kvalifikatora može kombinovati s tim da `DISTINCT` prepisuje sve `ALL` kvalifikatore koji su prethodno navedeni. `ORDER BY` je dozvoljeno koristiti u svakoj `SELECT` naredbi spojenoj sa `UNION` operatorom, ali na rezultat uticaj ima samo `ORDER BY` naveden u posljednjoj `SELECT` naredbi. `U ORDER BY` dijelu takvog upita se ne može koristiti atribut referenciran nazivom relacije (`tbl_name.col_name`) nego se treba navesti *alias* naziv za atribut u prvoj `SELECT` naredbi ili redni broj atributa u listi za selekciju.

Operacija presjeka relacija (`predmet ∩ preduslov`) kao rezultat daje sve n-torke iz relacije `predmet` koje postoje i u relaciji `preduslov`, odnosno ako bismo prema ovoj definiciji napisali upit:

```
SELECT * FROM predmet
  WHERE EXISTS (SELECT * FROM preduslov
                  WHERE predmet.oznFak = preduslov.oznFak
                  AND predmet.oznPred = preduslov.oznPred
                  AND predmet.nazPred = preduslov.nazPred)
```

S obzirom da obje relacije imaju isti ključ (oznFak, oznPred), tada ukoliko je ispunjen uslov da je:

```
    predmet.oznFak = preduslov.oznFak
AND predmet.oznPred = preduslov.oznPred
```

onda je ispunjen i uslov da je predmet.nazPred = preduslov.nazPred te se taj dio u upitu može izostaviti pa se presjek relacija pomoću SELECT naredbe određuje sljedećim upitom:

```
SELECT * FROM predmet
  WHERE EXISTS (SELECT * FROM preduslov
                  WHERE predmet.oznFak = preduslov.oznFak
                  AND predmet.oznPred = preduslov.oznPred)
```

S obzirom na činjenicu da će se n-torka iz relacije predmet pojaviti u rezultatu samo ukoliko postoji odgovarajuća n-torka iz relacije preduslov iz ovoga se može zaključiti da se presjek relacija može odrediti i prirodnim spajanjem relacija jer će se n-torka relacije predmet u rezultatu takvog upita pojaviti samo ukoliko ima odgovarajuću n-torku s kojom se može spojiti iz relacije preduslov.

```
SELECT predmet.* FROM predmet
  INNER JOIN preduslov ON (predmet.oznFak = preduslov.oznFak
                           AND predmet.oznPred = preduslov.oznPred)
```

Svaki od gornjih upita kao rezultat daje presjek relacija predmet i preduslov:

oznFak	oznPred	nazPred
FET	MAT1	Matematika I
FET	BP	Baze podataka
PMF	MAT1	Mat. analiza I

Kako je u prethodnom podpoglavlju prikazano da bi se spajanje relacija moglo zamijeniti upotrebom podupita sa IN kvalifikatorom to bi se moglo zaključiti da se i presjek relacija može odrediti upotrebom podupita sa IN kvalifikatorom umjesto spajanja relacija.

```
SELECT * FROM predmet
  WHERE predmet.oznFak IN (SELECT oznFak FROM preduslov)
  AND predmet.oznPred IN (SELECT oznPred FROM preduslov)
```

Kao rezultat prethodnog upita se dobija:

oznFak	oznPred	nazPred
FET	MAT1	Matematika I
FET	BP	Baze podataka
PMF	MAT1	Mat. analiza I
PMF	FIZ1	Opća fizika I

Posljednja n-torka se pojavljuje u rezultatu jer se vrijednost atributa `oznFak` te n-torke (`PMF`) pojavljuje u rezultatu prvog podupita kao i vrijednost atributa `oznPred` (`FIZ1`) u rezultatu drugog podupita. Dobijeni rezultat ne predstavlja presjek relacija `predmet` i `preduslov` te se u opštem slučaju presjek relacija ne može riješiti upotrebom podupita sa `IN` kvalifikatorom. Podupit sa `IN` kvalifikatorom se može koristiti jedino u slučaju kada se radi presjek relacija koje imaju jednostavan primarni ključ (sastavljen od samo jednog atributa).

S obzirom na podržani način ugradnje podupita u uslov dohvata sa upotrebom `IN` kvalifikatora opisanom u prethodnom podpoglavlju, u MySQL-u se presjek relacija u opštem slučaju može riješiti i upotrebom `IN` kvalifikatora:

MySQL
```
SELECT * FROM predmet
    WHERE (oznFak, oznPred) IN (SELECT oznFak, oznPred FROM preduslov)
```

Operacija razlike relacija (`predmet \ preduslov`) kao rezultat daje sve n-torke iz relacije `predmet` koje ne postoje u relaciji `preduslov`, odnosno ako bismo prema ovoj definiciji napisali upit:

```
SELECT * FROM predmet
  WHERE NOT EXISTS (SELECT * FROM preduslov
                      WHERE predmet.oznFak = preduslov.oznFak
                      AND predmet.oznPred = preduslov.oznPred
                      AND predmet.nazPred = preduslov.nazPred)
```

S obzirom da obje relacije imaju isti ključ (`oznFak`, `oznPred`), iz istog razloga kao u slučaju određivanja presjeka relacija, posljednji uslov poređenja u podupitu se može izostaviti te se razlika relacija određuje upitom:

```
SELECT * FROM predmet
  WHERE NOT EXISTS (SELECT * FROM preduslov
                      WHERE predmet.oznFak = preduslov.oznFak
                      AND predmet.oznPred = preduslov.oznPred)
```

Kao rezultat prethodnog upita dobija se presjek relacija `predmet` i `preduslov`:

oznFak	oznPred	nazPred
FET	AKT	Aktuatori
PMF	FIZ1	Opća fizika I

Ako bi se razlika relacija pokušala riješiti upotrebom podupita sa `NOT IN` kvalifikatorom:

```
SELECT * FROM predmet
  WHERE predmet.oznFak NOT IN (SELECT oznFak FROM preduslov)
  AND predmet.oznPred NOT IN (SELECT oznPred FROM preduslov)
```

kao rezultat ne bi se dobila niti jedna n-torka jer se svaka oznaka fakulteta iz relacije `predmet` nalazi i u relaciji `preduslov` te uslov dohvata nije ispunjen niti za jednu n-torku. Zbog toga, kao i u slučaju određivanja presjeka relacija, niti razlika relacija se u opštem slučaju ne može riješiti upotrebom podupita sa `NOT IN` kvalifikatorom. Podupit sa `NOT IN` kvalifikatorom se može koristiti jedino u slučaju kada se radi razlika relacija koje imaju jednostavan primarni ključ.

Iz istog razloga kao u slučaju presjeka relacija, u MySQL-u se razlika relacija u opštem slučaju može riješiti i upotrebom `NOT IN` kvalifikatora:

MySQL
```
SELECT * FROM predmet
  WHERE (oznFak, oznPred) NOT IN (SELECT oznFak, oznPred FROM preduslov)
```

Primjeri upita za vježbu

Q1: *Ispisati naslove i godine izdanja knjiga koje se ne koriste kao literatura niti na jednom predmetu na organizacionoj jedinici Elektroenergetske mreže i sistemi.*

```
SELECT naslov, godIzdanja FROM knjiga
  WHERE sifKnjiga NOT IN
    (SELECT sifKnjiga FROM literatura
       INNER JOIN pred ON literatura.sifPred = pred.sifPred
       INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
         WHERE nazOrgjed = 'Elektroenergetske mreže i sistemi')
```

Q2: *Ispisati nazive organizacionih jedinica u kojima je zaposlen bar jedan nastavnik mlađi od 35 godina a nije zaposlen nijedan nastavnik stariji od 50 godina.*

```
SELECT nazOrgjed FROM orgjed
  WHERE sifOrgjed IN
    (SELECT sifOrgjed FROM nast
       INNER JOIN clan ON nast.sifClan = clan.sifClan
         WHERE datRod > DATE_SUB(CURRENT_DATE, INTERVAL 35 YEAR))
  AND sifOrgjed NOT IN
    (SELECT sifOrgjed FROM nast
       INNER JOIN clan ON nast.sifClan = clan.sifClan
         WHERE datRod < DATE_SUB(CURRENT_DATE, INTERVAL 50 YEAR))
```

Q3: *Ispisati ime i prezime autora koji nisu objavili niti jednu knjigu koja se koristi kao obavezna literatura na bilo kojem predmetu.*

```
SELECT imeAutor, prezAutor FROM autor
  WHERE sifAutor NOT IN (SELECT DISTINCT sifAutor FROM napisao
       INNER JOIN literatura ON napisao.sifKnjiga = literatura.sifKnjiga
         WHERE obavezna)
```

Q4: *Ispisati ime prezime i datume rođenja studenata koji su rođeni u Tuzlanskom kantonu prije svih studenata rođenih u Brčko distriktu.*

```
SELECT imeClan, prezClan, datRod FROM clan
  INNER JOIN stud ON clan.sifClan = stud.sifStud
  INNER JOIN mjesto ON clan.pbrRod = mjesto.pbr
  INNER JIN kanton ON mjesto.sifKanton = kanton.sifKanton
    WHERE nazKanton = 'Tuzlanski kanton'
    AND datRod < ALL
      (SELECT datRod FROM clan
         INNER JOIN stud ON clan.sifClan = stud.sifStud
         INNER JOIN mjesto ON clan.pbrRod = mjesto.pbr
         INNER JIN kanton ON mjesto.sifKanton = kanton.sifKanton
           WHERE nazKanton = 'Brčko distrikt')
```

Q5: *Ispisati naslov i godinu izdanja knjiga koje se koriste kao neobavezna literatura na predmetima sa organizacione jedinice Automatika i robotika a starije su od bar jedne knjige koja se koristi kao neobavezna literatura na predmetima sa organizacione jedinice Telekomunikacije.*

```
SELECT naslov, godIzdanja FROM knjiga
  INNER JOIN literatura ON knjiga.sifKnjiga = literatura.sifKnjiga
  INNER JOIN pred ON literatura.sifPred = pred.sifPred
  INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
    WHERE nazOrgjed = 'Automatika i robotika'
    AND YEAR(CURRENT_DATE) - godIzdanja > ANY
      (SELECT YEAR(CURRENT_DATE) - godIzdanja FROM knjiga
         INNER JOIN literatura ON knjiga.sifKnjiga = literatura.sifKnjiga
         INNER JOIN pred ON literatura.sifPred = pred.sifPred
         INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
           WHERE nazOrgjed = 'Telekomunikacije')
```

Q6: *Ispisati naslove i godine izdanja knjiga koje imaju primjerke koji su nabavljeni bar 300 dana nakon što je prvu posudbu obavio bilo koji nastavnik.*

```
SELECT DISTINCT naslov, godIzdanja FROM knjiga
  INNER JOIN primjerak ON knjiga.sifKnjiga = primjerak.sifKnjiga
    WHERE DATE_ADD(datNabavka, INTERVAL 300 DAY) >=
      (SELECT MIN(datPosudba) FROM posudba
         INNER JOIN nast ON posudba.sifClan = nast.sifClan)
```

Q7: *Ispisati ime, prezime i datum rođenja nastavnika zaposlenih na organizacionoj jedinici 'Elektrotehnika i sistemi konverzije energije' čija je starost u danima veća od ukupnog broja dana zaduženja svih primjeraka knjiga koje se koriste kao literatura na predmetu 'Teorija elektromagnetnih polja'*

```
SELECT imeClan, prezClan, datRod FROM nast
  INNER JOIN clan ON nast.sifClan = clan.sifClan
  INNER JOIN orgjed ON nast.sifOrgjed = orgjed.sifOrgjed
    WHERE nazOrgjed = 'Elektrotehnika i sistemi konverzije energije'
```

```
    AND DATEDIFF(CURRENT_DATE, datRod) >
      (SELECT SUM(DATEDIFF(datVracanje, datPosudba)) FROM posudba
         INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj
         INNER JOIN knjiga ON primjerak.sifKnjiga = knjiga.sifKnjiga
         INNER JOIN literatura ON knjiga.sifKnjiga = literatura.sifKnjiga
         INNER JOIN pred ON literatura.sifPred = pred.sifPred
           WHERE nazPred ='Teorija elektromagnetnih polja')
```

Q8: *Ispisati naslov i godinu izdanja knjiga koje su starije od broja posudbi svih primjeraka te knjige u biblioteci.*

```
SELECT naslov, godIzdanja FROM knjiga
  WHERE YEAR(CURRENT_DATE) - godIzdanja >
    (SELECT COUNT(*) FROM posudba
       INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj
         WHERE primjerak.sifknjiga = knjiga.sifKnjiga)
```

Q9: *Ispisati naziv, skraćenicu i broj ECTS kredita predmeta čiji je broj ECTS kredita veći od svih ostalih predmeta sa iste organizacione jedinice.*

```
SELECT nazPred, skrPred, brojECTS FROM pred
  WHERE brojECTS >
    (SELECT AVG(brojECTS) FROM pred ostali
       WHERE pred.sifOrgjed = ostali.sifOrgjed
       AND pred.sifPred <> ostali.sifPred)
```

Q10: *Ispisati naslove i godine izdanja knjiga za koje nije posuđen niti jedan primjerak u godini u kojoj su izdate.*

```
SELECT naslov, godIzdanja FROM knjiga
  WHERE NOT EXISTS
    (SELECT * FROM posudba
       INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj
         WHERE primjerak.sifKnjiga = knjiga.sifKnjiga
         AND knjiga.godIzdanja = YEAR(datPosudba))
```

Q11: *Ispisati naslov knjige i ime i prezime autora za sve knjige koje imaju samo jednog autora.*

```
SELECT naslov, imeAutor, prezAutor FROM knjiga
  INNER JOIN napisao ON knjiga.sifKnjiga = napisao.sifKnjiga
  INNER JOIN autor ON napisao.sifAutor = autor.sifAutor
    WHERE NOT EXISTS
      (SELECT * FROM napisao nap
         WHERE nap.sifKnjiga = napisao.sifknjiga
         AND nap.sifAutor <> napisao.sifAutor)
```

Formiranje grupa zapisa

Formiranje grupa zapisa u upitu ustvari podrazumijeva da se n-torke u rezultatu podijele u manje grupe i obavlja se u `GROUP BY` dijelu `SELECT` naredbe. U `GROUP BY` dijelu se navodi popis atributa ili izraza u odnosu na čije vrijednosti će se obaviti grupisanje n-torki. Grupe n-torki se formiraju na način da sve n-torke koje imaju jednake vrijednosti atributa ili izraza navedenih u `GROUP BY` dijelu čine jednu grupu. Konačan rezultat upita sadrži onoliko n-torki koliko je formirano grupa što znači da se u rezultatu za svaku grupu pojavljuje po jedna n-torka.

LITERATURA

sifKnjiga	sifPred	rbrVaznost	obavezna
1234	101	1	TRUE
1211	107	2	TRUE
1227	103	1	TRUE
1234	102	5	FALSE
1218	107	1	TRUE
1234	103	3	FALSE
1211	101	2	TRUE
1218	102	2	TRUE
1227	105	3	FALSE
1234	105	1	TRUE
1218	103	2	TRUE
1234	107	3	TRUE
1227	101	3	FALSE
1218	105	2	TRUE

Primjer: *Ispisati šifre knjiga koje se koriste kao obavezna literatura*

```
SELECT sifKnjiga FROM literatura
  WHERE obavezna
  GROUP BY sifKnjiga
```

Može se smatrati da se upit obavlja tako da se najprije dohvate n-torke koje zadovoljavaju uslov dohvata, a potom se grupišu prema vrijednosti atributa navedenim u `GROUP BY` dijelu:

sifKnjiga	sifPred	rbrVaznost	obavezna
1234	101	1	TRUE
1234	105	1	TRUE
1234	107	3	TRUE
1211	107	2	TRUE
1211	101	2	TRUE
1227	103	1	TRUE
1218	107	1	TRUE
1218	102	2	TRUE
1218	103	2	TRUE
1218	105	2	TRUE

Kao što je rečeno, za svaku grupu se u rezultatu pojavljuje samo jedna n-torka:

sifKnjiga
1234
1211
1227
1218

Ovaj upit je poslužio samo za ilustraciju grupisanja u `SELECT` naredbi jer bi se isti rezultat dobio na jednostavniji (i korektniji) način upitom:

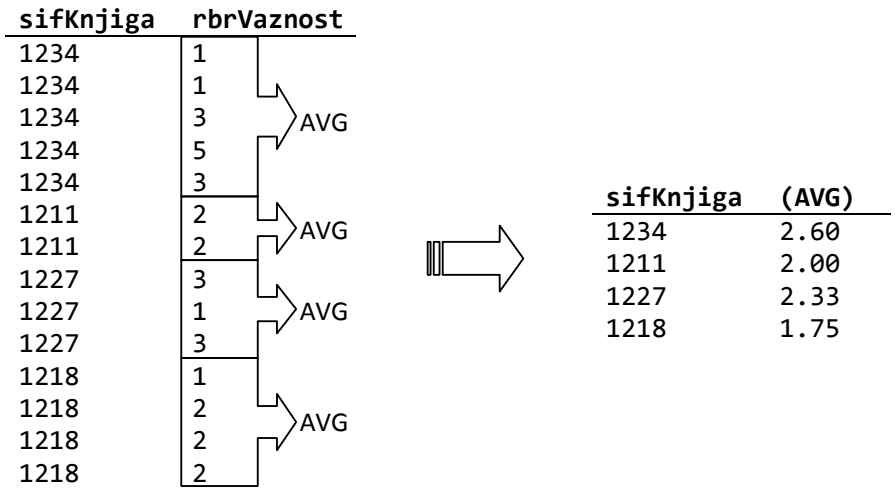
```
SELECT DISTINCT sifKnjiga FROM literatura
  WHERE obavezna
```

Međutim, grupisanje je vrlo korisno u kombinaciji sa agregatnim funkcijama s obzirom da se mogu upotrijebiti da bi se dobile agregirane informacije za svaku grupu. Zbog toga se `GROUP BY` dio vrlo rijetko koristi ukoliko u upitu ne postoji agregatna funkcija. U kombinaciji sa `GROUP BY` agregatna funkcija se primjenjuje individualno za svaku formiranu grupu.

Primjer: *Ispisati prosječnu važnost svake knjige.*

```
SELECT sifKnjiga, AVG(rbrVaznost) FROM literaura
  GROUP BY sifKnjiga
```

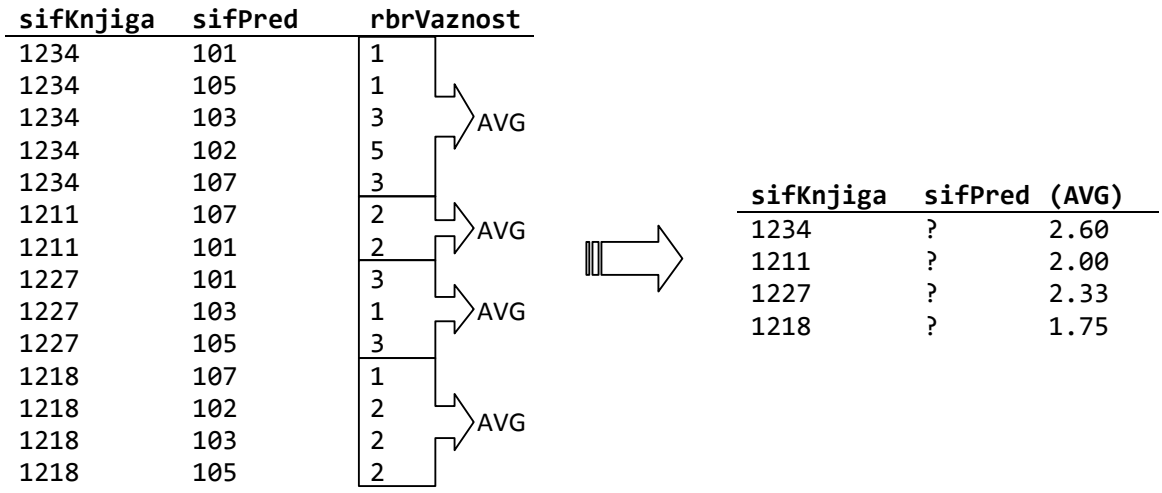
U ovom upitu agregatna funkcija `AVG` se primjenjuje na svaku grupu formiranu na osnovi vrijednosti atributa sifknjiga:



Na ovaj način je prevaziđeno ograničenje da se agregatne funkcije ne mogu koristiti u listi za selekciju u kombinaciji sa drugim atributima ili izrazima čime se omogućava postavljanje raznovrsnijih i korisnijih upita. Pri tome se mora voditi računa da se u `GROUP BY` dijelu navedu svi atributi i izrazi koji se koriste u listi za selekciju a nisu argumenti agregatnih funkcija.

```
SELECT sifKnjiga, sifPred, AVG(rbrVaznost) FROM literaura
  GROUP BY sifKnjiga
```

Gornji upit nije korektan jer se grupisanje obavlja samo po atributu `sifKnjiga` što znači da se u rezultatu za svaku šifru knjige (jednu vrijednost atributa `sifKnjiga`) pojavljuje samo jedna n-torka. Kako se u relaciji literatura uz istu vrijednost šifre knjige pojavljuju različite vrijednosti atributa `sifPred`, onda se ne može odrediti koju od tih vrijednosti treba ispisati u rezultatu.



Rezultat ovakvog upita bi imao smisla samo ukoliko bi se za svaku grupu pojavljivale iste vrijednosti atributa ili izraza koji nisu uključeni u `GROUP BY` dio, a to bi se desilo u slučaju da su ti atributi ili izrazi funkcijski zavisni o atributima navedenim u `GROUP BY` dijelu, odnosno da se grupisanje vrši na osnovu vrijednosti primarnog ključa relacije. Ova opciona osobina (što znači da je SUBP-i nisu obavezni implementirati) je dozvoljena od SQL99 standarda. Od verzije MySQL 5.7.5 implementirana je detekcija ove funkcijske zavisnosti ukoliko je uključena opcija `ONLY_FULL_GROUP_BY` te će u tom slučaju svi upiti koji u `GROUP BY` dijelu ne koriste atribute ili izraze iz liste za selekciju koji nisu funkcijski ovisni o onima navedenim u `GROUP BY` dijelu završiti sa pogreškom. U slučaju da je ova opcija isključena, gornji upiti će biti dozvoljeni i neće završiti sa pogreškom te će u rezultatu za svaku grupu biti ispisana vrijednost atributa koju server proizvoljno odabere iz grupe (obično koja se dohvati za prvu n-troku u toj grupi). Međutim, ovakav rezultat nema smisla niti koristi i ne treba ga pisati na ovaj način.

Nasuprot navedenom, u `GROUP BY` dijelu je dozvoljeno koristiti atribute koji se ne nalaze u listi za selekciju, npr.

```
SELECT AVG(rbrVaznost) FROM literaura
  GROUP BY sifKnjiga
```

(AVG)
2.60
2.00
2.33
1.75

U standardnom SQL-u u `GROUP BY` dijelu dozvoljeno je koristiti samo atribute relacija (i to isključivo njihove originalne nazive), dok je u većini SUBP-a, pa tako i u MySQL-u, u `GROUP BY` dijelu uz atribute dozvoljeno koristiti i izraze. U mjesto naziva atributa ili izraza u većini SUBP-a je dozvoljeno navoditi redne brojeve atributa ili izraza iz liste za selekciju čime se skraćuje pisanje upita. Osim rednih brojeva iz liste za selekciju, MySQL dozvoljava i korištenje zamjenskih naziva atributa ili izraza (*alias*) u `GROUP BY` dijelu.

MySQL
```
SELECT sifOrgjed, nazOrgjed, AVG(koef) FROM nast
  INNER JOIN orgjed ON nast.sifOrgjed = orgjed.sifOrgjed
    GROUP BY 1, 2
```

MySQL
```
SELECT koef*100 AS k, COUNT(*) FROM nast
  GROUP BY k  --ili GROUP BY koef*100 ili GROUP BY 1
```

Uslovi nad formiranim grupama zapisa

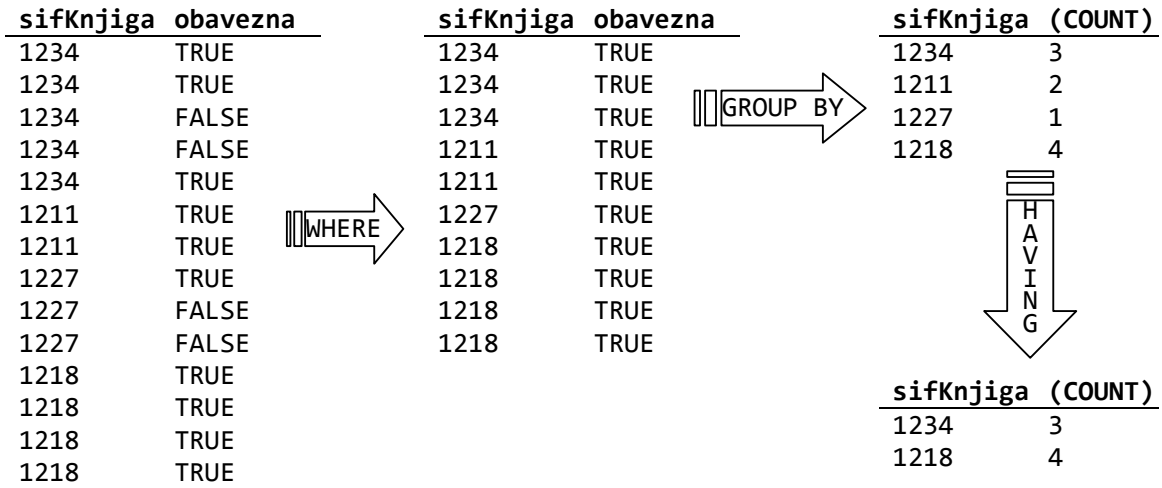
Ukoliko se u rezultatu upita u kojem je obavljeno grupisanje ne žele prikazati sve n-torke (za svaku grupu) tada je potrebno definisati uslov koji rezultujuće n-torke grupe trebaju ispuniti da bi se pojavile u rezultatu. Postavljanje uslova nad tim n-torkama se obavlja u `HAVING` dijelu `SELECT` naredbe. Dakle, u `WHERE` dijelu `SELECT` naredbe se postavljaju uslovi koje moraju zadovoljiti n-torke da bi bile kandidati za kreiranje grupa definiranih `GROUP BY` dijelom, dok se u `HAVING` dijelu `SELECT` naredbe postavljaju uslovi koje rezultujuće grupe trebaju zadovoljiti da bi se pojavile u rezultatu upita. U `HAVING` dijelu se za definisanje uslova mogu koristiti jedino atributi koji su navedeni u `GROUP BY` dijelu i agregatne funkcije pri čemu se kao argumenti agregatnih funkcija mogu koristiti i atributi iz relacija upita koji nisu navedeni u `GROUP BY` dijelu.

U MySQL-u se u `HAVING` dijelu mogu koristiti i atributi koji nisu navedeni u `GROUP BY` dijelu pod istim uslovima i na isti način kako je to prethodno objašnjeno za atribute u listi za selekciju (opcija `ONLY_FULL_GROUP_BY` u MySQL-u).

Primjer: *Ispisati šifre knjiga i broj predmeta na kojima se knjiga koristi kao obavezna literatura, ali samo onih knjiga koje se koriste kao obavezna literatura na više od 2 predmeta.*

```
SELECT sifKnjiga, COUNT(*) FROM literaura
  WHERE obavezna
  GROUP BY sifKnjiga
  HAVING COUNT(*) > 2
```

U ovom upitu agregatna funkcija `AVG` se primjenjuje na svaku grupu formiranu na osnovu vrijednosti atributa sifknjiga te se izvršavanje kompletnog upita može ilustrovati na sljedeći način:



U HAVING dijelu SELECT naredbe može se koristiti bilo koja vrsta uslova koja se koristi i u WHERE dijelu što znači da je dozvoljeno korištenje podupita, koreliranih podupita i kombiniranje uslova upotrebom logičkih operatora i zagrada.

Primjer: *Ispisati šifre knjiga i broj predmeta na kojima se knjiga koristi kao obavezna literatura, ali samo onih knjiga koje se koriste kao obavezna literatura na više predmeta nego što je njena prosječna važnost kao literature na svim predmetima.*

```
SELECT sifKnjiga, COUNT(*) FROM literaura
  WHERE obavezna
  GROUP BY sifKnjiga
  HAVING COUNT(*) > (SELECT AVG(rbrVaznost) FROM literatura druga
                       WHERE druga.sifKnjiga = literatura.sifKnjiga)
```

Ispitivanje postojanja funkcijske zavisnosti na osnovu trenutnog sadržaja relacije

Među atributima (ili skupovima atributa) X i Y relacije postoji funkcijska zavisnost X → Y (X funkcijski određuje Y ili Y funkcijski zavisi o X) ako je jednoj vrijednosti x od X u svakom trenutku pridružena jedna i samo jedna vrijednost y od Y. Ako bi se ova definicija funkcijske zavisnosti predstavila operacijama relacijske algebre to bi značilo da u relaciji postoji funkcijska zavisnost X → Y ako za svaku vrijednost x od X projekcija relacije po atributu Y (tj. $\pi_Y(\sigma_{X=x}(r))$) ima najviše jednu n-torku.

Kako je operacije relacijske algebre moguće implementirati korištenjem SQL-a to se onda može napisati upit koji bi se koristio za provjeru postojanja funkcijske zavisnosti u relaciji. Naravno, ova provjera se obavlja na osnovu n-torki koje trenutno postoje u relaciji te se postojanje funkcijske zavisnosti potvrđuje samo za trenutni sadržaj relacije, ali se ne može potvrditi i za relacijsku shemu. Stvarno postojanje funkcijskih zavisnosti se određuje na osnovu značenja atributa.

Kada se formira upit kojim se ispituje postojanje funkcijske zavisnosti X → Y potrebno je n-torke grupisati prema atributu (ili atributima) X te provjeriti da li u svakoj grupi postoji najviše jedna različita vrijednost od Y, odnosno ispisati one grupe za koje postoji

više od jedna različita vrijednost od Y. Ukoliko se u rezultatu takvog upita pojavi bar jedna n-torka (što znači da za jednu vrijednost od X za koju se formira grupa imamo više različitih vrijednosti od Y) onda funkcijska zavisnost X → Y u relaciji ne vrijedi.

Primjer: *Napisati naredbu kojom će se u relaciji literatura ispitati postojanje funkcijske zavisnosti { sifKnjiga, rbrVaznost } → sifPred*

```
SELECT sifKnjiga, rbrVaznost FROM literatura
  GROUP BY sifKnjiga, rbrVaznost
  HAVING COUNT(DISTINCT sifPred) > 1
```

sifKnjiga	rbrVaznost
1234	1
1211	2
1227	3
1218	2
1234	3

Za dobivene n-torke odnosno kombiacije atributa `sifKnjiga` i `rbrVaznost` se pojavljuju više od jedne različite vrijednosti atributa `sifPred` te je ovim upitom pokazano da ispitivana funkcijska zavisnost ne vrijedi.

Primjer: *Napisati naredbu kojom će se u relaciji literatura ispitati postojanje funkcijske zavisnosti { sifPred, rbrVaznost } → sifKnjiga*

```
SELECT sifPred, rbrVaznost FROM literatura
  GROUP BY sifPred, rbrVaznost
  HAVING COUNT(DISTINCT sifKnjiga) > 1
```

Kako prethodni upit ne vraća niti jednu n-torku, što znači da se za svaku kombinaciju atributa `sifPred` i `rbrVaznost` pojavljuje najviše jedna različita vrijednost atributa `sifKnjiga`, to je pokazano da ispitivana funkcijska zavisnost za trenutni sadržaj relacije vrijedi. Ovo ne znači da vrijedi i za relacijsku shemu te da (`sifPred`, `rbrVaznost`) može biti alternativni ključ u relaciji `literatura`, jer se može pojaviti n-torka (slučaj) da su na istom predmetu dvije različite knjige podjednako važne kao literatura (ista šifra predmeta i redni broj važnosti, a različite šifre knjiga).

S obzirom da u `HAVING` dijelu nije moguće koristiti `COUNT(DISTINCT A, B)`, ovakvim upitom nije moguće ispitati funkcijske zavisnosti u kojima desna strana sadrži više od jednog atributa, npr. Y = {A, B}. Međutim, kako za funkcijske zavisnosti vrijedi pravilo unije koje kaže da ukoliko u relaciji vrijedi da X → A i X → B, tada u relaciji vrijedi i da X → AB (X → A ∪ B), tj. X → Y, tada se upitima može pojedinačno ispitati postojanje funkcijskih zavisnosti X → A i X → B te ukoliko one postoje zaključiti da postoji i funkcijska zavisnost X → Y.

Pohrana rezultata upita u privremenu relaciju

Postoje slučajevi kada upiti postaju previše komplikovani da bi se mogli implementirati jednom `SELECT` naredbom. Takvi složeni upiti se najčešće razlažu na dva ili više jednostavnijih upita čiji se razultati međusobno kombinuju kako bi se dobio konačni rezultat. Da bi se rezultati tih jednostavnijih upita mogli koristiti u drugim upitima potrebno ih je privremeno negdje pohraniti te ih izbrisati nakon što više ne budu potrebni. U tu svrhu se u SQL-u najčešće koriste privremene relacije koje, kao što je već istaknuto, imaju sve karakteristike trajne relacije kreirane naredbom `CREATE TABLE`, osim što nisu vidljive u riječniku podataka i brišu se nakon što se završi sesija s bazom podataka tokom koje su kreirane.

Da bi se kreirala privremena relacija i napunila podacima (zapisima) koji su rezultat `SELECT` naredbe u MySQL-u se koristi `CREATE TEMPORARY TABLE` naredba u sljedećem opštem obliku:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    [IGNORE | REPLACE]
    [AS] query_expression
```

Kao što se može vidjeti, svi gore navedeni dijelovi naredbe izuzev naziva privremene relacije te `SELECT` naredbe (`query_expression`) kojom se puni privremena relacija su opcioni te ovdje neće biti predmet razmatranja. `SELECT` naredba koja se koristi za kreiranje privremene relacije može biti bilo koja `SELECT` naredba u obliku do sada opisanom u knjizi uz ograničenje da ne može imati `ORDER BY` dio. Privremena relacija koja nastaje kao rezultat će imati shemu koja je određena listom za selekciju `SELECT` naredbe što znači da će nazivi atributa te relacije odgovarati nazivima atributa iz liste za selekciju ili njihovim *alias* nazivima ukoliko su navedeni. Tipovi podataka će odgovarati tipovima podataka atributa iz relacija koje se koriste u `SELECT` naredbi ili rezultujućim tipovima podatka za izraze ako se koriste u listi za selekciju. U slučaju da lista za selekciju sadrži izraze moraju se navesti *alias* nazivi kako bi se definisali nazivi atributa u privremenoj relaciji.

Primjer: *Kreirati privremenu relaciju koefTmp koja sadrži šifre i nazive organizacionih jedinica i pripadajući prosječni koeficijent za platu nastavnika zaposlenih u toj organizacionoj jedinici te je istovremeno napuniti odgovarajućim podacima na osnovu sadržaja relacija nast i orgjed.*

```
CREATE TEMPORARY TABLE koefTmp AS
  SELECT orgjed.sifOrgjed, nazOrgjed, AVG(koef) prosjKoef FROM orgjed
    INNER JOIN nast ON orgjed.sifOrgjed = nast.sifOrgjed
    GROUP BY 1, 2
```

Privremena relacija `koefTemp` sa atributima `sifOrgjed`, `nazOrgjed` i `prosjKoef` se može tokom sesije koja ju je kreirala koristiti kao bilo koja druga relacija koja postoji u bazi podataka, sve dok se sesija ne završi ili dok se relacija eksplicitno ne izbriše naredbom `DROP`

TABLE. Sada se npr. može dahvatiti najveći maksimalni prosječni koeficijent što recimo nije bilo moguće napraviti na način MAX(AVG(koef)):

```
SELECT MAX(prosjKoef) FROM prosjTmp
```

Primjeri upita za vježbu

Q1: *Ispisati ime i prezime studenta i broj posudbi koje je obavio u toku tekuće godine. Zapise sortirati po broju posudbi a članove koji su eventualno obavili isti broj posudbi sortirati po prezimenu i imenu.*

```
SELECT imeClan, prezClan, COUNT(*) FROM stud
  INNER JOIN clan ON stud.sifClan = clan.sifClan
  INNER JOIN posudba ON clan.sifClan = posudba.sifClan
  WHERE YEAR(datPosudba) = YEAR(CURRENT_DATE)
  GROUP BY clan.sifClan, 1, 2
  ORDER BY 3 DESC, 2, 1
```

Q2: *Ispisati organizacione jedinice (šifru i naziv) i broj knjiga koji se koristi kao obavezna literatura na predmetima koji im pripadaju.*

```
SELECT orgjed.sifOrgjed, nazOrgjed, COUNT(DISTINCT sifknjiga) FROM orgjed
  INNER JOIN pred ON orgjed.sifOrgjed = pred.sifOrgjed
  INNER JOIN literatura ON pred.sifPred = literatura.sifPred
  WHERE obavezna
  GROUP BY 1, 2
```

Q3: *Ispisati nazive kantona i prosječan koeficijent za platu koji imaju nastavnici koji u njima stanuju.*

```
SELECT nazKanton, AVG(koef) FROM nast
  INNER JOIN clan ON nast.sifClan = clan.sifClan
  INNER JOIN mjesto ON clan.pbrStan = mjesto.pbr
  INNER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
  GROUP BY kanton.sifkanton, nazKanton
```

Q4: *Ispisati rang listu knjiga po broju obavljenih posudbi njihovih primjeraka. U rang listi se nalazi naziv knjige i godina izdanja te broj obavljenih posudbi primjeraka knjige, poredano tako da su na početku rang liste knjige koje su najviše posuđivane. Knjige koji imaju jednaki broj obavljenih posudbi, poredani su po naslovu.*

```
SELECT naslov, godIzdanja, COUNT(*) FROM knjiga
  INNER JOIN primjerak ON knjiga.sifKnjiga = primjerak.sifKnjiga
  INNER JOIN posudba ON ON posudba.invBroj = primjerak.invBroj
  GROUP BY knjiga.sifKnjiga, 1, 2
  ORDER BY 3 DESC, 2, 1
```

Q5: *Za svaki radni dan u sedmici ispisati broj rođenih studenata u tom danu. Rezultat sortirati po broju rođenih članova u opadajućem poretku.*

```
SELECT WEEKDAY(datRod), COUNT(*) FROM clan
  INNER JOIN stud ON clan.sifClan = stud.sifClan
  WHERE WEEKDAY(datRod) < 5
  GROUP BY 1
  ORDER BY 2 DESC
```

Q6: *Za svaku knjigu koja nikada nije posuđena ispisati naslov, godinu izdanja, broj primjeraka knjige u biblioteci i prosječnu starost primjeraka.*

```
SELECT naslov, godIzdanja, COUNT(*),
       AVG(DATEDIFF(CURRENT_DATE, datNabavka)) FROM knjiga
  INNER JOIN primjerak ON knjiga.sifKnjiga = primjerak.sifKnjiga
  WHERE sifKnjiga NOT IN
    (SELECT DISTINCT sifKnjiga FROM primjerak
       INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj)
GROUP BY knjiga.sifKnjiga, 1, 2
```

Q7: *Ispisati sve podatke o predmetima koji imaju više od 2 predmeta za preduslov.*

```
SELECT pred.* FROM pred
  INNER JOIN preduslov ON pred.sifPred = preduslov.sifPred
  GROUP BY 1, 2, 3, 4, 5
  HAVING COUNT(*) > 2
```

Q8: *Ispisati šifru i naziv kantona i broj rođenih članova u tom kantonu. Ispisati samo podatke za one kantone u kojima je broj rođenih članova veći od broja članova koji u njima stanuju.*

```
SELECT kanton.sifKanton, nazKanton, COUNT(pbrRod) FROM kanton
  INNER JOIN mjesto ON kanton.sifKanton = mjesto.sifKanton
  INNER JOIN clan ON mjesto.pbr = clan.pbrRod
  GROUP BY 1, 2
  HAVING COUNT(pbrRod) >
         (SELECT COUNT(pbrStan) FROM clan
            INNER JOIN mjesto stan ON clan.pbrStan = mjesto.pbr
            WHERE stan.sifKanton = kanton.sifKanton)
```

Q9: *Ispisati naziv, skraćenicu i broj ECTS kredita predmeta čije knjige koje se koriste literatura imaju najveći broj posudbi.*

```
SELECT nazPred, skrPred, brojECTS FROM pred
  INNER JOIN literatura ON pred.sifPred = literatura.sifPred
  INNER JOIN primjerak ON primjerak.sifKnjiga = literatura.sifKnjiga
  INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
  GROUP BY pred.sifPred, 1, 2, 3
  HAVING COUNT(*) >= ALL
       (SELECT COUNT(*) FROM posudba
        INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj
        INNER JOIN literatura ON primjerak.sifKnjiga = literatura.sifKnjiga
          GROUP BY sifPred)
```

Q10: *Napisati SELECT naredbe kojima se može ispitati postojanje sljedećih funkcijskih zavisnosti u relacijama na osnovu njihovog trenutnog sadržaja:*

a) `sifKnjiga`, `rbrAutor` → `sifAutor` *u relaciji* `napisao`

```
SELECT sifKnjiga, rbrAutor FROM autor
GROUP BY 1, 2
HAVING COUNT(DISTINCT sifAutor) > 1
```

b) `sifKnjiga`, `datNabavka` → `invBroj` *u relaciji* `primjerak`

```
SELECT sifKnjiga, datNabavka FROM primjerak
GROUP BY 1, 2
HAVING COUNT(DISTINCT invBroj) > 1
```

c) `sifKnjiga`, `sifPred` → `rbrVaznost`, `obavezna` *u relaciji* `literatura`

```
SELECT sifKnjiga, sifPred FROM literatura
GROUP BY 1, 2
HAVING COUNT(DISTINCT rbrVaznost) > 1
```

```
SELECT sifKnjiga, sifPred FROM literatura
GROUP BY 1, 2
HAVING COUNT(DISTINCT obavezna) > 1
```

Q11: *Kreirati privremenu relaciju autorTmp čija je shema jedanka shemi relacije autor uz dodatak atributa brojPos koji sadrži broj obavljenih posudbi knjiga koje je autor napisao te je napuniti podacima na osnovu trenutnog sadržaja baze podataka.*

```
CREATE TEMPORARY TABLE autorTmp AS
  SELECT autor.*, COUNT(*) brojPos FROM autor
    INNER JOIN napisao ON autor.sifAutor = napisao.sifAutor
    INNER JOIN primjerak ON napisao.sifKnjiga = primjerak.sifKnjiga
    INNER JOIN posudba ON primjerak.invBroj = posudba.invBroj
    GROUP BY 1, 2, 3
```

Q12: *Ispisati podatke o organizacionim jedinicima koje imaju veći broj zaposlenih nastavnika od prosječnog proja zaposlenih nastavnika po ostalim organizacionim jedinicama ili je prosječni koeficijent njihovih zaposlenih nastavnika veći od prosječnih koeficijenata ostalih organizacionih jedinica.*

```
CREATE TEMPORARY TABLE orgjedTmp AS
  SELECT orgjed.sifOrgjed, COUNT(*) brojNast, AVG(koef) prosjKoef
    FROM orgjed
    INNER JOIN nast ON orgjed.sifOrgjed = nast.sifOrgjed
    GROUP BY 1
```

```
SELECT orgjed.* FROM orgjed
  INNER JOIN nast ON orgjed.sifOrgjed = nast.sifOrgjed
  GROUP BY 1, 2
  HAVING (COUNT(*) > (SELECT AVG(brojNast) FROM orgjedTmp
                        WHERE orgjedTmp.sifOrgjed <> orgjed.sifOrgjed)
         OR AVG(koef) > (SELECT MAX(prosjKoef) FROM orgjedTmp
                           WHERE orgjedTmp.sifOrgjed <> orgjed.sifOrgjed))
```

SQL NAREDBE ZA MANIPULACIJU PODACIMA U BAZI PODATAKA

Od predstavljene četiri osnovne SQL naredbe za rad sa podacima u bazi podataka, jedino `SELECT` naredba ne utiče na sadržaj baze podataka i kako je opisano u prethodnom poglavlju koristi se za dohvat i prikaz podataka. Preostale tri osnovne SQL naredbe služe za manipuliranje podacima mijenjajući sadržaj baze podataka na način da stvaraju nove podatke ili mijenjaju (ažuriraju) i brišu postojeće podatke. U nastavku ovog poglavlja će biti detaljno predstavljene svaka od ovih naredbi.

INSERT naredba za unos podataka u relaciju

`INSERT` naredba se koristi za unos jednog ili više zapisa u relaciju te u skladu s tim postoje dva osnovna oblika upotrebe te naredbe koja će biti predstavljena u nastavku.

Jednostavna INSERT naredba

Opšti oblik `INSERT` naredbe kojom se unosi jedna n-torka u relaciju, zbog čega se i naziva jednostavni `INSERT` (*single row INSERT*) je sljedeći:

```
INSERT INTO table_name [(col_name, ...)]
  VALUES ({expr | NULL | DEFAULT}, ...)
```

`table_name` je naziv relacije u koju se unosi n-torka. Nakon naziva relacije u okruglim zagradama se može navesti lista naziva atributa (`col_name`) za koje se unose vrijednosti navedne u `VALUES` listi. Lista naziva atributa obavezno se navodi ukoliko se ne unose vrijednosti za sve atribute relacije ili ukoliko se vrijednosti atributa unose drugačijim redoslijedom od redoslijeda atributa u relaciji (navedenog prilikom kreiranja relacije). Ukoliko se lista naziva atributa izostavi, smatra se da se unose vrijednosti za sve atribute relacije i u `VALUES` listi se mora navesti onoliko vrijednosti koliko je atributa u relaciji te redoslijed mora odgovarati redoslijedu atributa navedenom prilikom kreiranja relacije.

S obzirom da se u listi naziva atributa ne moraju navesti svi atributi relacije u koju se unosi n-torka, za one atribute koji nisu navedeni u listi sistem će unijeti podrazumijevanu (*default*) vrijednost ukoliko je ona definisana za atribut, a ukoliko nije unijeti će `NULL`

vrijednost. Ukoliko za atribut koji nije naveden u listi naziva atributa prilkom kreiranja relacije nije definisana *default* vrijednost, a definisano je da atribut ne može poprimiti `NULL` vrijednost (`NOT NULL` ograničenje), ovakva `INSERT` naredba će završiti sa greškom. Osim toga, za svaki atribut relacije moguće je eksplicitno unijeti podrazumijevanu ili `NULL` vrijednost navođenjem ključnih riječi `DEFAULT` ili `NULL` na odgovarajućem mjestu u `VALUES` listi.

Ukoliko se u `VALUES` listi navode konstantne vrijednosti onda znakovne i datumske vrijednosti moraju biti navedene pod jednostrukim navodnicima (`'const'`), dok se numeričke vrijednosti navode bez navodnika. Bilo da je riječ o konstantim vrijednostima ili se one dobijaju izračunavanjem izraza (*expression*), tipovi podataka navedeni u `VALUES` listi moraju biti kompatibilni sa tipovima podataka kojim je definisan odgovarajući atribut. Premda će većina SUBP-a u ovom slučaju napraviti implicitnu konverziju tipova podataka, ovakav unos je logički pogrešan i nedopustiv te korisnik o ovome mora voditi računa kako bi se očuvao integritet podataka u bazi.

Primjer: *U relaciju orgjed unijeti podatke o organizacionoj jedinci sa šifrom 1005 i nazivom Tehnički odgoj i informatika.*

```
INSERT INTO orgjed VALUES (1005, 'Tehnički odgoj i informatika')
```

```
INSERT INTO orgjed (nazOrgjed, sifOrgjed)
  VALUES ('Tehnički odgoj i informatika', 1005)
```

Primjer: *U relaciju orgjed unijeti podatke o organizacionoj jedinici sa šifrom 1006 i nepoznatim nazivom.*

```
INSERT INTO orgjed (sifOrgjed) VALUES (1006)
```

U posljednjoj naredbi za vrijednost atributa `nazOrgjed` u relaciji `orgjed` će biti unesena podrazumijevana (*default*) vrijednost, a ukoliko ona nije navedena biće unesena `NULL` vrijednost. Ako je atribut definisan sa ograničenjem `NOT NULL` naredba će završiti sa pogreškom.

Složena INSERT naredba

Za razliku od prethodno opisanog jednostavnog oblika `INSERT` naredbe kojim se mogla unijeti samo jedna n-torka u relaciju, složeni oblik `INSERT` naredbe (*multiple row INSERT*) omogućava unos većeg broja n-torki i ima sljedeći oblik:

```
INSERT INTO table_name [(col_name, ...)]
  SELECT ...
```

Kao što se može primijetiti, ovim oblikom se u relaciju upisuju sve n-torke koje su dobivene kao rezultat navedne `SELECT` naredbe. Pravila koja su vrijedila za jednostavni oblik `INSERT` naredbe sada vrijede i za ovaj složeni oblik, tako da lista za selekciju `SELECT` naredbe koja se koristi treba ispoštovati pravila koja su vrijedila za `VALUES` listu. `SELECT` naredba koja se koristi da bi se odredile n-torke koje će biti upisane u relaciju može imati

bilo koji oblik opisan u poglavlju o SELECT naredbi s tim da u svom FROM dijelu ne može sadržavati naziv relacije u koju se INSERT naredbom unose podaci.

Primjer: *Postojeću relaciju orgjedNast koja ima istu strukturu kao relacija orgjed napuniti podacima o organizacionim jedincama u kojima su zaposleni nastavnici koji stanuju u Tuzli.*

```
INSERT INTO orgjedNast (sifOrgjed, nazOrgjed)
  SELECT DISTINCT orgjed.sifOrgjed, nazOrgjed FROM orgjed
    INNER JOIN nast ON orgjed.sifOrgjed = nast.sifOrgjed
    INNER JOIN clan ON nast.sifClan = clan.sifClan
    INNER JOIN mjesto ON clan.pbrStan = mjesto.pbr
    WHERE nazMjesto = 'Tuzla'
```

MySQL implementacija INSERT naredbe

Način na koji MySQL implementira INSERT naredbu, osim dodatnih opcija i mogućnosti koje nudi, ne slijedi u potpunosti pravila SQL standarda koja su prethodno opisana. Opšti oblik INSERT naredbe u MySQL-u je sljedeći:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name
  [PARTITION (partition_name,...)]
  { [(col_name,...)] {VALUES Clause | SELECT Clause} | SET Clause }
  [ ON DUPLICATE KEY UPDATE
      col_name=expr [, col_name=expr] ... ]
```

Kao što se iz opšteg oblika naredbe može primijetiti, uz dva opisana oblika INSERT naredbe, MySQL podržava i oblik sa SET Clause koji ima sljedeći oblik:

```
SET col_name={expr | DEFAULT | NULL}, ...
```

Osim ovoga, oblik INSERT naredbe sa VALUES omogućava unos više od jedne n-torke te ima sljedeći opšti oblik:

```
[(col_name,...)] VALUES ({expr | NULL | DEFAULT}, ...), (...), ...
```

Primjer: *U relaciju orgjed unijeti podatke o dvije nove organizacione jedinice sa šiframa 1005 i 1006 i nazivima Biomedicinski inženjering i Tehnički odgoj i informatika.*

```
MySQL INSERT INTO orgjed (sifOrgjed, nazOrgjed)
      VALUES (1005, 'Biomedicinski inženjering'),
             (1006, 'Tehnički odgoj i informatika')
```

S obzirom da se u knjizi ne bavimo particijama relacija, dio PARTITION će biti zanemaren, kao i opcija DELAYED jer je zastarila u novijim verzijama MySQL-a.

Ukoliko se u MySQL-u ne koristi striktni način rada (*strict SQL mode*), za atribute koji prilikom kreiranja relacije nemaju definisanu default vrijednost i NOT NULL ograničenje ukoliko se u INSERT naredbi ne navedu vrijednosti, sistem će postaviti podrazumjevanu

vrijednost za tip podatka kojeg je atribut. Dakle, za razliku od standardnog SQL-a ovakva `INSERT` naredba u MySQL-u ne završava sa pogreškom. Izraz (`expr`) koji se koristi za dodjeljivanje vrijednosti može referisati na bilo koji atribut čija je vrijednost prethodno postavljena u listi. Npr.

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

je dozvoljeno, ali

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

nije jer izraz koji postavlja vrijednost za `col1` referiše na `col2` čija se vrijednost postavlja nakon `col1`.

Kada se koristi `INSERT` naredba kojom se unosi više n-troki (sa `VALUES` ili `SELECT`), izvršavanje naredbe daje sljedeće infromacije:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Records` indicira broj n-torki koji je obradila naredba, a koji ne mora biti jednak broju unesenih n-torki jer `Duplicates` može imati vrijednost različitu od 0. `Duplicates` indicira broj n-torki koje nisu unesene jer bi došlo do dupliranja neke jedinstvene vrijednosti atributa. `Warnings` indicira broj pokušaja unosa vrijednosti koje su bile problematične na određeni način. Upozorenja se mogu pojaviti zbog jednog od sljedećih slučajeva:

- Unos `NULL` vrijednosti za atribut koji je deklarisan kao `NOT NULL`. Za `INSERT` naredbu koja unosi više n-torki vrijednost ovog atributa se postavlja na implicitnu podrazumjevanu vrijednost za tip podatka (ako se ne koristi *strict SQL mode*): 0 za numerički tip, prazan string ('') za znakovni tip i "nulta" vrijednost za datumske i vremenske tipove. Za `INSERT` naredbu koja unosi jednu n-torku upozorenje se ne pojavljuje nego naredba završava sa pogreškom.
- Postavljanje numeričkog atributa na vrijednost izvan njegovog opsega. Vrijednost se odsijeca na najbližu krajnju vrijednost opsega.
- Dodajeljivanje vrijednosti poput `'10.34 a'` numeričkom atributu. Prateći nenumerički tekst se odsijeca a preostali numerički dio se unosi. Ukoliko string nema vodeći numerčki dio unosi se vrijednost 0.
- Unos stringa koji prelazi maksimalnu dužinu stringa definisanu za atribut. Vrijednost se odsijeca na maksimalnu definisanu dužinu definisanu za atribut.
- Unos vrijednosti za datumski ili vremenski atribut koja nije validna. Vrijednost se postavlja na odgovarajuću "nultu" vrijdenost za tip.

Ukoliko se u `INSERT` naredbi koristi `LOW_PRIORITY` modifikator, izvršavanje naredbe se odgađa sve do trenutka dok ne postoji nijedan drugi proces koji čita podatke iz relacije u koju se `INSERT` naredbom unose podaci. U ovom slučaju je moguće da proces koji je postavio `INSERT` naredbu čeka vrlo dugo. `HIGH_PRIORITY` modifikator prepisuje efekat `--low_priority_updates` opcije te također onemogućava konkurentan unos.

Ukoliko se koristi IGNORE modifikator ignorišu se greške koje se javljaju izvršavanjem INSERT naredbe. Npr. bez IGNORE opcije n-torka čiji bi unos doveo do dupliciranja postojeće UNIQUE INDEX ili PRIMARY KEY vrijednosti uzrokuje grešku i naredba završava neuspješno. Sa IGNORE opcijom ovakva n-torka se jednostavno odbacuje i ne pojavljuje se greška nego se generiše upozorenje. Konverzije podataka koje bi dovele do greške prekidaju INSERT naredbu ukoliko se ne koristi IGNORE opcija. Ukoliko se koristi IGNORE opcija, neispravne vrijednosti se prilagođavaju na najbliže dozvoljene vrijednosti i unose, javlja se upozorenje ali se naredba ne poništava.

Ukoliko se specificira DUPLICATE KEY UPDATE i unosi se n-torka koja bi uzrokovala dupliciranje vrijdnosti za UNIQUE INDEX ili PRIMARY KEY, izvodi se ažuriranje stare n-torke u relaciji (sa tom jednistvenom vrijednošću atributa).

Ukoliko se koristi INSERT naredba sa SELECT naredbom, naziv relacije u koju se unose podaci se može koristiti u FROM dijelu SELECT naredbe (što nije u skladu sa SQl standardom a ni nekim starijim verzijama MySQL-a). Međutim, ne može se unositi u relaciju i dohvatati iz iste relacije u podupitu. Kada se dohvataju podaci iz relacije u koju se istovremeno unosi, MySQL kreira privremenu relaciju u koju smješta rezultat SELECT naredbe a potom ove n-troke ubacije u ciljnu relaciju. Kako bi se u ovom slučaju izbjegli problemi sa istim nazivima atributa kada INSERT i SELECT naredba referišu na istu relaciju, trebalo bi koristiti alias nazive za sve relacije koje se koriste u SELECT naredbi.

INSERT naredba i SERIAL tip podatka

SERIAL tip podatka u MySQL-u je zamjenski naziv (*alias*) za BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE. Dakle, za atribut koji ima ovaj tip podatka opseg je od 0 do 18.446.744.073.709.551.615, ne može poprimiti NULL vrijednost, automatski se inkrementira njegova vrijednost u novoj n-torci koja se unosi i ima jedinstvene vrijednosti. Vrijednost atributa SERIAL tipa se inkrementira ukoliko se prilikom INSERT naredbe za taj atribut unosi vrijednost 0 ili se izostavi vrijednost za taj atribut. Sistem određuje novu vrijednost kao prvi sljedeći cijeli broj nakon najveće do tada upisane vrijednosti, bez obzira da li je n-torka sa tom najvećom upisanom vrijednošću u međuvremenu obrisana. Ukoliko se prilikom INSERT naredbe za atribut SERIAL tipa podatka navede vrijednost različita od 0, sistem ne generiše sljedeći cijeli broj nego se u relaciju unosi navedena vrijednost ukoliko ne narušava UNIQUE ograničenje.

```
CREATE TABLE zapis(
  brZapisa SERIAL,
  nazZapisa CHAR(15)
);

INSERT INTO zapis VALUES (0, '1. zapis')          -- uneseno: 1   1. zapis
INSERT INTO zapis VALUES (0, '2. zapis')          -- uneseno: 2   2. zapis
INSERT INTO zapis VALUES (50, '3. zapis')         -- uneseno: 50  3. zapis
INSERT INTO zapis (nazZapisa) VALUES ('4. zapis')  -- uneseno: 51  4. zapis
```

```
INSERT INTO zapis VALUES (0, '5. zapis')          -- uneseno: 52  5. zapis
DELETE FROM zapis WHERE nazZapisa = '5. zapis'    -- obrisano: 52 5. zapis
INSERT INTO zapis VALUES (0, '5. zapis')          -- uneseno: 53  5. zapis
INSERT INTO zapis VALUES (52, '6. zapis')         -- uneseno: 52  6. zapis
INSERT INTO zapis VALUES (53, '7. zapis')         -- greška: 53 postoji
```

DELETE naredba za brisanje podataka

Za brisanje jedne ili više n-torki relacije koristi se `DELETE` naredba. Opšti oblik `DELETE` naredbe je sljedeći:

```
DELETE FROM table_name
  [WHERE where_condition]
```

Treba naglasiti da se `DELETE` naredbom brišu kompletne n-torke (zapisi) iz relacije i da je nemoguće izvršiti brisanje pojedinih atributa relacije. To se može primijetiti i iz opšteg oblika naredbe jer se između `DELETE` i `FROM` ključnih riječi ne navode nikakvi atributi ili izrazi.

Kao i u svim dosadašnjim naredbama `table_name` je naziv relacije iz koje se brišu podaci (n-troke) dok je `where_condition` uslov koje n-torke moraju ispuniti da bi bile obrisane. `where_condition` može biti sastavljen na isti način kao bilo koji uslov koje n-torke moraju ispuniti da bi se pojavile u rezultatu u `SELECT` naredbi, s tim da ne smije imati podupite koji u svom `FROM` dijelu sadrže naziv relacije iz koje se brišu n-torke. Kao i u slučaju `SELECT` naredbe, ukoliko se `WHERE` dio ne navede, `DELETE` naredbom se brišu svi zapisi iz relacije. Treba naglasti da se brišu svi podaci (zapisi) iz relacije i da relacija ostaje prazna, ali da se ne briše struktura relacije i da njena definicija ostaje u riječniku podataka. Relacija se može obrisati jedino `DROP TABLE` naredbom.

Primjer: *Izbrisati podatke o organizacionim jedinicama u kojima nije zaposlen nijedan nastavnik i kojima ne pripada nijedan predmet.*

```
DELETE FROM orgjed
  WHERE sifOrgjed NOT IN (SELECT DISTINCT sifOrgjed FROM nast)
  AND sifOrgjed NOT IN (SELECT DISTINCT sifOrgjed FROM pred)
```

Primjer: *Izbrisati podatke o posudbama koje su u tekućem mjesecu obavili nastavnici zaposleni u organizacionoj jedinici Telekomunikacije*

```
DELETE FROM posudba
  WHERE YEAR(datPosudba) = YEAR(CURRENT_DATE)
  AND MONTH(datPosudba) = MONTH(CURRENT_DATE)
  AND sifClan IN (SELECT sifClan FROM nast
                    INNER JOIN orgjed ON nast.sifOrgjed = orgjed.sifOrgjed
                    WHERE nazOrgjed = 'Telekomunikacije')
```

U MySQL-u `DELETE` naredba može sadržavati `ORDER BY` dio kojim se određuje redoslijed brisanja n-torki iz relacije. Ovo proširenje nema nikakav uticaj na krajnji rezultat `DELETE` naredbe ukoliko se koristi samostalno. Međutim, korisno je ukoliko se želi ograničiti broj n-torki koji će biti obrisan, odnosno ukoliko se koristi u kombinaciji sa `LIMIT` dijelom. Način upotrebe je isti kao i u slučaju `SELECT` naredbe.

Primjer: *Izbrisati 5 najstarijih primjeraka knjige Uvod u relacijske baze podataka koji se nalaze u biblioteci.*

MySQL
```
DELETE FROM primjerak
    WHERE sifKnjiga IN (SELECT sifKnjiga FROM knjiga
                        WHERE naslov = 'Uvod u relacijske baze podataka')
    ORDER BY datNabavke
    LIMIT 5
```

UPDATE naredba za ažuriranje podataka

Jednom kada su podaci uneseni u bazu podataka, osim mogućnosti njihovog dohvata i brisanja vjerovatno će se nekada pojaviti i potreba za izmjenom određenih podataka. Izmjena ili ažuriranje podataka u relaciji se obavlja `UPDATE` naredbom. Za razliku od `INSERT` i `DELETE` naredbi koje djeluju na sve atribute odnosno na n-torke relacije u cijelosti, `UPDATE` naredba može izvoditi ažuriranje vrijednosti svih ili samo određenih atributa relacije. Opšti oblik `UPDATE` naredbe je sljedeći:

```
UPDATE table_name
SET col_name = { expr | (SELECT Clause) | DEFAULT | NULL } [, ...]
[WHERE where_condition]
```

`table_name` je naziv relacije u kojoj se vrši promjena vrijednosti atributa, a `col_name` su nazivi atributa te relacije kojima će biti promijenjena vrijednost. `where_condition` je uslov koje n-torke relacije moraju ispuniti da bi vrijednosti njihovih atributa navedenih u SET dijelu bili izmijenjeni. Kao i u slučaju `SELECT` i `DELETE` naredbe ukoliko se `where_condition` izostavi `UPDATE` naredba se izvršava za sve n-torke relacije i promjena vrijednosti atributa navedenih u `SET` dijelu se obavlja za svaku n-torku relacije. `where_condition` može biti sastavljen na isti način kao bilo koji uslov u slučaju `SELECT` i `DELETE` naredbe, s tim da ne smije imati podupite koji u svom `FROM` dijelu sadrže naziv relacije koja se ažurira.

Atributima navedenim u `SET` dijelu mogu se dodijeliti podrazumjevane (*default*) i `NULL` vrijednosti upotrebom ključnih riječi `DEFAULT` i `NULL`, vrijednosti koje se dobiju izračunavanjem izraza (`expr`) koji se mogu izraditi na isti način kao i izrazi u listi za selekciju `SELECT` naredbe te vrijednosti koje se dobiju kao rezultat `SELECT` naredbe. Treba voditi računa da `SELECT` naredba kojom se određuje nova vrijednost atributa mora vratiti tačno jednu vrijednost (*single-valued query*) što znači da u listi za selekciju smije sadržavati samo jedan atribut ili izraz i kao rezultat vraća tačno jednu n-torku.

Primjer: *Predmetima čiji naziv počinje slovima M, N, P, R ili S i imaju manje od 5 ECTS kredita povećati broj ECTS kredita za 1 a ispred naziva predmeta dodati skraćenicu predmeta sa crticom.*

```
UPDATE pred
SET brojECTS = brojECTS + 1, nazPred = CONCAT(skrPred, '-', nazPred)
  WHERE brojECTS < 5
  AND nazPred RLIKE '^[MNPRS]'
```

Primjer: *Nastavnicima koji imaju najviše obavljenih posudbi povećati koeficijent za 10%.*

```
UPDATE nast SET koef = koef * 1.1
  WHERE sifClan IN
    (SELECT sifClan FROM posudba
       GROUP BY sifClan
       HAVING COUNT(*) >= ALL (SELECT COUNT(*) FROM posudba
                                 GROUP BY sifClan))
```

Primjer: *Nastavnicima koji nikada nisu prekoračili rok zaduženja povećati koeficijent za onoliko procenata koliko su posudbi obavili u tekućem mjesecu.*

```
UPDATE nast
SET koef = koef * (1 + 0.01 *
                  (SELECT COUNT(*) FROM posudba
                     WHERE MONTH(datPosudba) = MONTH(CURRENT_DATE)
                     AND posudba.sifClan = nast.sifClan)
  WHERE sifClan NOT IN (SELECT sifClan FROM posudba
                          WHERE datVracanje > datDo)
```

U slučajevima kada se u izrazima korištenim u `SET` dijelu pristupa atributu relacije koji se ažurira, `UPDATE` naredba će koristiti trenutnu vrijednost atributa, odnosno vrijednost atributa iz n-torke relacije prije njenog ažuriranja. Za razliku od standardnog SQL-a, u MySQL-u će svako sljedeće pristupanje atributu u `SET` dijelu koristiti ažuriranu vrijednost. Tako će kao rezultat sljedeće naredbe:

```
UPDATE tab1 SET col1 = col1 + 1, col2 = col1;
```

atributi `col1` i `col2` relacije `tab1` imati iste vrijednosti jer se u dodjeljivanju `col2 = col1` koristi ažurirana vrijednost `col1` iz prethodnog dodjeljivanja, a ne originalna vrijednost iz relacije.

Kao i u slučaju `DELETE` naredbe, MySQL podržava proširenje `UPDATE` naredbe sa `ORDER BY` i `LIMIT` dijelovima koji imaju istu namjenu kao kod `DELETE` naredbe. Osim toga, u MySQL-u se može koristiti oblik `UPDATE` naredbe kojom se ažuriraju n-torke u više relacija na način da se na mjestu `table_name` koristi spajanje relacija koje će biti ažurirane na isti način kako se obavlja u `FROM` dijelu `SELECT` naredbe. U tom slučaju se ažuriraju n-torke iz svih spojenih relacija koje ispunjavaju uslov i to svaka po jedan puta bez obzira koliko puta učestvuju u spajanju. Međutim, s obzirom da je ovakav oblik `UPDATE` naredbe svojstven za MySQL (podržavaju ga i neki drugi SUBP-i), a da recimo MySQL ne podržava grupisanje

atributa i izraza u SET dijelu što neki drugi SUBP-i podržavaju, u svim primjerima u knjizi će UPDATE naredba biti korištena u obliku podržanom standardnim SQL-om kako je opisano na početku poglavlja.

Primjeri upita za vježbu

Q1: *Kreirati privremenu relaciju* `pregled` *sa atributima redni broj unosa (tipa SERIAL), šifra člana, broj posudbi i prosječno vrijeme trajanja posudbi u danima. Potom INSERT naredbom napuniti relaciju* `pregled` *podacima o svim članovima biblioteke kojima je broj posudbi bar dva puta veći od broja posudbi koje su obavili u tekućem mjesecu.*

```
CREATE TEMPORARY TABLE pregled(
  rbrUnos   SERIAL,
  sifClan   INTEGER,
  brojPos   INTEGER,
  prosjTraj DECIMAL(3,1)
);
```

```
INSERT INTO pregled
  SELECT 0, sifClan, COUNT(*), AVG(DATEDIFF(datVracanja, datPosudba))
  FROM posudba
  GROUP BY sifClan
  HAVING COUNT(*) >= 2 * (SELECT COUNT(*) FROM posudba pos
                          WHERE MONTH(datPosudba) = MONTH(CURRENT_DATE)
                          AND pos.sifClan = posudba.sifClan)
```

Q2: *U relaciju* `pregled` *iz prethodnog primjera ubaciti podatke o nastavnicima koji imaju najveći broj posudbi na svojim organizacionim jedinicama.*

```
INSERT INTO pregled
  SELECT 0, sifClan, COUNT(*), AVG(DATEDIFF(datVracanja, datPosudba))
  FROM posudba
  INNER JOIN nast ON posudba.sifClan = nast.sifClan
  GROUP BY sifClan
  HAVING COUNT(*) >= ALL
         (SELECT COUNT(*) FROM posudba
            INNER JOIN nast ostali ON posudba.sifClan = ostali.sifClan
            WHERE ostali.sifOrgjed = nast.sifOrgjed
            GROUP BY sifClan)
```

Q3: *Izbrisati sve podatke o literaturi koja nije obavezna za one knjige koje nikada nisu posuđene u biblioteci.*

```
DELETE FROM literatura
  WHERE sifKnjiga NOT IN
    (SELECT sifknjiga FROM posudba
       INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj)
  AND NOT obavezna
```

Q4: *Izbrisati sve podatke o autorima čije knjige nisu posuđivane u biblioteci.*

```
DELETE FROM napisao
WHERE sifKnjiga NOT IN
  (SELECT sifKnjiga FROM posudba
     INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj)

DELETE FROM autor
WHERE sifAutor NOT IN (SELECT sifAutor FROM napisao)
```

Q5: *Izbrisati sve podatke o organizacionim jedinicama čiji zaposleni nastavnici nisu posuđivali knjige.*

```
DELETE FROM orgjed
WHERE NOT EXISTS
  (SELECT * FROM posudba
     INNER JOIN nast ON posudba.sifClan = nast.sifClan
     WHERE nast.sifOrgjed = orgjed.sifOrgjed)
```

Q6: *Izbrisati podatke o studentima koji imaju manje posudbi od svih nastavnika*

```
DELETE FROM stud
  WHERE NOT EXISTS
    (SELECT nast.sifClan FROM posudba posN
       INNER JOIN nast ON posN.sicClan = nast.sifClan
       GROUP BY 1
       HAVING COUNT(*) < (SELECT COUNT(*) FROM posudba posS
                            WHERE posS.sifClan = stud.sifClan))
```

Q7: *Svim mjestima iz Tuzlanskog kantona promijeniti naziv tako da se iza naziva doda oznaka kantona (TK) i poštanski broj uvećati za 100.*

```
UPDATE mjesto SET nazMjesto = CONCAT(nazMjesto, ' (TK)'), pbr = pbr + 100
  WHERE sifKanton IN
    (SELECT sifKanton FROM kanton WHERE nazKanton = 'Tuzlanski kanton')
```

Q8: *Iz upotrebe izbaciti sve primjerke knjiga (atribut koristenje postaviti na FALSE) koji su posuđeni više od 200 puta ili je ukupno trajanje zaduženja veće od 1000 dana.*

```
UPDATE primjerak SET koristenje = FALSE
  WHERE invBroj IN
    (SELECT invBroj FROM posudba
       GROUP BY invBroj
       HAVING (COUNT(*) > 200
               OR SUM(DATEDIFF(datVracanje, datPosudba)) > 1000))
```

Q9: *Svu obaveznu literaturu predmeta sa organizacione jedinice Automatika i robotika koja nikada nije posuđena u biblioteci proglasiti neobaveznom (atribut obavezna postaviti na FALSE)*

```
UPDATE literatura SET obavezna = FALSE
  WHERE sifPred IN
    (SELECT sifPred FROM pred
       INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
       WHERE nazOrgjed = 'Automatika i robotika')
  AND sifKnjiga NOT IN
    (SELECT sifknjiga FROM posudba
       INNER JOIN primjerak ON posudba.invBroj = primjerak.invBroj)
  AND obavezna = TRUE
```

Q10: *Svim primjercima knjiga izdatih prije 2000 godine koje se ne koriste kao obavezna literatura postaviti datum nabavke na datum prve posudbe primjeraka te knjige u biblioteci.*

```
UPDATE primjerak SET datNabavka =
  (SELECT MIN(datPosudba) FROM posudba
     INNER JOIN primjerak primj ON posudba.invBroj =primj.invBroj
     WHERE primj.sifKnjiga = primjerak.sifKnjiga)
  WHERE sifKnjiga IN
    (SELECT sifKnjiga FROM knjiga
       WHERE godIzdanja < 2000
       AND sifKnjiga NOT IN (SELECT sifKnjiga FROM literatura
                             WHERE obavezna))
```

Q11: *Svim nastavnicima sa organizacione jedinice Računarstvo i informatika povećati koeficijent za 20% prosječnog koeficijenta nastavnika sa ostalih organizacionih jedinica.*

```
UPDATE nast SET koef = koef + 0.2 *
  (SELECT AVG(koef) FROM nast ostali
     WHERE ostali.sifOrgjed <> nast.sifOrgjed)
  WHERE sifOrgjed IN (SELECT sifOrgjed FROM orgjed
                      WHERE nazOrgjed = 'Računarstvo i informatika')
```

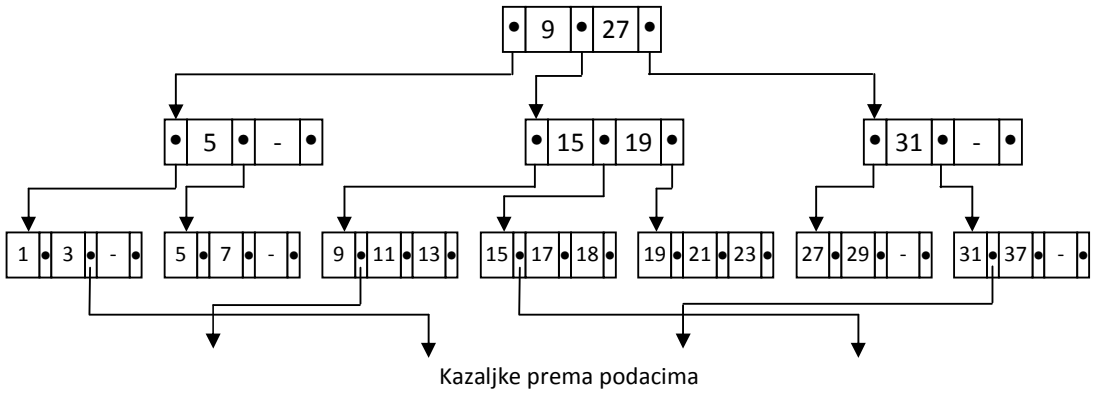
POGLAVLJE 5

INDEKSI U SQL-u

Cilj ovog poglavlja nije detaljno upoznati korisnka sa načinima pohrane i dohvata podataka u relacijskim bazama podataka ili kreiranjem, manipulacijom i izmjenom podatkovnih struktura koje nastaju kreiranjem indeksa u bazama podataka. Cilj poglavlja je upoznati korisnika sa načinom kreiranja i upotrebe indeksa u SQL-u te značaja njihovog korištenja prilikom pristupa podacima. U slučaju kada u bazama podataka postoje ogromne količine podataka, brzina pristupa tim podacima u relacijama je vrlo značajno svojstvo svakog SUBP-a. Sekvencijalni pristup (pretraga) kao najjednostavniji način pristupa podacima u većini slučajeva je spor i nije prikladan. To je razlog zbog kojeg se u bazama podataka nad relacijama kreiraju indeksi da bi se brže pronašli zapisi sa odgovarajućim vrijednostima atributa.

Bez indeksa SUBP, pa tako i MySQL, mora koristiti sekvencijalnu pretragu, odnosno krenuti od prvog zapisa u relaciji i čitati cijelu relaciju zapis po zapis kako bi pronašao one koji odgovaraju uslovu pretrage. Što je veća relacija, odnosno što je više podataka (zapisa) u relacijama, ovakva pretraga više košta. Međutim, ukoliko relacija posjeduje indeks kreiran nad atributima nad kojima se postavlja uslov pretrage, SUBP može brzo odrediti poziciju u podacima od koje počinje pretragu bez potrebe za pregledom svih podataka u relaciji što je mnogo brže od sekvencijalne pretrage svakog zapisa u relaciji.

Kada se nad atributom relacije kreira indeks, sistem stvara odgovarajuću podatkovnu strukturu koja omogućava nesekvencijalan pristup zapisima u relaciji. Većina MySQL indeksa (a isti slučaj je i sa ostalim SUBP-ima) kreira strukturu B-stabla, dok je još podržana i hash struktura za MEMORY relacije i struktura R-stabla za indekse nad prostornim podacima. Na slici je prikazana (strogo logički posmatrano) struktura B-stabla koja se kreira kada postoji indeks nad cjelobrojnim atributom (npr. sifClana u relaciji clan).



Kao što je na početku poglavlja istaknuto, organizacija B-stabla (balansirano stablo - balanced tree), pravila kreiranja indeksnih blokova, dinamička promjena i manipulacija same podatkovne strukture, nije predmet razmatranja u ovoj knjizi. Kada postoji struktura B-stabla nad nekim atributom, onda pristup zapisima na osnovu vrijednosti tog atributa počinje od korijena stabla (indeksni blok sa zapisima 9 i 27) i procedura pronalaska zapisa je rekurzivna. Svaki korak rekurzije predstavlja pretragu jednog nivoa B-stabla u kojem se nalazi najveća vrijednost atributa na tom nivou koja je manja ili jednaka traženoj vrijednosti. Nakon što se ta vrijednost pronađe, prati se pripadajuća kazaljka do bloka na sljedećem nivou. Ovaj postupak se rekurzivno ponavlja sve dok se ne dođe do podatkovnog bloka.

Nad jednom relacijom se može izgraditi više indeksa od kojih svaki može biti izgrađen nad jednim ili više atributa. Ukoliko je indeks kreiran nad samo jednim atributom `col1` relacije onda sistem može nesekvencijalno pristupiti zapisima relacije korištenjem atributa `col1` kao ključa za dohvat. Ukoliko je indeks kreiran nad više atributa relacije `col1`, `col2`, `col3` (kompozitni indeks) onda sistem može nesekvencijalno pristupiti zapisima relacije korištenjem bilo koje od kombinacija atributa (`col1`), (`col1`, `col2`) ili (`col1`, `col2`, `col3`) kao ključa za dohvat. Dakle, kompozitni indeksi se koriste u upitima koji u uslovima dohvata koriste sve atribute indeksa ili samo prvi atribut ili prva dva atributa ili prva tri atributa, itd. Kada se postavi upit, sistem sam određuje koji od postojećih indeksa će koristiti za dohvat zapisa.

U MySQL-u indeksi se koriste prilikom izvođenja sljedećih operacija:

- Brzi pronalazak zapisa koji odgovaraju uslovima dohvata navedenim u `WHERE` dijelu.
- Eliminacija zapisa koji će se razmatrati. Ukoliko postoji mogućnost izbora između više indeksa, MySQL će koristiti indeks koji ronalazi najmanji broj zapisa (najselektivniji indeks).
- Ukoliko u relaciji postoji indeks nad više atributa, optimizator može koristiti bilo koji lijevi prefiks atributa indeksa za pronalazak zapisa (kako je prethodno objašnjeno za tri atributa).
- Dohvat zapisa iz drugih relacija prilikom obavljanja operacije spajanja. MySQL može efikasnije koristiti indekse nad atributima ukoliko su atributi deklarisani sa istim nazivom i tipom podatka (`VARCHAR` i `CHAR` se smatraju istim ukoliko su deklarisani iste dužine, string atributi su jednaki ukoliko koriste isti set karaktera npr. `utf8` ili `latin1`). Poređenja atributa koji nisu slični (npr. poređenje string atributa sa vremenskim ili brojčanim atributom) mogu spriječiti upotrebu indeksa ukoliko se vrijednosti ne mogu porediti direktno bez konverzije.
- Pronalazak `MIN()` ili `MAX()` vrijednosti za određeni indeksirani atribut.
- Za sortiranje ili grupisanje relacije ukoliko se sortiranje ili grupisanje obavlja po lijevom prefiksu korištenog indeksa
- U nekim slučajevima upit može biti optimiziran da se dohvate vrijednosti bez pristupa podatkovnim zapisima. Indeks koji obezbjeđuje sav potreban rezultat za upit se naziva pokrivajući (*covering*) indeks. Ukoliko upit iz relacije koristi samo atribute koji su uključeni u odgovarajući indeks, selektovane vrijednosti mogu biti dohvaćene iz indeksnog stabla radi veće brzine dohvata.

Indeksi nemaju prevelik značaj kada se upiti obavljaju nad malim relacijama ili nad velikim relacijama gdje upiti obrađuju većinu ili sve zapise (upiti koji se koriste za izvještavanje). Kada upit mora pristupiti većini zapisa, sekvencijalni pristup je brži od pristupa korištenjem indeksa.

Osim što se indeksi koriste u svrhu poboljšanja performansi sistema prethodno opisanih, oni se kreiraju i da bi se osigurala jedinstvenost vrijednosti atributa u relaciji (npr. da bi se osiguralo da u relaciji član ne postoje dva člana sa jednakim jedinstvenim matičnim brojem). Ukoliko se indeks kreira kao indeks sa jedinstvenim vrijednostima (*unique index*), sistem neće dozvoliti da se u relaciji pojave dvije n-torke koje bi imale jednake vrijednosti atributa nad kojim je izgrađen takav indeks. Između ostalog, ovakva karakteristika indeksa se koristi da bi se osigurala jedinstvenost ključa u relaciji.

Upotreba indeksa za sortiranje ili grupisanje je ograničena načinom izrade (definisanja) indeksa, s obzirom da vrijednosti u indeksnim blokovima mogu biti sortirane uzlazno i silazno. Sistem će koristiti indeks za sortiranje zapisa u smjeru u kojem su sortirane vrijednosti u indeksnim blokvima, ali i u obrnutom smjeru. To znači da ukoliko je indeks izgrađen nad samo jednim atributom poredak vrijednosti u njegovim indeksnim blokovima nije bitan. Međutim, ukoliko se radi o kompozitnom indeksu, a prema atributima nad kojim je izgrađen indeks se obavlja sortiranje, da li će taj indeks biti korišten za sortiranje ovisi o poretku vrijednosti u indeksnim blokovima. Ukoliko su vrijednosti u kompozitnom indeksu za sve atribute sortirane u istom smjeru (npr. `col1 DESC`, `col2 DESC`), indeks će se koristiti za sortiranja atributa u istom smjeru (`col1 DESC`, `col2 DESC` ili `col1 ASC`, `col2 ASC`), ali ne i za kombinovanje smjerova sortiranja (`col1 ASC`, `col2 DESC` ili `col1 DESC`, `col2 ASC`).

MySQL može koristiti indeks čak i ako `ORDER BY` lista atributa ne odgovara u potpunosti atributima nad kojim je izrađen indeks, sve dok su neiskorišteni atributi iz indeksa i dodatni atributi u `ORDER BY` dijelu konstante u `WHERE` dijelu. Postoje još neki slučajevi u kojima MySQL jednostavno ne može koristiti indekse za sortiranje, ali ih naravno može koristiti za pronalazak zapisa koji odgovaraju uslovu u `WHERE` dijelu:

- Kada upit koristi `ORDER BY` nad različitim indeksima
- Kada upit koristi `ORDER BY` nad dijelovima (atributima) indeksa koji nisu uzastopni
- Kada se indeks korišten za dohvat zapisa razlikuje od indeksa korištenog u `ORDER BY`
- Kada upit koristi `ORDER BY` sa izrazom koji uključuje termine drugačije od naziva atributa indeksa (npr. `ABS(col)` ili `-col`)
- Kada upit ima različite `ORDER BY` i `GROUP BY` dijelove
- Kada u `ORDER BY` dijelu postoji atribut nad kojim je izgrađen indeks samo nad njegovim dijelom

Pored svih navedenih dobrobiti koje uvode indeksi svojim kreiranjem i upotrebom, oni imaju i neke negativne aspekte koje svakako treba uzeti u obzir, a neki od njih su:

- Indeksi zauzimaju značajan prostor

- Ažuriranje vrijednosti atributa nad kojima je izgrađen indeks traje znatno duže nego ažuriranje vrijednosti atributa nad kojim nema indeksa

Uzimajući u obzir sve navedeno, indekse bi trebalo kreirati u sljedećim slučajevima:

- Za atribute prema kojima se obavlja spajanje relacija
- Za atribute koji se često koriste u uslovima dohvata u `WHERE` dijelu
- Za atribute prema kojima se često obavlja grupisanje i sortiranje

Indekse svakako ne bi trebalo koristiti u sljedećim slučajevima:

- Atribut nad kojim se izgrađuje indeks ima relativno mali broj različitih vrijednosti (npr. redni broj autora sa vrijednostima od 1 do npr. 5 ili 6 u relaciji `napisao` sa 30 000 n-torki)
- Kada u relacijama predstoji veliki broj unosa, izmjena ili brisanja n-torki. U takvim slučajevima se preposučuje brisanje postojećih indeksa pa njhovo ponovno kreiranje nakon obavljenih operacija
- Relacije sadrže mali broj n-torki (npr. do stotinu). Kao što je već rečeno u tim slučajevima se brže pristupa korištenjem sekvencijalne pretrage.

Naredba za kreiranje indeksa

Kao i za sve ostale objekte u SQL-u, naredba za kreiranje indeksa je `CREATE INDEX`. Međutim, ova naredba nije dio SQL standarda te se njena sintaksa razlikuje od sistema o sistema. I pored toga, može se reći da neki osnovni dio te naredbe u svim sistemima ima sljedeći opšti oblik:

```
CREATE [ UNIQUE ] INDEX index_name
ON table_name (col_name [ ASC | DESC ] [, ...])
```

`index_name` je naziv indeksa koji se pohranjuje u riječniku podataka i kojem se može pristupiti u ostalim naredbama. `table_name` je naziv relacije nad kojom se kreira indeks a `col_name` naziv(i) atributa nad kojim(a) je kreiran indeks. Ključna riječ `UNIQUE` osigurava jedinstvenost vrijednosti atributa nad kojim je izgrađen indeks (*unique index*). Navođenjem ključnih riječi `ASC` ili `DESC` nakon naziva atributa prilikom kreiranja indeksa definiše se u kojem smjeru će biti sortirane vrijednosti u indeksnim blokovima.

Primjer: *Napisati naredbu za kreiranje indeksa kojom će se osigurati da se dvije iste knjige ne mogu koristiti kao literatura na istom predmetu.*

```
CREATE UNIQUE INDEX literatura_ind ON literatura (sifKnjiga, sifPred)
```

Kao i ostali objekti u SQL-u i indeksi se brišu `DROP` naredbom čiji je opšti oblik:

```
DROP INDEX index_name
```

Kompletna naredba za kreiranje indeksa u MySQL-u ima sljedeći opšti oblik:

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

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

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

index_type:
    USING { BTREE | HASH }

algorithm_option:
    ALGORITHM [=] { DEFAULT | INPLACE | COPY }

lock_option:
    LOCK [=] { DEFAULT | NONE | SHARED | EXCLUSIVE }
```

Ovdje ćemo samo istaći, da za atribute tipa nizova znakova indeksi se mogu kreirati korištenjem samo vodećeg dijela vrijednosti atributa korištenjem `col_name(index)` sintakse kojom se definiše dužina prefiksa nad kojim se gradi indeks. Pri tome vrijedi sljedeće:

- Prefiksi mogu biti specificirani za `CHAR`, `VARCHAR`, `BINARY` i `VARBINARY` atribute
- Prefiksi moraju biti specificirani za `BLOB` i `TEXT` atribute
- Dužina prefiksa se interpretira kao broj karaktera za `CHAR`, `VARCHAR` i `TEXT` tipove, a broj bajta za `BINARY`, `VARBINARY` i `BLOB` tipove

Primjer: *Napisati naredbu za kreiranje indeksa nad prvih 15 karaktera vrijednosti atributa nazPred u relaciji pred.*

```
CREATE INDEX nazPred_dio_ind ON pred (nazPred(15))
```

Ukoliko se nazivi obično razlikuju u prvih 15 karaktera, ovaj indeks ne bi trebao biti mnogo sporiji od indeksa koji bi bo kreiran nad kompletnim `nazPred` atributom. Osim toga, korištenje prefiksa atributa u indeksima može dovesti do značajnog smanjenja indeksnog fajla, što bi moglo uštediti dosta prostora na disku te također ubrzati `INSERT` operacije.

Upotreba pogleda (VIEW)

Pogled (virtuelna relacija, *view*) predstavlja objekat u bazi podataka čija je upotreba i ponašanje vrlo slično upotrebi i ponašanju relacija. Pogled ustvari predstavlja upit koji se može trajno pohraniti i čiji rezultat postaje sadržaj pogleda, što znači da su shema i sadržaj pogleda definisani `SELECT` naredbom. Jednom kada je defnisan, pogled se može tretirati kao bazna relacija te se mogu obavljati `SELECT`, `UPDATE`, `INSERT` i `DELETE` naredbe nad pogledom kao i spajanje sa drugim relacijama i pogledima.

Kreiranje pogleda se obavlja `CREATE VIEW` naredbom u sljedećem opštem obliku:

```
CREATE VIEW view_name [ (column_list) ]
    AS select_statement
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

`view_name` je naziv pogleda koji će se koristiti u operacijama nad pogledom. `column_list` je lista naziva atributa od kojih će se sastojati pogled (shema pogleda) i može se izostaviti prilikom kreiranja pogleda. Ukoliko se izostavi kao nazivi atributa pogleda će se koristiti nazivi atributa ili alias nazivi navedeni u listi za selekciju `SELECT` naredbe. Ako se navede `column_list` broj naziva atributa mora odgovarati broju atributa ili izraza korištenih u listi za selekciju. `select_statement` je `SELECT` naredba kojom se definira pogled, a u kojoj se mogu koristiti bazne tabele i drugi pogledi.

Definicija pogleda postaje trajna u trenutku kreiranja te na nju ne utiču naknadne izmjene u definicijama relacija na kojim je zasnovan pogled. Npr. ukoliko se pogled definiše sa `SELECT *` nad nekom relacijom, novi atributi koji se naknadno dodaju u relaciju ne postaju dio pogleda, a atributi koji budu izbrisani iz relacija će dovesti do greške kada im se pokuša pristupiti korištenjem pogleda. Nasuprot tome sadržaj pogleda se evaluira u trenutku izvođenja upita na temelju sadržaja relacija i pogleda nad kojima je pogled definisan. Svaka promjena nad sadržajem u tim pozadinskim relacijama i pogledima automatski će se odraziti i na rezultatima SQL naredbi u kojima se koristi pogled koji je definisan nad njima.

Da bi korisnik kreirao pogled mora imati odgovarajuće privilegije nad atributima iz liste za selekciju `SELECT` naredbe te dozvolu pristupa (`SELECT`) atributima koji se koriste u ostatku `SELECT` naredbe.

`WITH CHECK OPTION` klauzula se može navesti prilikom kreiranja pogleda kako bi se onemogućio unos zapisa za koje `WHERE` uslov u `select_statement` nije istinit, odnosno unos zapisa koji ne zadovoljavaju definiciju pogleda. Osim toga, sprječava se i izmjena podataka upotrebom pogleda nad onim zapisima za koje je `WHERE` uslov istinit, ali bi izmjena dovela do toga da postane lažan. U `WITH CHECK OPTION` klauzuli ključne riječi `LOCALE` i `CASCADED` (`CASCADED` je podrazumijevano) određuju doseg provjere za slučaj da je pogled kreiran na osnovu drugih pogleda. Ukoliko se navede opcija `LOCAL`, provjerava se `WHERE` dio u definiciji pogleda, a potom provjeravaju resursi pozadinskih pogleda na koje se

primjenjuju ista pravila. Ukoliko se navede opcija `CASCADED`, provjerava se `WHERE` dio u definiciji pogleda, a potom provjeravaju resursi pozadinskih pogleda, dodaje `WITH CASCADED CHECK OPTION` na njih i primjenjuju se ista pravila. Bez navođenja `WITH CHECK OPTION` ne provjerava se `WHERE` dio u definiciji pogleda, a potom provjeravaju resursi pozadinskih pogleda i primjenjuju ista pravila.

Kao što se iz navedenog može primijetiti, kreiranje pogleda je vrlo slično kreiranju privremene relacije sa `SELECT` naredbom, čak im je i namjena u dosta slučajeva slična, te bi se moglo pogrešno zaključiti da pogled može biti zamjena za privremenu relaciju i obrnuto. Zbog toga će u nastavku biti ukratko navedene neke osnovne razlike između ovih objekata u SQL-u:

- Pogled je definicija i ne pohranjuje podatke i kada se postavi upit nad pogledom sigurno se dohvataju trenutni podaci iz relacija nad kojima je kreiran
- Privremena relacija sadrži kopiju podataka, ali se nakon njenog kreiranja ne obavlja sinhronizacija sa trajnim tabelama na osnovu kojih je nastala. Kada se podaci u trajnim relacijama promijene, te promjene se neće odraziti u privremenoj relaciji osim ako se ponovo ne kreira.
- Definicija pogleda se trajno pohranjuje u bazi podataka i pogled postoji i nakon što se okonča sesija tokom koje je kreiran.
- Definicija privremene relacije se ne pohranjuje u bazi podataka i privremena relacija postoji dok se ne okonča sesija tokom koje je kreirana.
- Nakon što je kreiran, pogled je vidljiv (mogu mu pristupiti) svim sesijama uspostavljenim sa bazom podataka dok je privremena relacija vidljiva samo sesiji tokom koje je kreirana.
- Kroz poglede je moguće vršiti ažuriranja trajnih relacija nad kojima je kreiran dok se ažuriranja koja se obave u privremenim relacijama ne reflektuju na trajne relacije na osnovu kojih su nastale.
- Kada se koriste podaci kroz poglede vrši se postavljanje odgovarajućih ključeva nad zapisima u trajnim relacijama
- Pogledi koriste indekse definisane nad trajnim relacijama te se u određenoj mjeri može ubrzati pristup podacima korištenjem pogleda

Uzimajući u obzir navedene razlike između pogleda i privremenih relacija postoji nekoliko situacija u kojima bi trebalo koristiti jedne ili druge:

- Privremenu relaciju koristiti u slučajevima kada se kreirana struktura neće ponovo koristiti više puta nego će se nad njom izvršiti samo nekoliko upita
- Privremenu relaciju koristiti kada se žele koristiti podaci nastali kao rezultat nekoliko uzastopnih upita (kada se puni u nekoliko koraka ili se izbjegava korištenje prevelikog `UNION`)
- Privremenu relaciju koristiti kada se želi raditi nad konzistentnim skupom podataka koji se ne treba mijenjati tokom rada. To je efikasnije od postavljanja ključeva kao posljedica korištenja pogleda nad relacijama u kojima se trajno nalaze podaci.

- Pogled koristiti kada je upit dovoljno komplikovan i kao takav će se koristiti više puta
- Pogled koristiti u slučajevima da se želi smanjiti kompleksnost modela baze podataka u odnosu na krajnjeg korisnika
- Pogled koristiti u slučajevima dodjele odgovarajućih dozvola korisnicima s obzirom da sistem također dozvoljava postavljanje sigurnosnih ograničenja i nad pogledima (ovo je najčešća upotreba pogleda u relacijskim bazama podataka).

RUKOVANJE NULL VRIJEDNOSTIMA U SQL-u

Ponekad se može desiti, naročito ukoliko se radi o bazama podataka koji su podrška velikim sistemima koji proizvode velike količine podataka koji se u njima trebaju pohraniti, da neke od informacija koje treba unijeti u bazu podataka nisu potpune. Razlog za to može biti što neke informacije jednostavno trenutno nisu poznate (ukoliko se podaci unose sa obrazaca, a neka od polja nisu popunjena), neke informacije uopće ne postoje ili nisu primjenjive (za studenta ne postoji koeficijent za platu), dok neke informacije postoje, ali do njih nije moguće doći (podaci koji se čuvaju u tajnosti i nisu javno dostupni). Sve takve informacije u bazi podataka prikazuju se kao poseban oblik podatka - `NULL` vrijednost.

`NULL` vrijednost se razlikuje i interno se pohranjuje od bilo koje druge dozvoljene vrijednosti za bilo koji tip podatka - nije vrijednost 0 za numerički tip podatka, nije prazan niz za podatke tipa niza znakova, itd. Način na koji se interno pohranjuje `NULL` vrijednost nije ni bitan jer je jedinstvena na nivou baze podataka nezavisno od tipa podatka kojeg predstavlja. Tako se i u SQL naredbama, bez obzira na tip podatka, koristi `NULL` konstanta.

```
INSERT INTO posudba VALUES (100004, '2017-04-15', NULL, '2017-04-30', 1010)
```

```
UPDATE posudba SET datumDo = NULL
  WHERE ADDDATE(datumVrac, INTERVAL 10 DAYS) > CURRENT_DATE
```

Bitno je istaći da svaki aritmetički izraz u kojem bilo koji operand poprimi `NULL` vrijednost će u cijelosti biti evaluiran kao `NULL`. Svi izrazi poređenja u dosadašnjim primjerima su evaluirani kao tačni ili netačni (istiniti ili lažni, *true* ili *false*) jer se pretpostavljalo da niti jedan od operanada neće porimiti `NULL` vrijednost. Međutim, ukoliko u izrazu poređenja bilo koji operand poprimi `NULL` vrijednost tada je rezultat izraza logička vrijednost nepoznato (*unknown*). Zbog toga i osnovne logičke operacije `AND`, `OR` i `NOT` osim rezultata *true* i *false* mogu imati rezultat *unknown* (trovalentna logika).

S obzirom da sistemi za upravljanje bazama podataka nisu u stanju međusobno razlikovati `NULL` vrijednosti smatra se da je pri rukovanju s `NULL` vrijednostima u skupovima dopuštena pojava jedne i samo jedne `NULL` vrijednosti u skupu te će dvije n-torke relacije biti smatrane kopijama ukoliko njihovi odgovarajući (korespondentni) atributi ili imaju iste vrijednosti ili imaju `NULL` vrijednosti.

IS NULL operator

Uzimajući u obzir sve aprijed navedeno, ukoliko se u uslovima dohvata želi ispitati da li neki atribut ima (ili nema) `NULL` vrijednost, korištenje relacijskih operatora poređenja neće dati željeni rezultat jer izrazi:

```
imeAtributa = NULL i imeAtributa != NULL
```

bez obzira na vrijednost atributa `imeAtributa` uvijek kao rezultat daju *unknown* jer je vrijednost jednog od operanada `NULL`. Da bi se ispravno ispitalo da li je vrijednost nekog atributa postavljena na `NULL`, u SQL naredbama se koristi operator `IS NULL`, odnosno `IS NOT NULL`. Upotrebom ova dva operatora rezultat poređenja će uvijek biti *true* ili *false* (nikada *unknown*).

POSUDBA

invBroj	datumPos	datumVrac	datumDo	sifClan
100001	29.03.2017	NULL	05.04.2017	1001
101001	13.04.2017	20.04.2017	25.04.2017	1002
101005	22.03.2017	NULL	31.03.2017	1003
100004	08.04.2017	22.04.2017	20.04.2017	1004
100010	16.03.2017	21.03.2017	23.03.2017	1001
101001	21.04.2017	NULL	03.05.2017	1005
100002	30.03.2017	07.04.2017	06.04.2017	1006

Primjer: *Ispisati sve podatke o psudbama primjeraka koji još nisu vraćeni.*

```
SELECT * FROM posudba
  WHERE datumVrac IS NULL
```

invBroj	datumPos	datumVrac	datumDo	sifClan
100001	29.03.2017	NULL	05.04.2017	1001
101005	22.03.2017	NULL	31.03.2017	1003
101001	21.04.2017	NULL	03.05.2017	1005

Primjer: *Ispisati datum vraćanja i datum posudbe za sve vraćene primjerke.*

```
SELECT datumPos, datumVrac FROM posudba
  WHERE datumVrac IS NOT NULL
```

invBroj	datumPos	datumVrac	datumDo	sifClan
101001	13.04.2017	20.04.2017	25.04.2017	1002
100004	08.04.2017	22.04.2017	20.04.2017	1004
100010	16.03.2017	21.03.2017	23.03.2017	1001
100002	30.03.2017	07.04.2017	06.04.2017	1006

NULL vrijednosti u uslovima dohvata

U svim SQL naredbama (`SELECT`, `UPDATE`, `DELETE`) sve n-torke za koje se uslov dohvata nakon uvrštavanja vrijednosti njihovih atributa evaluira kao `true` će biti prikazane, ažurirane ili obrisane. Sve n-torke za koje se uslov dohvata evaluira kao *false* ili *unknown* neće biti prikazane, ažurirane ili obrisane.

Primjer: *Ispisati šifre članova koji posuđene primjerke nisu vratili na vrijeme.*

```
SELECT sifClan FROM posudba
  WHERE datumVrac > datumDo
```

invBroj	datumPos	datumVrac	datumDo	sifClan
100004	08.04.2017	22.04.2017	20.04.2017	1004
100002	30.03.2017	07.04.2017	06.04.2017	1006

Primjer: *Ispisati inventarni broj primjerka i šifru člana za sve posudbe koje su obavljene u aprilu mjesecu 2017 godine a vraćene nakon 20.04.2017.*

```
SELECT invBroj, sifClan FROM posudba
  WHERE MONTH(datumPos) = 4 AND YEAR(datumPos) = 2017
  AND datumVrac > '20.04.2017'
```

1. n-torka:

`3 = 4` ∧ `2017 = 2017` ∧ `NULL > '20.04.2017'` → `false` ∧ `true` ∧ `unknown` → `false`

2. n-torka:

`4 = 4` ∧ `2017 = 2017` ∧ `'20.04.2017' > '20.04.2017'` → `true` ∧ `true` ∧ `false` → `false`

3. n-torka:

`3 = 4` ∧ `2017 = 2017` ∧ `NULL > '20.04.2017'` → `false` ∧ `true` ∧ `unknown` → `false`

4. n-torka:

`4 = 4` ∧ `2017 = 2017` ∧ `'22.04.2017' > '20.04.2017'` → `true` ∧ `true` ∧ `true` → `true`

5. n-torka:

`3 = 4` ∧ `2017 = 2017` ∧ `'21.03.2017' > '20.04.2017'` → `false` ∧ `true` ∧ `false` → `false`

6. n-torka:

`4 = 4` ∧ `2017 = 2017` ∧ `NULL > '20.04.2017'` → `true` ∧ `true` ∧ `unknown` → `unknown`

7. n-torka:

`3 = 4` ∧ `2017 = 2017` ∧ `'07.04.2017' > '20.04.2017'` → `false` ∧ `true` ∧ `false` → `false`

invBroj	datumPos	datumVrac	datumDo	sifClan
100004	08.04.2017	22.04.2017	20.04.2017	1004

Spajanje relacija u prisustvu NULL vrijednosti

Slično kao u slučaju selekcije, i prilikom obavljanja operacija spajanja (prirodnog, uz uslov ili sa izjednačavanjem) spojiće se samo one n-torke za koje se uslov spajanja evaluira kao istinit (*true*).

MJESTO

pbr	nazMjesto	sifKanton
75000	Tuzla	3
75300	Lukavac	NULL
76120	Brčko	12

KANTON

sifKanton	nazKanton
3	Tuzlanski kanton
12	Brčko distrikt

```
SELECT mjesto.*, nazKanton FROM mjesto INNER JOIN kanton
  ON mjesto.sifKanton = kanton.sifKanton

SELECT * FROM mjesto NATURAL JOIN kanton

SELECT * FROM mjesto INNER JOIN kanton USING(sifKanton)
```

pbr	nazMjesto	sifKanton	nazKanton
75000	Tuzla	3	Tuzlanski kanton
76120	Brčko	12	Brčko distrikt

NULL vrijednosti u agregatnim funkcijama

Sve agregatne funkcije, bez obzira na to da li se koristi kvalifikator `DISTINCT` ili ne, ignorišu `NULL` vrijednosti atributa ili izraza koji se koristi kao argument i izračunavaju vrijednost na osnovu ostalih vrijednosti. Ukoliko su sve vrijednosti atributa ili izraza jednake `NULL` rezultat primjene agregatne funkcije će biti `NULL` izuzev `COUNT` agregatne funkcije koja će kao rezultat dati vrijednost 0. Jedini slučaj kada pojava `NULL` vrijednosti ne utiče na rezultat agregatne funkcije je `COUNT(*)` u kojem slučaju se kao rezultat dobija broj n-torki.

Primjeri:

```
SELECT COUNT(*) FROM posudba                                    → 7

SELECT COUNT(datumVrac) FROM posudba                            → 4

SELECT COUNT(DISTINCT MONTH(datumVrac)) FROM posudba            → 2

SELECT SUM(DATEDIFF(datumVrac, datumPos)) FROM posudba          → 34

SELECT SUM(DATEDIFF(datumVrac, datumPos)) FROM posudba          → NULL
  WHERE DAY(datumPos) BETWEEN 20 AND 29
```

Primjer: *Ispisati prosječan broj dana zaduženja svih primjeraka knjiga koje su trebale biti vraćene u aprilu mjesecu.*

```
SELECT AVG(DATEDIFF(datumVrac, datumPos)) FROM posudba
  WHERE MONTH(datumDo) = 4
```

AVG(Expression)
9,667

NULL vrijednosti u izrazima za formiranje grupa zapisa

U slučaju kada se obavlja grupisanje n-torki, treba voditi računa o definiciji kopije n-torki, a to podrazumijeva da će u istu grupu ulaziti sve n-torke čije su vrijednosti svih atributa prema kojima se obalja grupisanje međusobne kopije.

```
SELECT MONTH(datumPos) mjesecPos, MONTH(datumVrac) mjesecVrac, COUNT(*)
  FROM posudba
  GROUP BY 1, 2
```

mjesecPos	mjesecVrac	COUNT(*)
3	NULL	2
4	4	2
3	3	1
4	NULL	1
3	4	1

Poredak rezultata upita u prisustvu NULL vrijednosti

Iako se prilikom poređenja `NULL` vrijednosti u uslovima uvijek dobija rezultat *unknown*, prilikom sortiranja rezultata upita prema atributima u kojima postoje `NULL` vrijednosti, `NULL` vrijednost se smatra "manjom" od svih ostalih vrijednosti. To znači da će se prilikom sortiranja rezultata u uzlaznom poretku `NULL` vrijednost naći na prvom mjestu, dok će se prilikom sortiranja rezultata u silaznom poretku `NULL` vrijednost naći na posljednjem mjestu u rezultatu upita.

```
SELECT * FROM posudba
  ORDER BY datumVrac, datum Pos
```

POSUDBA

invBroj	datumPos	datumVrac	datumDo	sifClan
101005	22.03.2017	NULL	31.03.2017	1003
100001	29.03.2017	NULL	05.04.2017	1001
101001	21.04.2017	NULL	03.05.2017	1005
100010	16.03.2017	21.03.2017	23.03.2017	1001
100002	30.03.2017	07.04.2017	06.04.2017	1006
101001	13.04.2017	20.04.2017	25.04.2017	1002
100004	08.04.2017	22.04.2017	20.04.2017	1004

NULL vrijednosti u rezultatima podupita

Ukoliko rezultat podupita sadrži `NULL` vrijednosti tada je potrebno voditi računa kada se koriste sljedeći načini ugradnje podupita u uslov dohvata:

```
WHERE expression relationalOperator ALL (subquery)
```

```
WHERE expression NOT IN (subquery)
```

U prvom slučaju, da bi se n-torka vanjskog upita pojavila u rezultatu poređenje izraza u uslovu dohvata (`expression`) sa svakom od vrijednosti koja se pojavljuje u rezultatu podupita se mora evaluirati kao true. Ukoliko se u rezultatu podupita pojavi `NULL` vrijednost, poređenje izraza (`expression`) sa tom vrijednošću se evaluira kao *unknown* te se zbog toga cjelokupan uslov dohvata evaluira kao `unknown` te takav upit neće vratiti niti jednu n-torku.

U drugom slučaju, da bi se n-torka vanjskog upita pojavila u rezultatu izraz u uslovu dohvata (`expression`) mora biti različit od svake vrijednosti koja se pojavljuje u rezultatu podupita (ekvivalentan uslov bi bio `expression != ALL (subquery)`). Kao i u prethodno opisanom slučaju, ukoliko se u rezultatu podupita pojavi `NULL` vrijednost, poređenje izraza (relacijski operator `!=`) sa tom vrijednošću se evaluira kao *unknown* te se zbog toga cjelokupan uslov dohvata evaluira kao *unknown* te takav upit neće vratiti niti jednu n-torku.

```
SELECT * FROM posudba
  WHERE datumDo < ANY (SELECT datumVrac FROM posudba)
```

invBroj	datumPos	datumVrac	datumDo	sifClan
100001	29.03.2017	NULL	05.04.2017	1001
101005	22.03.2017	NULL	31.03.2017	1003
100004	08.04.2017	22.04.2017	20.04.2017	1004
100010	16.03.2017	21.03.2017	23.03.2017	1001
100002	30.03.2017	07.04.2017	06.04.2017	1006

```
SELECT * FROM posudba
  WHERE datumDo < ALL (SELECT datumVrac FROM posudba)

0 row(s) returned

SELECT * FROM posudba
  WHERE datumDo IN (SELECT datumVrac FROM posudba)
```

invBroj	datumPos	datumVrac	datumDo	sifClan
100004	08.04.2017	22.04.2017	20.04.2017	1004

```
SELECT * FROM posudba
  WHERE datumDo NOT IN (SELECT datumVrac FROM posudba)

0 row(s) returned
```

Operacije unije, presjeka i razlike u prisustvu NULL vrijednosti

Slično kao u slučaju grupisanja rezultata upita, u operacijama unije, presjeka i razlike se mora voditi računa da li dvije n-torke iz različitih relacija predstavljaju međusobne kopije. Kako se operacija unije u SQL-u implementira `UNION` operatorom, prisustvo `NULL` vrijednosti u određenim atributima ne utiče na oblik `SELECT` naredbe kojom se implementira operacija unije. S druge strane, operacije presjeka i razlike se implementiraju na već opisani način u kojem se koriste uslovi poređenja, te se sve n-torke koje imaju `NULL` vrijednosti atributa koji učestvuju u uslovima poređenja ne bi pojavile u rezultatu (`NULL = NULL` kao rezultat daje *unknown*). Iz tog razloga prilikom implementacije operacija presjeka i razlike u SQL-u u slučaju kada postoje `NULL` vrijednosti uz standardni uslov poređenja (jednakosti) atributa potrebno je uključiti i uslov kada oba atributa koji se porede imaju `NULL` vrijednost.

PREDMET

oznFak	oznPred	nazPred
FET	MAT1	Matematika I
FET	AKT	Aktuatori
FET	NULL	Baze podataka
PMF	MAT1	Mat. analiza I
NULL	FIZ1	Opća fizika I

PREDUSLOV

oznFak	oznPred	nazPred
NULL	MAT1	Matematika I
FET	FIZ1	Fizika I
FET	BP	Baze podataka
PMF	MAT1	Mat. analiza I

Primjer: *Ispisati predmete koji imaju preduslove i koji se koriste kao preduslovi za druge predmete (napraviti presjek relacija predmet i preduslov).*

```
SELECT * FROM predmet
  WHERE EXISTS
    (SELECT * FROM preduslov
       WHERE (predmet.oznFak = preduslov.oznFak
              OR (predmet.oznFak IS NULL AND preduslov.oznFak IS NULL))
       AND (predmet.oznPred = preduslov.oznPred
            OR (predmet.oznPred IS NULL AND preduslov.oznpred IS NULL))
       AND (predmet.nazPred = preduslov. nazPred
            OR (predmet. nazPred IS NULL AND preduslov. nazPred IS NULL)))
```

Isti rezultat se može dobiti i pomoću spajanja sa izjednačavanjem.

```
SELECT * FROM predmet, preduslov
  WHERE (predmet.oznFak = preduslov.oznFak
         OR (predmet.oznFak IS NULL AND preduslov.oznFak IS NULL))
  AND (predmet.oznPred = preduslov.oznPred
       OR (predmet.oznPred IS NULL AND preduslov.oznpred IS NULL))
  AND (predmet.nazPred = preduslov. nazPred
       OR (predmet. nazPred IS NULL AND preduslov. nazPred IS NULL))
```

Primjer: *Ispsati predmete koji se ne koriste kao preduslov niti jednom drugom predmetu (napraviti razliku relacija predmet i preduslov).*

```
SELECT * FROM predmet
  WHERE NOT EXISTS
    (SELECT * FROM preduslov
       WHERE (predmet.oznFak = preduslov.oznFak
              OR (predmet.oznFak IS NULL AND preduslov.oznFak IS NULL))
       AND (predmet.oznPred = preduslov.oznPred
            OR (predmet.oznPred IS NULL AND preduslov.oznpred IS NULL))
       AND (predmet.nazPred = preduslov. nazPred
            OR (predmet. nazPred IS NULL AND preduslov. nazPred IS NULL))
```

Vanjsko spajanje relacija

U slučaju svih do sada prezentovanih oblika spajanja relacije koje se spajaju se tretiraju simetrično, odnosno n-torka jedne relacije će se pojaviti u rezultatu upita samo ukoliko postoji odgovarajuća n-torka druge relacije s kojom se može spojiti (n-torke relacija zadovoljavaju uslov spajanja). N-torka jedne relacije za koje ne postoji odgovarajuća n-torka druge relacije (ne zadovoljava uslov spajanja niti za jednu n-torku druge relacije) se neće pojaviti u rezultatu upita. I obrnuto.

MJESTO

pbr	nazMjesto	sifKanton
75000	Tuzla	3
72000	Zenica	4
76120	Brčko	12
10000	Zagreb	NULL

KANTON

sifKanton	nazKanton
3	Tuzlanski kanton
12	Brčko distrikt
1	Sarajevski kanton

```
SELECT mjesto.*, kanton.* FROM mjesto
  INNER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
```

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
76120	Brčko	12	12	Brčko distrikt

U prethodnom primjeru n-torka `<72000, Zenica, 4>` iz relacije `mjesto` se nije pojavila u rezultatu jer ne postoji odgovarajuća n-torka u relaciji `kanton` (za svaku od n-torki relacije `kanton` uslov spajanja nije zadovoljen, evaluira se kao `false`). Iz istog razloga u rezultatu se nije pojavila niti n-torka `<1, Sarajevski kanton>` iz relacije `kanton`. Ukoliko je vrijednost stranog ključa neke n-torke relacije postavljena na `NULL`, ta n-torka se prilikom spajanja sa odgovarajućom relacijom (korištenjem tog stranog ključa) nikada neće pojaviti u rezultatu jer se uslov spajanja za svaku n-torku druge relacije evaluira kao unknown. Zbog toga se n-torka `<10000, Zagreb, NULL>` nije pojavila u rezultatu prethodnog upita.

Ukoliko se želi da se u rezultatu upita pojave sve n-torke određene relacije bez obzira da li postoje odgovarajuće n-torke druge relacije mora se primijeniti vanjsko spajanje relacija ukojem se relacije više ne tretiraju simetrično. U slučaju vanjskog spajanja zbog nesimetričnog tretiranja relacija jedna će relacija biti dominantna i njene će se n-torke pojaviti u rezultatu čak i ako ne postoji odgovarajuća n-torka druge relacije sa kojom se

mogu spojiti dok će druga relacija biti podređena i njene n-torke će se pojaviti u rezultatu jedino ukoliko postoje odgovarajuće n-torke dominantne relacije sa kojima se mogu spojiti. Kako se u rezultatu upita pojavljuju atributi obje relacije, uz vrijednosti atributa dominantne relacije pojaviće se `NULL` vrijednosti za atribute podređene relacije ukoliko ne postoji odgovrajuća n-torka podređene relacije.

Koja relacija će biti dominantna se u SQL-u određuje tipom vanjskog spajanja u `FROM` dijelu `SELECT` naredbe. Ukoliko se koristi `LEFT [OUTER] JOIN` radi se o lijevom vanjskom spajanju te se sve n-torke iz lijeve relacije koja je u tom slučaju dominantna pojavljuju u rezultatu upita. Analogno, desno vanjsko spajanje u kojem je relacija sa desne strane dominantna se implementira sa `RIGHT [OUTER] JOIN` dok se obostrano vanjsko spajanje u kojem su obje relacije dominantne implementira sa `FULL [OUTER] JOIN`. U MySQL-u obostrano vanjsko spajanje (`FULL OUTER JOIN`) nije podržano, ali bi se moglo simulirati sa unijom (`UNION` ili `UNION ALL` u zavisnosti od toga da li se pojavljuju duple n-torke) lijevog i desnog vanjskog spajanja. Osim toga, kao što se može primijetiti, `OUTER` je opciono i može se izostaviti kod određivanja tipa vanjskog spajanja.

```
SELECT mjesto.*, kanton.* FROM mjesto
  LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
```

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
72000	Zenica	4	NULL	NULL
76120	Brčko	12	12	Brčko distrikt
10000	Zagreb	NULL	NULL	NULL

```
SELECT mjesto.*, kanton.* FROM mjesto
  RIGHT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
```

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
76120	Brčko	12	12	Brčko distrikt
NULL	NULL	NULL	1	Sarajevski kanton

```
SELECT mjesto.*, kanton.* FROM mjesto
  FULL OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
```

MySQL
```
SELECT mjesto.*, kanton.* FROM mjesto
    LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
  UNION
  SELECT mjesto.*, kanton.* FROM mjesto
     RIGHT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
```

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
72000	Zenica	4	NULL	NULL
76120	Brčko	12	12	Brčko distrikt
10000	Zgreb	NULL	NULL	NULL
NULL	NULL	NULL	1	Sarajevski kanton

Na koji način će se NULL vrijednost prikazati u rezultatu upita zavisi od programskog alata koji se koristi za postavljanje upita i prikaz rezultata (prazna "kućica", <null>, NULL, itd.). Ukoliko se umjesto podrazumijevanog prikaza NULL vrijednosti u programskom alatu želi prikazati neka druga konkretna vrijednost (koja se ne pojavljuje kao vrijednost atributa) SUBP-i omogućavaju da se ista definiše u listi za selekciju SELECT naredbe. U MySQL-u se u tu svrhu koriste CASE izraz i IF funkcija (napomena: ne CASE i IF naredbe).

Opšti oblik CASE izraza je:

MySQL
```
CASE value
  WHEN [compare_value] THEN result
  [WHEN [compare_value] THEN result ...]
  [ELSE result]
END
```

MySQL
```
CASE
  WHEN [condition] THEN result
  [WHEN [condition] THEN result ...]
  [ELSE result]
END
```

Prvi oblik CASE izraza vraća result za prvo poređenje value = compare_value koje je istinito (true). Drugi oblik CASE izraza vraća result za prvi uslov koji je istinit (true). Ukoliko niti jedno poređenje ili uslov nije istinit (true), vraća se rezultat naveden pod ELSE ili NULL ukoliko ELSE nije navedeno. S obzirom da se u prvom slučaju koristi poređenje sa operatorom jednakosti (=), ovaj oblik CASE izraza nije moguće koristiti za slučaj poređenja NULL vrijednosti jer se NULL = NULL evaluira kao unknown a ne true. U tom slučaju mora se koristiti drugi oblik sa uslovom u kojem se upotrebljava IS NULL operator.

Primjer: *Za svako mjesto koje se pojavljuje u relaciji mjesto ispisati poštanski broj, naziv mjesta i naziv kantona kojem pripada. Mjestima kojima je šifra kantona postavljena na NULL vrijednost kao naziv kantona ispisati 'Nepoznat kanton'.*

MySQL
```
SELECT pbr, nazMjesto,
       CASE WHEN mjesto.sifKanton IS NULL THEN 'Nepoznat kanton'
         ELSE nazKanton
       END AS nazivKantona
  FROM mjesto LEFT OUTER JOIN kanton
  ON mjesto.sifKanton = kanton.sifKanton
```

MySQL IF funkcija prihvata tri parametra. Prvi parametar je izraz koji predstavlja uslov koji se ispituje, drugi parametar je izraz čija se vrijednost vraća ukoliko je uslov istinit (*true*), a treći izraz čija se vrijednost vraća ukoliko uslov nije istinit (*false* ili *unknown*).

MySQL
```
SELECT pbr, nazMjesto,
       IF(mjesto.sifKanton IS NULL , 'Nepoznat kanton', nazKanton)
       AS nazivKantona
  FROM mjesto LEFT OUTER JOIN kanton
  ON mjesto.sifKanton = kanton.sifKanton
```

pbr	nazMjesto	nazivKantona
75000	Tuzla	Tuzlanski kanton
72000	Zenica	Nepoznat kanton
76120	Brčko	Brčko distrikt
10000	Zagreb	Nepoznat kanton

Kao što je poznato, uslov dohvata naveden u `WHERE` dijelu `SELECT` naredbe se može postaviti nad atributima svih relacija navedenim u `FROM` dijelu, i atributima dominantne i atributima podređene relacije. U standardnom SQL-u ukoliko se uslov postavi nad atributima dominantne relacije u vanjskom spajanju pojaviće se samo one n-torke dominantne relacije koje ispunjavaju uslov i sa njima spojene n-torke podređene relacije na osnovu uslova spajanja.

```
SELECT mjesto.*, kanton.* FROM mjesto
  LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
  WHERE pbr < 76000
```

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
72000	Zenica	4	NULL	NULL
10000	Zagreb	NULL	NULL	NULL

Ukoliko se uslov dohvata postavi nad atributima podređene relacije, u vanjskom spajanju će se i dalje pojaviti sve n-torke dominantne relacije i sa njima spojene samo one n-torke podređene relacije koje zadovoljavaju i uslov spajanja i uslov dohvata.

```
SELECT mjesto.*, kanton.* FROM mjesto
  LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
  WHERE nazKanton LIKE 'T%'
```

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
72000	Zenica	4	NULL	NULL
76120	Brčko	12	NULL	NULL
10000	Zagreb	NULL	NULL	NULL

Međutim, u MySQL -u ovakav upit se pretvara u upit sa običnim (*inner*) spajanjem umjesto upita sa vanjskim (*outer*) spajanjem. Ovo se dešava za svaki upit sa vanjskim spajanjem u kojem je WHERE uslov odbijen zbog `NULL` vrijednosti (*null-rejected*), a to se dešava u slučaju kada se uslov evaluira kao *false* ili *unknown* za bilo koju n-torku sa `NULL` vrijednostima generisanu u operaciji vanjskog spajanja u MySQL-u. Za gornji primjer takvi *null-rejected* uslovi bi bili:

```
kanton.nazKanton IS NOT NULL
```

```
kanton.sifKanton > 3

kanton.sifKanton <= mjesto.sifKanton

kanton.sifKanton > 5 OR kanton.nazKanton LIKE 'S%'
```

U skladu sa navedenim primjerima, opšte pravilo kojim se određuje da li je uslov odbačen zbog `NULL` vrijednosti (*null-rejected*) u operaciji vanjskog spajanjau MySQL-u je prilično jednostavno:

- Ukoliko je u formi `A IS NOT NULL` gdje je `A` atribut bilo koje tabele koja se obično spaja
- Ukoliko je predikat koji sadrži atribute tabele koja se obično spaja a koji se evaluira kao *unknown* kada je jedan od njegovih argumenata `NULL`
- Ukoliko se radi o konjunkciji (`AND` operator) u kojoj postoji *null-rejected* uslov
- Ukoliko se radi o disjunkciji (`OR` operator) *null-rejected* uslova

Zbog toga je rezultat posljednjeg upita u MySQL-u:

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton

Ukoliko se ipak i u MySQL-u žele dohvatiti sve n-torke dominantne relacije bez obzira da li n-torke zadovoljavaju uslov dohvata postavljen nad atributima podređene relacije ili ne (kao u standardnom SQL-u) prethodni upit je potrebno postaviti u sljedećem obliku:

```
MySQL SELECT mjesto.*, kanton.* FROM mjesto
        LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
        WHERE nazKanton LIKE 'T%' OR nazKanton IS NULL
```

U ovom slučaju rezultat upita u MySQL-u je sljedeći:

pbr	nazMjesto	sifKanton	sifKanton	nazKanton
75000	Tuzla	3	3	Tuzlanski kanton
72000	Zenica	4	NULL	NULL
76120	Brčko	12	NULL	NULL
10000	Zagreb	NULL	NULL	NULL

Vanjsko spajanje tri ili više relacija

Ukoliko se u upitu spajaju dvije relacije takvo spajanje može biti obično spajanje (*inner*) ili vanjsko spajanje (*outer*). Međutim, ukoliko u upitu učestvuje više relacija tada su moguće kombinacije običnog i vanjskog spajanja. Te kombinacije bi se mogle svesti na tri oblika spajanja koji će biti opisani u nastavku, a u svim primjerima će uz relacije mjesto (izuzimajući n-torku za mjesto Zagreb) i kanton biti korištena i relacija `clan`:

CLAN

sifClan	imeClan	prezClan	pbrStan
1001	Mensur	Kasumović	75300
1002	Student	Studentić	72000
1003	Marina	Pejić	75000

U prvom obliku se relacije vanjski spajaju, odnosno relacija `mjesto` se vanjski spaja sa relacijom `kanton`, a potom neovisno o tom spajanju se vanjski spaja sa relacijom `clan`. Takav upit bi imao sljedeći oblik:

```
SELECT mjesto.*, kanton.*, pbrRod, prezClan FROM mjesto
  LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton
  LEFT OUTER JOIN clan ON mjesto.pbr = clan.pbrRod
```

S obzirom da je u ovakvom obliku vanjskog spajanja relacija `mjesto` dominantna relacija u odnosu na relaciju `kanton` i na relaciju `clan`, u rezultatu upita će se pojaviti sve n-torke relacije `mjesto` te odgovarajuće n-torke iz relacija `kanton` i `clan` ukoliko postoje, odnosno ukoliko zadovoljavaju uslov spajanja.

pbr	nazMjesto	sifKanton	nazKanton	pbrRod	prezClan
75000	Tuzla	3	Tuzlanski kanton	75000	Pejić
72000	Zenica	4	NULL	72000	Studentić
76120	Brčko	12	Brčko Distrikt	NULL	NULL

U drugom obliku se relacija `clan` vanjski spaja sa obično (prirodno) spojenim relacijama `mjesto` i `kanton`. Takav upit bi imao sljedeći oblik:

```
SELECT prezClan, pbrRod, mjesto.*, nazKanton FROM clan
  LEFT OUTER JOIN
  (mjesto INNER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton)
  ON clan.pbrRod = mjesto.pbr
```

S obzirom da je u ovakvom obliku vanjskog spajanja relacija `clan` dominantna relacija u odnosu na rezultat običnog (prirodnog) spajanja relacija `mjesto` i `kanton`, u rezultatu upita će se pojaviti sve n-torke relacije `clan` te uz njih n-torke relacija `mjesto` i `kanton` ukoliko postoji odgovarajući rezultat običnog (prirodnog) spajanja relacija `mjesto` i `kanton`.

prezClan	pbrRod	pbr	nazMjesto	sifKanton	nazKanton
Kasumović	75300	NULL	NULL	NULL	NULL
Studentić	72000	NULL	NULL	NULL	NULL
Pejić	75000	75000	Tuzla	3	Tuzlanski kanton

U trećem obliku se relacija `clan` vanjski spaja sa vanjski spojenim relacijama `mjesto` i `kanton`. Takav upit bi imao sljedeći oblik:

```
SELECT prezClan, pbrRod, mjesto.*, nazKanton FROM clan
  LEFT OUTER JOIN
  (mjesto LEFT OUTER JOIN kanton ON mjesto.sifKanton = kanton.sifKanton)
  ON clan.pbrRod = mjesto.pbr
```

S obzirom da je u ovakvom obliku vanjskog spajanja relacija `clan` dominantna relacija u odnosu na rezultat vanjskog spajanja relacija `mjesto` i `kanton`, u rezultatu upita će se pojaviti sve n-torke relacije `clan` te uz njih odgovarajuće n-torke relacije `mjesto`, odnosno ukoliko zadovoljavaju uslov spajanja. Pored toga, relacija `mjesto` je dominantna u odnosu

na relaciju kanton te će se u rezultatu uz n-torke relacije mjesto pojaviti i odgovarajuće n-torke relacije kanton.

prezClan	pbrRod	pbr	nazMjesto	sifKanton	nazKanton
Kasumović	75300	NULL	NULL	NULL	NULL
Studentić	72000	72000	Zenica	4	NULL
Pejić	75000	75000	Tuzla	3	Tuzlanski kanton

Primjeri upita za vježbu

Q1: *Knjigama čiji primjerci nisu posuđivani u tekućoj godini postaviti vrijednost atributa vrijednost na NULL.*

```
UPDATE knjiga SET vrijednost = NULL
  WHERE sifKnjiga NOT IN
    (SELECT sifKnjiga FROM primjerak INNER JOIN posudba
       ON primjerak.invBroj = posudba.invBroj
       WHERE YEAR(datumPos) = YEAR(CURRENT_DATE))
```

Q2: *Članovima koji imaju inicijale E.M. i ne stanuju u Tuzlanskom kantonu postaviti poštanski broj rođenja na NULL.*

```
UPDATE clan SET pbrStan = NULL
  WHERE SUBSTRING(imeClan, 1, 1) = 'E'
  AND SUBSTRING(prezClan, 1, 1) = 'M'
  AND pbrStan NOT IN
    (SELECT pbr FROM mjesto INNER JOIN kanton
       ON mjesto.sifKanton = kanton.sifKanton
       WEHERE nazKanton = 'Tuzlanski kanton')
```

Q3: *Ispisati broj knjiga čiji primjerci još nisu vraćeni ili nije postavljen datum do kojeg se primjerak mora vratiti.*

```
SELECT COUNT(DISTINCT sifKnjiga) FROM primjerak INNER JOIN posudba
  ON primjerak.invBroj = posudba.invBroj
  WHERE datumVrac IS NULL OR datumDo IS NULL
```

Q4: *Ispisati šifru i naziv organizacionih jedinica i broj predmeta koji im pripadaju za koje je evidentirana skraćenica predmeta ali nije broj ECTS kredita.*

```
SELECT orgjed.sifOrgjed, nazOrgjed, COUNT(sifPred) FROM orgjed
  INNER JOIN pred ON orgjed.sifOrgjed = pred.sifOrgjed
  WHERE skrPred IS NOT NULL AND brojECTS IS NULL
  GROUP BY 1, 2
```

Q5: *Ispisati sve podatke o predmetima koji imaju veći broj ECTS kredita od svih svojih preduslova. Voditi računa da za neke od predmeta broj ECTS kredita nije evidentiran.*

```
SELECT * FROM pred
  WHERE brojECTS > ALL
    (SELECT brojECTS FROM pred uslov INNER JOIN preduslov
       ON uslov.sifPred = preduslov.sifPreduslov
       WHERE preduslov.sifPred = pred.sifPred
       AND brojECTS IS NOT NULL)
```

Q6: *Ispisati nazive mjesta u kojima nije rođen niti jedan član koji iznajmljeni primjerak nije vratio na vrijeme. Voditi računa da za neke članove poštanski broj mjesta rođenja nije evidentiran.*

```
SELECT nazMjesto FROM mjesto
  WHERE pbr NOT IN (SELECT pbrRod FROM clan INNER JOIN posudba
                      ON clan.sifClan = posudba.sifClan
                      WHERE datumDo < datumVrac
                      AND pbrRod IS NOT NULL)
```

Q7: *Za svaki predmet koji je evidentiran u bazi podataka, a preduslov je bar jednom predmetu, ispisati skraćenicu i naziv predmeta i naziv organizacione jedinice kojoj predmet pripada. Za predmete za koje nije evidentirana pripadna organizaciona jedinica kao naziv organizacione jedinice ispisati 'Nepoznato'.*

```
SELECT skrPred, nazPred IF(pred.sifOrgjed IS NULL, 'Nepoznato', nazOrgjed)
  FROM pred LEFT OUTER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
  WHERE sifPred IN (SELECT sifPreduslov FROM preduslov)
```

```
SELECT skrPred, nazPred
       CASE WHEN pred.sifOrgjed IS NULL THEN 'Nepoznato'
         ELSE nazOrgjed
       END
  FROM pred LEFT OUTER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed
  WHERE sifPred IN (SELECT sifPreduslov FROM preduslov)
```

Q8: *Za svaki predmet koji pripada organizacionoj jedinici Računarstvo i informatika ispisati naziv predmeta i naslove njegove obavezne literature. U listi se trebaju pojaviti svi predmet bez obzira da li za neki od predmeta nije evidentirana obavezna literatura. Ispis sortirati po nazivu predmeta, a unutar toga po važnosti literature.*

```
SELECT nazPred, naslov, rbrVaznost FROM
  (pred INNER JOIN orgjed ON pred.sifOrgjed = orgjed.sifOrgjed)
  LEFT OUTER JOIN
  (literatura INNER JOIN knjiga ON literatura.sifKnjiga = knjiga.sifKnjiga)
  ON pred.sifPred = literatura.sifPred
    WHERE nazOrgjed = 'Računarstvo i informatika'
    ORDER BY 1, 3
```

Q9: *Za svakog evidentiranog studenta u bazi podataka ispisati broj indeksa, prezime i ime i broj obavljenih posudbi u tekućoj godini, te prosječno i ukupno trajanje tih posudbi u danima. U ispisu ne prikazivati NULL nego konkretne brojne vrijednosti. Ispis sortirati prema ukupnm broju obavljenih posudbi a studente koji eventualno imaju isti broj posudbi sortirati prema prezimenu i imenu.*

```
SELECT brIndex, prezStud, imeStud, COUNT(invBroj),
  IF(COUNT(invBroj) = 0, 0, AVG(DATEDIFF(datumVrac, datumPos)) prosjBrDana,
  IF(COUNT(invBroj) = 0, 0, SUM(DATEDIFF(datumVrac, datumPos)) ukBrDana
  FROM (stud INNER JOIN clan ON stud.sifClan = clan.sifClan)
  LEFT OUTER JOIN posudba ON clan.sifClan = posudba.sifClan
    WHERE YEAR(datumPos) = YEAR(CURRENT_DATE) OR datumPos IS NULL
    GROUP BY 1, 2, 3
    ORDER BY 4 DESC, 2, 3
```

Q10: *Za svaku knjigu evidentiranu u bazi podataka ispisati naslov i prezime i ime prvog i drugog autora knjige u istom zapisu. U rezultatu se moraju pojaviti sve knjige bez obzira da li imaju evidentirane autore (jednog ili oba).*

```
SELECT naslov, prvi.prezAutor, prvi.imeAutor, drugi.prezAutor,
       drugi.imeAutor
  FROM knjiga LEFT OUTER JOIN
  (autor prvi INNER JOIN napisao nap1
   ON prvi.sifAutor = nap1.sifAutor AND nap1.rbrAutor = 1)
  ON knjiga.sifKnjiga = nap1.sifKnjiga LEFT OUTER JOIN
  (autor drugi INNER JOIN napisao nap2
   ON drugi.sifAutor = nap2.sifAutor AND nap2.rbrAutor = 2)
```

IMPLEMENTACIJA PRAVILA INTEGRITETA U SQL-u

U SUBP-ima, integritet podataka osigurava tačnost, pouzdanost i konzistentnost podataka tokom bilo koje operacije koja podrazumijeva manipuliranje podacima, dohvat podataka, pohranu podataka, operacije kreiranja sigurnosne kopije ili obnove baze podataka, itd. Da ne bi došlo do narušavanja konzistentnosti, tačnosti i pouzdanosti podataka prilikom svih gore navednih operacija u bazi podataka se definiraju integritetska ograničenja koja postaju sastavni dio sheme baze podataka i pohranjuju se u riječnik podataka baze podataka (*data dictionary*, *catalogue*, *repository*). Korištenjem ovih pohranjenih integritetskih ograničenja SUBP kontrolira sve operacije kojima se mijenja sadržaj baze podataka te osigurava da nakon obavljanja tih operacija baza podataka bude u konzistentnom stanju, odnosno u stanju u kojem su opet zadovoljena sva integritetska ograničenja.

- Integritetska ograničenja podrazumijevaju:
- Integritet ključa (*key integrity*)
- Entitetski integritet (*entity integrity*)
- Ograničenja `NULL` vrijednosti (*constraints on NULL*)
- Referencijski integritet (*referential integrity*)
- Domentski integritet (*domain integrity*)
- Opšta integritetska ograničenja (*general integrity constraints*)

U SQL-u se integritetska ograničenja implementiraju prilikom kreiranja relacije u `CREATE TABLE` naredbi ili naknadno definirati upotrebom `ALTER TABLE` naredbe. Ukoliko se integritetsko ograničenje definira prilikom kreiranja relacije onda se može definirati neposredno uz definiciju atributa (ukoliko se odnosi samo na dati atribut) ili nakon definicije svih atributa (ukoliko se odnosi na relaciju, odnosno na više atributa relacije).

`column_constraint` iz osnovnog oblika `CREATE TABLE` naredbe navedenom u prvom poglavlju ima sljedeći opšti oblik:

```
[CONSTRAINT constraintName]
{ NOT NULL | UNIQUE | PRIMARY KEY | CHECK (condition) |
REFERENCES ref_table[(ref_column)] [ON DELETE action] [ON UPDATE action] }
```

gdje se `condition` odnosi samo da atribut uz koji se definira ograničenje, a `action` može poprimiti jednu od vrijednosti: `NO ACTION`, `CASCADE`, `SET NULL`, `SET DEFAULT`.

Slično, `table_constraint` iz osnovnog oblika `CREATE TABLE` naredbe ima sljedeći opšti oblik:

```
[CONSTRAINT constraintName]
{ UNIQUE (column_name [, ...]) |
PRIMARY KEY UNIQUE (column_name [, ...])| CHECK (condition) |
FOREIGN KEY UNIQUE (column_name [, ...]) REFERENCES ref_table[(ref_column)]
   [ON DELETE action] [ON UPDATE action] }
```

gdje `condition` može uključivati sve atribute relacije.

Opšti oblik naknadnog definiranja integritetskih ograničenja uz pomoć `ALTER TABLE` naredbe već je naveden u prvom poglavlju.

Integritet ključa

Za svaki od mogućih ključeva relacije, odnosno atribut ili skup atributa koji mogu predstavljati ključ relacije, ne smiju postojati dvije n-torke relacije sa jednakim vrijednostima ključa. Drugim riječima, vrijednost ključa u svakoj n-torci relacije mora biti jedinstvena (*unique*). Tako npr. u relaciji:

```
CLAN(sifClan, imeClan, prezClan, pbrRod, pbrStan, datRod, jmbgClan)
```

mogući ključevi relacije su `K1 = {sifClan}` i `K2 = {jmbgClan}` te da bi bio zadovoljen integritet ključa u relaciji `clan` ne smiju postojati dvije n-torke koje imaju jednake vrijednosti atributa `sifClan` ili atributa `jmbgClan`.

Ukoliko je ključ složen kao npr. u relaciji:

```
POSUDBA(invBroj, datumPos, datumVrac, datumDo, sifClan)
```

u kojoj je ključ `K = {invBroj, datumPos}`, da bi bio zadovoljen integritet ključa ne smiju postojati dvije n-torke koje imaju istu kombinaciju vrijednosti atributa `invBroj` i `datumPos`.

Na osnovu prethodno navedenog i osnovnog oblika definiranja ograničenja integriteta, može se zaključiti da se integritet ključa definira pomoću naznake UNIQUE prilikom definicije atributa:

```
...
, jmbgClan      CHAR(13)      UNIQUE
, ...
```

ili na kraju definicije relacije sa `CREATE TABLE` naredbom:

```
...
, UNIQUE (invBroj, datumPos)
, ...
```

Većina SUBP-a će prilikom definiranja integriteta ključa (`UNIQUE` ograničenja) automatski kreirati `UNIQUE` indeks nad atributima nad kojima se definira ograničenje.

Entitetski integritet

Iako relacija može imati više (mogućih) ključeva, samo jedan od njih će biti odabran i definiran kao primarni ključ relacije. Entitetski integritet se upravo odnosi na primarni ključ relacije i definira da niti jedan dio (atribut) primarnog ključa ne smije poprimiti `NULL` vrijednost. To znači da za dvije relacije iz prethodnog primjera u relaciji `clan` atribut `sifClan` koji je primarni ključ relacije ne smije poprimiti `NULL` vrijednost niti u jednoj n-torki relacije, odnosno niti jedan od atributa `invBroj` i `datumPos` koji čine primarni ključ relacije `posudba` niti u jednoj n-torki relacije ne smiju poprimiti `NULL` vrijednost.

U SQL-u se entitetski integritet definira pomoću naznake `PRIMARY KEY` čime se istovremeno osigurava i jedinstvenost vrijednosti primarnog ključa (nije potrebno dodatno navoditi i `UNIQUE` ograničenje). Kao i u slučaju definiranja integriteta ključa, entitetski integritet se definira prilikom definicije atributa:

```
...
, sifClan      INTEGER        PRIMARY KEY
, ...
```

ili na kraju definicije relacije sa `CREATE TABLE` naredbom:

```
...
, PRIMARY KEY (invBroj, datumPos)
, ...
```

Pravilo entitetskog integriteta nužno mora zadovoljiti za svaku relaciju i sve n-torke relacije te se niti u jednoj relaciji baze podataka ne može pojaviti n-torka kod koje bi primarni ključ ili bilo koji njegov dio mogao poprimiti `NULL` vrijednost. Većina SUBP-a će prilikom definiranja entitetskog integriteta (primarnog ključa sa `PRIMARY KEY`) automatski kreirati `UNIQUE` indeks nad atributima nad kojima se definira ograničenje.

Ograničenja NULL vrijednosti

Već je ranije navedeno da se za pojedine atribute relacije može definirati ograničenje prema kojem vrijednosti atributa ne mogu poprimiti `NULL` vrijednost. Ovo ograničenje integriteta se definira pomoću naznake `NOT NULL` uz definiciju atributa i jedino je ograničenje integriteta koje se ne može definirati na nivou relacije na kraju `CREATE TABLE` naredbe.

```
...
, imeClan      NCHAR(25)      NOT NULL
, prezClan     NCHAR(25)      NOT NULL
, ...
```

Referencijski integritet

Osim primarnog ključa u relaciji može biti definiran i strani ključ, a to je atribut ili skup atributa koji referencira (se odnosi, pokazuje) na primarni ključ iste (u slučaju postojanja refleksivne veze) ili druge relacije. Referencijalni integritet se upravo odnosi na strani ključ relacije i osigurava konzistentnost između n-torki relacija (ili iste relacije) na način da strani ključ u relaciji (atributi koji čine strani ključ) može poprimiti samo vrijednosti primarnog ključa relacije na koju referencira ili `NULL` vrijednost. Postoje slučajevi kada strani ključ ne može poprimiti `NULL` vrijednost, a to je u situaciji kada se pravilo referencijskog integriteta sukobljava sa pravilom entitetskog integriteta, odnosno kada je strani ključ ujedno i dio primarnog ključa relacije, ili ukoliko je za strani ključ već definirano ograničenje `NULL` vrijednosti (`NOT NULL`).

U SQL-u se referencijski integritet definira pomoću naznake `REFERENCES` uz definiciju atributa koji je strani ključ:

```
...
, sifClan      INTEGER        REFERENCES clan(sifClan)
, ...
```

ili pomoću naznake `FOREIGN KEY` nakon definicije svih atributa u `CREATE TABLE` naredbi:

```
...
, FOREIGN KEY (invBroj) REFERENCES primjerak(invBroj)
, ...
```

Na ovaj način SUBP osigurava referencijalni integritet i neće dozvoliti izvođenje operacije kojom bi se postavila vrijednost stranog ključa na vrijednost koja ne postoji kao vrijednost primarnog ključa relacije na koju referencira. Konkretno:

- Nije dozvoljeno unijeti n-torku u relaciju posudba sa vrijednošću atributa invBroj koja ne postoji u relaciji primjerak
- Nije dozvoljeno ažurirati vrijednost atributa invBroj u nekoj n-torci relacije posudba na vrijednost koja ne postoji za atribut invBroj u relaciji primjerak
- Nije dozvoljeno brisanje n-torke iz relacije primjerak ukoliko u relaciji posudba već postoji n-torka sa vrijednošću atributa invBroj koja je jednaka vrijednosti primarnog ključa (atributa invBroj) u n-torci koja treba biti obrisana

Međutim, iako u većini slučajeva ovakva održavanja pravila referencijalnog integriteta zadovoljavaju, odnosno u većini slučajeva se ne dozvoljava evidentiranje pozajmljivanja nepostojećeg primjerka knjige u fakultetskoj biblioteci, može se desiti situacija kada određeni primjerak ili svi primjerci jedne knjige više nisu upotrebljivi za pozajmljivanje te se žele arhivirati podaci o takvim primjercima i knjigama koji se više ne koriste (ne

iznajmljuju) te ih izbrisati iz baze podataka. Da bi se ovakva operacija mogla izvesti, a da bi se i dalje očuvalo pravilo referencijalnog integriteta, uz brisanje (ista je situacija i sa ažuriranjem) određene n-torke (ciljna n-torka) potrebno je izvršiti odgovarajuće kompenzacijske akcije nad n-torkama relacija (pozivajuće n-torke) koje imaju vrijednosti stranog ključa jednake vrijednosti primarnog ključa n-troke koja će biti obrisana (ažurirana). U skladu s tim referencijski integritet dozvoljava sljedeće strategije:

- Ciljna n-torka se ne može obrisati (ne može se promijeniti vrijednost primarnog ključa ciljne n-torke) ukoliko u bazi postoje odgovarajuće pozivajuće n-torke
- Uz brisanje ciljne n-troke (ažuriranje vrijednosti primarnog ključa ciljne n-torke) treba izvesti brisanje svih pozivajućih n-torki (ažuriranje vrijednosti stranog ključa pozivajuće n-torke na novu vrijednost primarnog ključa ciljne n-torke)
- Kao dio operacije brisanja ciljne n-torke (ažuriranja primarnog ključa ciljne n-torke), vrijednosti stanog ključa u pozivajućim n-torkama se postavljaju na `NULL` vrijednost.
- Kao dio operacije brisanja ciljne n-torke (ažuriranja primarnog ključa ciljne n-torke), vrijednosti stanog ključa u pozivajućim n-torkama se postavljaju na `DEFAULT` vrijednost.

Koja od navedenih strategija će se primijeniti za očuvanje referencijskog integriteta definira se u SQL-u prilikom definiranja stranog ključa sa naznakama `ON { DELETE | UPDATE } NO ACTION`, `ON { DELETE | UPDATE } CASCADE`, `ON { DELETE | UPDATE } SET NULL` i `ON { DELETE | UPDATE } SET DEFAULT`, respektivno. Ukoliko se ne navede niti jedna od naznaka prilikom definiranja stranog ključa, SUBP-i primjenjuju prvu strategiju, odnosno kao da je definirano `ON { DELETE | UPDATE } NO ACTION`.

Neki SUBP-i će prilikom definiranja ograničenja referencijskog integriteta (stranog ključa sa `FOREIGN KEY`) automatski kreirati indekse nad atributima nad kojima se definira ograničenje.

Primjer: *Napisati naredbu za kreiranje relacije posudba u kojoj će biti definirana pravila entitetskog i referecijalnog integriteta sa različitim strategijama očuvanja.*

```
CREATE TABLE posudba
  ( invBroj       INTEGER     NOT NULL
  , datumPos      DATE        NOT NULL
  , datumVrac     DATE
  , datumDo       DATE        NOT NULL
  , sifClan       INTEGER
  , PRIMARY KEY (invBroj, datumPos)
  , FOREIGN KEY (invBroj) REFERENCES primjerak(invBroj) ON DELETE CASCADE
  , FOREIGN KEY (sifClan) REFERENCES clan(sifClan) ON DELETE SET NULL
  );
```

Domenski integritet i opšta integritetska ograničenja

Kao što i sam naziv govori, domenski integritet definira domenu atributa, odnosno skup vrijednosti koje atribut može poprimiti. Djelimično se definira samom definicijom tipa

podatka za atribut pa tako definiranjem tipa podatka `SMALLINT` određuje se domena tog atributa kao skup cijelih brojeva u intervalu od -32767 do 32767. Preciznije određivanje domene atributa definira se u SQL-u pomoću `CHECK` ograničenja:

```
...
, rbrVaznost      SMALLINT    CHECK (rbrVaznost BETWEEN 1 AND 5)
, ...
```

Sa CHECK ograničenjem se definiraju i sva ostala opšta integritetska ograničenja koja podrazumijevaju ograničenja odnosa među vrijednostima atributa. Npr. rok do kojeg student mora vratiti knjigu ne može biti manji od 2 dana niti veći od 15 dana. Ovo opšte integritetsko ograničenje se definira na sljedeći način:

```
...
, CHECK (datumDo - datumPos >= 2 AND datumDo - datumPos <= 15)
, ...
```

Za svako ograničenje integriteta koje se definira korisno je dati vlastiti naziv koji će se korisniku dojaviti prilikom pokušaja obavljanja naredbe koja bi narušila to integritetsko ograničenje, a osim toga korisnik može koristiti taj naziv da bi uklonio odgovarajuće integritetsko ograničenje. Npr. ukoliko se imenuje prethodno definirano CHECK ograničenje:

```
...
, CONSTRAINT checkPosudba
  CHECK (datumDo - datumPos >= 2 AND datumDo - datumPos <= 15)
, ...
```

Korisnik za uklanjanje tog definiranog ograničenja može izvesti naredbu:

```
ALTER TABLE posudba DROP CONSTRAINT chkPosudba
```

Primjer: *Napisati naredbu za kreiranje relacije LET u koju se pohranjuju podaci o letovima aviona. Atributi relacije su šifra leta (cijeli broj), datum obavljanja leta, registarski broj aviona koji leti na letu (niz od 5 karaktera), šifre aerodroma sa kojeg se polijeće i na koji se slijeće (cijeli brojevi) kao i vrijeme polijetanja i slijetanja. Primarni ključ relacije čine atributi šifra leta i datum leta. U naredbi implementirati pravila entitetskog i refernecijalnog integriteta te ograničenja da let ne može trajati manje od pola sata i da se ne može poletjeti i sletjeti na isti aerodrom.*

```
CREATE TABLE let (
    sifLet        INTEGER     NOT NULL
  , datumLet      DATE        NOT NULL
  , regBr         CHAR(5)     NOT NULL  REFERENCES avion (regBr)
  , sifAerPol     INTEGER     NOT NULL  REFERENCES aerodrom (sifAer)
  , vrijemePol    TIME
  , sifAerSlet    INTEGER     NOT NULL  REFERENCES aerodrom (sifAer)
  , vrijemeSlet   TIME
  , PRIMARY KEY (sifLet, datumLet)
  , CHECK (TIMEDIFF(vrijemeSlet, vrijemePol) >= '00:30:00')
  , CHECK (sifAerPol != sifAerSlet)
  );
```

DODATAK A

MODEL OGLEDNE BAZE PODATAKA

U svim primjerima u knjizi korištena je baza podataka fakultetske biblioteke u kojoj se pohranjuju podaci o posudbama primjeraka knjiga njenih članova koji su studenti i zaposlenici fakulteta.

U bazi podataka vodi se evidencija o svim knjigama koje fakultetska biblioteka posjeduje, a za svaku knjigu se evidentira šifra koja ju jedinstveno određuje, naslov knjige, godina njenog izdavanja, bodovi koji odražavaju značaj ili popularnost knjige u skladu sa brojem posudbi primjeraka te knjige i njene zastupljenosti kao obavezne ili neobavezne literature na predmetima te autori koji potpisuju knjigu. Za autora se evidentira šifra koja ga jedinstveno određuje, prezime i ime autora te koji je po redu autor koji potpisuje određenu knjigu. Biblioteka posjeduje određeni broj primjeraka svake knjige za koje se evidentira inventarni broj, kratak opis stanja u kojem se primjerak nalazi te da li je primjerak i dalje u upotrebi (da li se još uvijek izdaje u biblioteci).

U biblioteci se za svaku knjigu evidentira i da li se koristi kao literatura na nekom od predmeta koji se izvode na fakultetu. Evidentira se da li se knjiga na određenom predmetu koristi kao obavezna ili neobavezna literatura i koja je po redu po važnosti na tom predmetu. Za predmete se evidentira šifra predmeta koja ih jednoznačno određuje, skraćenica predmeta, naziv predmeta, broj ECTS kredita koje predmet nosi u studijskom programu i organizaciona jedinica (odnosno uža naučna oblast) kojoj predmet matično pripada. Za predmet se evidentiraju i predmeti koji su mu preduslov, odnosno iz kojih se moraju ostvariti ECTS krediti da bi se pristupilo nastavi i polaganju ispita iz tog predmeta. Jedan predmet može imati više predmeta kao preduslove, a također i sam može biti preduslov za više predmeta (npr. za predmet Operativni sistemi preduslovi su predmeti Strukture podataka i Arhitekutra računara, dok je sam predmet Operativni sistemi preduslov za predmete Računarske mreže i Sistemsko programiranje). Za organizacione jedinice se evidentira šifra i naziv organizacione jedinice.

Članovi biblioteke su studenti koji su upisani na fakultet i nastavnici koji izvode nastavu na fakultetu. Za svakog člana se evidentira šifra koja ga jedinstveno određuje, ime, prezime, datum rođenja, jedinstveni matični broj te mjesto rođenja i mjesto stanovanja. Za studente se dodatno evidentiraju matični broj (koji ih također jdinstveno određuje) i broj indeksa, dok se za nastavnike dodatno evidentira koeficijent za platu i organizaciona jedinica u kojoj je zaposlen odnosno ima izbor. Mjesta su opisana poštanskim brojem koji ih

jedinstveno identificira, nazivom i kantonom u kojem se nalaze. Za kanton se evidentira šifra i naziv kantona.

U bazi podataka se evidentiraju sve posudbe primjeraka knjiga od strane njenih članova. Prilikom posudbe evidentira se koji član je posudio koji primjerak, datum posudbe i datum do kojeg se primjerak knjige mora vratiti u biblioteku. Prilikom vraćanja primjerka knjige evidentira se datum vraćanja. Svaki član isti primjerak knjige može posuditi više puta, ali ne istog dana. U istom danu jedan član može posuditi više različitih primjeraka knjige.

U skladu sa detaljnim opisom sistema identifikovani su entiteti i veze, opisani vlastitim atributima te je kreiran ER (*Entity-Relationship*) model baze podataka.

Opis entiteta (primarni ključevi entiteta su istaknuti):

KANTON
sifKanton **- šifra kantona**
nazKanton - naziv kantona

MJESTO
pbr **- poštanski broj mjesta**
nazMjesto - naziv mjesta

CLAN
sifClan **- šifra člana u biblioteci**
imeClan - ime člana
prezClan - prezime člana
datRod - datum rođenja člana
jmbgClan - jedinstveni matični broj člana

STUD
mbrStud **- matični broj studenta**
brIndex - broj indeksa studenta

NAST
sifNast **- šifra nastavnika**
koef - koeficijent za platu

ORGJED
sifOrgjed **- šifra organizacione jedinice**
nazOrgjed - naziv organizacione jedinice

PRED
sifPred **- šifra predmeta**
skrPred - skraćenica predmeta
nazPred - naziv predmeta
brojECTS - broj ECTS kredita koje predmet nosi u studijskom programu

AUTOR
sifAutor **- šifra autora**
imeAutor - ime autora
prezAutor - prezime autora

KNJIGA
sifKnjiga **- šifra knjige**
naslov - naslov knjige
godIzdanja - godina izdavanja knjige
vrijednost - bodovi koji izražavaju značaj ili važnost knjige

PRIMJERAK
invBroj **- inventarni broj primjerka**
stanje - kratak opis stanja u kojem se primjerak nalazi
koristenje - da li se primjerak još uvijek izdaje ili ne

POSUDBA
invBroj **- inventarni broj posuđenog primjerka**
datumPos **- datum posudbe primjerka**
datumVrac - datum vraćanja primjerka
datumDo - datum do kojeg se primjerak mora vratiti

Opis veza (primarni ključevi veza su istaknuti):

seNalazi
pbr **- poštanski broj mjesta**
sifKanton - šifra kantona u kojem se mjesto nalazi

rođen
sifClan **- šifra člana**
pbr - poštanski broj mjesta rođenja člana

stanuje
sifClan **- šifra člana**
pbr - poštanski broj mjesta stanovanja člana

jeStud
mbrStud **- matični broj studenta**
sifClan - šifra člana

jeNast
sifNast **- šifra nastavnika**
sifClan - šifra člana

radi
sifNast **- šifra nastavnika**
sifOrgjed - šifra organizacione jedinice na kojoj je nastavnik zaposlen (izabran)

pripada
sifPred **- šifra predmeta**
sifOrgjed - šifra organizacione jedinice kojoj predmet matično pripada

ima
invBroj **- inventarni broj primjerka**
sifKnjiga - šifra knjige čiji je primjerak

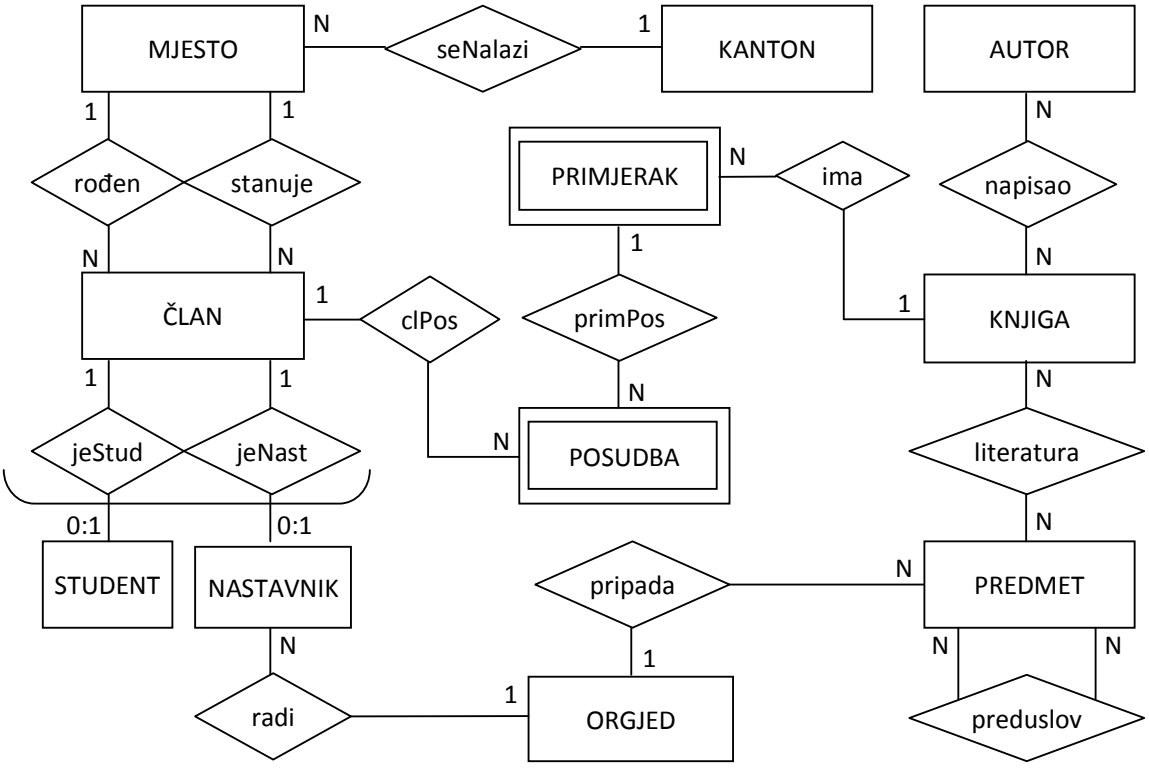
napisao
sifKnjiga **- šifra knjige**
sifAutor **- šifra autora knjige**
rbrAutor - redni broj koji je autor po redu koji potpisuje knjigu

literatura
sifKnjiga **- šifra knjige**
sifPred **- šifra predmeta na kojem se knjiga koristi kao literatura**
rbrVaznost - redni broj koja je literatura po važnosti na predmetu
obavezna - da li se knjiga koristi kao obavezna literatura ili ne

preduslov
sifPred **- šifra predmeta**
sifPred - šifra predmeta koji je preduslov

primPos
invBroj **- inventarni broj primjerka**
datumPos **- datum posudbe primjerka**

clPos
invBroj **- inventarni broj primjerka**
datumPos **- datum posudbe primjerka**
sifClan - šifra člana koji je posudio primjerak



Na osnovu kreiranog ER modela baze podataka jednostavno se (unijom shema sa jednakim ključevima) kreira relacijski model baze podataka. Naredbe za kreiranje baze podataka i relacija sa ugrađenim općim pravilima integriteta su date u nastavku:

```
CREATE DATABASE bibliotekaFET CHARACTER SET = 'utf8' COLLATE =
'utf8_croatian_ci';

CREATE TABLE kanton
  ( sifKanton     SMALLINT  PRIMARY KEY
  , nazKanton     NCHAR(40) NOT NULL
  );

CREATE TABLE mjesto
  ( pbr           INTEGER    NOT NULL
  , nazMjesto     NCHAR(40)  UNIQUE
  , sifKanton     SMALLINT   REFERENCES kanton (sifKanton)
  );

CREATE TABLE clan
  ( sifClan       SMALLINT    PRIMARY KEY
  , imeClan       NCHAR(25)   NOT NULL
  , prezClan      NCHAR(25)   NOT NULL
  , pbrRod        INTEGER     REFERENCES mjesto (pbr)
  , pbrStan       INTEGER     REFERENCES mjesto (pbr)
  , datRod        DATE
  , jmbgClan      CHAR(13)
  );

CREATE TABLE stud
  ( mbrStud       INTEGER     PRIMARY KEY
  , brIndex       CHAR(13)
  , sifClan       INTEGER     REFERENCES clan (sifClan)
  );

CREATE TABLE orgjed
  ( sifOrgjed     INTEGER     PRIMARY KEY
  , nazOrgjed     NCHAR(60)   UNIQUE
  );

CREATE TABLE nast
  ( sifNast       INTEGER      PRIMARY KEY
  , sifOrgjed     INTEGER      REFERENCES orgjed (sifOrgjed)
  , koef          DECIMAL(3,2) NOT NULL
  , sifClan       INTEGER      REFERENCES clan (sifClan)
  );

CREATE TABLE pred
  ( sifPred       INTEGER     PRIMARY KEY
  , skrPred       CHAR(8)
  , nazPred       NCHAR(60)   NOT NULL
  , sifOrgjed     INTEGER     REFERENCES orgjed (sifOrgjed)
  , brojECTS      INTEGER
  );
```

```
CREATE TABLE knjiga
  ( sifKnjiga     INTEGER     PRIMARY KEY
  , naslov        NCHAR(80)   NOT NULL
  , godIzdanja    INTEGER     NOT NULL
  , vrijednost    SMALLINT
  );

CREATE TABLE primjerak
  ( invBroj       INTEGER     PRIMARY KEY
  , stanje        NCHAR(150)
  , sifKnjiga     INTEGER     REFERENCES knjiga (sifKnjiga)
  , koristenje    BOOLEAN
  );

CREATE TABLE posudba
  ( invBroj       INTEGER     NOT NULL
  , datumPos      DATE        NOT NULL
  , datumVrac     DATE
  , datumDo       DATE        NOT NULL
  , sifClan       INTEGER     REFERENCES clan (sifClan)
  , PRIMARY KEY (invBroj, datumPos)
  );

CREATE TABLE autor
  ( sifAutor      INTEGER     PRIMARY KEY
  , imeAutor      NCHAR(25)   NOT NULL
  , prezAutor     NCHAR(25)   NOT NULL
  );

CREATE TABLE napisao
  ( sifKnjiga     INTEGER     REFERENCES knjiga (sifKnjiga)
  , sifAutor      INTEGER     REFERENCES autor (sifAutor)
  , rbrAutor      SMALLINT    DEFAULT 1
  , PRIMARY KEY (sifKnjiga, sifAutor)
  );

CREATE TABLE literatura
  ( sifKnjiga     INTEGER     REFERENCES knjiga (sifKnjiga)
  , sifPred       INTEGER     REFERENCES pred (sifPred)
  , rbrVaznost    SMALLINT    DEFAULT 1
  , obavezna      BOOLEAN
  , PRIMARY KEY (sifKnjiga, sifPred)
  );

CREATE TABLE preduslov
  ( sifPred       INTEGER     REFERENCES pred (sifPred)
  , sifPreduslov  INTEGER     REFERENCES pred (sifPred)
  , PRIMARY KEY (sifPred, sifPredUslov)
  );
```

DODATAK B

MySQL WORKBENCH INTERAKTIVNI ALAT ZA RAD SA MySQL BAZOM PODATAKA

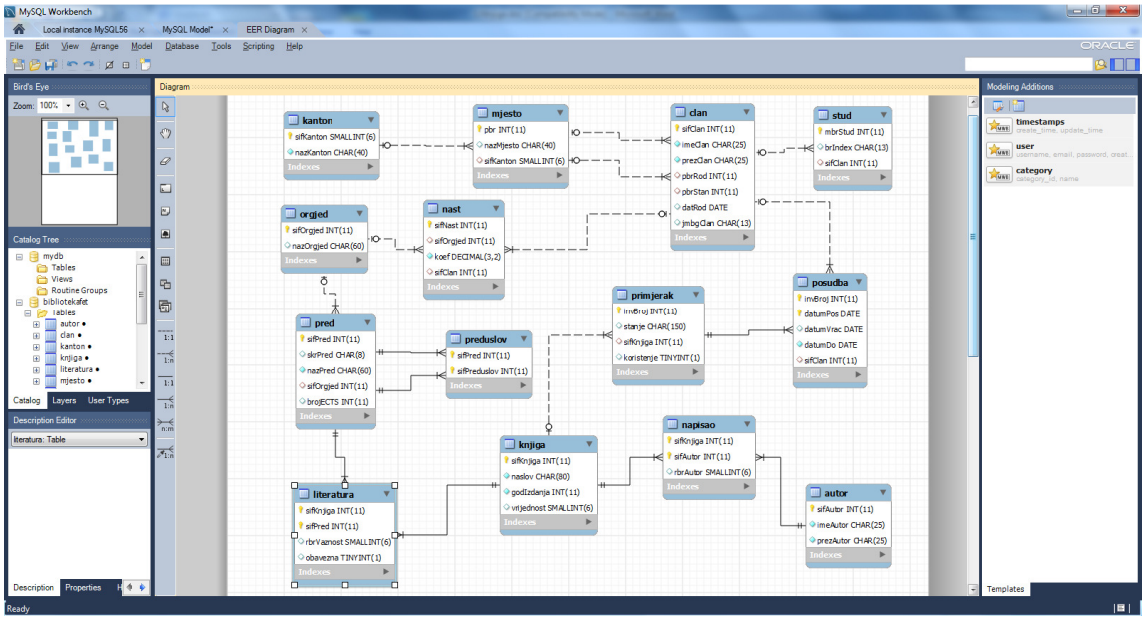
MyQSL Workbench predstavlja moćan vizuelni alat za modeliranje i dizajniranje baze podataka koji pomaže programerima i administratorima baze podataka (DBA) u kreiranju novih te modifikovanju postojećih modela MySQL baza podataka upotrebom direktnog/reverznog inženjeringa kao i promjenu funkcija upravljanja bazama podataka. MySQL Workbench je dizajniran kako bi povećao produktivnost korisnika pružajući sredstva za konceptualizaciju, komunikaciju, izgradnju i održavanje ključnih poslovnih metapodataka, baza podataka visokih performansi i skladišta podataka.

MySQL Workbench grafičko sučelje i automatizirani procesi osiguravaju podršku prilagođenu za različite profile u okviru modernog poduzeća, koji uključuju administratore baze podataka, programere aplikacija, arhitekte podataka i IT upravljanje. Proizvod je dostupan za Windows, Linux i Mac operativne sisteme pružajući prilagođeni *look and feel* kako bi korisnici mogli dizajnirati svoje baze podataka iz svih popularnih desktop operativnih sistema.

Dizajniranje baze podataka

Temeljna namjena MySQL Workbench-a je da programerima i DBA-ima obezbijedi podršku za vizualno kreiranje dizajna baze podataka koji u konačnici rezultira MySQL bazom podataka. Modeli su daleko najbolji i najučinkovitiji način za razumijevanje i stvaranje valjane baze podataka sa dobrim karakteristikama i MySQL Workbench pomaže korisnicima ubrzati njihov rad kroz razne funkcije i alate.

MySQL Workbench omogućuje programeru ili DBA-u kreiranju jednog ili više modela u svom sučelju, a obezbjeđuje različite načine prikaza dizajniranih objekata (tablica, pogleda, pohranjene procedure, itd).



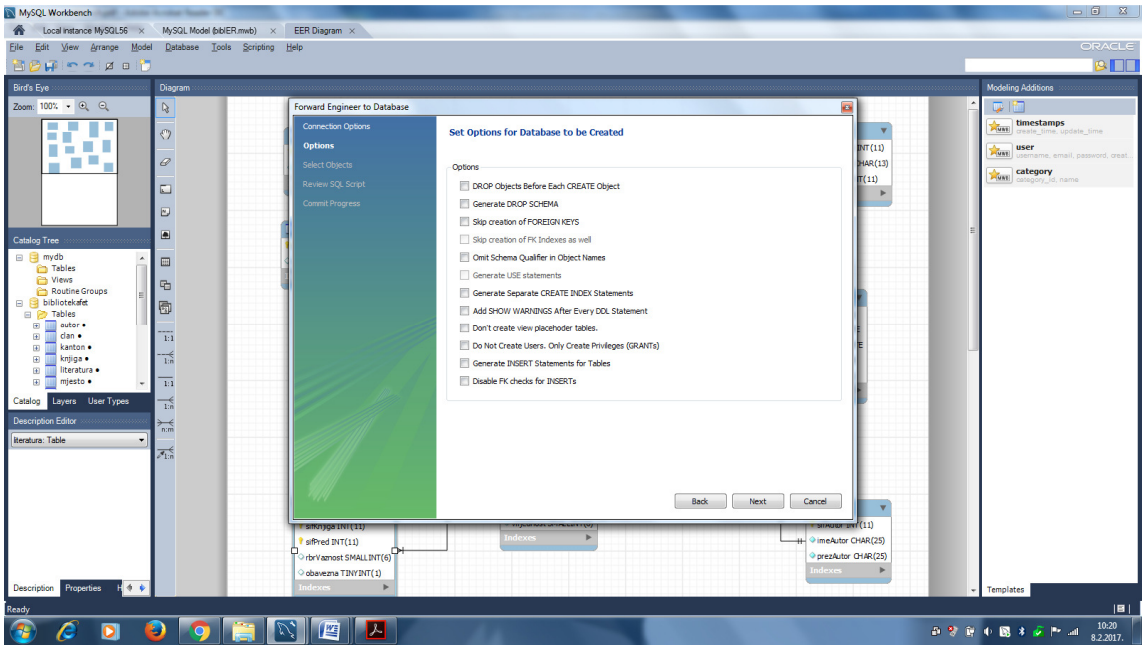
Mnoge različite mogućnosti i potpore u okviru MySQL Workbench pomažu brzom dizajniranju i implementiranju modela baza podataka. MySQL Workbench sadrži sve osnovne elemente modeliranja podataka koji se mogu očekivati u jednom alatu za dizajn baze podataka: vizualni dizajn svih objekata koji čine bazu podataka, uključujući tablice, veze između tablica (stranih ključeva), poglede, okidače, pohranjene procedure, funkcije, dozvole za objekte i još mnogo toga. Editori koji su jednostavni za korištenje omogućuju vrlo jednostavno dodavanje ili izmjenu elemenata poput kolona, indeksa, stranih ključeva, privilegija, itd. Za izmjene koje su napravljene u modelu a koje se trebaju poništiti, dostupna je Undo funkcija koja vraća stvari kako su izgledale prije nego su izmjene napravljene.

Osim toga, MySQL Workbench ima niz drugih pomagala koja omogućuju DBA-ima i programerima brzo kreiranje dizajna baze podataka po prvi put. Njegov alat Model Validation provjerava model podataka za eventualne pogreške i dojavljuje korisniku sve pronađene probleme. Za velike modele koje su teški za navigaciju, Zoom osobina omogućuje lako povećavanje i smanjivanje pogleda na model od ptičje perspektive na cijeli model baze podataka do fokusiranja na jednom određenom dijelu modela. Za pronalaženje raznih objekata (relacija, naziva atributa, itd.) na velikom modelu, napredna Find opcija pronalazi sva pojavljivanja po bilo kakvom kriteriju za pretraživanje da korisnik unese, s rezultatima prikazanim u obliku point-and-click navigacije do bilo čega što je izabrano u izlazu.

Konačno, u alatu postoji niz drugih korisnih funkcija poput dodatka za različite notacije modeliranja, Auto-layout funkcije koja automatski organizira tablice na dijagramu, a ima ugrađenu i sposobnost skriptiranja koja omogućuje naprednim korisnicima proširenje alata u Pythonu.

Direktni i obrnuti inženjering

Jedan od najčešće korištenih funkcija u dobrim alatima za modeliranje uključuje direktni inženjering dizajna baze podataka, što znači da alat generira sav SQL kod potreban za stvaranje fizičke baze podataka na ciljanom serveru. MySQL Workbench pruža upravo takvu mogućnost, tako da se vizualni model baze podataka izrađen u ovom alatu lako može brzo pretvoriti u fizičku bazu podataka na ciljanom MySQL Server-u. Ovakav način dizajna baze podataka osigurava da sav generisani SQL kod radi ispravno prvi put, što eliminira normalan proces ručnog pisanja složenog SQL koda koji je sklon greškama.



Osim toga, MySQL Workbench također omogućujei obrnuti inženjering, kako bi korisnici mogli razumjeti postojeće baze podataka bez obzira da li se radi o zasebnoj bazi podataka ili bazi podataka kao dijelu zapakirane aplikacije. MySQL Workbench podrška za obrnuti inženjering se spaja na bilo koji postojeći MySQL Server te gradi novi model baze podataka bilo iz cijelog ili dijela izvornih MySQL baza podataka. Nakon što se jednom smjeste u alat, svi objekti i njihove veze se mogu lako vidjeti, razumjeti, upravljati ili mijenjati.

Validacija scheme

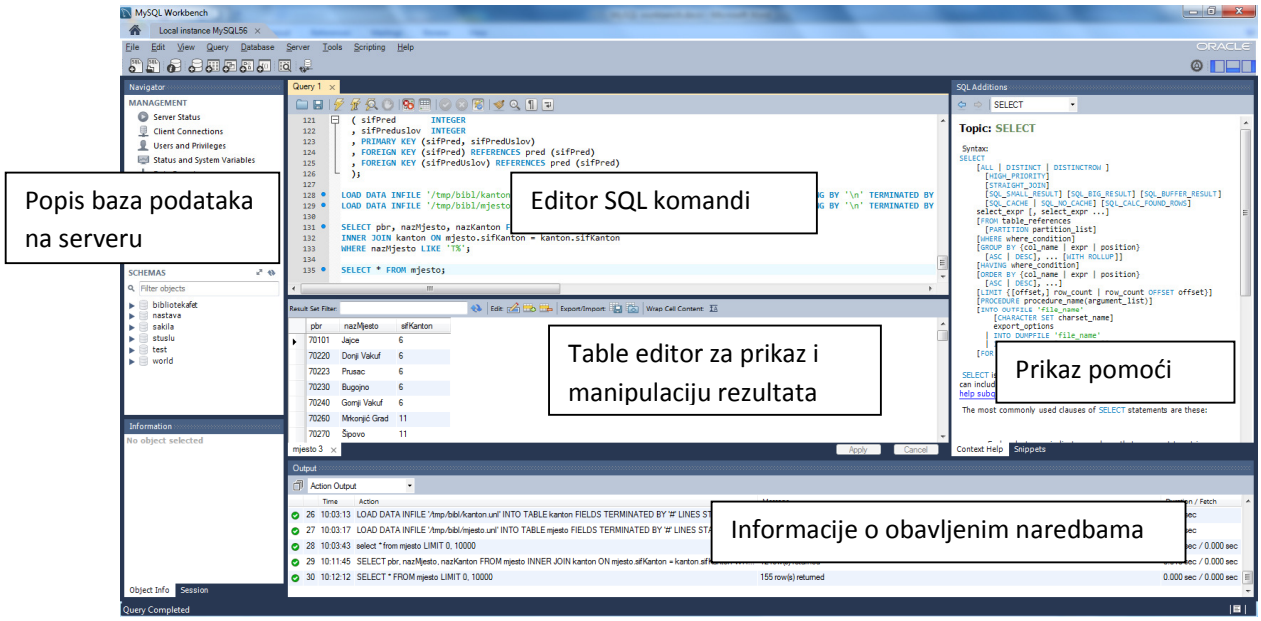
MySQL Workbench pruža napredne module provjere sheme koji omogućuju DBA-ima i programerima testiranje svojih dizajniranih modela prije implementacije. Moduli za potvrdu općih i MySQL specifičnih shema su dizajnirani na način da testiraju probleme oko praznog sadržaja, tablica kojima nedostaje primarni ključ, dupliciranje imena objekata ili atributa, stranih ključeva koji upućuju na neključne atribute i druge probleme koji bi mogli dovesti pada optimalne performanse, skalabilnosti ili pouzdanosti aplikacije.

SQL development

MySQL Workbench obezbjeđuje programerima kompletan skup vizualnih alata za izradu, uređivanje, formatiranje, debagiranje i upravljanje SQL upitima, konekcijama na baze podataka i objektima, kao i pojednostavljivanje upravljanja podacima u bazama podataka. Omogućena je pomoć ovisna o sadržaju (context help) koja se aktivira postavljanjem pokazivača iznad SQL iskaza o kojem su potrebni detaljnije informacije. Obezbjeđen je načina rada automatskim dovršavanjem iskaza (auto-complete) što pomaže učinkovitosti - kako smanjenjem količine otkucanog teksta tako i pronalaženjem i osiguravanjem ispravnih SQL naredbi i imena objekata.

Vizuelni SQL editor

Vizualni SQL Editor omogućuje programerima postavljanje, uređivanje i izvršavanje upita, te kreiranje i uređivanje podataka, kao i pregled i izvoz rezultata. Obojena sintaksa (isticanje ključnih riječi plavom bojom) pomaže u pisanju i ispravljanju pogrešaka u SQL naredbama. EXPLAIN planovi koji mogu pomoći u optimizaciji upita se također lako mogu izgraditi. U SQL editoru može se izvršiti više upita istovremeno i rezultati mogu se pogledati na pojedinačnim karticama (tabovima) u prozoru s rezultatima.



Neke od važnijih opcija SQL editora su:

 - izvršava označeni (selektovani) SQL kod ili sve naredbe u SQL editoru ukoliko ništa nije označeno

 - izvršava SQL iskaz nad kojim je postavljen kursor

 - izvršava EXPLAIN naredbu za SQL iskaz nad kojim je postavljen kursor

 - zaustavlja izvođenje upita koji se trenutno izvršava (korisno u slučaju predugačkih pogrešno postavljenih upita, te zastoja koji se mogu pojaviti kod paralelnog pristupa)

Table editor olakšava izmjenu podataka i potvrđivanje promjena korištenjem jednostavnog grid formata, a istovremeno obezbjeđuje programerima SQL DML naredbe koje se ustvari koriste da bi se obavile te promjene. Mogućnost izvoza rezultata dopušta programerima da rezultate sačuvaju u uobičajenim formatima (CSV, XML, JSON, i sl.). Neke od važnijih opcija Table editora su:

 - izmjena (edit) trenutnog zapisa

 - unos novog i izmjena označenih zapisa

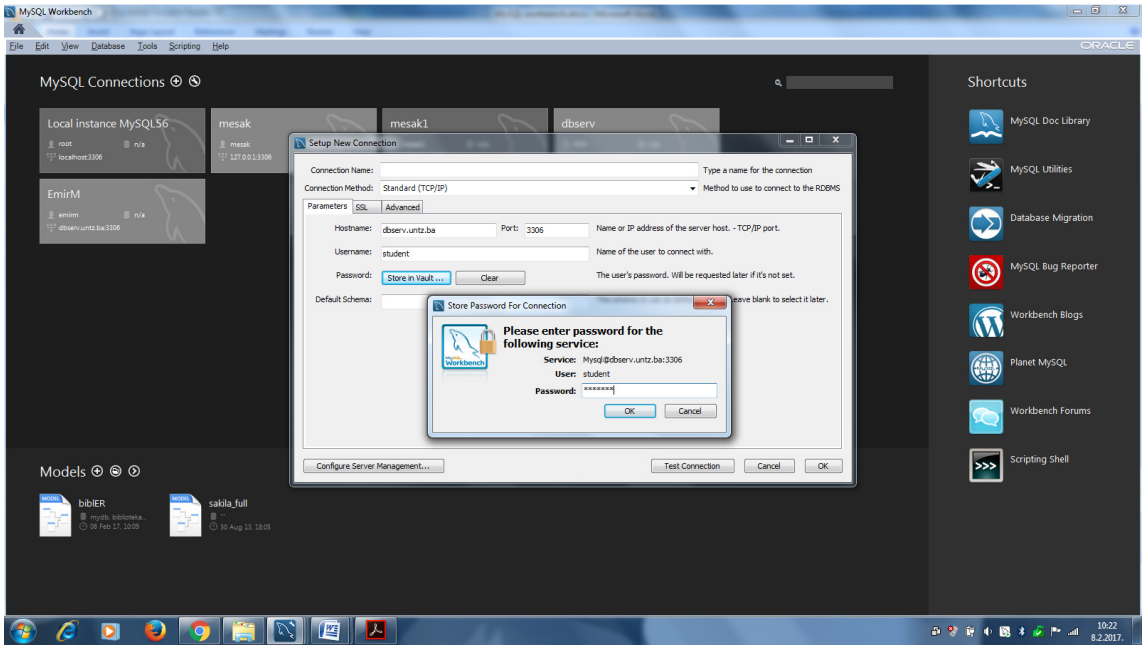
 - uvoz i izvoz (export/import) zapisa

History Panel osigurava prikaz kompletne istorije sesije upita i iskaza koja pokazuje koji upiti su postavljani i kada su pokrenuti. Na ovaj način programeri mogu lako dohvatiti, pregledati, ponovo izvršiti, dodati ili mijenjati prethodno izvršene SQL iskaze.

Osim toga, SQL Snippet panel omogućuje programerima spremanje i jednostavno ponovno korištenje Select, DML i DDL koda. Najčešći upiti mogu biti smješteni u isječcima (snippet), tako da se brzo mogu dohvatiti i ponovno upotrijebiti kasnije jednostavnim dvostrukim klikom na popisu isječaka.

Upravljanje konekcijama

Iz MySQL Workbench-a korisnici mogu lako vidjeti sve svoje baze podataka i konekcije. Za dodavanje i uređivanje konekcija, Database Connections Panel i Connections Wizard omogućavaju programerima stvaranje, organizaciju i upravljanje konekcijama baze podataka. Napredni korisnici mogu koristiti Manage Connections dijaloški okvir za unos parametara konekcije kao što su IP adresa, port, korisničko ime i lozinka. Upravitelj konekcijama korisnicima omogućuje testiranje novokreirane konekcije i podržava razne napredne opcije sigurnosti i daljinskog upravljanja, koje uključuju podršku za SSH tuneliranje - vrlo zgodan za korištenje MySQL Workbench-a za pristup i upravljanje MySQL bazama podataka koje se izvode na udaljenim podatkovnim centrima ili u cloud okruženju.


MySQL Workbench
File Edit View Database Tools Scripting Help
ORACLE
MySQL Connections
Local instance MySQL56
mesak
mesak1
dbserv
EmirM
Setup New Connection
Connection Name:
Type a name for the connection
Connection Method:
Standard (TCP/IP)
Method to use to connect to the RDBMS
Parameters
SSL
Advanced
Hostname:
dbserv.untz.ba
Port:
3306
Name or IP address of the server host. - TCP/IP port.
Username:
student
Name of the user to connect with.
Password:
Store in Vault ...
Clear
The user's password. Will be requested later if it's not set.
Default Schema:
Leave blank to select it later.
Store Password For Connection
Please enter password for the following service:
Service:
Mysql@dbserv.untz.ba:3306
User:
student
Password:
OK
Cancel
Configure Server Management...
Test Connection
Cancel
OK
Models
bibIER
sakila_full
Shortcuts
MySQL Doc Library
MySQL Utilities
Database Migration
MySQL Bug Reporter
Workbench Blogs
Planet MySQL
Workbench Forums
Scripting Shell
10:22
8.2.2017.