Slobodan Obradović

Siniša Ilić

SQL
Strukturirani upitni jezik u sistemima za upravljanje relacionim bazama podataka

Beograd, 2016.

Slobodan Obradović, Siniša Ilić
SQL – Strukturirani upitni jezik u sistemima za upravljanje relacionim bazama podataka

Recenzenti:
dr. Zoran Ćirović
dr. Branimir Trenkić

Izdavač: Visoka škola elektrotehnike i računarstva strukovnih studija, Beograd

Za izdavača: dr Vera Petrović, direktor

Korice: Gabrijela Dimić

ISBN 978-86-7982-246-8

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд

004.652.4:004.42(075.8)
004.655.3(075.8)

ОБРАДОВИЋ, Слободан, 1955-
SQL : strukturirani upitni jezik u savremenim sistemima za upravljanje relacionim bazama podataka / Slobodan Obradović, Siniša Ilić. - Beograd : Visoka škola elektrotehnike i računarstva strukovnih studija, 2016 (Niš : Sven). - 229 str. : ilustr. ; 24 cm

Tiraž 15. - Bibliografija: str. 227.

ISBN 978-86-7982-246-8

1. Илић, Синиша, 1968- [аутор]
a) Релационе базе података - Програмирање b) Програмски језик "SQL"
COBISS.SR-ID 222007052

SADRŽAJ

PREDGOVOR

Jedan od najvažnijih razloga velike upotrebe računara jeste potreba za obradom sve većeg broja podataka. Da bi obrada podataka bila što efikasnija podaci se organizuju i logički povezuju u baze podataka. Sistemi za upravljanje bazama podataka (SUBP) predstavljaju sredstvo za čuvanje i obradu podataka. Danas su još uvek dominantni u upotrebi relacioni sistemi, čiji osnovni element su relacije, odnosno tabele.

Za obradu podataka u relacionim bazama podataka koristi se SQL, strukturirani upitni jezik (Structured Query Language). SQL (izgovara se "es-ku-el") je implementiran u praktično sve komercijalne SUBP, i mada svaki proizvođač ima svoj dijalekt SQL-a rad sa svim verzijama je gotovo identičan. Udžbenik je podeljen u šest poglavlja.

SQL je uvodno poglavlje u kojem su date opšte definicije i podele naredbi prema nameni.

Naredbe za definisanje podataka predstavljaju poglavlje u kojem su date osnovne karakteristike i način primene naredbi kojima se kreiraju, menjaju i uklanjaju objekti i ograničenja baze podataka (tabele, atributi, veze, itd.). Ove naredbe po pravilu koriste projektanti i administratori baze podataka.

Naredbe za rukovanje podacima upotrebljavaju programeri, i ono što je najbitnije vrlo često i krajnji korisnici baze podataka. Ove naredbe su vrlo jednostavne, i korisnik u njima određuje šta želi da uradi a ne i kako se to ostvaruje. Ovaj rad najčešće je interaktivan, minimalno je proceduralan. U osnovi obrade, manipulacije podacima nalazi se upitni blok koji čine SELECT, FROM i WHERE naredbe. Ove naredbe omogućavaju izdvajanje skupa željenih podataka, koji se čuvaju i jednoj ili više tabela baze podataka. Rezultati upita mogu biti i izračunate veličine, a mogu se i grupisati po nekim osobinama ili urediti po nekom pravilu.

Ažuriranje baze podataka je neophodno jer podaci nastaju, menjaju se u vremenu i prestaju da budu od interesa. Dakle podaci se dodaju (INSERT), menjaju (UPDATE) i brišu (DELETE).

Kreiranje i korišćenje okidača pokazuje kako se neki poslovni procesi mogu automatski izvršavati bez intervencije korisnika, a aktiviraju ih događaji ažuriranja tabela. Dodavanjem, menjanjem ili brisanjem podataka menja se sadržaj tabela čime se aktiviraju posebni programi sastavljeni od SQL naredbi.

Zapamćene, uskladištene procedure predstavljaju imenovane blokove naredbi sa opcionim nizom ulaznih i izlaznih parametara. One se izvršavaju na serveru. Vrlo su korisne jer njima ne prethodi sintaksna i semantička analiza (kao što je to u slučaju prosleđivanja skupa SQL naredbi od strane klijenta), procedura je već kompajlirana na serveru zajedno sa planom izvršenja iste.

Funkcije definisane od strane korisnika su potprogrami koji na osnovu liste ulaznih parametara izračunavaju izlazne parametre (procedura za razliku od funkcije ne vraća ni jednu vrednost).

Kursori omogućavaju rad sa pojedinačnim redovima za razliku od tipičnih SQL naredbi koje rade sa skupovima slogova, redova. Kursor je u suštini pokazivač koji redom pokazuje na svaki slog (red) i na taj način omogućava pojedinačnu obradu svakog sloga.

Kreiranje i korišćenje pogleda je od posebnog interesa, jer pogledi olakšavaju rad krajnjih korisnika i programera, a ujedno su i moćno sredstvo za zaštitu podataka.

Upravljačke naredbe obezbeđuju sigurnost i integritet podataka. Baze podataka omogućavaju jednovremeni pristup podacima od strane većeg broja korisnika. Samim tim bezbednost podataka je ugrožena i moguće je narušavanje integriteta i konzistentnosti podataka. Ove naredbe uglavnom koriste administratori baze podataka.

Sve pokazane primere izvršili smo u četiri izabrana SUBP – MS Access, MS SQL Server, MySQL Server i Oracle DB Server. Jednostavniji primeri, zasnovani na ANSI SQL sintaksi, su isprobani u MS Access-u i rade u neizmenjenoj sintaksi i u ostala tri SUBP-a. Složeniji primeri koriste blokove sastavljene od SQL naredbi koji omogućavaju izvršenje raznih kontrola i obrada podataka i oni nisu izvršeni na SUBP MS Access 2010.

Ti blokovi mogu da se aktiviraju izvršavanjem bilo koje od naredbi ažuriranja podataka ili direktnim pokretanjem izvršenja bloka. To su: okidači, procedure, paketi i funkcije - objekti koji se najčešće upotrebljavaju u modernim bazama podataka. Verzije pomenutih baza nad kojima smo izveli primere su MS SQL 2012 Express Edition instaliran na OS MS Windows 7, MySQL Server 5.0.45 instaliran na OS CentOS linuxu ver 5.3 i Oracle Database 10g instaliran na Red Hat Enterprise Linux-u ver 4. Primeri su kreirani na odgovarajućim klijentskim aplikacijama za pomenute baze podataka i to:

- MS SQL Management Studio za MS SQL server,
- MySQL Workbench za MySQL server i
- Oracle SQL Developer za Oracle Database.

Pomenute tri verzije baza podataka i njihovi klijentski softveri su besplatni za preuzimanje na odgovarajućim web stranicama.

Primeri pokazani u ovoj knjizi su zasnovani na našem, za ovu priliku izmišljenom skupu podataka čije je uputstvo za generisanje dato u Prilogu 1.

Udžbenik je namenjen svima koji žele da koriste relacione baze za smeštanje i obradu podataka. U pedagoškom smislu, od posebnog značaja je izbor velikog broja primera koji su korišćeni kod svakog objašnjenja. Gotovo svi pojmovi i objašnjenja potkrepljena su primerima koji su metodički poređani, od vrlo jednostavnih do vrlo složenih. Za rešavanje i najsloženijih zadataka dovoljna su znanja stečena u prethodno obrađenoj materiji i primerima.

U Beogradu, mart 2016. Autori

SQL

Sastavni deo svakog sistema za upravljanje bazama podataka čine i specijalni jezici za opis i korišćenje baza podataka koji sadrže sledeće sastavne delove: *jezik za opis podataka* (**DDL – Data Description Language**), *jezik za opis fizičke strukture podataka* (**DMCL – Device Media Control Language**), *jezik za rad sa podacima* (**DML – Data Manipulation Language**) i *jezik upita* (**QL – Query Languages**). "Jedna od najvažnijih osobina baza podataka jeste da je opis podataka potpuno odvojen od programa za obradu podataka" [1].

Jezik za opis podataka može biti poseban neproceduralni jezik ili proširenje postojećeg programskog jezika (Cobol, C, Pascala ili C++). Njegova osnovna funkcija je specifikacija logičke strukture podataka time što se definišu: objekti i/ili tabele, logičke veze između objekata, atributi i dozvoljeni interval vrednosti za svaki atribut. Jezik za opis fizičke strukture podataka definiše: način memorisanja i dodele memorijskog prostora na disku, redosled i fizičku organizaciju podataka, dodelu privremene memorije i načine adresiranja i pretraživanja podataka. Jezik za rad sa podacima obezbeđuje vezu između podataka i aplikacije (programa za rad sa podacima: unos, obradu, prikaz i sl.), a njegove osnovne funkcije su: otvaranje i zatvaranje datoteka (naredbe **OPEN** i **CLOSE**), pronalaženje željenog sloga (**FIND**), izmena sadržaja nekog polja (**MODIFY**), dodavanje sloga (**INSERT**), brisanje sloga (**DELETE**, **REMOVE**) itd.

Jezici upita omogućavaju realizaciju proizvoljnih funkcija – upita nad relacijama. U zavisnosti od toga da li se zasnivaju na relacionoj algebri ili na predikatskom (relacionom) računu, postoje dve klase ovih jezika. U relacionoj algebri definisane su operacije pomoću kojih je moguće dobiti željenu relaciju (tabelu) iz skupa datih relacija (tabela). Relacionim računom definišu se osobine relacija koje se žele dobiti. U relacionoj algebri definisane su sledeće operacije: unija, diferencija, presek, Kartezijev (Dekartov) proizvod, projekcija, selekcija (restrikcija), spajanje (join), deljenje. Postoje još neke izvedene operacije uslovljene **NULL** vrednostima. Relacioni račun je neproceduralni jezik, te programer umesto da definiše proceduru pomoću koje će se dobiti željeni rezultat, samo specificira željeni rezultat. Postoje dva oblika relacionog računa: relacioni račun n-torki i relacioni račun domena.

"Primer relacionog računa n-torki:

x ∈ RADNIK
x.SIFRA, x.IME GDE_JE x.PLATA > *2 000*
AND x.KVALIF = *'VKV'*

Primer relacionog računa domena:

x,y GDE_JE ∃ z > *2 000* **AND**
RADNIK (SIFRA: x, IME: y, PLATA: z, KVALIF = *'VKV'*) ".[2]

Zajednička karakteristika ovim računima jeste ta da su njihove konstrukcije mnogo sličnije prirodnim jezicima nego što je to slučaj sa jezicima treće generacije. Najpoznatiji primeri ovih jezika su SQL, Quel i QBE [25], [27]. "Oni se zasnivaju na različitim principima: **QBE** se zasniva na relacionom računu domena, **Quel** se zasniva na relacionom računu (ne

[1] [16] Дончев, А., Обрадович, С.: "База *от данни",* Технически университет – Габрово, ISBN – 954-683-223-5, Габрово, 2004.

[2] [27] Slobodan Obradović "Poslovno informatička obuka", *Nacionalna služba za zapošljavanje, ISBN 86-902585-8-2*, Beograd, 2004.

podržava relacionu algebru), a **SQL** se zasniva na relacionom računu n-torki tj. na kombinaciji relacione algebre i relacionog računa" [3].

Navedena tri jezika nisu značajna samo u oblasti istraživanja baza podataka, već imaju značajne komercijalne implementacije. Iako smo ih označili kao jezike upita, to je u stvari netačno, jer ovi jezici imaju ugrađene i sve druge mogućnosti: definisanje strukture podataka, modifikacija podataka u bazi i mogućnosti za definisanje bezbednosnih ograničenja. Iako postoje komercijalni proizvodi zasnovani na sva tri jezika (Ingres je zasnivan na Quel-u), detaljnije ćemo obraditi SQL jer je on našao najširu primenu, a u narednom poglavlju biće opisane mogućnosti QBE jer je njegova primena vrlo prosta, a on je implementiran u sve moderne RDBMS i alate za rad sa bazama podataka.

SQL je programski paket koji se zasniva na relacionoj algebri i relacionom računu. Skraćenica SQL dolazi iz engleskog **Structured Query Language,** što bi u slobodnom prevodu značilo **strukturni jezik za postavljanje upita** [13], [16], [25], [27], [30]. Teorijske osnove, kao i koncept prvog SQL programskog paketa, dao je E.F. Codd utvrdivši u svom radu:

"*...**da koncept relacionog modela dozvoljava razvoj univerzalnog jezika za manipulisanje podacima baziranog na primeni relacione algebre**"*.

Razvoj SQL programskog paketa tekao je u skladu sa tempom kojim se danas prihvataju nove ideje u tehnici, a posebno u računarskoj tehnici. Godine 1974, dakle četiri godine posle definisanja teorijskih principa relacionih baza podataka, nastaje programski paket nazvan **SEQUEL,** da bi ubrzo došlo do njegove modifikacije i pojave paketa **SQUARE**, a potom, kako to onda obično biva, dolazi **SEQUEL2**. IBM razvija sopstveni paket pod imenom **SISTEM-R**. Neka vrsta sinteze iskustava navedenih programskih paketa ostvarena je u verziji koja se danas nalazi pod imenom **MySQL**.

Osnovna ili standardna verzija SQL programskog paketa je verzija predviđena za manipulisanje podacima u relacionim bazama koja je implementirana danas praktično u svim programskim paketima za obradu podataka kao njihov sastavni deo, a u svojoj sintaksi ima za sve osnovne relacione operatore (spajanje, razliku, množenje, restrikciju i projekciju) ekvivalentne SQL naredbe. Tri preostala relaciona operatora (deljenje, presek i unija) ne smatraju se osnovnim, jer se te operacije mogu izvesti kombinovanjem pomenutih, osnovnih i, za razliku od većine verzija SQL-a, upitni jezik Quel ih ne podržava. SQL je standardizovan od strane kompetentnih međunarodnih institucija, kao što su **ISO** (*International Standardization Organization*) i **ANSI** (*American National Standards Institute*). U ovoj knjizi poštovana je sintaksa propisana po **ANSI** standardu. Jezik SQL je skup naredbi pomoću kojih korisnik određuje šta želi da uradi i ne razmišlja o tome kako će to biti ostvareno. Pri izvršavanju naredbi najbitnija je uloga optimizatora naredbi. Pre izvršenja svake naredbe on određuje optimalan način izvršavanja sa stanovišta brzine i potrebnih resursa. Jednom naredbom moguće je obraditi grupu slogova, a optimalan pristup grupi bolji je od grupe optimalnih pristupa pojedinačnim slogovima.

Relacioni jezik koji komunicira sa bazom i podacima zasnovan je na relacionoj algebri i na predikatskom računu prvog reda. Zahvaljujući tome, korisnik postavlja logičke uslove na osnovu kojih se izdvajaju podaci. Sistem za upravljanje bazom podataka izdvaja sve slogove koji zadovoljavaju postavljene uslove, a da mu pri tome korisnik nije saopštio kako da ih izdvoji i gde se ti podaci fizički nalaze.

Osnovni nedostatak SQL paketa ogleda se u činjenici da sve njegove mogućnosti i opcije ne mogu savladati korisnici - laici u smislu projektovanja, eksploatacije, obrade i neposrednog korišćenja informacionih sistema. SQL**,** naime, zahteva od korisnika poznavanje

[3] [25] Obradović, S., Pandurov, T., Vučinić, B.: "SQL *Strukturirani upitni jezik",* Visoka škola elektrotehnike i računarstva, ISBN – 978-86-85081-17-3, Beograd, 2008.

konfiguracije informacionog sistema (logički model baze, odnosno spisak relacija, veza među njima, kao i raspored atributa u njima), te poznavanje osnova tehnike programiranja.

U SQL naredbama mogu se na proizvoljnom mestu pojaviti beli znakovi: razmak (Space), tabulator (Tab) ili nova linija. Sam tekst naredbe, odnosno upita, biće mnogo čitljiviji ako se svaka odredba tipa **SELECT**, **FROM**, **WHERE**, **GROUP BY**, **HAVING**, **ORDER BY**, i sl. navede u posebnom redu. Takođe je korisno u naredbama pisati komentare. Komentari se mogu pojaviti na svakom mestu gde se mogu pojaviti i beline. SQL spada u tzv. ***case-insensitive*** programske pakete, što znači da se naredbe, ključne reči, imena objekata i promenljivih mogu pisati i malim i velikim slovima, a da sistem pri tome ne pravi razliku među njima. Međutim, i pored toga, zbog preglednosti napisanog programa, preporučljivo je dosledno koristiti mala i velika slova. Mi ćemo se, stoga, ubuduće pridržavati sledećih oznaka:

- velika slova - ključne reči, funkcije i imena relacija,
- mala slova - imena atributa itd.,
- kosa (italic) mala slova - vrednosti atributa (podaci),
- u uglastim zagradama - neobavezne opcije,
- komentari - dva znaka minus (- -).

"Skoro svaki ozbiljniji informacioni sistem danas se zasniva na relacionoj bazi podataka, bilo da se radi o Oracle-ovom serveru ili Access-ovoj datoteci. Ubedljivo najpopularniji sistem za upravljanje bazama podataka pod Linuxom je MySQL, koji razvija istoimena švedska kompanija."[4]. "Na tržištu postoji veliki broj SUBP (Sistem za upravljanje bazama podataka, DBMS - Database Management System) [24], [26]. Skoro svi komercijalni proizvodi imaju open source alternative (MySQL, Firebird, PostgreSQL, ...)"[5].

Pri imenovanju objekata treba imati u vidu da svi SUBP postavljaju određena ograničenja u pogledu dužine imena za objekte (najčešće ne mogu biti duža od 30 karaktera), nazive samih baza (često ne mogu biti duži od 8 karaktera) i veza ka bazama (data base link, dužine do 128 karaktera). Nazivi moraju, po pravilu, početi slovom, a mogu da sadrže slova, cifre, specijalne znakove, ali ne i apostrofe i navodnike. U linkovima se mogu koristiti i znakovi . i @. Nazivi objekata ne mogu biti rezervisane reči SUBP-a, niti drugih proizvoda koji su integrisani u njega (na primer nekog programskog jezika). Pod navodnicima se mogu koristiti svi znakovi, osim samih navodnika i nove linije. Sam SUBP, po pravilu, prevodi sve nazive objekata u velika slova. U nekim SUBP, na primer u MS SQL Serveru, sam korisnik pri instalaciji može da izabere opciju da li da se pravi razlika između velikih i malih slova u imenima objekata. Dužina naziva objekata u SQL Serveru do verzije 7.0 bila je ograničena na 20 karaktera, a za kasnije verzije na 128 karaktera (116 za privremene objekte). SUBP MySQL koji je instaliran na MS Windows platformi ne pravi razliku između velikih i malih slova u nazivima objekata, dok isti instaliran na Linux platformi pravi tu razliku. Predlog autora je da nazive objekata (relacija, atributa, ograničenja itd) treba uraditi malim slovima i stalno se referencirati istim pri pozivu tih objekata. Tada ni jedan sistem neće prijaviti grešku da taj objekat ne postoji u bazi podataka.

Svaki tip objekta ima pripadnu oblast imenovanja. Nazivi različitih objekata u jednoj oblasti imenovanja moraju biti različiti. U jednoj šemi podataka ne mogu postojati dve tabele, pogleda, zapamćene procedure, sekvence ili snapshot sa istim imenom. Nazivi kolona u istim

[4] [24] Svetlana Andjelic, Slobodan Obradovic, Branislav Gacesa, "AN PERFORMANCE ANALYSIS OF THE DBMS - MySQL vs PostgreeSQL" -mr Communications-Scientific Letters of the University Zilina, Slovakia, pages 53-59, ISSN 1335-42-05 Art No 4/2008 Networks for new generations.

[5] [26] S. Obradović, S. Slavnić, A. Donchev, Comparison of the free DBMS alternatives for MS SQL, International scientific conference UNITECH'08, TU-Gabrovo, 21-22 November 2008, Gabrovo, proceedings, vol.1, pp.I 455-I 460, ISSN:1313-230X.

tabelama i pogledima moraju biti različiti, ali se mogu ponavljati u različitim tabelama i pogledima.

Programski paket SQL omogućava jednostavno :

- kreiranje relacija,
- unos podataka,
- brisanje,
- ažuriranje,
- pretraživanje podataka, te
- prikazivanje novih informacija.

Osnovne naredbe SQL-a koje se koriste u manipulisanju podacima u relacionoj bazi podataka omogućavaju definisanje, obradu i zaštitu podataka. Sve SQL naredbe, po pravilu, se završavaju interpunkcijskim znakom (**;**) i mogu se podeliti u tri grupe.

a) ***Naredbe za definisanje podataka*** (Data Definition Statements, DDL) omogućavaju definisanje resursa i logičkog modela relacione baze podataka [25], [27]:
   - **CREATE TABLE**– kreiranje fizičke tabele baze podataka,
   - **CREATE VIEW** – kreiranje virtuelne imenovane tabele, "pogled",
   - **CREATE INDEX** – kreiranje indeksa nad jednom ili više kolona tabele ili pogleda,
   - **ALTER TABLE** – izmena definicije tabele, izmena, dodavanje ili uklanjanje kolone (atributa),
   - **DROP TABLE** – uklanjanje tabele iz baze podataka,
   - **DROP VIEW** – uklanjanje pogleda iz baze podataka.

b) ***Naredbe za rukovanje podacima*** (Data Manipulation Statements) omogućavaju ažuriranje podataka u širem smislu (izmenu, dodavanje i brisanje) i izveštavanje (pribavljanje postojećih i izračunavanje novih informacija) iz baze podataka [25], [27]:
   - **SELECT** – pristup podacima i prikaz sadržaja baze podataka,
   - **INSERT** – unošenje podataka, dodavanje redova u tabelu,
   - **DELETE** – brisanje podataka, izbacivanje redova iz tabele**,**
   - **UPDATE** – ažuriranje, izmena vrednosti podataka u koloni.

c) ***Naredbe za upravljanje bezbednošću podataka*** (Data Control Functions) omogućavaju oporavak, konkurentnost, sigurnost i integritet relacione baze podataka [25], [27]:
   - **GRANT** – dodela prava korišćenja tabele drugim korisnicima od strane vlasnika tabele,
   - **REVOKE** – oduzimanje prava korišćenja tabele drugim korisnicima,
   - **BEGIN TRANSACTION** – početak transakcije koji se može završiti jednom od dveju sledećih naredbi,
   - **COMMIT WORK** – prenos dejstva transakcije na bazu podataka,
   - **ROLLBACK WORK** – poništavanje dejstva transakcije na bazu podataka.

Bez poznavanja navedenih komandi za projektovanje, definisanje podataka i rukovanje podacima ne može se ozbiljno računati na održavanje i izradu elementarnih aplikacija za obradu podataka uz korišćenje moderne relacione tehnologije. Štaviše, za najveći broj standardnih zahteva dovoljno je poznavanje samo ovih nekoliko naredbi, pa da se postigne željeni cilj.

Sa druge strane, ove osnovne naredbe predstavljaju uvod u šira znanja za one koji nameravaju da se profesionalno bave projektovanjem i održavanjem informacionih sistema, a kompletan spisak SQL naredbi može se naći u svakom priručniku SQL-a.

Naredbe za definisanje podataka

Naredbe za definisanje podataka služe za definisanje strukture objekata, izmenu i brisanje definicije [13], [25], [27], [30]. Pomoću ovih naredbi opisuje se šema podataka. Kod nekih SUBP u ovu grupu naredbi se svrstavaju i naredbe kojima se definišu korisnici i njihova prava. Svako izvršenje DDL naredbe implicitno potvrđuje tekuću transakciju, kao i samu komandu nakon njenog izvršenja. Osnovna karakteristika tih naredbi jeste da se njihovo izvršenje ne može otkazati komandom **ROLLBACK**. One, dakle, implicitno potvrđuju tekuću transakciju, kao i samu komandu posle njenog izvršenja. U ovu grupu naredbi spadaju i naredbe definisanja prava pristupa, uloga i korisnika, ali mi ćemo ove naredbe posmatrati sa stanovišta njihove funkcije koja se odnosi na zaštiti i bezbednost podataka. Ove naredbe omogućavaju:

- kreiranje, izmenu i brisanje objekata baze, samu bazu i njene korisnike (**CREATE**, **ALTER** i **DROP**),
- preimenovanje objekata baze (**RENAME**),
- brzo brisanje podataka u tabelama bez brisanja definicije objekata,
- praćenje korisnika sistema i prikupljanje statistika o objektima i
- dodavanje komentara za objekte u rečnik podataka.

Nova relacija (tabela) kreira se pomoću programskog paketa SQL naredbom **CREATE TABLE**. Opšti oblik ove naredbe glasi:

CREATE TABLE ime_relacije, **lista imena i tipova atributa**

uz definisanje eventualnih ograničenja njihovih vrednosti.

Ime relacije (ili **ime_tabele**) ne sme biti nijedna od ključnih reči programskog paketa SQL i mora počinjati slovom engleskog alfabeta. Od specijalnih znakova može sadržati samo znak **_**. Dužina imena nije precizirana.

Ime atributa bira se na isti način kao i ime relacije. Broj atributa jedne relacije ograničen je mogućnostima računara, a zavisi i od implementacije. U praksi su relacije sa više od 30 atributa retkost. Obično je taj broj, u dobro projektovanim sistemima, od 10 do 30.

U jednoj bazi podataka, kao što smo videli, može postojati više atributa sa istim imenom, ali oni ne smeju pri tom biti u istoj relaciji (tabeli). Preporučljivo je ovu opciju (ista imena atributa u različitim tabelama, a da pri tom označavaju različite stvari) radi preglednosti i eliminacije grešaka, izbegavati kad god je to moguće.

Tip atributa služi da se pobliže opiše podatak i na taj način odredi optimalan način njegovog memorisanja - sa jedne strane, a definicijom **ograničenja vrednosti** s druge strane, sprečavaju se moguće greške prilikom unošenja podatka.

Tipovi podataka

Programski paketi za rad sa bazama podataka omogućavaju definisanje različitih tipova podataka [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [15], ali svi koriste nekoliko osnovnih tipova.

- **Slovni** ili **znakovni tip** podatka koristi se kada podatak predstavlja niz alfanumeričkih znakova. Takav podatak je, na primer, ime i prezime, adresa ili zanimanje radnika, broj telefona itd. Slovni ili **string** podatak može biti svaki niz alfanumeričkih i nekih od speci-

jalnih znakova. Broj znakova može biti ograničen na *n* (tip podatka je u tom slučaju **CHAR(n)**) ili je promenljive dužine (**VARCHAR**).

- **Numerički (NUMERIC) tip** podatka (na primer visina plate ili dužina neke ulice) može, slično kao i u drugim programskim jezicima, da bude celobrojan (**SMALLINT,** ako je dužina 16 bita, **INTEGER**, ako je dužina 32 bita i **QUADWORD,** sa dužinom od 64 bita), ali i realan broj sa fiksnom ili pomičnom decimalnom tačkom razne preciznosti koja zavisi od dužine, to jest broja cifara. Tako je tip podataka **FLOAT(n)** realan broj do 6 cifara, odnosno **DOUBLE PRECISION** dužine od 7 do 15 cifara.
- **Datumski (DATE)** i **vremenski (TIME) tip** podatka često se koristi u informacionim sistemima jer je čitav niz podataka vezan za vreme, za neke rokove, bez obzira da li su iskazani danima, mesecima i godinama ili satima, minutima i sekundama.
  Datumski tip **DATE** je jedna celina, jedan podatak, iako u sebi sadrži tri numerička polja (za *dan, mesec* i *godinu*), međusobno odvojena tačkom, crtom ili kosom crtom, a što zavisi od zemlje u kojoj će se koristiti. Za ovakav tip podatka važi i posebna aritmetika, koja omogućava korisniku da računa vremenske intervale.
  Za vremenski tip podatka **TIME** važe ista pravila kao i za datumski, s tim što se koriste različiti postupci za obradu zasnovani na različitim aritmetikama: jedna aritmetika za datumski tip, a druga za vremenski, jer godina ima 12 meseci, a mesec 28 (29), 30 ili 31 dan, dok je za sat, minut i sekundu taj odnos 24:60:60, pa se i odgovarajuće aritmetike, shodno tome, moraju razlikovati.
- **Logički tip** podatka (**LOGICAL**) koristi se kod atributa kod kojih je domen ograničen na dve vrednosti.

  *Na primer, takav atribut je **prisutnost,** kod koga mogu postojati samo vrednosti **prisutan** ili **nije_prisutan,** ili atribut **pol,** kod koga mogu postojati vrednosti **muški** i **ženski**.*
- **Memo tip** podatka je, u principu, slovni (alfanumerički) tip, ali sa većom dužinom (na primer do 64 kilobajta) a koristi se za unošenje opisnih podataka (dijagnoza ili anamneza pacijenta, i dr.).
- **Bit-mapa** podatak nosi neku grafičku informaciju, dijagram ili sliku.

U tehnici definisanja podataka postoji i slučaj **nepostojećeg podatka** kada vrednost podatka (bilo kog tipa) nije poznata ili nije nastupio momenat njegovog prisustva u bazi, a koji u drugim programskim jezicima nije poznat, takozvana ***NULL*** vrednost. Nepostojeći podatak se koristi i u slučajevima kada neki entiteti nemaju neko svojstvo, a neki drugi entiteti koji pripadaju istom prostoru objekata imaju. Na primer, neki lekari su specijalisti, a neki nisu.

Dakle, atribut *specijalnost* u prostoru objekata LEKAR neki pojedinačni primerci ne mogu imati jer nisu specijalisti. Na ovaj način se omogućava da u istoj tabeli budu zapisani podaci o svim lekarima (sa i bez specijalizacije). Time se dobija manji broj objekata, odnosno relacija (tabela) u bazi podataka. SQL omgućuje svakom podatku (sem primarnog ključa) da ima i nepostojeću, **NULL** vrednost.

Svaki sistem za upravljanje bazama podataka (SUBP) ima svoj skup ugrađenih tipova podataka, a većina omogućava i samom korisniku da definiše svoje tipove podataka.

a) Ugrađeni tipovi podataka u SQL serveru

TIP PODATKA	OPIS
bigint	Integer od -2^63 do 2^63-1
int	Integer od -2^31 do 2^31 - 1

smallint	Integer od -2^15 do 2^15 - 1
tinyint	Integer od 0 do 255
bit	Integer sa vrednošću 0 ili 1
decimal	Numerički podataka sa fiksnom preciznošću od 10^38 +1 do 10^38 -1
numeric	Numerički podataka sa fiksnom preciznošću od 10^38 +1 do 10^38 -1
money	Novčane vrednosti od -2^63 do 2^63 - 1
smallmoney	Novčane vrednosti od -214,748.3648 do +214,748.3647
float	Broj sa pokretnom preciznošću od -1.79E + 308 do 1.79E + 308
real	Broj sa pokretnom preciznošću od -3.40E + 38 do 3.40E + 38
datetime	Datum i vreme od 1. januara 1753. do 31. decembra 9999. sa korakom od 3.33 ms
smalldatetime	Datum i vreme od 1. januara 1900. do 6. juna 2079. sa korakom od 1 min
char	Karakteri fiksne dužine sa maksimumom od 8 000 karaktera
varchar	Karakteri promenljive dužine sa maksimumom od 8 000 karaktera
text	Karakteri promenljive dužine sa maksimumom od 2^31 - 1 karaktera
nchar	Unikod podaci fiksne dužine sa maksimalnom dužinom od 4 000 karaktera
nvarchar	Unikod podaci promenljive dužine sa maksimalnom dužinom od 4 000 karaktera
ntext	Unikod podaci promenljive dužine sa maksimalnom dužinom od 2^30 – 1 karaktera
binary	Binarni podaci fiksne dužine sa maksimalnom dužinom od 8 000 bajta
varbinary	Binarni podaci promenljive dužine sa maksimalnom dužinom od 8000 bajta
image	Binarni podaci promenljive dužine sa maksimalnom dužinom od 2^31 – 1 bajta
cursor	Referenca na kursor
sql_variant	Tip podatka koji čuva vrednosti raznih tipova podataka, osim text, ntext, timestamp i sql_variant
table	Specijalan tip podatka koji čuva result set za kasnije procesiranje
timestamp	Unikatni broj koji se menja svaki put kada se radi promena nad vrstom.
uniqueidentifier	Globalni identifikator koji nikada ne može da se ponovi

Korisnički definisani tipovi podataka

SQL server takođe podržava korisnički definisane tipove podataka. Oni obezbeđuju mehanizam za definisanje imena tipa podataka koji je restriktivniji od ugrađenih tipova podataka. Baziraju se na ugrađenim tipovima podataka.

b) tipovi podataka u MY SQL

Tekstualni tipovi podataka

TIP PODATKA	OPIS
CHAR( )	Sekcija fiksne dužine od 0 do 255 karaktera
VARCHAR( )	Sekcija promenljive dužine od 0 do 255 karaktera
TINYTEXT	String maksimalne dužine do 255 karaktera
TEXT	String maksimalne dužine do 65 535 karaktera
BLOB	String maksimalne dužine do 65 535 karaktera
MEDIUMTEXT	String maksimalne dužine do 16 777 215 karaktera
MEDIUMBLOB	String maksimalne dužine do 16 777 215 karaktera
LONGTEXT	String maksimalne dužine do 4 294 967 295 karaktera
LONGBLOB	String maksimalne dužine do 4 294 967 295 karaktera

Zagrade omogućavaju da se unese maksimalan broj karaktera koji će biti upotrebljen u koloni, na primer **VARCHAR**(20)

CHAR i **VARCHAR** su tipovi podataka koji se najčešće upotrebljavaju. **CHAR** je string fiksne dužine i najčešće se koristi kod podataka koji ne variraju u dužini, dok se **VARCHAR** koristi kod podataka koji mogu varirati u dužini. **CHAR** tip podatka je brži za procesiranje zato što su svi podaci u koloni iste dužine. **VARCHAR** je malo sporiji zbog proračunavanja dužine karaktera u koloni, ali čuva memorijski prostor. Koji ćemo tip izabrati, zavisi od konkretne situacije.

BLOB predstavlja Binary Large Object (veliki binarni objekat). I **TEXT** i **BLOB** su varijabilne dužine i služe da čuvaju velike količine podataka. Slični su većoj verziji **VARCHAR** tipa. Ovi tipovi podataka služe da sačuvaju velike količine informacija, ali se procesiraju mnogo sporije.

Numerički tipovi podataka

TIP PODATKA	OPIS
TINYINT( )	-128 do 127 **označeni** 0 to 255 **neoznačeni**
SMALLINT( )	-32 768 do 32 767 **označeni** 0 do 65 535 **neoznačeni**
MEDIUMINT( )	-8 388 608 do 8 388 607 označeni 0 do 16 777 215 neoznačeni
INT( )	-2 147 483 648 do 2 147 483 647 označeni 0 do 4 294 967 295 neoznačeni
BIGINT( )	-9 223 372 036 854 775 808 do 9 223 372 036 854 775 807 označeni 0 do 18 446 744 073 709 551 615 neoznačeni
FLOAT	Mali broj u pokretnom zarezu
DOUBLE(,)	Veliki broj u pokretnom zarezu
DECIMAL(,)	Broj tipa DOUBLE sačuvan kao string, koji dozvoljava fiksan decimalni zarez

Integer tipovi podataka imaju dodatnu opciju UNSIGNED (NEOZNAČEN). Normalno, integer ide od negativne do pozitivne vrednosti. Upotreba komande UNSIGNED pomera rang integer broja, tako da počinje od nule umesto od negativnog broja.

Datumski tipovi podataka - Date types

TIP PODATKA	OPIS
DATE	Datum u formatu YYYY-MM-DD.
DATETIME	Datum i vreme u formatu YYYY-MM-DD HH:MM:SS.
TIMESTAMP	Datum i vreme u formatu YYYYMMDDHHMMSS
TIME	Vreme u formatu HH:MM:SS.

Ostali tipovi podataka

TIP PODATKA	OPIS
ENUM ( )	Skraćenica za "ENUMERACIJU", što znači da svaka kolona može da ima jednu od navedenih mogućih vrednosti.
SET	Slično tipu "ENUM", osim što svaka kolona može više od jedne vrednosti sa liste mogućih vrednosti.

ENUM predstavlja skraćenicu za listu ENUMERACIJE, odnosno to je nabrojivi tip podatka. Atribut ovog tipa može sadržati samo jednu od vrednosti koje su definisane u okviru liste između () zagrada. Na primer, lista sa dve moguće vrednosti: **ENUM** ('y','n'). Može se dodati maksimalno 65 535 vrednosti u enumeracionu listu. Ako vrednost koja je uneta nije u listi, biće uneta blanko vrednost.

SET je slična vrednosti **ENUM**, osim što **SET** može sadržati 64 vrednosti i može uneti više od jedne vrednosti sa liste.

Vrednosti podataka jednog tipa mogu se eksplicitno konvertovati u vrednosti drugog tipa. Ali većina **SUBP** vrši i implicitnu konverziju jednog tipa u drugi, mada se uvek preporučuje da izvrši eksplicitnu konverziju. Najčešće se implicitna konverzija vrši pri unosu (**INSERT**) i promeni (**UPDATE**). Tada se podaci koji se unose konvertuju u tip koji je definisan za kolonu (tip kolona). Implicitna konverzija se vrši i pri poređenju podataka različitog tipa i tada se, na primer, pri poređenju niza znakova i broja, niz znakova konvertuje u broj.

c) tipovi podataka u Oracle

Tekstualni tipovi podataka

TIP PODATKA	OPIS
CHAR( )	Sekcija fiksne dužine od 0 do 2000 bajta
VARCHAR2( )	Sekcija promenljive dužine od 0 do 4000 bajta
NCHAR()	Unicode string maksimalne dužine do 2000 bajta
NVARCHAR2()	Unicode string maksimalne dužine do 4000 bajta
CLOB	String maksimalne dužine do 8 terabajta
NCLOB	Unicode string maksimalne dužine do 8 terabajta
LONG	String maksimalne dužine do 2 gigabajta

Zagrade omogućavaju da se unese maksimalan broj karaktera koji će biti upotrebljen u koloni, na primer **VARCHAR2**(20)

CHAR i **VARCHAR2** su tipovi podataka koji se najčešće upotrebljavaju. **CHAR** je string fiksne dužine i najčešće se koristi kod podataka koji ne variraju u dužini, dok se **VARCHAR2** koristi kod podataka koji mogu varirati u dužini. **CHAR** tip podatka je brži za procesiranje zato što su svi podaci u koloni iste dužine. **VARCHAR2** je malo sporiji zbog proračunavanja dužine karaktera u koloni, ali čuva memorijski prostor. **NCHAR** i **NVARCHAR2** su Unicode ekvivalenti tipova **CHAR** i **VARCHAR2** koji čuvaju Unicode karaktere. Definisani skup karaktera (character set) za **NCHAR** i **NVARCHAR2** tipove podataka mogu biti ili AL16UTF16 ili UTF8 i oni se definišu prilikom kreiranja baze izborom odgovarajućeg nacionalnog skupa karaktera (national character set). AL16UTF16 i UTF8 su Unicode standardi za predstavljanje karaktera.

CLOB i **NCLOB** predstavljaju **C**haracter **L**arge **O**bject (veliki objekat karaktera) i National Character Large Object. I **CLOB** i **NCLOB** su varijabilne dužine i služe da čuvaju velike količine podataka. Slični su većoj verziji **VARCHAR2** tipa. Ovi tipovi podataka služe da sačuvaju velike količine informacija, ali se procesiraju mnogo sporije. Kolone sa tipom podataka **LONG** se koriste da u rečniku podataka u Oracle-u čuvaju tekst definicija pogleda (view).

Numerički tipovi podataka

TIP PODATKA	OPIS
PLS_INTEGER	Označena celobrojna vrednost u opsegu -2147483647 do 2147483647
NUMBER (precision, scale)	Od 1 x 10^{-130} to 9.99...9 x 10^{125} **pozitivni**, od -1 x 10^{-130} to 9.99...99 x 10^{125} **negativni** i **nula**
BINARY_FLOAT	Mali broj u pokretnom zarezu (single-precision floating-point)
BINARY_DOUBLE	Veliki broj u pokretnom zarezu (double-precision floating-point)

Za kolone u kojima treba sačuvati brojčanu vrednost dovoljno je navesti tip podataka **NUMBER** bez preciznosti i skale. Međutim, ako želimo da „formatiramo“ brojčanu vrednost onda to možemo uraditi tako što ćemo definisati preciznost (ukupan broj numeričkih vrednosti) i skalu (broj numeričkih vrednosti iza decimalne tačke). Maksimalni broj numeričkih mesta je 38. Umesto tačne preciznosti može se koristiti specijalni karakter „*“ pa tako možemo pisati i NUMBER(*, 6) gde se podrazumeva da je preciznost 38. PLS_INTEGER zauzima manje prostora za smeštaj vrednosti u bazi podataka, tako da se ovaj tip promenljive više preporučuje nego upotreba tipa NUMBER.

Datumski tipovi podataka - Date types

TIP PODATKA	OPIS
DATE	Datum i vreme do nivoa sekunde
TIMESTAMP	Datum i vreme do nivoa delića sekunde
INTERVAL YEAR (year_precision) TO MONTH	Vreme izraženo u godinama i mesecima gde year_precision je broj numeričkih vrednosti u granicama 0-9 (podrazumevana vrednost je 2)
INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision	Vreme izraženo u danima, satima, minutama i sekundama, day_precision je maksimalni broj numeričkih mesta u delu DAY (od 0 – 9 podrazumevana vrednost je 2) i fractiona_seconds_precision je maksimalni broj numeričkih mesta u delu SECOND (od 0 – 9, podrazumevana vrednost je 6)

Prilikom unosa datumske promenljive u neku tabelu koristi se funkcija za unos datuma TO_DATE koja ima format kao na sledećem primeru: TO_DATE(‘15/10/2012 15:45:33’, 'DD/MM/YYYY HH24:MI:SS'). Znakovi za odvajanje delova datuma (/:-) se mogu uzeti po želji, ali se mogu i izostaviti – važno je da se u stringu u kome je smeštena datumska vrednost znakovi za odvajanje delova datuma unesu na isti način kao i u stringu u kome je definisan format. Tako se datumska vrednost iz prethodnog primera može zapisati i u obliku: TO_DATE(‘20121015 154533’, 'YYYYMMDD HH24MISS').

Ostali tipovi podataka

TIP PODATKA	OPIS
BLOB	Binary Large Object (do 8 Terabajta za Oracle verzije 9-10, a do 128 TB za 11g)
BFILE	Pokazivač na binarni fajl na disku
ROWID	Tip podataka koji koristi Oracle baza kako bi sačuvala adresu za svaki red
XMLType	XML podaci veličine do 4 gigabajta

BLOB tip podataka služi da sačuva ne-strukturirane binarne podatke u bazi. **BFILE** tip podataka čuva ne-strukturirane binarne fajlove van baze podataka u fajlovima operativnog

sistema, tako što se u koloni ovog tipa smešta pokazivač na binarni fajl u operativnom sistemu.

Oracle baza podataka koristi vrednosti tipa podataka **ROWID** interno za kreiranje indeksa. Svaki ključ u indeksu je povezan sa rowid koji pokazuje na početak traženog reda u tabeli radi bržeg pristupa podacima.

XMLType se može upotrebiti kao bilo koji drugi tip promenljive i upotrebljava se u kolonama tabela i u pogledima. Isto tako, korisniku se nudi i određeni broj funkcija koje rade sa XML podacima. Tako npr, funkcija extract može da izdvoji određeni čvor iz instance tipa XMLType. U upitima se koriste standardne SQL komande za ovaj tip podataka kao i za ostale tipove podataka

Domen podataka

Navođenjem tipa podatka automatski se ograničava skup dozvoljenih vrednosti, odnosno određuje se domen [13], [16], [25], [27], [30]. Domen (engl. *domain*) nekog svojstva predstavlja skup mogućih (dozvoljenih) vrednosti koje to svojstvo (atribut) može da ima. Domen ima sopstveno ime i pri tome više atributa mogu biti definisani nad istim domenom.

Napomena: Domen svojstva nije isto što i domen u matematici, gde predstavlja skup originala funkcije. Formalno gledano, atribut predstavlja funkciju koja preslikava skup entiteta u neki domen. Dakle, svaki atribut je opisan skupom parova (atribut, vrednost). Svaki radnik pojedinačno opisan je skupom {(idbr, *5932*), (ime, *Mitar*), (prezime, *Vuković*), … (brod, *20*)}.

Definisanje domena je osnovni oblik definisanja integriteta podataka, ali se dodatno mogu definisati uslovi koje vrednosti moraju da zadovolje.

Domeni se često mešaju s tipovima podataka. To su dva različita pojma. Domen je uži pojam od tipa podataka jer definicija domena zahteva precizniji opis ispravnih, prihvatljivih vrednosti podataka. Na primer, domen kvalifikacija predstavlja stručnu spremu radnika. Ako se definiše kao podatak tipa Text, onda to može biti bilo koji tekst. Ali, ispravne su samo vrednosti iz skupa (NKV, KV, VKV, VS, VSS, MR, DR) ili (niža, srednja, viša, visoka, magistar, doktorat). Domen se ne može uvek svesti na nabrojiv skup vrednosti, ali se mogu definisati pravila kojima se određuje skup prihvatljivih vrednosti. Ali, domen nije ni prosta kombinacija tipa podataka i pravila za ispravnost podataka.

Tipovi podataka predstavljaju fizički pojam, dok je domen logički pojam. Ima smisla upoređivati atribute koji su definisani nad istim domenima, i nad njima se mogu obavljati i sve druge relacione operacije (unija, presek, diferencija, spajanje). Plata radnika i njegova šifra mogu biti tipa Number (broj). Nema nikakvog smisla upoređivati te dve vrednosti. Šifra radnika i šifra njegovog šefa ili rukovodioca odeljenja su atributi sa istim domenom i ima smisla obavljati relacione operacije nad njima.

Upotreba NULL vrednosti

Jedna od najvećih novina koje su donele relacione baze podataka jeste mogućnost prikazivanja nepostojećeg podatka čija vrednost je nedefinisana. To su **NULL** vrednosti i one ponekad moraju postojati [13], [16], [25], [27], [30].

Kada u bazi postoje atributi za koje su **NULL** vrednosti "normalne" jer to svojstvo nije primenljivo na sve primerke nekog entiteta. Takav je na primer atribut *premija* u tabeli RADNIK. Ako to svojstvo nije primenljivo na većinu primeraka entiteta onda taj atribut treba eliminisati iz tabele još u fazi projektovanja. U slučaju da je taj podatak vrlo važan za one koji to svojstvo imaju, onda se kreira nova tabela samo za one objekte koji to svojstvo poseduju (u ovom slučaju može se napraviti tabela PREMIJA<*idbr#*, *premija*>).

1. Ako vrednost nekog atributa za neke objekte još nije poznata ili nije dozvoljena (neki radnici još uvek nemaju telefon, rukovodioca, ili nisu raspoređeni ni u jedno odeljenje).
2. Kada nije nastupio momenat delovanja nekog atributa (plata ili premija za mesec mart biće poznati su tek tokom aprila).

Kod nekih baza podataka upotreba null vrednosti za neki atribut se podrazumeva (implicitna), a kod nekih se mora neposredno (eksplicitno) dozvoliti. Pošto nema pravila, za svaki SUBP (sistem za upravljanje bazom podataka), pri definisanju objekata baze podataka i svakog atributa treba se odlučiti za odgovarajući mogućnost.

Opšta preporuka jeste da upotrebu Null vrednosti za neki podatak treba dozvoliti uvek, sem kada za to postoje posebni razlozi (primarni ključ, spoljni ključ, tj. ne sme se narušiti integritet entiteta i referencijalni integritet). Null vrednosti treba dozvoliti gde god u stvarnom svetu postoji razuman razlog da vrednost bude nepoznata, ili barem tamo gde Null vrednost neće učiniti da podatak bude potpuno neupotrebljiv u sistemu.

Ako ne dopustite Null vrednosti, moraćete da smislite način kako će sistem pomoći korisniku koji pokušava da unese nov zapis a nema sve potrebne podatke. Problem je *kada* će biti potrebni podaci koje korisnik unosi. Iz analize radnih procesa sistema zna se da su određeni podaci neophodni pre završetka određenog posla. Ali to ne znači da su *svi* podaci neophodni za obavljanje svih *poslova* obavezno potrebni i kada se dodaje nov zapis.

Podaci o plati i premiji (ili o bankovnom računu) zaposlenog potrebni su vam da biste mu isplatili zaradu. Prema tome, polja za podatke o plati i premiji (bankovnom računu) ne mogu biti Null kada dođe dan isplate. Ali ne bi trebalo da sistem sprečava korisnika da unese zapis o novom zaposlenom, niti da obavlja druge poslove, samo zato što nema baš sve podatke. Ako ne želite da prihvatite Null vrednosti, čak ni privremeno, najlakši način da rešite taj problem jeste da zadate neku razumnu podrazumevanu vrednost (engl. *default value*).

Objekti baze podataka

Za svaki problem koji se rešava pomoću računara postoji ograničen skup objekata koji su od interesa. Ovi objekti se nazivaju **elementarni objekti** ili **entiteti**. Skup elementarnih objekata predstavlja **prostor objekata** ili **tip**. Svaki prostor objekata mora imati ime i ono je jedinstveno u jednom sistemu (ili bazi podataka). Ne mogu postojati u jednoj bazi podataka dva prostora objekata sa istim imenom. Reč entitet potiče iz grčkog jezika i označava bitnost odnosno suštinu. Entitet je nešto što postoji i što se na jedinstven način može identifikovati. Po svojoj prirodi entitet može biti realni objekat (zgrada, osoba, radnik, itd.), apstraktni pojam (zvanje, količina, itd.), događaj (zapošljavanja, upis, udes, itd.) ili veza dva ili više objekata (radnik-odeljenje, student-predmet-profesor, lekar-pacijent, itd.) [1], [2], [3], [4], [13], [14], [16], [25], [27], [30].

U realnom svetu svaki objekat (čovek, dokument, prevozno sredstvo...) ima obilje osobina. Pri projektovanju informacionih sistema (IS), vrši se apstrakcija, to jest zanemarivanje nebitnih stvari. Svakom objektu pripisuju se određeni kvalitet u vezi s

problemom koji se rešava i koji su od značaja za rešenje problema. Ti kvaliteti nazivaju se **osobine**, **svojstva** ili **atributi**).

Na primer, ako je reč o IS nekog preduzeća, za objekat **čovek** od značaja su nam osobine kao što su stručna sprema i radni staž, a kad bi se radio IS nekog sportskog društva, onda bi od značaja za čoveka bile osobine sport kojim se bavi, visina, težina i sl.

Osobine objekta koje nisu od značaja za rešenje problema se zanemaruju (apstrahuju, apstrahovanje podataka[6]). Pošto za jedan problem može biti od značaja više raznih atributa istog ili raznih objekata, svaki atribut mora imati **ime**, a takođe i *vrednost tj. kvantitativni* odraz kvaliteta i predstavlja **elementarni podatak**. Atributsko ime je naziv određenog kvaliteta (svojstva objekta), a atributska vrednost je kvantitativna, brojna vrednost koja pripada izvesnom skupu vrednosti (domen). Svakom elementarnom objektu pripisuje se elementarni podatak koji ima samo jednu vrednost (skalar).

Jedan objekat može da karakteriše više atributa od važnosti za rešenje datog problema i oni se nazivaju **elementarna obeležja**. Elementarno obeležje je obeležje koje se dalje ne može dekomponovati, ili koje se u datom slučaju dalje ne dekomponuje na delove. To je odluka projektanta i zavisi od problema koji se rešava i načina korišćenja.

Na primer:

- datum se može posmatrati kao jedan atribut *Datum* sastavljen od 3 dela (dan, mesec, godina).
- Tri atributa: *Dan*, *Mesec*, *Godina*.

Jedan atribut, ili grupa od nekoliko atributa, koji su jedinstveni u svom pojavljivanju, kandidati su za identifikatora objekata datog tipa. Skup, niz ili logički proizvod elementarnih obeležja predstavlja složeno obeležje i njemu se ne pridružuje ime. Između obeležja-atributa i tipa entiteta nema jasne razlike, jer imenovano složeno obeležje ustvari predstavlja tip entiteta. Jedna definicija tipa objekta, odnosno entiteta mogla bi biti: **Entitet je imenovani skup atributa**. Na primer: plata i premija mogu biti atributi *Plata* i *Premija* tipa **Radnik**, a mogu biti tip (prostor objekata) **Zarada** sa dva atributa *Plata* i *Premija*. Status objekta informacionog sistema imaju oni entiteti koji pored identifikatora objekta (*primarnog ključa*) imaju još neka svojstva koja se opisuju atributima. Pod atributom se smatra ono svojstvo objekta koje se opisuje jednim podatkom. Ukoliko je za opisivanje atributa potrebno više podataka, onda taj atribut predstavlja novi objekat. Skup svih vrednosti elementarnih obeležja koja se odnose na jedan objekat predstavlja zapis ili logički slog, pri tome se svaki elementarni podatak (pojedinačna vrednost) smešta u jedno polje zapisa.

Objekat: RADNIK					
Atributi:	**Br.Rad.**	**Ime**	**Pol**	**Odeljenje**	**Radni_staž**
Zapis:	**53125**	**Ana**	**Ž**	**Prodaja**	**3G. 2M. 17D.**

Skup zapisa koji odgovaraju jednom prostoru objekata čini datoteku, i pri tome mora važiti sledeće:

- Jednom prostoru objekata odgovara jedna ili više datoteka.
- Broj zapisa u datoteci jednak je broju objekata u prostoru.
- Broj elementarnih obeležja objekta jednak je broju polja u zapisu.

Ukoliko za jednu aplikaciju treba definisati više prostora objekata (skupova objekata), obrazuje se biblioteka datoteka, a kada se između tih datoteka uspostave određene logičke veze pomoću softvera koji se zove DBMS (Data Base Management System) nastaje **baza podataka**.

[6] Apstrakcija, zanemarivanje nebitnog, isticanje bitnog. Misaono izdvajanje nekih osobina i svojstava predmeta od samog predmeta

Kolekcija podataka organizovanih za brzo pretraživanje i pristup, koja zajedno sa sistemom za administraciju, organizovanje i memorisanje tih podataka, čini sistem baze podataka. Iz ugla korisnika, podaci su na neki logički način povezani.

Baza podataka je logički integrisani skup podataka sa centralizovanim upravljanjem. Danas su dominantne relacione baze podataka koje predstavljaju skup vremenski promenljivih tabela [1], [2], [3], [4], [13], [14], [16], [25], [27], [30].

Relacija definisana na **n** skupova je podskup Dekartovog proizvoda tih skupova:

$$R \subseteq D1 \times D2 \times \ldots \times Dn$$

podskup sadrži one n-torke Dekartovog proizvoda koje zadovoljavaju zadatu relaciju. Redosled elemenata u jednoj n-torki je bitan, a redosled n-torki je proizvoljan.

Neka su dati skupovi A={1,2,3,4} i B={4,6,8} tako da za$\forall(a \in A)$ i $\forall(b \in B)$, i n neka je nad skupovima A i B zadata relacija:

$$R: A \times B = \{(a,b) \mid a = b/2\}$$

$$R = \{(2,4),(3,6),(4,8)\}.$$

Skupovi **D1, D2,...,Dn** nazivaju se domenima relacije **R**, dok broj domena određuje stepen relacije (unarna, binarna, n-arna relacija). Broj n-torki u relaciji naziva se kardinalnost.

Atribut relacije predstavlja imenovani domen relacije. Definisani su domeni - atributi:

SIFRA = {123 ,234, 345}
NAZIV = {stolica, sto}
CIJENA = {100,200,300}

i relacija:

$$PROIZVOD \subseteq SIFRA \times NAZIV \times CIJENA.$$

Koncept atributa omogućava predstavljanje relacije kao tabele, pa se relacija PROIZVOD može predstaviti sledećom tabelom:

SIFRA	NAZIV	CENA
123	stolica	100
234	sto	200
345	sto	300

Relacija je osnovni element relacionog modela. U bazama podataka pojam relacija je u stvari sinonim, uz neka ograničenja za tabelu. E. F. Codd je u svom radu, kojim je po prvi put "promovisao" relacioni model, pod relacijom definisao pravougaono područje - tabelu koje se sastoji od kolona (atributa i vrednosti atributa - podataka) i redova (n-torki, odnosno slogova). Pošto je relacija skup, a svaka tabela nije, definišu se određeni uslovi koje tabela mora da zadovolji da bi bila relacija.

Tabela se definiše kao niz naziva kolona sa nula ili više redova (vrsta, n-torki, torki, slogova, zapisa) kojima su zadate vrednosti kolona. U jednoj bazi ime tabele je jedinstveno (ne mogu postojati dve tabele sa istim imenom). Svaka relacija se može prikazati kao tabela, ali svaka tabela nije relacija. Kada u relacionim bazama koristimo pojam tabele onda ustvari govorimo o relacijama, odnosno o tabelama koje zadovoljavaju sledeće uslove:

Kolone predstavljaju osobine objekata, i pri tome:

- naziv kolone je ime kojim se predstavljaju vrednosti unete u tu kolonu;
- naziv kolone je jedinstven za jednu tabelu;
- u raznim tabelama mogu postojati kolone sa istim imenom;
- sve vrednosti unutar jedne kolone su istog tipa;
- redosled kolona nije bitan, ali je isti za sve redove u tabeli.

Redovi tabele odgovaraju logičkim slogovima datoteke i imaju sledeće osobine:

- svaki red ima jednu i samo jednu vrednost za svaku kolonu tabele;
- redovi tabele imaju isti skup kolona i pri tome neke vrednosti mogu biti **NULL**, odnosno ne moraju imati vrednosti za svaku kolonu;
- redovi su jedinstveni, odnosno ne smeju postojati identični redovi unutar tabele (svaki entitet mora da se identifikuje jednoznačno).

Da bi redovi bili jedinstveni, svaka tabela mora da ima ključ, odnosno primarni ključ (engl. *primary key*) [1], [2], [3], [4], [13], [14], [16], [19], [29], [27], [30]. Ključ tabele je neprazan skup kolona čije vrednosti jedinstveno određuju vrednosti svih drugih kolona. To je takozvani super ključ koji obezbeđuje da sve vrste budu među sobom razlučite (*jedinstvenost,* engl. *unique*). Ako ne postoji ni jedan podskup ovog skupa koja ima predhodno svojstvo (*minimalnost),* onda je to **primarni ključ** (*jedinstvenost* i *minimalnost*). Na primer, grupa atributa šifra radnika (idbr), ime i prezime je svakako jedinstvena i predstavlja super ključ. Naime, šifra radnika je sama jedinstvena pa su atributi ime i prezime suvišni.

Svi skupovi atributa koji imaju svojstvo jednoznačnosti i minimalnosti su kandidati da budu primarni ključ tabele. Jedna tabela može da ima nekoliko skupova atributa koji su pogodni za primarni ključ, to su ekvivalentni ključevi. Jedan od ekvivalentnih ključeva se bira za primarni ključ. Ukoliko ne postoji nijedan pogodan skup atributa koji bi mogao da predstavlja primarni ključ, može se dodati veštački ključ, koji obično predstavalja neku šifru ili neki u nizu uzastopnih brojeva (autonumber). Primarni ključ je: jedinstven, dostupan (za korišćenje) i nepromenljiv. Primarni ključ obezbeđuje integritet entiteta (slogova, n-torki), odnosno da se svaki pojedinačni primerak nekog prostora objekata može razlikovati od svih drugih. Samim tim, primarni ključ, ili bilo koji njegov deo, ne mogu imati vrednost **NULL**. Ovo ujedno predstavlja i pravilo integriteta entiteta.

Svaka relacija mora da ima bar jednog kandidata za ključ: to je skup svih atributa koji čine jedan slog (n-torku, torku). Kandidat za ključ može da se sastoji samo od jednog svojstva (prost ključ) ili od više atributa (složen ključ). Složeni primarni ključevi su potpuno prihvatljivi i ne treba ih izbegavati. Ponekad je iz praktičnih razloga razumno zameniti previše složen ključ veštačkim identifikatorom-poljem tipa identity ili autonumber.

"Primarnim ključem obezbeđuje se integritet entiteta, a spoljni (strani) ključevi omogućavaju da se uvede referencijalni integritet. Ovim se znatno smanjuje mogućnost unosa pogrešnih podataka, ali se nepogodnim izborom primarnih i spoljnih ključeva može blokirati baza. "[7]

Dva entiteta, pojedinačna objekta, zapisa u tabeli, su različita ako imaju različit primarni ključ. Svi ostali atributi mogu imati iste vrednosti, odnosno ne moraju se razlikovati kod svih entiteta. Na primer, ako jedan student istovremeno studira na dva fakulteta, on će imati sve opšte podatke iste u oba slučaja, jedina razlika će biti u broju indeksa.

[7] [19] Anton Dončev, Slobodan Obradović, Svetlana Anđelić, *"METODOLOGIJA PROJEKTOVANJA BAZA PODATAKA - Algoritam za izbor primarnog ključa",* Festival informatičkih dostignuća INFOFEST 2006, Budva

Spoljni, strani ključ je atribut u nekoj tabeli (spoljna, zavisna tabela) čiji je skup vrednosti podskup skupa vrednosti primarnog ključa neke druge tabele (primarna, roditeljska, ili referentna tabela). Ovo ujedno predstavlja i **pravilo referencijalnog integriteta**. ″Spoljni ključ je atribut A u tabeli T1 koji služi za povezivanje sa tabelom T2, u kojoj je taj atribut primarni ključ K. Spoljni ključ može da ima vrednost **NULL**.″ [8]

Strani ključ (engl. foreign key) je skup kolona u jednoj tabeli, čije vrednosti moraju da odgovaraju vrednostima primarnog ključa, ili ograničenja tipa unique u drugoj tabeli (roditeljska, ili referentna tabela). Ograničenje stranog ključa definiše kolone stranog ključa (u istoj tabeli kao i ograničenje), kao i kolone primarnog ključa, ili ograničenja unique koja se nalaze u tabeli roditelju. Većina SUBP traži da roditeljska tabela i zavisna tabela budu u istoj bazi podataka i da svaka kolona stranog ključa mora biti istog tipa i veličine kao odgovarajuća kolona primarnog ključa, ili ograničenja unique. Spoljni kluč koristi se da iskaže vezu između dva prostora objekata (dve relacije, dve tabele). Radnici rade u nekim organizacionim jedinicama-odeljenjima. Da bi iskazali tu vezu i da bi se znalo u kom odeljenju radi određeni radnik, neophodno je entitetu radnik dodati atribut šifra odeljenja (brod). Šifra odeljenja-brod je primarni ključ, identifikator entiteta odeljenje [1], [2], [3], [4], [13], [14], [16], [19], [25], [27], [30].

Da bi se izbeglo višestruko pamćenje istih podataka (redundansa), vrši se razbijanje jedne relacije na dve ili više relacija. Te novonastale relacije se povezuju međusobno pomoću spoljnih ključeva. Ako se te veze prekinu, originalna relacija se ne može rekonstruisati (razbijanje bez gubitaka), odnosno dolazi do gubitaka informacija. Narušeno je pravilo referencijalnog integriteta. U tom slučaju sistem u najboljem slučaju postaje nepouzdan a najčešće i neupotrebljiv. Osnovno pravilo referencijalnog integriteta jeste da nijedan zapis u spoljnoj tabeli ne sme da sadrži spoljni ključ za koji ne postoji odgovarajući zapis u primarnoj tabeli. Drugim rečima, u spoljnoj tabeli ne smeju da postoje zapisi ''siročići''. Entiteti siročići mogu da nastanu zbog sledećih razloga[9]:

1. Spoljnoj tabeli dodat je zapis koji nema odgovarajućeg parnjaka u kandidatu za ključ u primarnoj tabeli.
2. Spoljnom ključu dodeljena je vrednost koja ne postoji u primarnoj tabeli.
3. Izmenjena je vrednost kandidata za ključ u primarnoj tabeli.
4. Izbrisan je zapis iz primarne tabele

Prvi slučaj je jednostavno zabranjuje i pri tome se ne odnosi na nepoznate i nepostojeće vrednosti. Ako veza nije obavezna može se uneti proizvoljan broj nepostojećih ili nepoznatih vrednosti (Null).

Drugi i treći slučaj se prosto sprečavaju tako što se zabrani izmena kandidata za ključ. Ovo je ujedno i uslov integriteta entiteta (nije dozvoljena izmena vrednosti kandidata za ključeve). Ako to baš mora da se dogodi, treba nametnuti lančano ažuriranje. Takođe treba obezbediti mehanizam da nova vrednost bude ispravna. Ovaj mehanizam ima većina postojećih mašina baze podataka, ili se taj uslov može ispuniti.

Četvrti slučaj spoljnih ključeva siročića se dešava kada se obriše zapis u primarnoj tabeli. Na primer, obrišemo zapis o nekom odeljenju, šta će se desiti sa radnicima koji su radili u tom odeljenju? Rešenje može biti ili zabrana brisanja entiteta u primarnoj tabeli ili

[8] [18] Slobodan Obradović *"Poslovno informatička obuka"*, Nacionalna služba za zapošljavanje, ISBN 86-902585-8-2, Beograd, 2004.

[9] Postoje i specijalne vrste uslova za očuvanje referencijalnog integriteta kada je maksimalna kardinalnost ograničena na neku vrednost. Na primer, rukovodilac može imati najviše 5 podređenih, tim u košarci može imati najviše 12 igrača, i sl.

lančano brisanje. Kod lančanog brisanja pri brisanju zapisa u primarnoj tabeli, brišu se svi povezani zapisi u spoljnoj tabeli. Ovo može biti vrlo opasno rešenje (gubljenje podataka) pa je moguće i treće rešenje. Zavisne zapise u spoljnoj tabeli povezati sa nekim drugim zapisom u primarnoj tabeli. Ovo najčešće nije preporučljivo.

Kreiranje tabela (CREATE TABLE)

Svaki SUBP ima svoju sintaksu za kreiranje (CREATE) i izmenu (ALTER) tabela. Sintaksa za kreiranje tabela za svaki SUBP je prilično složena i ima dosta detalja koji su namenjeni vrlo naprednim korisnicima. S toga ćemo ovde predstaviti što je moguće jednostavniju varijantu u cilju sagledavanja osnovnih funkcionalnosti ove naredbe.

Za MS Access naredba Create table ima sledeću sintaksu

```
CREATE [TEMPORARY] TABLE ime_tabele
(ime_kolone tip [(veličina)] [NOT NULL] [WITH
COMPRESSION | WITH COMP] [index1]
[, ime_kolone tip [(veličina)] [NOT NULL]
[index2] [, …]]
[, CONSTRAINT indeks_nad_više_kolona [, …]])
```

gde je:

CREATE [TEMPORARY] TABLE ime_tabele	Kada se koristi ključna reč TEMPORARY tada se kreira tzv. privremena tabela (u kojoj se smeštaju privremeni podaci a zatim brišu iz nje).
(ime_kolone tip [(veličina)]	Definiše se naziv kolone, tip podataka koji će se čuvati u toj koloni kao i veličina tog tipa (npr. JMBG char(13))
[NOT NULL] [WITH COMPRESSION | WITH COMP] [index1]	Ako se kolona definiše bez ključnih reči NOT NULL onda se podrazumeva da kolona može imati prazne (NULL) vrednosti. Atribut WITH COMPRESSION se koristi za kompresiju nad kolonama tipa character ili memo (TEXT). WITH COMP je alternativa (skraćeno pisanje) ključnih reči WITH COMPRESSION. Ako kolona treba da bude indeksirana (zbog bržeg pretraživanja) onda se pored naziva kolone i tipa podatka dodaje i naziv indeksa nad tom kolonom.
CONSTRAINT indeks_nad_više_kolona [, …]])	Ova ključna reč definiše indeks nad više kolona. Tako ako hoćemo da pretražujemo kolone prezime i ime onda pišemo: *CONSTRAINT prezime_ime (prezime, ime)*

U MS SQL serveru kreiranje tabele ima sledeću sintaksu.

```
CREATE TABLE
   [ ime_baze . [ ime_šeme ] . | ime_šeme . ]
ime_tabele
   [ AS FajlTabele ]
   (          {          <definicija_kolone>          |
<definicija_izračunate_kolone>
      |      <definicija_skupa_kolona>      |      [
<ograničenje_tabele> ] [ ,...n ] } )
 [ ; ]
```

gde je:

[ime_baze . [ime_šeme] . | ime_šeme .] ime_tabele	Definiše opciono ime_baze i ime_šeme i obavezan naziv tabele (npr. preduzece.dbo.odeljenje)
[AS FajlTabele]	Kada postoji ova opcija onda se ne navode kolone iz kojih se tabela sastoji, već se format tabele automatski preuzima iz FajlTabele
definicija_kolone	Definiše za svaku kolonu tip podataka, način upoređivanja (COLLATION), da li vrednost može da bude prazna (NULL) ili ne, ograničenja (CONSTRAINT) o primarnom, jedinstvenom ili spoljnjem ključu u koloni ili o nekoj proveri (CHECK), podrazumevana vrednost (DEFAULT), indikator kolone sa automatski generisanim vrednostima (IDENTITY), proređena kolona (SPARSED) sa dosta NULL vrednosti itd. Npr. ako kolona treba da sačuva broj odeljenja (brod) celobrojnog tipa i ako ta kolona predstavlja primarni ključ, tada se to piše: *brod smallint NOT NULL PRIMARY KEY*
<definicija_izračunate_kolone>	Predstavlja jednačinu po kojoj se vrednost u ovoj koloni automatski izračunava u zavisnosti od vrednosti u ostalim kolonama (npr. *kolona4 as 2*kolona2*3.14*)
<definicija_skupa_kolona>	Definiše ime skupa kolona kao XML reprezentacije koja kombinuje sve „retke“ (SPARSE) kolone tabele u jedan strukturirani izlaz
<ograničenje_tabele>	Slično ograničenjima koja se mogu postaviti nad kolonom, ovde se postavljaju ograničenja nad tabelom – primarni, jedinstveni, spoljni ključ, ograničenja provere. Tako npr. ako imamo relaciju između kolona kolona2 i kolona3 u tabeli koju kreiramo sa kolonama kolona1 i kolona2 u tabeli TABELA2 onda pišemo: *CONSTRAINT fk_ovatabela_tabela2 FOREIGN KEY (kolona2, kolona3) REFERENCES tabela2(kolona1, kolona2).*

Za SUBP MySQL kreiranje tabele ima sledeću sintaksu.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
ime_tabele
   (definicija_kolona,...)
   [opcije_nad_tabelom]
```

gde je:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] ime_tabele	Kada se koristi ključna reč TEMPORARY tada se kreira tzv. privremena tabela (u kojoj se smeštaju privremeni podaci a zatim brišu iz nje). Kada se koriste ključne reči IF NOT EXIST tada se tabela kreira i pod uslovom da postoji tabela sa istim imenom. Tada se tabela sa istim imenom briše i kreira nova sa novom definicijom kolona.
definicija_kolona	Sastoji se iz naziva kolone, tipa podataka koji se čuva u toj koloni, da li vrednost može da bude prazna (NULL) ili ne, ograničenja (CONSTRAINT) o primarnom, jedinstvenom ili spoljnjem ključu u koloni ili o nekoj proveri (CHECK), indikator kolone sa automatski generisanim vrednostima (AUTO_INCREMENT), načinom organizovanja indeksa nad tom kolonom itd. Npr. ako kolona treba da predstavlja auto inkrementirajuću vrednost celobrojnog tipa i ako ta kolona predstavlja primarni ključ, tada se to piše: *autoid int NOT NULL AUTO_INCREMENT, PRIMARY KEY (autoid)*
opcije_nad_tabelom	Ovde se definiše ENGINE - tip organizovanja tabele, (ima ih više, najčešće se koriste: MyISAM, InnoDB, Memory, Merge, Archive), skup karaktera (character set), način upoređivanja (COLLATION), fizički prostor tabela (TABLESPACE) gde se tabela nalazi itd. Tako npr. ako se tabela kreira sa engine-om InnoDB i sa skupom karaktera utf8, onda pišemo: *ENGINE=InnoDB DEFAULT CHARSET=utf8*

SUBP Oracle ima sledeću sintaksu za kreiranje relacione tabele.

```
CREATE [ GLOBAL TEMPORARY ] TABLE [
ime_šeme. ]ime_tabele
  [ (relacione_osobine) ]
  [ ON COMMIT { DELETE | PRESERVE } ROWS ]
  [ fizičke_osobine ]
  [ osobine_tabele ] ;
```

gde je:

CREATE [GLOBAL TEMPORARY] TABLE [ime_šeme.]ime_tabele	Kada se koristi ključna reč GLOBAL TEMPORARY tada se kreira privremena tabela (u kojoj se smeštaju privremeni podaci a zatim brišu iz nje). Ime_šeme je opcioni naziv šeme a ime_tabele je naziv tabele koja se kreira.
relacione_osobine	Sastoji se iz naziva kolone, tipa podataka koji se čuva u toj koloni, da li vrednost može da bude prazna (NULL) ili ne, podrazumevana vrednost (DEFAULT), ograničenja o primarnom, jedinstvenom ili spoljnjem ključu u koloni. Npr. ako kolona treba da čuva podatke o plati zaposlenih i ako se na početku ne zna tačno plata, tada se to piše: *plata number(17,2) DEFAULT 0.00*
fizičke_osobine	Ovde se definiše TABLESPACE – fizički prostor u kome se tabela nalazi, indeksna organizacija (ORGANIZATION INDEX) nad tabelom itd. Ove opcije se uglavnom popunjavaju od strane DB administratora.
osobine_tabele	Ovde se definiše način particionisanja tabele (PARTITION) po kolonama po opsezima vrednosti, hash vrednostima itd. To se uglavnom koristi radi optimizacije kreiranja upita (performansi sistema) i aktivnost definisanja ovih osobina je posao DB administratora.

Opis baze PREDUZEĆE

Svi primeri će biti urađeni na jednoj bazi koja sadrži četiri međusobno povezane tabele. Pri kreiranju tabele za svaki atribut mora se navesti tip podatka. U sledećem primeru možemo videti kako se jednostavno pomoću SQL-a kreira nova tabela. Pretpostavimo da u informacionom sistemu nekog preduzeća treba kreirati tabelu **RADNIK,** u kojoj bi se nalazili podaci o kvalifikaciji, imenu i prezimenu radnika, poslu koji obavlja, njegovom rukovodiocu, datumu zaposlenja, premiji, plati i broju odeljenja u kome radi [25], [27], [30]..

*Tabelu **RADNIK** kreiramo naredbama:*

MS Access:

```
CREATE TABLE RADNIK (
  idbr# INTEGER NOT NULL,
  ime CHAR(25) NOT NULL,
  prezime CHAR(25) NOT NULL,
  posao CHAR(20),
  kvalif CHAR(3),
  rukovodilac$ INTEGER,
  datzap DATE,
  premija FLOAT(1),
  plata FLOAT(1),
  brod$ SMALLINT
);
```

MS SQL:

```
CREATE TABLE radnik(
  idbr int NOT NULL PRIMARY KEY,
  ime NVARCHAR (25) NOT NULL,
  prezime NVARCHAR (25) NOT NULL,
  posao NVARCHAR(20),
  kvalif NVARCHAR(3),
  rukovodilac int,
  datzap SMALLDATETIME,
  premija MONEY ,
  plata MONEY DEFAULT 0.00,
  brod smallint
);
```

MySQL:

```
CREATE TABLE radnik(
  idbr int NOT NULL,
  ime VARCHAR(20) NOT NULL,
  prezime VARCHAR (40) NOT NULL,
  posao VARCHAR(20),
  kvalif VARCHAR(10),
  rukovodilac int,
  datzap DATE,
  premija DECIMAL(17,2) ,
  plata DECIMAL(17,2) DEFAULT 0.00,
  brod smallint,
  PRIMARY KEY  (IDBR)
  ) ENGINE=InnoDB
DEFAULT CHARSET=utf8
COLLATE=utf8_general_ci;
```

Oracle:

```
CREATE TABLE RADNIK(
  idbr NUMBER(6,0) PRIMARY KEY,
  ime VARCHAR2 (20) NOT NULL,
  prezime VARCHAR2 (40) NOT NULL,
  posao VARCHAR2(20),
  kvalif VARCHAR2(10),
  rukovodilac NUMBER (6,0),
  datzap DATE,
  premija NUMBER(17,2) ,
  plata NUMBER(17,2) DEFAULT 0.00,
  brod NUMBER(6,0)
);
```

a kao rezultat dobijamo relaciju:

RADNIK <idbr#, ime, prezime, posao, kvalif, rukovodilac$, datzap, premija, plata, brod$>

u kojoj atributi idbr, ime i prezime ne mogu imati vrednosti **NULL**.

Za potpuniju informaciju o preduzeću, sem tabele RADNIK, potrebno je kreirati i tabelu ODELJENJE u kome radnik radi i tabelu PROJEKAT koja sadrži informacije o poslovima kojima se

preduzeće trenutno bavi. To se postiže sledećim naredbama **CREATE TABLE** (primeri su pokazani samo za SUBP MS Access):

```
CREATE TABLE ODELJENJE
        (brod# SMALLINT NOT NULL,
        imeod CHAR(15) NOT NULL,
        mesto CHAR(20),
        sefod$ INTEGER);

CREATE TABLE PROJEKAT
        (brproj# INTEGER NOT NULL,
        imeproj CHAR(25) NOT NULL,
        sredstva FLOAT(2)
        rok DATE);
```

Ovako definisana tabela RADNIK sadrži i neke relacije između pojedinih radnika – *unarne veze* (neki radnici su istovremeno rukovodioci nekim drugim radnicima).

Takođe, ostvarene su i dve relacije, veze sa jednim drugim entitetom – *binarne veze* sa tabelom ODELJENJE.

Jedna veza pokazuje da su radnici zaposleni, odnosno *rade* u nekom od odeljenja koja se nalaze u sastavu preduzeća. Pri tome, jedan radnik pripada samo jednom odeljenju, a u jednom odeljenju radi više radnika. Ovo je veza tipa 1:N (jedan prema više), a ostvaruje se tako što se u tabeli RADNIK na strani više (više radnika) uvodi kao atribut spoljni, strani ključ *brod$,* koji predstavlja primarni ključ u tabeli ODELJENJE (na strani **1**). Spoljni ključ prepoznajemo po znaku $ iza imena atributa, a primarni ključ iza imena ima znak #. Atribut *brod* u tabeli RADNIK može imati samo neku od vrednosti koja postoji u koloni *brod* u tabeli ODELJENJE.

Druga veza pokazuje da neki radnici *upravljaju* odeljenjima, odnosno da su šefovi pojedinih odeljenja. Jedan radnik može biti šef najviše u jednom odeljenju, a jedno odeljenje može imati najviše jednog šefa. Dakle, ova veza je tipa 1:1. I veze tipa 1:1 mogu se ostvariti preko stranog ključa. Ostvaruju se tako što se u tabeli ODELJENJE uvodi kao atribut (spoljni ključ) primarni ključ u tabeli RADNIK. U ovom slučaju primarni ključ je preimenovan i zove se *šefod,* kako bi njegova namena bila očiglednija. Atribut *šefod* može imati samo neku od vrednosti koja postoji u koloni *idbr* u tabeli RADNIK. Atributi *idbr* u tabeli RADNIK i *šefod* u tabeli ODELJENJE imaju isti domen (skup vrednosti). Ali, svi radnici rade na nekim konkretnim poslovima, projektima i, pri tome, jedan radnik može raditi na više projekata, a istovremeno na jednom velikom projektu radi više radnika. Dakle, ovo je relacija M:N (više prema više). Da bismo ostvarili ovu relaciju, između dva entiteta treba kreirati novu tabelu, nazovimo je UČEŠĆE, koja ima **složeni primarni ključ** (*idbr#,brproj#*), koji čine primarni ključevi iz tabela RADNIK (*idbr#*) i PROJEKAT (*brproj#*).

```
CREATE TABLE UCESCE
        (idbr# INTEGER NOT NULL,
        brproj# INTEGER NOT NULL,
        brsati SMALLINT,
        funkcija CHAR(15));
```

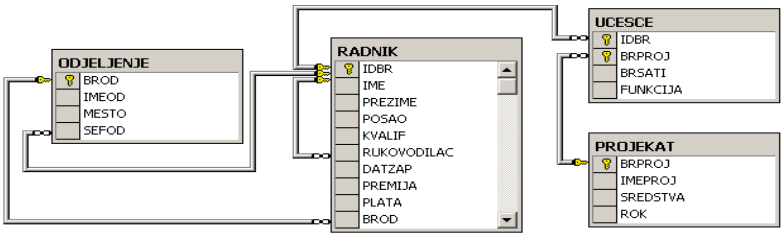
Dakle, bazu podataka čine četiri fizičke tabele (Slika 1):

RADNIK <idbr#, ime, prezime, posao, kvalif, rukovodilac$, datzap, premija, plata, brod$>

ODELJENJE <brod#, imeod, mesto, sefod$>

PROJEKAT <brproj#, imeproj, sredstva, rok>

UCESCE<idbr#, brproj#, brsati, funkcija>



Slika 1. Baza podataka preduzeća – tabele i veze između njih

Moderne verzije RDBMS-a (**Relational Data Base Management System**) omogućavaju da se u naredbi **CREATE TABLE** definišu primarni i strani, spoljni ključ, kao i da se u definiciju tabele unesu dodatna pravila, koja se odnose na očuvanje integriteta podataka pri ažuriranju baze (pri unosu i brisanju podataka-pravila referencijalnog integriteta). *"*Postoje i opšta ograničenja koja se nazivaju *pravila integriteta relacionog modela*, a to su: *integritet entiteta* i *referencijalni integritet."*[10] Ona određuju kako se operacije ažuriranja jedne tabele prenose na druge tabele, koje su u nekoj relaciji-vezi sa tabelom koja se ažurira [4], [5], [6], [7], [8], [9], [10], [11], [12], [15], [18], [21], [25], [27], [30], [31].

"Prvo ograničenje ostvaruje se preko zadavanja tipa podataka i definisanja skupa dozvoljenih vrednosti (validation rules)."[11]

"**Integritet entiteta**: Primarni ključ ili neki njegov deo ne mogu imati Null-vrednost. Kako je primarni ključ jedinstven i neponovljiv, to se na ovaj način obezbeđuje da svaki red, odnosno objekat koji je opisan podacima, u toj tabeli bude jedinstven."[12]

"**Referencijalni integritet**: Skup vrednosti spoljnjeg ključa u nekoj tabeli T1, koja je u relaciji N:1 (povezana) sa nekom drugom tabelom T2, mora biti podskup skupa vrednosti primarnog ključa u tabeli T2 sa kojom je tabela T1 povezana. Na primer atribut BROD u tabeli RADNIK može imati samo one vrednosti koje postoje u tabeli ODELJENJE. "[13]

Kreiranje objekata baze i dodavanje ograničenja - CONSTRAINT

Za kreiranje tabela koristi se naredba **CREATE TABLE**, a neophodno je navesti ime tabele, kao i imena atributa i odgovarajući tip podataka za svaki atribut. Prilikom kreiranja tabele mogu se navesti i neka druga ograničenja koja imaju atributi. Tako npr., ako vrednost nekog atributa mora biti uneta tj. atribut ne sme imati **NULL** vrednost, to se može obezbediti odmah prilikom kreiranja tabele dodajući deo **NOT NULL** odmah iza tipa podatka.

Ograničenje **DEFAULT** služi za automatsko upisivanje vrednosti u određenu kolonu kada nikakva vrednost nije zadata. Time se znatno ubrzava unos podataka (kada se neka vrednost često ponavlja), smanjuje se mogućnost pojave pogrešnih podataka i obezbeđuje se integritet podataka (da ne bi podatak bio **NULL**). Prilikom kreiranja tabele naredbom

[10] [18] Slobodan Obradović *"Poslovno informatička obuka",* Nacionalna služba za zapošljavanje, ISBN 86-902585-8-2, Beograd, 2004.

[11] [21] S. Obradović, I. Vujičić, A. Žorić: *"Primena računara u poslovanju",* S. Obradović, ISBN-86-904623-4-1, Beograd, 2006.

[12] [30] Obradović, S., Pandurov, T., Vučinić, B.: "MS-Access *Projektovanje baza podataka i aplikacija",* Visoka škola elektrotehnike i računarstva, ISBN – 978-86-85081-99-6, Beograd, 2009.

[13] [31] Obradović, S., Barbarić, Ž.: " *Informatika i informacione* tehnologije", Visoka škola elektrotehnike i računarstva, Beograd, 2009.

CREATE TABLE može se odmah dodati i primarni ključ (**PRIMARY KEY**) za tu tabelu ili postaviti neko ograničenje za polje koje treba da ima jedinstvenu vrednost (**UNIQUE**), a nije primarni ključ. Takođe, odmah je moguće i uraditi povezivanje tabela dodavanjem spoljnjeg, stranog ključa (**FOREIGN KEY**) i slično.

Pri definisanju stranog ključa može se postaviti i referencijalni integritet, čime se u bazu unose značajna ograničenja. Na primer, ne mogu se obrisati podaci iz tabele PROJEKAT dok se ne obrišu podaci iz tabele UCESCE (ukoliko nije definisano kaskadno brisanje, koje treba oprezno koristiti). Još opasnije je uvođenje referencijalnog integriteta između tabela RADNIK i ODELJENJE, između kojih postoje dve veze. Ako se za strane ključeve (brod u tabeli RADNIK i sefod u tabeli ODELJENJE) ne dozvoli upotreba **NULL** vrednosti, nije moguć unos ni jednog podatka u ove tabele, odnosno baza se ne može inicijalizovati.

Primer 1. Kreirati tabelu RADNIK.[14]

Tabela treba da sadrži sledeće atribute:

atribut	tip podatka	dužina
idbr	integer	
ime	char	25
prezime	char	25
posao	char	10
kvalif	char	3
rukovodilac	integer	
datzap	datetime	
premija	float	1
plata	float	1

Potrebno je obezbediti da atributi **idbr** i **ime** moraju biti uneti, tj. da ne mogu imati **NULL** vrednost. Podrazumevana (**DEFAULT**) vrednost za platu je 0.

```
CREATE TABLE RADNIK
        (idbr INTEGER NOT NULL,
        ime CHAR(25) NOT NULL,
        prezime CHAR(25),
        posao CHAR(10),
        kvalif CHAR(3),
        rukovodilac INTEGER,
        datzap DATETIME,
        premija FLOAT(1),
        plata FLOAT(1) DEFAULT 0);
```

Primer 2. Kreirati tabelu ODELJENJE.

Tabela treba da sadrži sledeće atribute:

Atribut	Tip podatka	Dužina
brod	smallint	
imeod	char	15
mesto	char	20
sefod	integer	

[14] Većina primera urađena je u MS SQL Serveru, a primeri koji su urađeni u drugim SUBP (Oracle, MS Access, MySQL) biće posebno naglašeni.

Potrebno je obezbediti da atributi **brod** i **imeod** moraju biti uneti, tj. da ne mogu imati **NULL** vrednost. Postaviti da je atribut brod primarni ključ.

CREATE TABLE ODELJENJE
(brod **SMALLINT NOT NULL PRIMARY KEY**,
imeod **CHAR**(15) **NOT NULL**,
mesto **CHAR**(20),
sefod **INTEGER**);

Napomena: Atribut koji se definiše kao primarni ključ ne može imati **NULL** vrednost, pa bi bilo dovoljno napisati: brod **SMALLINT PRIMARY KEY.**

Primer 3. Kreirati tabelu PROJEKAT.
Tabela treba da sadrži sledeće atribute:

atribut	tip podatka	dužina
brproj	integer	
imeproj	char	25
sredstva	float	2
rok	datetime	

Potrebno je obezbediti da atribut **imeproj** mora biti unet i da ima jedinstvenu vrednost. Postaviti da je atribut **brproj** primarni ključ.

CREATE TABLE PROJEKAT
(brproj **INTEGER PRIMARY KEY**,
imeproj **CHAR**(25) **NOT NULL UNIQUE**,
sredstva **FLOAT**(2),
rok **DATETIME**);

Napomena: Za atribut imeproj koji ima jedinstvenu vrednost, a ne sme da ima **NULL** vrednosti, neophodno je staviti **NOT NULL**, jer u protivnom moguće bi bilo da samo jedan zapis atributa **imeproj** ima vrednost **NULL**.

Primer 4. Kreirati tabelu UCESCE.
Tabela treba da sadrži sledeće atribute:

atribut	tip podatka	dužina
idbr	integer	
brproj	integer	
brsati	integer	
funkcija	char	15

Potrebno je obezbediti da atributi **idbr** i **brproj** moraju biti uneti, tj. da ne mogu imati **NULL** vrednost. Postaviti da je atribut brproj spoljni ključ u odnosu na atribut brproj tabele PROJEKAT.

CREATE TABLE UCESCE
(idbr **INTEGER NOT NULL**,
brproj **INTEGER NOT NULL FOREIGN KEY REFERENCES** PROJEKAT (brproj),
brsati **INTEGER**,
funkcija **CHAR**(15));

Napomena: Ovaj SQL upit se ne bi mogao izvršiti ako prethodno nije kreirana tabela PROJEKAT koja sadrži atribut **brproj** jer se u ovom upitu vrši povezivanje tabela PROJEKAT i UCESCE

preko atributa **brproj**. Pri definisanju atributa **brproj** u tabeli UCESCE voditi računa da ima isti domen kao atribut **brproj** u tabeli PROJEKAT, u kojoj je atribut **brproj** primarni ključ.

Ali jednom kreirana tabela naredbom **CREATE** podložna je izmenama u svojoj strukturi, a to se postiže naknadno korišćenjem naredbe **ALTER TABLE**. Na taj način se neka ograničenja mogu naknadno dodati, kada je baza već inicijalizovana. Moderne verzije RDBMS-a, pored naredbe **CREATE TABLE,** imaju i mogućnost kreiranja relacije bez pisanja ove naredbe korišćenjem unapred pripremljenih opcija, "čarobnjaka" (**WIZARD**) i "alata" (**TOOLS**), ugrađenih u sistem za manipulisanje bazom. Korisnik u tom slučaju mora samo uneti naziv relacije, broj atributa, naziv atributa i tip podatka koji će poprimiti taj atribut, kao i da li atribut može poprimiti vrednost **NULL** ili ne.

Izmena definicije objekata u postojećoj tabeli - ALTER TABLE

Izmene u strukturi tabele moguće je izvršiti korišćenjem naredbe **ALTER TABLE,** tako se u već postojeće tabele mogu dodati novi atributi, izmeniti postojeći, postaviti razna ograničenja, dodati primarni ključevi, izvršiti povezivanje tabela i slično.

Za SUBP MS Access ova komanda ima oblik:

```
ALTER TABLE ime_tabele
{ADD {COLUMN ime_kolone tip[(veličina)] [NOT
NULL]    [CONSTRAINT index] |
ALTER    COLUMN    kolona    tip[(veličina)]    |
CONSTRAINT    indeks_nad_više_kolona    }    |
DROP {COLUMN ime_kolone I
CONSTRAINT ime_indeksa} }
```

gde je:

{ADD {COLUMN ime_kolone tip[(veličina)] [NOT NULL] [CONSTRAINT index] |	ADD COLUMN se koristi u cilju dodavanja nove kolone. Ključna reč CONSTRAINT se koristi kako bi novoj koloni dodelili indeks. Npr. add column JMBG char(13) NOT NULL CONSTRAINT ixJMBG
ALTER COLUMN kolona tip[(veličina)]	Ključne reči ALTER COLUMN se koriste kada je potrebno izmeniti format neke već postojeće kolone. Tako npr. ako smo imali kolonu prezime (20) i naknadno utvrdili da nam treba barem 40 karaktera da sačuvamo prezime tada pišemo ALTER COLUMN prezime char(40)
CONSTRAINT indeks_nad_više_kolona [, ...]])	Ova ključna reč definiše indeks nad više kolona. Tako ako hoćemo da pretražujemo kolone prezime i ime onda pišemo: *CONSTRAINT prezime_ime (prezime, ime)*

DROP {COLUMN ime_kolone I CONSTRAINT ime_indeksa} }	Ključne reči DROP COLUMN znače da u postojećoj tabeli treba ukloniti (izbaciti) kolonu ime_kolone, a DROP CONSTRAINT znače da u postojećoj tabeli treba izbaciti neko ograničenje. Npr. ako želimo da izbacimo kolonu JMBG pišemo: DROP COLUMN JMBG

U SUBP MS SQL ova naredba ima oblik:

```
ALTER TABLE [ ime_baze . [ ime_šeme ] . |
ime_šeme . ] ime_tabele
{
   ALTER COLUMN ime_kolone
   {
      [ ime_šeme_tipa. ] ime_tipa [ ( { preciznost
[ , skala ]
         [opcije_alter]}
   | ADD
   {
      <definicija_kolone>
    | [opcije_add]}
   | DROP
   {
      [opcije_drop]
   } [ ,...n ]
  [check_opcije]
   | { ENABLE | DISABLE } TRIGGER
      { ALL | ime_okidača [ ,...n ] }
  | [ostale_opcije]
}
[ ; ]
```

gde je:

[ime_baze . [ime_šeme] . | ime_šeme .] ime_tabele	Ime_baze je opcioni naziv baze u kojoj se nalazi tabela, ime_šeme je takođe opciono i ime_tabele je naziv tabele čiju definiciju menjamo
ALTER COLUMN ime_kolone { [ime_šeme_tipa.] ime_tipa [({ preciznost [, skala] [opcije_alter]}	ALTER COLUMN ime_kolone znači da se menja definicija kolone sa nazivom ime_kolone i postavlja se da je novi tip podataka ime_tipa koji se može nalaziti u šemi ime_šeme_tipa. opcije_alter sadrže svojstva promenjene kolone (COLLATE, NULL, NOT NULL itd)

ADD { <definicija_kolone> | [opcije_add]}	Ključna reč ADD se koristi da bi se dodala nova kolona unutar postojeće tabele. Pod definicija_kolone podrazumeva se tip i veličina podataka, podrazumevana vrednost, NULL, NOT NULL itd. U delu opcije_add nalaze se definicije kolone ako je ona računska, ograničenja itd.
| DROP { [opcije_drop] } [,...n]	Ključna reč DROP se koristi kako bi se izbacila kolona ili ograničenje iz postojećeg skupa kolona i ograničenja koja postoje u tabeli. U delu opcije_drop se navodi ime kolone ili ime ograničenja
check_opcije	Podrazumevaju proveru ograničenja nakon što se izvrše promene u tabeli
{ ENABLE | DISABLE } TRIGGER { ALL | ime_okidača [,...n] }	Privremeno isključuje (DISABLE) odnosno uključuje (ENABLE) okidač nad tabelom
[ostale_opcije]	Opcije rezervisane za administratora baze kojim se vrši particionisanje fajlova baze podataka koji sadrže tabelu koja se menja

Promena definicije tabele u SUBP MySQL se vrši preko sledeće sintakse

```
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE
ime_tabele
  [alter_specifikacije [, alter_specifikacije] ...]
  [opcije_particionisanja]
```

gde je:

ALTER [ONLINE | OFFLINE] [IGNORE] TABLE ime_tabele	Opcione ključne reči ONLINE ili OFFLINE označavaju da se akcija promene tabele može desiti u normalnom režimu rada baze podataka ili kada je baza otvorena isključivo od strane jednog korisnika
alter_specifikacije	Ovde pripadaju sledeće ključne reči: ADD COLUMN (dodaj kolonu), ADD CONSTRAINT (dodaj ograničenje), ADD INDEX (dodaj index), ALTER COLUMN (promeni DEFAULT vrednost kolone), CHANGE COLUMN (promeni ime koloni), MODIFY COLUMN (promeni definiciju kolone), DROP COLUMN (izbaci kolonu), DROP INDEX (izbaci indeks) itd.
opcije_particionisanja	Način fizičkog organizovanja podataka u tabeli radi pružanja boljih performansi, rezervisano za DB administratora

U SUBP Oracle se promene definicije tabele vrše na sledeći način:

```
ALTER TABLE [ ime_šeme. ] ime_tabele
  [ svojstva_alter_table
  | specifikacije_kolone
  | specifikacije_ograničenja
  | alter_table_particionisanje ]
  [ enable_disable_clause
  | { ENABLE | DISABLE } { TABLE LOCK | ALL
TRIGGERS }
  ] ... ;
```

gde je:

[ime_šeme.] ime_tabele	Opciono ime šeme i obavezno ime tabele čiju definiciju menjamo
svojstva_alter_table	Definisanje kompresije tabele, koji korisnik se prijavljuje za rad sa tabelom, mogući paralelni rad sa tabelom (od strane paralelnih baza), mogućnost promene naziva tabele (RENAME TO), definisanje opcija READ ONLY, READ WRITE itd.
specifikacije_kolone	Ovde se nalaze specifikacije dodavanja (ADD), promene (MODIFY), izbacivanja (DROP) i promene (RENAME) imena kolone. Prilikom dodavanja i promene kolone prisutne su sve ključne reči kao i kod definisanja kolona u naredbi CREATE TABLE
specifikacije_ograničenja	Ovde se nalaze specifikacije dodavanja (ADD), promene (MODIFY), izbacivanja (DROP) i promene (RENAME) imena ograničenja (CONSTRAINT). Prilikom dodavanja i promene ograničenja prisutne su sve ključne reči kao i kod definisanja ograničenja u naredbi CREATE TABLE
alter_table_particionisanje	Opcije rezervisane za administratora baze kojim se vrši particionisanje fajlova baze podataka koji sadrže tabelu koja se menja
enable_disable_clause | { ENABLE | DISABLE } { TABLE LOCK | ALL TRIGGERS }	Privremeno isključuje (DISABLE) odnosno uključuje (ENABLE) ograničenja (PRIMARY KEY, UNIQUE KEY, CONSTRAINT itd) i okidača. Operacije unutar ALTER TABLE komande mogu da se izvode samo u slučaju ako tabela može da se privremeno zaključa (TABLE LOCK)

U nastavku je dato nekoliko opštih primera upotrebe naredbe ALTER TABLE i konkretnih primera prvenstveno za SUBP MS SQL Server (ako se u primeru ne navede za konkretni SUBP).

Dodavanje novog atributa, odnosno nove kolone u tabelu:
ALTER TABLE ime_tabele
ADD (atrib tip [, atrib tip]);

Izmena definicije postojećeg atributa, tj. postojeće kolone u tabeli u SUBP Oracle i MySQL vrši se naredbom:

ALTER TABLE ime_tabele
MODIFY (atrib modifikacija [, atrib modifikacija]);

Dodavanje ograničenja

- *provera (**CHECK**),*
- *jedinstven (**UNIQUE**),*
- *primarni ključ (**PRIMARY KEY**),*
- *spoljnji, strani ključ (**FOREIGN KEY**):*

ALTER TABLE ime_tabele
ADD CONSTRAINT ime_ograničenja
CHECK uslov;

ALTER TABLE ime_tabele
ADD CONSTRAINT ime_ograničenja
UNIQUE (ime_atributa[, ime_atributa]);

ALTER TABLE ime_tabele
ADD CONSTRAINT ime_ograničenja
PRIMARY KEY (ime_atributa[, ime_atributa]);

ALTER TABLE ime_tabele
ADD CONSTRAINT ime_ograničenja
FOREIGN KEY (ime_atributa[, ime_atributa])
REFERENCES ime_referentne_tabele (ime_atributa[, ime_atributa]);

Primer 5. U tabelu ODELJENJE dodati atribut *brzap* (broj zaposlenih), u uglastoj zagradi navesti kako se to piše za SUBP Oracle:

ALTER TABLE ODELJENJE
ADD (brzap INTEGER) [**ADD** (brzap NUMBER(8))] ;

Primer 6. Dodati atribut brod u tabelu RADNIK, u uglastoj zagradi navesti kako se to piše za SUBP Oracle:

ALTER TABLE RADNIK
ADD brod SMALLINT [**ADD** (brod NUMBER(5))];

Primer 7. Promeniti tip polja brsati u tabeli UCESCE na **SMALLINT**, za SUBP MySQL navesti kako se to piše u uglastoj zagradi.

ALTER TABLE UCESCE
ALTER COLUMN brsati SMALLINT [MODIFY brsati SMALLINT];

Primer 8. Dodati ograničenje **NOT NULL** nad atributom prezime u tabeli RADNIK.

ALTER TABLE RADNIK
ALTER COLUMN prezime char(25) **NOT NULL**;

Primer 9. Dodati ograničenje nad atributom rok u tabeli projekat da datum mora biti veći od trenutnog datuma, u uglastoj zagradi navesti kako se to piše u SUBP Oracle

```
ALTER TABLE PROJEKAT
ADD CONSTRAINT ck_rok_projekta
CHECK (rok>GETDATE ()) [CHECK (rok> CURRENT_DATE)] ;
```

Napomena: Funkcija koja vraća trenutni datum u MS SQL Serveru je **GETDATE**(), a u ORACLE-u i MySQL **CURRENT_DATE**.

Primer 10. Ograničiti unos za polje **kvalif** u tabeli RADNIK tako da mogu biti unete samo vrednosti VKV, KV ili VSS.

```
ALTER TABLE RADNIK
ADD  CONSTRAINT ck_kvalif
CHECK (kvalif in ('VKV', 'KV', 'VSS'));
```

Primer 11. U tabeli ODELJENJE dodati UNIQUE ograničenje nad imenom odeljenja.

```
ALTER TABLE ODELJENJE
ADD CONSTRAINT uk_odeljenje_imeod
UNIQUE (imeod);
```

Primer 12. Dodati primarni ključ nad atributom **idbr** u tabeli RADNIK.

```
ALTER TABLE RADNIK
ADD CONSTRAINT PRIMARY KEY pk_radnik
PRIMARY KEY (idbr);
```

Primer 13. Dodati primarni ključ u tabeli UCESCE.

Tabela UČEŠĆE ima složeni primarni ključ, tj. sastoji se od dva atributa **idbr** i **brproj**.

```
ALTER TABLE UCESCE
ADD CONSTRAINT pk_ucesce
PRIMARY KEY (idbr, brproj);
```

Primer 14. Kreirati vezu između tabela RADNIK i ODELJENJE postavljanjem ograničenja stranog ključa.

U tabeli RADNIK treba postaviti strani ključ nad atributom **brod** koji je povezan sa atributom **brod** koji je primarni ključ u tabeli ODELJENJE.

U tabeli ODELJENJE treba postaviti strani ključ nad atributom **sefod** koji je povezan sa atributom **idbr** koji je primarni ključ u tabeli RADNIK.

```
ALTER TABLE RADNIK
ADD CONSTRAINT fk_radnik_odeljenje
FOREIGN KEY (brod)
REFERENCES ODELJENJE (brod);

ALTER TABLE ODELJENJE
ADD CONSTRAINT fk_odeljenje_radnik
FOREIGN KEY (sefod)
REFERENCES RADNIK (idbr);
```

Primer 15. Kreirati vezu između tabela RADNIK I UCESCE.

```
ALTER TABLE UCESCE
ADD CONSTRAINT fk_ucesce_radnik
FOREIGN KEY (idbr)
REFERENCES RADNIK (idbr);
```

Primer 16. Kreirati unarnu vezu između polja **rukovodilac** i **idbr** u tabeli RADNIK.

```
ALTER TABLE RADNIK
ADD CONSTRAINT fk_radnik_radnik
FOREIGN KEY (rukovodilac)
REFERENCES RADNIK (idbr);
```

Izbacivanje objekata iz baze podataka - DROP

Objekti baze podataka su tabele, atributi (kolone), indeksi, kao i razna druga ograničenja. Izbacivanje definicije objekata i objekata iz baze vrši se naredbom **DROP**. Ovom naredbom se izbacuje ne samo definicija tabele, već i svi njeni indeksi i podaci koje ona sadrži, za razliku od naredbe **DELETE,** koja može obrisati sve podatke iz tabele, ali sama tabela ostaje u bazi podataka. Izbacivanje definicije tabele iz baze podataka vrši se naredbom čiji je opšti oblik:

DROP TABLE ime_tabele

Neki sistemi za upravljanje bazama podataka podržavaju i naredbu **DROP DATABASE** *ime_baze*, kojom se izbacuje cela baza. Sistemi orijentisani na jedan fajl, kao što je Microsoft Access, ne podržavaju ovu komandu. To se radi jednostavnim brisanjem fajla (baze podataka) sa diska naredbom za brisanje (Delete), iz operativnog sistema.

Primer 17. Iz tabele ODELJENJE izbaciti atribut **brzap**.

```
ALTER TABLE ODELJENJE
DROP COLUMN brzap;
```

Primer 18. Ukinuti ograničenje ck_rok_projekat nad atributom rok u tabeli PROJEKAT.

```
ALTER TABLE PROJEKAT
DROP CONSTRAINT ck_rok_projekta;
```

Rad sa indeksima

Indeks je objekat šeme, definisan nad jednom ili više kolona neke tabele. Indeks sadrži po jednu stavku za svaku različitu vrednost indeksirane kolone (ili kolona) u tabeli. U svakoj stavci indeksa pamti se i fizička adresa slogova koji imaju datu vrednost u indeksiranoj koloni (kolonama). Podaci u indeksu (stavke) čuvaju se u obliku balansiranog binarnog stabla. Samim tim, pristup svakoj stavci indeksa je vrlo brz. Indeksi se koriste za brzi pristup po kolonama koje se indeksiraju i omogućavaju jedinstvenu vrednost indeksiranih kolona kada te kolone imaju ulogu primarnog ključa, posebno u sistemima kod kojih se naredbom **CREATE** ne mogu definisati primarni ključevi. Ali, u tabelama mogu postojati jedinstveni indeksi koji nisu primarni ključevi (kada postoji više kandidata za primarni ključ).

Postojanje indeksa usporava sve naredbe za ažuriranje podataka koje menjaju, brišu ili dodaju nove vrednosti u indeksirane kolone. Posle svake izmene podataka SUBP mora da ažurira i indeks nad kolonama koje su izmenjene. Dakle, indeksi ubrzavaju pristup slogovima, a usporavaju naredbe **INSERT**, **UPDATE** i **DELETE**. Indeksi su objekti fizičkog nivoa baze podataka i njihovo postojanje ne utiče na ispravnost rada programa, već samo na performanse. Kreiranje jednog ili više indeksa (ne preporučuje se više od 3) predstavlja kompromis između ubrzavanja pristupa i usporavanja ažuriranja. Svaki od SUBP ima svoju sintaksu za kreiranje indeksa, način filtriranja slogova koji se indeksiraju, fizičku organizaciju indeksa, restrikcije, kompresije i druge osobine. U opštem obliku, a dovoljno za ovaj kurs, indeksi se kreiraju naredbom:

CREATE [UNIQUE] INDEX naziv_indeksa
ON ime_tabele (atr [, atrib])

Primer 19. U tabeli ODELJENJE kreirati indeks nad artibutom **imeod**.

CREATE INDEX ind_imeod
ON ODELJENJE (imeod);

Primer 20. U tabeli ODELJENJE kreirati primarni ključ jedinstveni indeks nad artibutom **brod**.

CREATE UNIQUE INDEX ind_brod
ON ODELJENJE (brod);

Indeksi se izbacuju naredbom **DROP INDEX** naziv_indeksa.

Primer 21. U tabeli ODELJENJE ukloniti indeks **ind_imeod** nad atributom **imeod**.

DROP INDEX ODELJENJE.ind_imeod;

Napomena: Potrebno je navesti i ime tabele na koju se indeks odnosi, pa zatim ime indeksa.

Podaci se iz baza podataka mogu dobiti na dva načina. Prvi, sekvencijalni metod (**Sequential Access Metod**), zahteva da se pročitaju svi slogovi jedne tabele prilikom pretraživanja. Dakle, čitava datoteka od prvog do zadnjeg zapisa. Ovaj metod je neefikasan, ali je jedini mogući da biste bili sigurni u tačnost dobijenog odgovora. Drugi, direktni metod (**Direct Access Metod**), zahteva da podaci budu indeksirani po određenom atributu i u tom slučaju moguće je direktno pristupiti podatku bez pretraživanja čitave tabele. U tu svrhu se kreira jedna struktura nalik na izokrenuto drvo, takozvano binarno stablo. Na vrhu stabla (u korenu) nalazi se pokazivač na grupe podataka - *čvorove* (**nodes**). Svaki čvor – roditelj (**parent**) sadrži najviše dva pokazivača na druge čvorove – decu. Levo se nalaze čvorovi koji imaju vrednost manju od čvora roditelja, a desno su čvorovi sa vrednošću koja je veća od čvora roditelja.

"Jedna od osnovnih operacija pri manipulisanju velikim brojem podataka je pomenuto **pretraživanje,** to jest dobijanje neke informacije na osnovu određenog broja prethodno poznatih vrednosti nekih atributa. Međutim, ako je broj podataka veliki (stotine miliona, na primer), onda pretraživanje nije ni najmanje jednostavan i kratak postupak i za najbrže savremene računare. Na primer, naći neki podatak o građaninu ***XY*** u bazi podataka zemlje sa nekoliko stotina miliona stanovnika mora da potraje, jer računar mora da "pročešlja" bazu od početka do kraja. A ukoliko je upit logički složen, vreme za dobijanje odgovora postaje nedopustivo dugo."[15]

Iz toga razloga se, pri pretraživanju velikih baza podataka, pribegava njihovom **indeksiranju** po atributu (ili atributima) po kome se vrši pretraživanje. Sve tabele su, po pravilu, indeksirane po primarnom ključu i tada **Indeksna tabela** (uvek je izvedena od osnovne), ima samo dva atributa koji definišu drugačije, po nekoj zakonitosti, složene slogove ili n-torke (poređane po primarnom ključu). Inače, svaka indeksna tabela ima dva dela:

- prvi deo-lista atributa, po kojima se vrši pretraživanje (i po kojima se vrši uređivanje indeksa) i
- drugi deo, tzv. **indeks**, koji služi za vezu sa osnovnom tabelom.

[15] [27] Kaluđerčić, P., Obradović, S.. "Projektovanje *informacionih sistema - Relacione baza podataka",* Visoka škola elektrotehnike i računarstva, Beograd, 2008.

Pokažimo postupak kreiranja indeksa nad jednim atributom, koji nije primarni ključ, na jednom jednostavnom primeru. Pretpostavimo da tabela GRAĐANIN, ima oblik:

GRAĐANIN < matbr#, prezime, ime, dat. rođ, adresa, .. >

a jedan njen deo se vidi u sledećoj tabeli.

INDGRAD

prez	index
Antić	1
Babić	4
Jović	2
Ljujić	6
Marić	3
Perić	7
Rodić	5

GRAĐANIN

Redbr	matbr#	prez	ime	datrođ	adresa
1	1324764	Antić	Ante	10.07.54	Beograd
2	9763421	Jović	Jovan	212.33	Valjevo
3	4513298	Marić	Maks	13.03.76	Bor
4	3344228	Babić	Miro	02.02.77	Užice
5	3524999	Rodić	Ana	05.10.84	Zemun
6	7623087	Ljujić	Vera	23.11.49	Beograd
7	6653129	Perić	Petar	17.03.11	Trebinje

Tabela GRAĐANIN i njoj pridruženi indeks INGRAD

Redni broj zapisa - *Redbr* (**Record number**) vodi se u većini programskih paketa za obradu baza podataka za svaku tabelu - relaciju automatski inkrementiranjem nekog brojača (ili interno), i taj broj se najčešće upotrebljava za kreiranje indeksne tabele (nazovimo je INDGRAD) po nekom atributu koji nije ključni, na primer **prezime**.

U indeksnoj tabeli INDGRAD su podaci (n-torke) složeni drugim redom (u ovom slučaju po abecedi, po prezimenima, ali se to može uraditi i po nekoj drugoj promenljivoj po veličini ili po datumu) i pored vrednosti atributa **prezime** indeksirana tabela ima samo još i vrednost indeksa, to jest broja zapisa u tabeli iz koje je izvedena – dakle GRAĐANIN**.** Nalaženje građanina poznatog prezimena izvodi se sada u dva koraka.

Pošto su u indeksiranoj tabeli podaci (prezimena) složeni po poznatom zakonu (abeceda), to se traženi podatak prvo nalazi u njoj u nekoliko koraka (pretraživanjem binarnog stabla pošto se znaju pravila slaganja po abecedi, postupak je isti kao pri traženju reči u rečniku ili broja u telefonskom imeniku) bez potrebe da se pretražuje cela tabela, a onda uz pomoć vrednosti indeksa-rednog broja zapisa, dolazimo opet u samo jednom koraku (direktnim pristupom) i do drugih podataka toga građanina u osnovnoj tabeli GRAĐANIN**.**

Na primer, traže se podaci o građaninu koji se preziva Babić. Pristupi se stredini tabele indeksa. Kako ima 7 redova sredina je na (7+1)/2 tj,. red 4. Tu piše Ljujić. Kako je Lj veće od B, traženi podatak je u donjoj polovini. (Odmah smo eliminisali pola tabele. Da je bilo 100000 zapisa otpalo bi 50000). Sada pristupimo sredini donje polovine (3+1)/2 tj,. red 2. Pročitamo podatak, Babić i sada saznajemo da je to 4 zapis u tabeli građanin i direktno ga uzimamo iz tabele (tačnije iz RAM memorije ili sa diska računara).

Da nije bilo tabele sa indeksima, pristupali bismo rer po red tabeli građanin. Posle 4 pokušaja našli bi traženi zapis. Ako tog prezimena nema morali bi da pročitamo sve zapise iz tabele Građanin da bi utvrdili da ga nema.

Pravljenje indeksnih tabela (od jedne osnovne tabele može se napraviti više indeksnih tabela po raznim atributima, ili po više atributa) podleže nekim pravilima. Međutim, uvek mora biti zadovoljen uslov da postoji kriterijum po kome se vrednosti u indeksnoj tabeli mogu poređati.

Svaka tabela treba da ima bar jedan indeks i njega pravi mašina baze podataka (BP) automatski za primarni ključ (PK). Po pravilu se indeks pravi za svako polje (polja) koja će se koristiti za spajanje sa drugim tabelama (po pravilu za spoljne ključeve-FK).

Ako polja spoljnog ključa učestvuju u primarnom ključu ali to nije ceo ključ, onda se polje spoljnog ključa definiše poseban indeks.

Sve operacije i odredbe jezika SQL-a prikazaćemo na primerima realizovanim na već isprojektovanoj i kreiranoj bazi podataka jednog preduzeća koju čine tabele:

RADNIK <idbr#, ime, prezime, posao, kvalif, rukovodilac$, datzap, premija, plata, brod$>
ODELJENJE <brod#, imeod, mesto, sefod$>
PROJEKAT < BrProj#, imeproj, sredstva, rok>
UČEŠĆE <idbr#, BrProj#, brsati, funkcija>[16]

Napomena: Da bismo istakli atribute koji predstavljaju ključeve, njima su na kraju dodati specijalni znaci # i $ koje inače nećemo koristiti u samoj bazi podataka. Njihova upotreba se ne preporučuje.

Na primer: Tabela UCESCE ima složeni primarni ključ *BrPoj, Idbr*, treba dodatno napraviti indeks nad poljem *idbr* zbog spajanja sa glavnom tabelom RADNIK.

Treba indeksirati i sva polja po kojim se vrši sortiranje da bi bilo efikasnije i lakše. Sortiranje se, doduše, može obaviti i sa ORDER BY.

Napomena: Održavanje indeksa je dodatni posao koji potreban samo kada se doda zapis ili se ažurira sadržaj indeksiranog polja. Broj indeksa po tabeli zavisi od toga koliko se tabela često ažurira. Tabela koja se često ažurira ne bi trebala da ima više od 10 do 15 indeksa, uključujući primarni ključ i one koji služe za spajanje sa drugim tabelama.

Na primer, upotreba indeksa je poželjna u tabeli PROJEKAT (retko se ažurira) a nije za **Ucesce** (ako se ona ažurira cesto).

Naravno, nisu svi atributi dobri kandidati za indeks. Nije moguće praviti indekse nad kolonama tipa bit-map, tekst ili slika, a kako je veličina indeksa ograničena, nisu pogodne (ni dozvoljene) velike kolone tipa CHAR, VARCHAR, BINARY i sl. Za indeks su kandidati kolone po kojima se najčešće vrši pretraživanje, grupisanje, sortiranje i selekcija, a po pravilu su to: spoljni (strani) ključevi i kolone koje učestvuju u odredbama GROUP BY i ORDER BY (koje će kasnije biti detaljno objašnjene).

Na kraju, spomenimo još jednom, da indeksiranje ima i svoje nedostatke. Naime, za pretraživanje, zbog čega se ovakve tabele prvenstveno i prave, indeksirane datoteke su izuzetno efikasne, ali se zato prilikom ažuriranja (dodavanje i brisanje podataka) osnovne tabele gubi računarsko vreme, jer se indeksne datoteke, svaki put posle izmene osnovne tabele, moraju ažurirati i balansirati (balansiranje binarnog stabla). To je i osnovni razlog zašto se tabele sa malim brojem podataka – zapisa, ne indeksiraju. Sa brzim računarom, i neindeksirana manja tabela može se u veoma kratkom vremenu pretražiti od početka do kraja.

Pri pristupu podacima indeks se koristi na osnovu uslova selekcije. Ako je selektivnost indeksa loša, pristup preko indeksa je sporiji čak i od pretrage čitave tabele. Redosled kolona u indeksu takođe utiče na mogućnost upotrebe indeksa i brzinu pretrage stabla. Kada se kreira indeks nad više kolona, na prvom mestu treba da se navede kolona koja ima najveću selektivnost, na drugom mestu treba da je kolona sa sledećom manjom selektivnošću. U tom slučaju, dobija se najveća brzina pretrage indeksa. Ovo je posebno bitno kod tabela sa složenim ključevima. Tada je redosled kolona u ključu vrlo bitan. U tabeli UCESCE bolje je

[16] Da bi istakli atribute koji predstavljaju ključeve njima su na kraju dodati specijalni znaci # i $ koje inače nećemo koristiti u samoj bazi podataka. Njihova upotreba se ne preporučuje jer se položaj ovih znakova razlikuje na raličitim tastaturama, odnosno zavisi od izbora regionalnih podešavanja.

da na prvom mestu bude *idbr* nego *brproj* jer je broj različitih vrednosti u koloni *idbr* mnogo veći pa će selektivnost biti veća.

Redosled uslova za pojedine kolone u odredbi WHERE nema uticaja na korišćenje indeksa. Dakle, indeks se upotrebljava na isti način nezavisno od redosleda iskaza u uslovu za selekciju. Pri pretrazi stabla indeksa uvek se prvo upoređuje vrednost prve kolone. Ako se ova ne slaže, ne gledaju se vrednosti drugih kolona. Zbog toga je važno da prva kolona bude najselektivnija.

Opravdanost upotrebe indeksa zavisi od namene sistema. Ako je sistem prvenstveno namenjen za tekuću obradu transakcija (**OLTP, On-Line Transaction Processing**), sve promene se registruju u bazi u trenutku njihovog nastajanja. Pri tome se u toku jedne transakcije menja mali broj slogova, pa samim tim i stavki indeksa (jedna ili dve, retko desetak i više). To znači da će vreme koje je potrebno da se ažuriraju indeksi takođe biti vrlo malo.

Ali ako se tabele popunjavaju ili ažuriraju u paketima, kao na primer u **OLAP** sistemima (**On-Line Analitical Processing**) broj slogova koji se menjaju je vrlo veliki (nekoliko stotina slogova). Tada treba vrlo racionalno kreirati indekse da vreme održavanja tabele indeksa i obrade ne bi bilo preterano dugo.

Pri masovnim operacijama ažuriranja bolje je da ne postoje indeksi definisani nad tim tabelama. Indekse je onda bolje kreirati posle punjenja i to nakon što se tabele sortiraju. U tim slučajevima naknadno kreiranje indeksa je mnogo brže od njihovog ažuriranja (zbog balansiranja stabla).

Pored indeksiranja i ciljanim grupisanjem podataka na fizičkom medijumu (disku), efikasnost i brzina rada mogu se bitno povećati. Rešenje se sastoji u tome da se slogovi koji se najčešće uzastopno koriste smeste na disk na iste, ili susedne segmente. Tako, ako imamo slogove **S1** i **S2** smeštene na isti segment diska **D1,** onda će se pri dohvatu **S1** jednovremeno u međumemoriji računara naći i podaci sloga **S2,** pa ako nam i oni odmah nakon što smo iskoristili podatke sa **S1** zatrebaju, oni su već tu, "pri ruci", u operativnom delu memorije i pristup njima je zato brz.

Ponekad se svi zapisi na disku smeštaju ne po redosledu unosa, već uređeni po nekom redosled u. To su takozvane sortirane tabele i imaju najbrži pristup podacima, ali se one posle svakog ažuriranja moraju iznova sortirati. Dakle, kao i kod upotrebe indeksa produžava se vreme održavanja baze.

Naredbe za rukovanje podacima

Naredba **SELECT** služi za "pribavljanje" jednog ili više podataka iz jedne, ili iz više tabela. Rezultat ove naredbe je neka informacija koja opet ima najčešće strukturu relacije ili bar tabele, pa tako ima svoje atribute i vrednosti atributa, ali ne postoji kao fizička tabela stalno prisutna na disku. Dakle, rezultat ovakvog upita je virtuelna neimenovana tabela koja postoji samo u radnoj, operativnoj memoriji.

Međutim, i pored toga što ne postoji fizički na disku u formi tabele, rezultat naredbe SELECT može se koristiti kao argument neke druge naredbe SELECT (koja je sastavni deo prve), prilikom pravljenja složenih upita, jer za sve vreme izvršavanja naredbe SELECT parcijalni rezultati postoje kao tabele u memoriji računara. Rezultat upita može biti prost neizmenjeni sadržaj jedne ili više tabela, ali isto tako može biti i neka nova vrednost koja je izračunata na bazi podataka koji postoje u bazi. Prava snaga koncepta baza podataka i SQL-a upravo leži u tome da možemo dobijati i nove, izračunate informacije koje ne postoje kao zapis (podatak) u bazi. Analizom tih novih podataka u interaktivnom radu sa bazom na licu mesta donosimo odluke i kreiramo nove upite sa ciljem dobijanja novih informacija. Rezultat upita je najčešće tabela, relacija (složeni SELECT), a ne samo jedan podatak ili slog (prosti SELECT) ([25], [27], [30]).

U svim daljim razmatranjima pretpostavićemo da je naredba SELECT tipa složeni SELECT**,** da se koristi u interaktivnom načinu rada, pa će se u primerima koji slede naći i takvi gde se jednovremeno pristupa podacima iz više tabela. Opšti oblik naredbe SELECT, odnosno upitnog bloka glasi:

SELECT [ALL DISTINCT]] lista atributa1 (ili lista izraza)
FROM lista tabela (relacija)
[**WHERE** lista uslova1]
[**GROUP BY** lista atributa 2]
[**HAVING** lista uslova 2]
[**ORDER BY** lista atributa 3]
[**UNION, MINUS**, **INTERSECT**, **EXIST**, **ANY**, **ALL**]

posle koje može da sledi i naredna SELECT naredba u slučaju složenog upita nad jednom ili više relacija. U naredbi SELECT se:

- definišu-selektuju atributi (SELECT) čije vrednosti želimo dobiti (odgovara **operatoru projekcije)**;
- zatim se izdvajaju relacije u kojima se nalaze vrednosti tih atributa (**FROM**) (odgovara **operatoru spajanja**);
- onda se definišu uslovi koje podaci treba da zadovoljavaju pri izdvajanju (**WHERE**), što odgovara **operatoru restrikcije** (**selekcije**);
- na kraju, mogu se postaviti i zahtevi kojima se rezultati grupišu (**GROUP BY**) po nekim atributima;
- onda se definišu uslovi koje grupe treba da zadovoljavaju (**HAVING**);
- ili na neki način uređuju (**ORDER BY**).

Prema tome, upitni blok predstavlja kompoziciju operacija projekcije, restrikcije i spajanja, ali njime nije određen redosled kojim se te operacije obavljaju. Zbog toga je upitni blok opštija, manje proceduralna konstrukcija od ovih operacija relacione algebre.

Odredbe SELECT i FROM su obavezne. Ako se ne postavi nikakav uslov za selekciju, grupisanje ili uređivanje, druge odredbe (WHERE, GROUP BY, HAVING, ...) jednostavno se izostavljaju.

Koristeći naredbu SELECT**,** moguće je :

- izdvojiti neke atribute;
- izmeniti redosled atributa. Redosled atributa u odgovoru isti je kao redosled atributa koji je naveden u naredbi **SELECT** i ne mora biti isti kao redosled atributa dat u definiciji tabele;
- sprečiti pojavu višestrukih n-torki.

Značenje opcije **DISTINCT** je sledeće:

- ako ova odredba nije navedena, upit prikazuje (vraća) sve podatke, to jest sve n-torke, koje ispunjavaju postavljeni uslov, tako da se u tom slučaju u rezultatu mogu pojaviti i identične n-torke, pa rezultat onda očito ne mora biti i relacija;
- DISTINCT eliminiše sve višestruke n-torke, u rezultatu se nalaze samo one koje su različite pa je rezultat prema tome opet relacija.

Ako se ne navede nijedan parametar, podrazumeva se ALL**,** ali ima verzija SQL-a koje u tom slučaju podrazumevaju i DISTINCT i na to treba obratiti pažnju, jer u suprotnom dobijeni rezultat možda neće biti relacija, što može izazvati greške u daljoj obradi.

Sve operacije i odredbe SQL-a prikazaćemo na primerima realizovanim na već kreiranoj bazi podataka jednog preduzeća:

RADNIK <*idbr#, ime, prezime, posao, kvalif, rukovodilac$, datzap, premija, plata, brod$*>

ODELJENJE <*brod#, imeod, mesto, sefod$*>

PROJEKAT <*brproj#, imeproj, sredstva, rok*>

UČEŠĆE <*idbr#, brproj#, brsati, funkcija*>.

Slika 3 prikazuje vrednosti koje treba uneti u tabele tako da one budu relacije i da imaju odgovarajući sadržaj pogodan za kreiranje upita. Tabele nisu urađene u potpunosti po pravilima normalizacije, ali to je učinjeno u nameri da se prikažu određene negativne pojave, tj. anomalije koje nastaju pri upisu, brisanju i ažuriranju podataka, a takođe i da bi bilo moguće na jednoj bazi prikazati što više naredbi i mogućnosti SQL-a.

Upiti nad jednom tabelom za prikaz neizmenjenog sadržaja tabele

Najjednostavniju grupu upita čine oni koji prikazuju prost, neizmenjen sadržaj jedne tabele.

Projekcija, izdvajanje pojedinih atributa

Projekcija je operacija kojom se iz skupa atributa izdvajaju samo oni koji su od interesa.

Primer 22. Prikaži celokupan sadržaj tabele ODELJENJE.

```
SELECT *
FROM ODELJENJE;
```

Rezultat će biti čitava bazna tabela (Slika 3 - tabela ODELJENJE) jer znak ***** je sinonim za traženje svih atributa, a kod nekih RDBMS (na primer MS Access), kao i u Windowsu i DOS-u, zamenjuje bilo koji niz znakova (džoker znak). Stavljanjem znaka * redosled atributa u rezultujućoj tabeli isti je kao u definiciji tabele.

Redosled atributa (kolona) i njihovi nazivi u odgovoru na upit ne moraju biti uvek isti i jednaki redosledu i imenima u definiciji tabele. Naime, redosled kolona u rezultatu upita jednak je redosledu njihovog navođenja u SELECT odredbi, a imena su ista kao ona koja se u njoj navedu.

U sledećem primeru promenjen je redosled kolona, a imena su ista kao u definiciji tabele. To se postiže navođenjem imena onih atributa u redosledu koji se želi u rezultatu:

SELECT ODELJENJE.imeod, ODELJENJE.mesto, ODELJENJE.adresa, ODELJENJE.brod
FROM ODELJENJE;

ODELJENJE : Table

BROD	IMEOD	MESTO	SEFOD
10	Komercijala	Novi Beograd	5662
20	Plan	Dorćol	5780
30	Prodaja	Stari Grad	5786
40	Direkcija	Banovo Brdo	5842
50	Računski centar	Zemun	

Record: 1 of 5

PROJEKAT : Table

BRPROJ	IMEPROJ	SREDSTVA	ROK
100	uvoz	3000000	5.5.2004
200	izvoz	2000000	22.8.2005
300	plasman	6000000	2.12.2004
400	projektovanje	5000000	14.4.2005
500	izgradnja	0	22.8.2005

Record: 1 of 5

RADNIK : Table

IDBR	IME	PREZIME	POSAO	KVALIF	RUKOVODILAC	DATZAP	PREMIJA	PLATA	BROD
5367	Petar	Vasić	vozač	KV	5780	1.1.1978	1900	1300	20
5497	Aleksandar	Marić	električar	KV	5662	17.2.1990	800	1000	10
5519	Vanja	Kondić	prodavac	VKV	5662	7.11.1991	1300	1200	10
5652	Jovan	Perić	električar	KV	5662	31.5.1980	500	1000	10
5662	Janko	Mančić	upravnik	VSS	6789	12.8.1993		2400	10
5696	Mirjana	Dimić	čistač	KV	5662	30.9.1991	0	1000	10
5780	Božidar	Ristić	upravnik	VSS	6789	11.8.1984		2200	20
5786	Pavle	Šotra	upravnik	VSS	6789	22.5.1983		2800	30
5842	Miloš	Marković	direktor	VSS		15.12.1981		3000	40
5867	Svetlana	Grubač	savetnik	VSS	5842	8.8.1970		2750	40
5874	Tomislav	Bogovac	električar	KV	5662	19.4.1971	1100	1000	10
5898	Andrija	Ristić	nabavljač	KV	5786	20.1.1980	1200	1100	30
5900	Slobodan	Petrović	vozač	KV	5780	3.10.2002	1300	900	20
5932	Mitar	Vuković	savetnik	VSS	5842	25.3.2000		2600	20
5953	Jovan	Perić	nabavljač	KV	5786	12.1.1979	0	1100	30
6234	Marko	Nastić	analitičar	VSS	5867	17.12.1990	3000	1300	30
6789	Janko	Simić	upravnik	VSS	5842	23.12.2003	10	3900	40
7890	Ivan	Buha	analitičar	VSS	5867	17.12.2003	3200	1600	20
7892	Luka	Bošković	analitičar	VSS	5867	20.5.2004		2000	

Record: 1 of 19

UCESCE : Table

IDBR	BRPROJ	BRSATI	FUNKCIJA
5497	400	2000	IZVRŠILAC
5652	100	1000	IZVRŠILAC
5652	300	1000	IZVRŠILAC
5662	300	2000	ŠEF
5696	200	2000	ŠEF
5696	300	2000	IZVRŠILAC
5780	200	2000	ORGANIZATOR
5786	100	2000	KONSULTANT
5842	100	2000	ŠEF
5867	200	2000	KONSULTANT
5898	200	2000	IZVRŠILAC
5900	100	2000	IZVRŠILAC
5932	100	500	KONSULTANT
5932	200	1000	ORGANIZATOR
5932	300	500	NADZORNIK
5953	100	1000	IZVRŠILAC
5953	300	1000	IZVRŠILAC
6234	100	500	NADZORNIK
6234	200	1200	IZVRŠILAC
6234	300	300	KONSULTANT
6789	200	2000	IZVRŠILAC
7890	300	2000	IZVRŠILAC

Record: 1 of 22

Slika 2. *Vrednosti podataka za pojedine tabele*

U ovom slučaju, kao i u svim slučajevima kada je u pitanju upit nad jednom tabelom u SELECT naredbi nije neophodno navoditi ime tabele pre imena atributa. Isto važi i kada u raznim tabelama nad kojima se primenjuje upit ne postoje kolone sa istim imenima. Uzimajući u obzir i to pravilo, ovaj zadatak bi bio urađen na sledeći način:

SELECT imeod, mesto, adresa, brod
FROM ODELJENJE;

Ako se želi da kolone u rezultatu budu prikazane u redosledu u kojem su date u definiciji tabele, najbolje je koristiti džoker znak *:

SELECT ODELJENJE.*
FROM ODELJENJE;

Primer 23. Prikazati sve podatke iz tabele RADNIK.
SELECT RADNIK.*
FROM RADNIK;

Primer 24. Prikazati identifikacione brojeve i imena svih zaposlenih.

SELECT idbr, ime
FROM RADNIK;

Kao rezultat ovog upita dobiju se dve kolone koje prikazuju *idbr* i *ime* zaposlenih, prikazane u polaznoj tabeli.

Ovaj zadatak urađen dodavanjem imena tabele pre imena atributa koji treba da se prikaže, glasio bi:

```
SELECT RADNIK.idbr, RADNIK.ime
FROM RADNIK;
```

U ovom primeru ne možemo koristi znak * jer se u zadatku ne traži da budu prikazani svi atributi (sve kolone) tabele RADNIK.

Id. br	Ime
5367	Petar
5497	Aleksandar
5519	Vanja
5652	Jovan
5662	Janko
5696	Mirjana
5780	Božidar
5786	Pavle
5842	Miloš
5867	Svetlana
5874	Tomislav
5898	Andrija
5900	Slobodan
5932	Mitar
5953	Jovan
6234	Marko
6789	Janko
7890	Ivan
7892	Luka

Primer 25. Prikazati broj (šifru), ime i mesto svih odeljenja.

```
SELECT brod, imeod, mesto
FROM ODELJENJE;
```

Prilikom navođenja imena atributa u upitu, neophodno je voditi računa o tome koje je stvarno ime tih atributa u tabeli (na primer, atribut ime odeljenja u tabeli ODELJENJE je *imeod*, a ne *ime*, *ime_odeljenja* ili slično).

Primer 26. Prikazati identifikacione brojeve, imena, datum zaposlenja i broj odeljanja za sve zaposlene.

```
SELECT idbr, ime, datzap, brod
FROM RADNIK;
```

Primer 27. Prikaži nazive svih odeljenja u preduzeću.

```
SELECT imeod
FROM ODELJENJE;
```

Odgovor sistema predstavlja sledeća tabela:

Ime od
Komercijala
Plan
Prodaja
Direkcija
Računski centar

Primer 28. Prikaži nazive svih poslova u preduzeću.

```
SELECT posao
FROM RADNIK;
```

Odgovor na upit ne mora biti relacija, odnosno mogu se pojaviti višestruki slogovi. U odgovoru na prethodni upit svaki, od poslova se pojavljuje onoliko puta koliko ima zaposlenih koji obavljaju dati posao.

Posao
vozač
električar
prodavac
električar
upravnik
čistač
upravnik
upravnik
direktor
savetnik
električar
nabavljač
vozač
savetnik
nabavljač
analitičar
upravnik
analitičar
analitičar

Primer 29. Prikaži nazive svih različitih poslova u preduzeću.

```
SELECT DISTINCT posao
FROM RADNIK;
```

Kao odgovor na ovaj upit dobijeni su samo različiti poslovi, znači svi poslovi u preduzeću bez ponavljanja, što je omogućeno upotrebom ključne reči DISTINCT koja vraća samo različite zapise, bez ponavljanja.

Posao
analitičar
čistač
direktor
električar
nabavljač
prodavac
savetnik
upravnik
upravnik
vozač

Isti efekat se kod nekih SUBP postiže odredbom **UNIQUE** (jedinstven, radi samo u Oracle DBMS), ali ovo ne važi kod svih. Recimo, u Accessu nije moguće:

```
SELECT UNIQUE posao
FROM RADNIK;
```

Kako u praksi, u većini slučajeva, postoji potreba za eliminacijom identičnih n-torki (slogova), to treba praktično uvek koristiti oblik SELECT DISTINCT**,** a ne samo SELECT, a mi ćemo u primerima koji slede smatrati da je to ponuđena, pretpostavljena opcija (engl. default) i nećemo je posebno navoditi**.**

Atributi se u naredbi SELECT mogu navoditi i tako što će im se pridodati i naziv relacije kojoj pripadaju (preporučljivo je uvek koristiti ovakav način pisanja, a obavezno ga moramo koristiti onda ako, u dve relacije koje pretražujemo, postoje atributi sa istim imenom).

Naziv relacije se tada može pisati i skraćeno, samo pomoću prvog slova naziva tabele, ali pod uslovom da se prvo slovo u nazivu relacije razlikuje, ako su nazivi atributa identični. Tako su sledeće naredbe potpuno identične sa naredbom u prethodnom primeru.

Primer 30. Prikazati sve različite kvalifikacije zaposlenih.

```
SELECT kvalif
FROM RADNIK;
```

Rezultat ovog upita su sve kvalifikacije za sve zaposlene, a kao rezultat se dobije onoliko zapisa koliko ima zaposlenih. Međutim, to nije bio zahtev u ovom zadatku, jer se tražilo da se prikažu samo različite kvalifikacije, pa ovaj upit treba uraditi na sledeći način:

```
SELECT DISTINCT kvalif
FROM RADNIK;
```

Kao rezultat ovog upita sa upotrebom DISTINCT dobili smo samo različite kvalifikacije koje postoje u tabeli RADNIK.

```
SELECT DISTINCT R.kvalif
FROM RADNIK R;
```

Prethodni primer pokazuje da se imena tabela mogu u naredbama zameniti odgovarajućim sinonimima ili skraćenicama (alias), sa ciljem da se ubrza pisanje naredbi. Tako je dugačko ime tabele RADNIK zamenjeno jednim slovom R.

Primer 31. Prikaži brojeve odeljenja u preduzeću:

a) u kojima ima zaposlenih,
b) svih odeljenja u preduzeću.

a) Ova informacija može se dobiti iz tabele RADNIK:

```
SELECT DISTINCT radnik.brod AS "Šifre odeljenja"
FROM RADNIK;
```

Šifre odeljenja
10
20
30
40

b) Isti podaci mogu se dobiti iz tabele ODELJENJE:

```
SELECT odeljenje.brod AS šifra
FROM ODELJENJE;
```

Šifra
10
20
30
40
50

Treba uočiti nekoliko detalja:

- u primeru a) moramo koristiti odredbu DISTINCT jer u jednom odeljenju radi više radnika pa bi u odgovoru bilo ponovljeno svako odeljenje više puta. Broj odeljenja iz tabele RADNIK jer u toj tabeli je evidencija koji zaposleni radi u kom odeljenju, a predikat DISTINCT je korišćen da bi se dobila samo različita odeljenja bez ponavljanja. U ovom slučaju nismo dobili odeljenje 50, jer ni jedan zaposleni ne radi u njemu. Pošto jedan radnik nije raspoređen ni u jedno odeljenje u odgovoru postoji jedno prazno polje (NULL vrednost).
- u primeru b) ne moramo koristiti odredbu DISTINCT jer je šifra odeljenja primarni ključ relacije i samim tim je jedinstven.
- u primeru a) moramo iza odredbe **AS** (kao) koristiti znake navoda ili uglaste zagrade [ ] (u Accessu) jer je novo ime sastavljeno od više reči, a u primeru b) se ne mora koristiti.

Napomena: U nekim sistemima za upravljanje bazama podataka (SUBP) nije potrebno navoditi odredbu AS**,** već se samo stavi razmak, tj. prazno mesto (space). To je ekvivalentno odredbi AS (što nije slučaj u Access-u). Tako su sledeće naredbe SELECT ekvivalentne prethodnim:

a)
```
SELECT O.brod šifra
FROM ODELJENJE O;
```

b)
```
SELECT DISTINCT R.brod [šifre odeljenja]
FROM RADNIK R;
```

Selekcija, izdvajanje slogova koji zadovoljavaju uslov - odredba WHERE

U svim dosadašnjim primerima u rezultatu su se pojavljivali svi redovi, slogovi jedne tabele. Moguće je primeniti naredbu SELECT samo na neke slogove (n-torke) koje zadovoljavaju ili ne zadovoljavaju zadate uslove. U tu svrhu koristimo odredbu **WHERE** (*gde je*) koja omogućuje:

- izdvajanje (selekciju) redova koji zadovoljavaju neki uslov,
- izdvajanje redova koji zadovoljavaju više uslova (**AND**),
- izdvajanje redova koji zadovoljavaju bar jedan od uslova (**OR**),
- izdvajanje redova koji zadovoljavaju složene uslove (**AND** i **OR**),
- izdvajanje redova čija je vrednost unutar nekih granica (**BETWEEN**),
- izdvajanje redova čija vrednost pripada nekoj listi vrednosti (**IN**),
- izdvajanje redova koji ne zadovoljavaju neke uslove (**NOT**, **IS NOT**),
- izdvajanje redova ako neka vrednost postoji (**EXISTS**) itd.

Sintaksa pisanja uslova (WHERE), pomoću izraza upoređivanja, u opštem slučaju je oblika:

argument1 ***operator upoređivanja*** **argument2**

a argumenti mogu biti:

- jedan atribut,
- skup atributa (u zagradama odvojeni zapetom),
- funkcija,
- izraz (sastoji se od imena atributa, funkcija i konstanti povezanih aritmetičkim operacijama)
- ili naredba **SELECT**

koja vraća najviše jednu n-torku, dok operator upoređivanja može biti bilo koji od operatora:

- **veći (>),**
- **manji (<),**
- **veći ili jednak (>=),**
- **manji ili jednak (<=),**
- **jednak (=),** i
- **različit (< >, !=).**

Ako se upoređuju argumenti koji se sastoje od više atributa, tada oba argumenta moraju imati jednak broj atributa, a upoređuje se prvi sa prvim, drugi sa drugim itd. Napomenimo da atributi koji se upoređuju moraju imati isti domen, odnosno moraju biti istog, ili kompatibilnog tipa podataka.

Prilikom postavljanja uslova mogu se koristiti sledeće odredbe:

- **BETWEEN** (između),
- **IN** (u),
- **EXISTS** (postoji),
- **ANY** (bilo koji, ma koji), **SOME** (neki),
- **ALL** (svi)**.**

Ne ulazeći u sve mogućnosti navedenih opcija, pokažimo na primerima kako se neke od njih koriste. Sintaksa operacije **BETWEEN** (u prevodu "između") ima opšti oblik

argument1 **[NOT] BETWEEN** argument2 **AND** argument3

a rezultat ovog testa je istinit ako se vrednost za *argument1* nalazi između vrednosti *argument2* i *argument3* uključujući i granice toga intervala (argument2 <= argument1 <= argument3). Kod NOT BETWEEN granice intervala nisu uključene (argument2 > argument1 ili argument1> argument3). Kod upotrebe BETWEEN i NOT BETWEEN intervali su disjunktni.

Opcija **EXISTS** ("postoji") koristi se za proveru postojanja najmanje jedne n-torke u rezultatu pretraživanja. Po pravilu, koristi se u ugnježdenim upitima. Sintaksa ove naredbe glasi:

EXISTS (relacioni izraz).

Odgovor je istinit (engl. **TRUE**) ako relacioni izraz daje barem jednu n-torku kao rezultat, u suprotnom je neistinit (engl. **FALSE**). Pored ovih, pomenutih opcija, šire verzije SQL-a imaju još neke mogućnosti, kao na primer:

- **LIKE,**
- **MATCH,**
- **ALL, ANY (SOME)**.

Njihovo značenje se može naći u korisničkim priručnicima (engl. **User manual**) za korišćenje programskog paketa SQL-a.

Složeni logički izrazi prave se pomoću binarnih logičkih operacija: OR (ili, disjunkcija, logičko sabiranje), AND (i, konjunkcija, logičko množenje), XOR (isključivo ili) i unarne operacije NOT (ne, negacija, komplementiranje). Istinitosne vrednosti ovih logičkih operacija zavise od istinitosnih vrednosti iskaza i definisane su odgovarajućim tabelama. Iskazi mogu biti tačni (TRUE, T), netačni (FALSE, F) ili nepoznate (Null). Ako su vrednosti iskaza poznate (T ili F) onda imamo dvovrednosnu (dvovalentnu) logiku ([4], [13]).

Tabele istinitosti iskaza i složenih israza u logici sa dve vrednosti (T, F)

NOT, negacija: suprotna vrednost od tačnog iskaza je netačan iskaz i obrnuto.

x	NOT
F	T
T	F

AND, i: Istovremeno zadovoljenje dva ili više uslova. Rezultujući izraz je tačan samo ako su tačni svi iskazi koji ga čine.

x	y	x AND y
F	F	F
F	T	F
T	F	F
T	T	T

AND	T	F
T	T	F
F	F	F

OR, ili: zadovoljenje bar jednog od dva ili više uslova. Rezultujući izraz je tačan samo ako je tačan bar jedan ili proizvoljan broj iskaza koji ga čine.

x	y	x OR y
F	F	F
F	T	T
T	F	T
T	T	T

OR	T	F
T	T	T
F	T	F

XOR, isključivo ili: Zadovoljenje samo jednog od dva uslova. Rezultujući izraz je tačan samo ako je tačan samo jedan iskaz od onih koji ga čine (ili neparan broj iskaza).

x	y	x XOR y
F	F	F
F	T	T
T	F	T
T	T	F

OR	T	F
T	F	T
F	T	F

Relacija jednakosti: **=** Rezultujući izraz je tačan samo ako su svi iskazi koji ga čine iste istinitosne vrednosti (oba tačna ili oba netačna).

x	y	x = y
F	F	T
F	T	F
T	F	F
T	T	T

=	T	F
T	T	F
F	F	T

Relacija nejednakosti: ≠ **(**<>, !=**)**. Rezultujući izraz je tačan samo ako su iskazi koji ga čine suprotne istinitosne vrednosti (jedan tačan a jedan netačan).

x	y	x ≠ y
F	F	F
F	T	T
T	F	T
T	T	F

≠	T	F
T	F	T
F	T	F

Relacije poređenja: =, <>, <, <=, >, >=, uglavnom ima smisla primenjivati nad numeričkim podacima (brojevima) mada su definisane i nad tekstualnim podacima.

Tekstovi se porede tako što se uporede ASCII kodovi prvih karaktera pa ako nisu jednaki, prelazi se na drugi karakter i tako redom, dok se ne donese zaključak ili se dođe do kraja teksta.

Primer 32. Prikazati imena zaposlenih koji rade u odeljenju 10.

```
SELECT ime
FROM RADNIK
WHERE brod=10;
```

Ime
Aleksandar
Vanja
Jovan
Janko
Mirjana
Tomislav

U ovom slučaju kao rezultat upita nećemo dobiti imena svih zaposlenih, već samo onih koji su iz odeljenja 10, jer smo taj uslov dodali u odredbu WHERE. Zavisno od toga koliko je zaposlenih u odeljenju 10, kao rezultat ovog upita može se dobiti:

- nula zapisa, ako nema ni jedan zaposleni u odeljenju 10,
- svi zapisi iz tabele RADNIK, ako su svi zaposleni u odeljenju 10,
- između 0 i ukupnog broja zapisa tabele RADNIK, ako su samo neki radnici zaposleni u odeljenju 10.

Primer 33. Prikazati imena zaposlenih koji rade u odeljenju 50.

```
SELECT ime
FROM RADNIK
WHERE brod = 50;
```

Rezultat ovog upita je prazna tabela, jer ne postoji ni jedan zaposleni u odeljenju 50.

Primer 34. Prikazati imena zaposlenih koji rade u odeljenju 70.

```
SELECT ime
FROM RADNIK
WHERE brod=70;
```

Rezultat ovog upita je prazna tabela, jer odeljenje 70 ne postoji.

Primer 35. Prikazati ime prezime i platu zaposlenog čiji je identifikacioni broj 5786.

```
SELECT ime, prezime, plata
FROM RADNIK
WHERE idbr=5786;
```

Ime	Prezime	Plata
Pavle	Šotra	2800

Rezultat ovog upita je jedan zapis jer postoji zaposleni datog identifikacionog broja i ne može se kao rezultat dobiti više od jednog zapisa, jer atribut *idbr* je primarni ključ u tabeli RADNIK i može postojati samo jedan zaposleni datog identifikacionog broja. Rezultat upita u kome se uslov postavlja tako da atribut koji je primarni ključ u tabeli može imati najviše jednu vrednost uvek je jedan ili ni jedan zapis, zavisno da li zadata vrednost tog atributa postoji. Ako ne postoji, rezultujuća tabela ima nula redova.

Primer 36. Prikazati ime i platu zaposlenog čiji je identifikacioni broj 5775.

```
SELECT ime, plata
FROM RADNIK
WHERE idbr=5775;
```

Rezultat ovog upita nema ni jedan slog, jer ne postoji radnik sa datim identifikacionim brojem.

Primer 37. Prikazati sve podatke o projektu 200.

```
SELECT *
FROM PROJEKAT
WHERE brproj=200;
```

Primer 38. Prikazati identifikacione brojeve zaposlenih koji učestvuju na projektu 100.

```
SELECT idbr
FROM UCESCE
WHERE brproj=100;
```

Idbr
5652
5786
5842
5900
5932
5953
6234

Rezultat ovog upita može biti nula ili više zapisa, zavisno od toga koliko zaposlenih radi na projektu 100. Treba primetiti da u tabeli UCESCE atribut *BrProj* jeste deo složenog primarnog ključa, ali sam nije primarni ključ.

Primer 39. Prikazati funkciju zaposlenog čiji je identifikacioni broj 5652 na projektu 100.

```
SELECT funkcija
FROM UCESCE
WHERE idbr=5652 AND brproj=100;
```

Funkcija
IZVRŠILAC

Rezultat ovog upita može biti jedan ili ni jedan zapis, a nikako više od jednog zapisa, jer atributi *idbr* i *BrProj* zajedno čine složeni primarni ključ tabele UCESCE, pa je njihova svaka kombinacija jedinstvena.

Primer 40. Prikazati sve podatke o zaposlenima koji imaju platu 2000.

```
SELECT *
FROM RADNIK
WHERE plata=2000;
```

Rezultat upita u kome se uslov odnosi na tačnu jednu vrednost atributa koji nije primarni ključ može biti nula ili više zapisa.

Primer 41. Prikaži kvalifikaciju, platu i ime zaposlenih u odeljenju 30.

```
SELECT R.kvalif, R.plata, R.ime, R.brod
FROM RADNIK R
WHERE R.brod = 30;
```

Kvalif	Plata	Ime	Brod
VSS	2800	Pavle	30
KV	1100	Andrija	30
KV	1100	Jovan	30
VSS	1300	Marko	30

Napomena: Uočimo da redosled atributa u rezultatu odgovara redosledu atributa kojim su oni navedeni u naredbi SELECT**,** a ne redosledu atributa u samoj fizičkoj tabeli.

Primer 42. Prikazati imena projekata i sredstava za projekte za koje se izdvajaju sredstva manja od 3 000 000.

```
SELECT imeproj, sredstva
FROM PROJEKAT
WHERE sredstva<3000000;
```

Imeproj	Sredstva
izvoz	2000000
izgradnja	0

Korišćen je operator "<" jer uslov zadatka zahteva sredstva manja od 3 000 000, ne uzimajući u obzir i vrednost 3 000 000.

Primer 43. Prikazati imena projekata i sredstva za projekte za koje se izdvajaju sredstva od 3 000 000 ili manja.

```
SELECT imeproj, sredstva
FROM PROJEKAT
WHERE sredstva<=3000000;
```

Imeproj	Sredstva
uvoz	3000000
izvoz	2000000
izgradnja	0

U ovom primeru je korišćen operator "<=" jer u uslovu zadatka se zahteva da sredstva budu 3 000 000 ili manje.

Primer 44. Prikazati imena i plate zaposlenih čija je plata veća od 2 000.

```
SELECT ime, plata
FROM RADNIK
WHERE plata>2000;
```

U ovom primeru korišćen je operator ">" (veće) tako da će u rezultatu upita biti prikazani samo oni zaposleni koji imaju platu veću od 2 000, ne uzimajući u obzir one čija je plata 2000.

Primer 45. Prikazati imena i plate zaposlenih čija plata je 2000 ili veća.

```
SELECT ime, plata
FROM RADNIK
WHERE plata>=2000;
```

U ovom primeru korišćen je operator ">=" da bi prikazali plate veće od 2 000, ali i one koje su 2 000.

Napomena: Operatori: =, <, >, <=, >= koriste se u MS Access-u, Oracle-u, MySQL-u, MS SQL Server-u. Operator "nije jednako", odnosno "različito" u raznim SUBP piše se na razne načine: "!= " u Oracle ili "<>" – Access i SQL Server.

Primer 46. Prikazati identifikacioni broj, ime, kvalifikaciju i broj odeljenja zaposlenih koji ne rade u odeljenju 10.

```
SELECT idbr, ime, kvalif, brod
FROM RADNIK
WHERE brod <>10;
```

Idbr	Ime	Kvalif	Brod
5367	Petar	KV	20
5780	Božidar	VSS	20
5786	Pavle	VSS	30
5842	Miloš	VSS	40
5867	Svetlana	VSS	40
5898	Andrija	KV	30
5900	Slobodan	KV	20
5932	Mitar	VSS	20
5953	Jovan	KV	30
6234	Marko	VSS	30
6789	Janko	VSS	40
7890	Ivan	VSS	20

Ovaj upit bi radio u MS Accessu, MS SQL Serveru, MySQL-u, jer se znak nejednakosti"<>" piše isto. Ovaj upit se može uraditi i na drugi način, gde se kao znak nejednakosti koristi "!= ", ali on neće raditi u MS Accessu (ali radi u MS SQL Serveru, MySQL-u i Oracle-u).

```
SELECT idbr, ime, kvalif, brod
FROM RADNIK
WHERE brod!=10;
```

Primer 47. Prikazati ime, datum zaposlenja, platu i premiju za zaposlene koji obavljaju posao savetnika.

```
SELECT ime, datzap, plata, premija
FROM RADNIK
WHERE posao='savetnik';
```

Ime	Datzap	Plata	Premija
Svetlana	1970-08-08 00:00:00	2750	NULL
Mitar	2000-03-25 00:00:00	2600	NULL

Savetnik je vrednost atributa *posao* i kako je u pitanju string potrebno je dodati apostrofe. Obeležavanje stringa na ovakav način je uobičajeno (MS Access, MySQL, MS SQL Server).

Ako vrednost atributa koja je string stavimo pod znake navoda ("*savetnik*"), upit će raditi u SUBP Oracle, MS Access i MySQL, dok MS SQL Server prihvata samo apostrofe.

Primer 48. Prikazati ime, datum zaposlenja, platu i premiju za zaposlene koji obavljaju posao vozača.

```
SELECT ime, datzap, plata, premija
FROM RADNIK
WHERE posao='vozac' ;
```

Na ovakav način urađen upit neće vratiti kao rezultat ni jedan zapis, iako u tabeli RADNIK postoje zaposleni koji obavljaju posao vozača, zbog toga što je u kriterijumu napisano *vozac* (slovo **c** umesto slova **č**), a u tabeli je upisana vrednost *vozač*. Zaposleni koji obavljaju posao vozača neće biti prikazani zbog pogrešno unetog uslova. Upit treba da izgleda ovako:

```
SELECT ime, datzap., plata, premija
FROM RADNIK
WHERE posao='vozač';
```

Napomena: SQL Server je osetljiv na kodne mape, odnosno neće prihvatiti naša slova Č, Đ, Š, Ć, Ž sem ako se pri definiciji tipa podatka ne upotrebi NVARCHAR. Tada se i u upitu ispred vrednosti mora staviti N (Natural). Prethodni upit u SQL Server-u ima oblik:

```
SELECT ime, datzap., plata, premija
FROM RADNIK
WHERE posao = N'vozač';
```

Primer 49. Prikazati ime, posao i kvalifikaciju zaposlenih koji su se zaposlili 17.12.1990 godine.

```
SELECT ime, posao, kvalif
FROM RADNIK
WHERE datzap='17.12.1990 ' ;

SELECT ime, posao, kvalif
FROM RADNIK
WHERE datzap=#17.12.1990# ;

U Oracle
SELECT ime, posao, kvalif
FROM RADNIK
WHERE datzap= to_date('19901217', 'YYYYMMDD');
```

Primer 50. Prikazati ime, prezime, posao, platu i premiju za zaposlene koji rade u odeljenju 10 i imaju kvalifikaciju KV.

```
SELECT ime, prezime, posao, plata, premija
FROM RADNIK
WHERE brod=10 AND kvalif='KV';
```

U ovom upitu imamo situaciju da treba da budu zadovoljena dva uslova istovremeno, pa je korišćen operator AND.

Ime	Prezime	Posao	Plata	Premija
Aleksandar	Marić	električar	1000	800
Jovan	Perić	električar	1000	500
Mirjana	Dimić	čistač	1000	0
Tomislav	Bogovac	električar	1000	1100

Primer 51. Prikazati imena zaposlenih koji rade u odeljenju 20, a imaju posao vozača.

```
SELECT ime
FROM RADNIK
WHERE brod=20 AND posao=N'vozač';
```

Primer 52. Prikaži ime, posao i platu zaposlenih u odeljenju 30, čija je plata veća od 2000.

```
SELECT R.ime, R.posao, R.plata, R. brod
FROM RADNIK R
WHERE R.brod=30 AND R.plata>2000;
```

Ime	Posao	Plata	Brod
Pavle	upravnik	2800	30

Primer 53. Prikazati ime, posao, platu i premiju zaposlenih, čiji je posao analitičar ili savetnik.

```
SELECT ime, posao, plata, premija
FROM RADNIK
WHERE posao='analitičar' OR posao='savetnik';
```

U ovom upitu potrebno je da bude zadovoljen jedan od dva uslova, pa je korišćen operator OR. Baza je osmišljena tako da jedan zaposleni može obavljati samo jedan posao u preduzeću.

Ovaj upit može da se uradi i na drugi način, korišćenjem operatora IN, i tada se kao elementi skupa koji zadovoljavaju uslove navode poslovi *'analitičar'* i *'savetnik'*.

```
SELECT ime, posao, plata, premija
FROM RADNIK
WHERE posao IN ('analitičar','savetnik');
```

Primer 54. Prikazati ime, posao, platu i premiju zaposlenih, čiji posao nije analitičar ni savetnik.

```
SELECT ime, posao, plata, premija
FROM RADNIK
WHERE posao<>'analitičar' AND posao<>'savetnik';
```

U ovom zadatku potrebno je da ne budu prikazani zaposleni koji obavljaju određene poslove, pa je korišćen operator NOT IN da bi bili eliminisani oni poslovi koji su navedeni u skupu.

```
SELECT ime, posao, plata, premija
FROM RADNIK
WHERE posao NOT IN ('analitičar','savetnik');
```

Primer 55. Prikaži ime, posao i broj odeljenja upravnika i direktora.

```
SELECT R.ime, R.posao, R.brod
FROM RADNIK R
WHERE (R.posao='direktor') OR (R.posao='upravnik');

SELECT R.ime, R.posao, R.brod
FROM RADNIK R
WHERE R.posao IN ('direktor', 'upravnik');
```

Ime	Posao	Brod
Janko	upravnik	10
Božidar	upravnik	20
Pavle	upravnik	30
Miloš	direktor	40

Primer 56. Prikaži ime i broj odeljenja zaposlenih koji rade u odeljenju 10 ili 20, a kvalifikacija im je VKV.

```
SELECT ime, brod
FROM RADNIK
WHERE (brod=10 OR brod =20) AND kvalif='VKV';
```

Ime	Brod
Vanja	10

U ovom upitu imamo kombinaciju uslova koji su povezani operatorima AND i OR, pa su zagrade neophodne da bi dobili tačno ono rešenje koje se traži u zadatku. U delu upita (BROD=10 OR BROD = 20) izdvojeni su zaposleni iz odeljenja 10 ili 20, a onda je dodat još i uslov da oni, pored toga, treba da imaju i kvalifikaciju VKV. Ovaj zadatak može se uraditi i na drugi način, a da i dalje bude ispravan:

```
SELECT ime, brod
FROM RADNIK
WHERE (brod=10 AND kvalif='VKV') OR (brod =20 AND kvalif='VKV');
```

Ako bismo ovaj upit uradili bez stavljanja zagrada tamo gde je to neophodno, prikazivao bi netačne rezultate, kao na primer:

```
SELECT ime, brod
FROM RADNIK
WHERE brod=10 OR brod =20 AND KVALIF='VKV';
```

U nekim SUBP (Oracle, SQL Server, i sl.) operacija AND ima viši prioritet od operacija OR, a NOT ima najviši prioritet. Zbog toga bi ovaj iskaz bio netačno interpretiran kao da je u pitanju uslov: brod=*10* OR (brod =*20* AND KVALIF='*VKV*'). Upotreba zagrada je preporučljiva da korisnik ne bi morao da pamti prioritete operacija.

Primer 57. Prikazati ime, posao i kvalifikaciju zaposlenih koji obavljaju posao upravnika ili savetnika, a zaposleni su u odeljenju 20.

```
SELECT ime, posao, kvalif
FROM RADNIK
WHERE posao IN ('upravnik', 'savetnik') AND brod=20;

SELECT ime, posao, kvalif
FROM RADNIK
WHERE (posao='upravnik' OR posao='savetnik') AND brod=20;

SELECT ime, posao, kvalif
FROM RADNIK
WHERE (posao='upravnik' AND brod=20) OR (posao='savetnik' AND brod=20);
```

Ime	Posao	Kvalif
Božidar	upravnik	VSS
Mitar	savetnik	VSS

U ovom primeru je prikazan zadatak urađen na tri načina, korišćenjem različitih kombinacija zagrada i operatora i svaki od njih je ispravan.

Primer 58. Prikaži ime i posao upravnika i analitičara iz odeljenja 10.

Bez korišćenja zagrada zadatak bi se dobio pogrešan rezultat!

```
SELECT R.ime, R.posao, R.brod
FROM RADNIK R
WHERE R.posao="upravnik" OR  R.posao="analitičar" AND R.brod=10;
```

Netačan rezultat:

Ime	Posao	Kvalif
Janko	upravnik	10
Božidar	upravnik	20
Pavle	upravnik	30

Rešenje zadatka urađeno na sledeći način je ispravno:

```
SELECT R.ime, R.posao, R.brod
FROM RADNIK R
WHERE (R.posao="upravnik" OR
       R.posao="analitičar") AND R.brod=10;
```

Tačan rezultat:

Ime	Posao	Brod
Janko	upravnik	10

Napomena: Poželjno je da redosled ispitivanja uslova regulišete upotrebom zagrada jer u protivnom, možete dobiti potpuno neočekivane i verovatno netačne odgovore.

Primer 59. Prikazati ime, datum zaposlenja, platu, premiju i broj odeljenja za zaposlene koji imaju platu između 1 000 i 2 000 (uključujući i te vrednosti).

```
SELECT ime, datzap, plata, premija, brod
FROM RADNIK
WHERE plata BETWEEN 1000 AND 2000;
```

Ime	Datzap	Plata	Premija	Brod
Petar	1978-01-01 00:00:00	1300	1900	20
Aleksandar	1990-02-17 00:00:00	1000	800	10
Vanja	1991-11-07 00:00:00	1200	1300	10
Jovan	1980-05-31 00:00:00	1000	500	10
Mirjana	1991-09-30 00:00:00	1000	0	10
Tomislav	1971-04-19 00:00:00	1000	1100	10
Andrija	1980-01-20 00:00:00	1100	1200	30
Jovan	1979-01-12 00:00:00	1100	0	30
Marko	1990-12-17 00:00:00	1300	3000	30
Ivan	2003-12-17 00:00:00	1600	3200	20
Luka	2004-05-20 00:00:00	2000	NULL	NULL

Operator **BETWEEN** se koristi onda kada je potrebno prikazati podatke iz nekog opsega vrednosti, uključujući i granične vrednosti. Ovaj zadatak bi se alternativno mogao uraditi i na drugi način:

```
SELECT ime, datzap, plata, premija, brod
FROM RADNIK
WHERE plata>=1000 AND plata<=2000;
```

Primer 60. Prikaži ime i platu zaposlenih čija je plata od 2600 do 3000.

```
SELECT R.ime, R.plata
FROM RADNIK R
WHERE R.plata >= 2600 AND R.plata <=3000;

SELECT R.ime, R.plata
FROM RADNIK R
WHERE R.plata BETWEEN 2600 AND 3000;
```

Primer 61. Prikazati ime, datum zaposlenja, platu, premiju i broj odeljenja za zaposlene koji imaju platu između 1000 i 2000 (ne uključujući te vrednosti).

```
SELECT ime, datzap, plata, premija, brod
FROM RADNIK
WHERE plata>1000 AND plata<2000;
```

Ime	Datzap	Plata	Premija	Brod
Petar	1978-01-01 00:00:00	1300	1900	20
Vanja	1991-11-07 00:00:00	1200	1300	10
Andrija	1980-01-20 00:00:00	1100	1200	30
Jovan	1979-01-12 00:00:00	1100	0	30
Marko	1990-12-17 00:00:00	1300	3000	30
Ivan	2003-12-17 00:00:00	1600	3200	20

Napomena: U ovom slučaju, kada ne treba da se uzmu u obzir granične vrednosti 1000 i 2000, ne može se koristi operator BETWEEN.

Neki SUBP, na primer Oracle, imaju na raspolaganju i odredbe **ANY** (**SOME**), **ALL** koje se mogu koristiti u WHERE odredbi. Kod drugih SUBP, na primer MS Access, MS SQL Server, ove odredbe se mogu koristiti samo u ugnježdenim upitima. U MySQL-u **ANY**( ), **SOME**( ), i **EVERY**( ) grupne su funkcije. Upotreba ovih odredi ima mnogo više smisla u ugnježdenim upitima. Sledeća dva primera urađena su u SUBP Oracle.

Primer 62. Prikazati zaposlene koji imaju platu jednaku bilo kojoj vrednosti iz skupa {1000, 1500, 2000}.

```
SELECT idbr, ime, plata
FROM RADNIK
WHERE plata = ANY (1000,1500, 2000);

SELECT idbr, ime, plata
FROM RADNIK
WHERE plata = SOME (1000,1500, 2000);
```

Primer 63. Prikazati zaposlene čija je plata veća od svih navedenih vrednosti u skupu {1000, 1500, 2000}.

```
SELECT idbr, ime, plata
FROM RADNIK
WHERE plata >ALL (1000,1500, 2000);
```

Uređivanje izveštaja po vrednosti izabranih atributa - odredba ORDER BY

Redosled slogova u tabelama podataka nije definisan u bazi podataka. Čak ni fizički redosled slogova na disku ne može da određuje redosled slogova u rezultatu upita. Odredba **ORDER BY** određuje eksplicitno redosled n-torke u rezultatu upita po nekom kriterijumu (po abecedi, po veličini itd.) u rastućem ili opadajućem poretku. Odredba ORDER BY je uvek poslednja odredba u SELECT bloku, jer najpre se selektuju n-torke, a na kraju se uređuju. Ova opcija može imati više atributa po kojima se vrši uređanje.

Primer 64. Prikazati ime, kvalifikaciju, posao, platu i premiju zaposlenih koji rade u odeljenju 10. Rezultate urediti po imenu u rastućem redosledu.

```
SELECT ime, kvalif, posao, plata, premija
FROM RADNIK
WHERE brod=10
ORDER BY ime ASC;
```

Ime	Kvalif	Posao	Plata	Premija
Aleksandar	KV	električar	1000	800
Janko	VSS	upravnik	2400	NULL
Jovan	KV	električar	1000	500
Mirjana	KV	čistač	1000	0
Tomislav	KV	električar	1000	1100
Vanja	VKV	prodavac	1200	1300

U rešenju ovog zadatka vidimo da su zapisi poređani po imenima zaposlenih u rastućem abecednom redosledu. Ovde smo uveli odredbu ORDER BY koja služi za sortiranje podataka i uvek ide na kraju upita. Iza atributa *ime* je dodato **ASC** (Ascending), što označava da treba urediti podatke po imenu u rastućem redosledu. Za opadajući redosled obavezno se mora navesti odredba **DESC**.

Ovaj primer smo mogli uraditi i ovako:

```
SELECT ime, kvalif, posao, plata, premija
FROM RADNIK
WHERE brod=10
ORDER BY ime;
```

Razlika u odnosu na prethodni slučaj je ta što iza imena ne stoji ASC. Ako je potrebno da podatke uredimo po rastućem redosledu, nije neophodno to posebno naglasiti, jer ako se ništa ne navede, podrazumeva se da je uređenje po rastućem redosledu.

Primer 65. Prikazati ime, platu i premiju zaposlenih koji obavljaju posao električara. Rezultate urediti po premiji u opadajućem redosledu.

```
SELECT ime, plata, premija
FROM RADNIK
WHERE posao='električar'
ORDER BY premija DESC;
```

Ime	Plata	Premija
Tomislav	1000	1100
Aleksandar	1000	800
Jovan	1000	500

Kada je potrebno urediti podatke po opadajućem redosledu, neophodno je to navesti navođenjem **DESC** iza imena atributa po kome se radi uređenje.

Napomena: U nekim SUBP Null je najmanja vrednost (Access, SQL Server) i kada se sortiranje vrši u rastućem redosledu onda su ovi redovi na početku tabele, a kada se sortira u opadajućem redosledu ovi redovi su na kraju. Kod nekih su kao nepostojeći, uvek izdvojeni na početak tabele koja predstavlja odgovor na upit (Oracle).

Primer 66. Prikazati ime, platu, premiju, posao i broj odeljenja za sve zaposlene. Rezultate urediti po broju odeljenja u rastućem redosledu, a zatim po plati u opadajućem.

```
SELECT ime, plata, premija, posao, brod
FROM RADNIK
ORDER BY brod, plata DESC;
```

Ako je potrebno urediti rezultujuću tabelu po više atributa, onda se oni navode onim redosledom kojim se želi uređenje uz navođenje tipa uređenja. Pri tome se umesto imena atributa mogu koristiti redni brojevi atributa u naredbi SELECT. Prethodni upit bi tada bio:

```
SELECT ime, plata, premija, posao, brod
FROM RADNIK
ORDER BY 5, 2 DESC;
```

Napomena: Redosled sortiranja odgovara redosledu navođenja atributa u odredbi ORDER BY. Najpre se sortira izveštaj po prvom navedenom atributu, zatim po drugom itd. Sortiranje može biti izvršeno u rastućem redosledu – ASC (Ascendant) i to je pretpostavljena (default) vrednost, pa se ne mora navoditi. Ako želimo da odgovor bude uređen u opadajućem redosledu po vrednosti nekog atributa moramo iza imena navesti reč DESC (Descendent).

Primer 67. Prikaži ime, kvalifikaciju, platu i premiju uređenu:

a) po kvalifikaciji u opadajućem, po plati u rastućem, a po premiji u opadajućem redosledu:

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY kvalif DESC, plata, premija DESC;
```

b) po plati u rastućem, a po kvalifikaciji i premiji u opadajućem redosledu:

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY plata, kvalif DESC, premija DESC;
```

Ime	Kvali	Plata	Premija
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Luka	VSS	2000	NULL
Božidar	VSS	2200	NULL
Janko	VSS	2400	NULL
Mitar	VSS	2600	NULL
Svetlana	VSS	2750	NULL
Pavle	VSS	2800	NULL
Miloš	VSS	3000	NULL
Janko	VSS	3900	10
Vanja	VKV	1200	1300
Slobodan	KV	900	1300
Tomislav	KV	1000	1100
Aleksandar	KV	1000	800
Jovan	KV	1000	500
Mirjana	KV	1000	0
Andrija	KV	1100	1200
Jovan	KV	1100	0
Petar	KV	1300	1900

a)

Ime	Kvalif	Plata	Premija
Slobodan	KV	900	1300
Tomislav	KV	1000	1100
Aleksandar	KV	1000	800
Jovan	KV	1000	500
Mirjana	KV	1000	0
Andrija	KV	1100	1200
Jovan	KV	1100	0
Vanja	VKV	1200	1300
Marko	VSS	1300	3000
Petar	KV	1300	1900
Ivan	VSS	1600	3200
Luka	VSS	2000	NULL
Božidar	VSS	2200	NULL
Janko	VSS	2400	NULL
Mitar	VSS	2600	NULL
Svetlana	VSS	2750	NULL
Pavle	VSS	2800	NULL
Miloš	VSS	3000	NULL
Janko	VSS	3900	10

b)

c) po premiji i kvalifikaciji u opadajućem, a po plati u rastućem redosledu:

SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY premija **DESC**, kvalif **DESC**, plata;

Ime	Kvalif	Plata	Premija
Ivan	VSS	1600	3200
Marko	VSS	1300	3000
Petar	KV	1300	1900
Vanja	VKV	1200	1300
Slobodan	KV	900	1300
Andrija	KV	1100	1200
Tomislav	KV	1000	1100
Aleksandar	KV	1000	800
Jovan	KV	1000	500
Janko	VSS	3900	10
Mirjana	KV	1000	0
Jovan	KV	1100	0
Luka	VSS	2000	NULL
Božidar	VSS	2200	NULL
Janko	VSS	2400	NULL
Mitar	VSS	2600	NULL
Svetlana	VSS	2750	NULL
Pavle	VSS	2800	NULL
Miloš	VSS	3000	NULL

Ako postoji više n-torki sa jednakim vrednostima atributa, njihov redosled je onda proizvoljan. Ovakav rezultat se može izbeći sortiranjem po atributu koji je primarni ključ, gde su dve iste vrednosti atributa isključene. Ako se sortiranje vrši po nekom atributu koji ima i vrednost Null (kao što bi mogao biti atribut *premija* u poslednjem primeru), onda se sve vrednosti Null grupišu zajedno.

Rad sa tekstualnim podacima - operator LIKE

Za rad sa tekstualnim podacima vrlo često se koristi odredba LIKE (liči na, kao, poput). Upotreba ove odredbe omogućava pronalaženje traženog teksta u nekom tekstualnom podatku (podatku tipa CHAR). U raznim verzijama SQL-a postoje džoker znaci, tj. znaci koji mogu zameniti jedan znak ili bilo koji niz znakova. U Oracle-u, MySQL i MS SQL Serveru proizvoljan niz znakova zamenjuje se džoker znakom **%** (u Access-u je to znak *****), a jedan znak zamenjuje se džokerom **_** (u MS Accessu je to znak **?**).

Navedimo nekoliko primera kako bismo izlistali imena koja zadovoljavaju neki od uslova:

- ime se završava slovom **a** — **WHERE** ime **LIKE** '%a' ;
- treće slovo imena je **V** — **WHERE** ime **LIKE** '_ _v%' ;
- treće slovo imena je **v** — **WHERE** ime **LIKE** '??v*' ; (u Access-u)
- u imenu nema slovo **N** — **WHERE** ime **NOT LIKE** '%n%' ;
- ime je dužine od 5 slova — **WHERE** ime **LIKE** '_ _ _ _ _ ' ;
- ime nije dužine od 5 slova — **WHERE** ime **NOT LIKE** '_ _ _ _ _ ' ;
- u imenu je slovo **l** ispred slova **N** — **WHERE** ime **LIKE** '%l %N %' ;
- broj ima **dve cifre** crticu **cifru** — **WHERE** tel **LIKE** '[0-9][0-9]-[0-9]' ;

Primer 68. Prikazati ime, prezime, datum zaposlenja i broj odeljenja za zaposlene čije ime počinje slovom M.

```
SELECT ime, prezime, datzap, brod
FROM RADNIK
WHERE ime LIKE '%M%';
```

Ime	Prezime	Datzap	Brod
Mirjana	Dimić	1991-09-30 00:00:00	10
Miloš	Marković	1981-12-15 00:00:00	40
Mitar	Vuković	2000-03-25 00:00:00	20
Marko	Nastić	1990-12-17 00:00:00	30

Operator **LIKE** je pogodan za pretraživanje tekstualnih podataka. Znak % je takozvani «džoker», znak koji zamenjuje proizvoljan broj znakova. 'M%' znači da prvo slovo treba da bude M, a znak % zamenjuje proizvoljan niz karaktera, tako da će u rezultatu upita biti prikazani svi zaposleni čije ime počinje slovom M, bez obzira na broj slova u imenu. Ovaj upit bi radio u MS SQL Serveru, dok za MS Access treba napisati upit na sledeći način:

```
SELECT ime, datzap, brod
FROM RADNIK
WHERE ime LIKE 'M*';
```

Dakle, u MS Accessu «džoker» znak koji zamenjuje proizvoljan niz znakova je: *****.

Primer 69. Prikazati broj i ime projekta čije ime završava slovom z.

```
SELECT brproj, imeproj
FROM PROJEKAT
WHERE imeproj LIKE '%z';
```

Brproj	Imeproj
100	uvoz
200	izvoz

```
SELECT brproj, imeproj
FROM PROJEKAT
WHERE imeproj LIKE '*z';
```

Prvi slučaj je napisan za MS SQL Server, a drugi za MS Access. Slično kao i u prethodnom zadatku, ovde je potrebno da poslednje slovo u stringu bude z, a nije bitno koja su i koliko ima slova pre njega.

Primer 70. Prikazati imena zaposlenih čije ime sadrži slovo t.

```
SELECT ime
FROM RADNIK
WHERE ime LIKE '%t%';
```

Ime
Petar
Svetlana
Tomislav
Mitar

```
SELECT ime
FROM RADNIK
WHERE ime LIKE '*t*';
```

Prvi slučaj je napisan za MS SQL Server, a drugi za MS Access. «Džoker» znaci zamenjuju bilo koji niz karaktera, tako da se u rezultatu dobijaju imena koja počinju, završavaju ili u sebi sadrže slovo ***t***.

Primer 71. Prikazati imena zaposlenih čije ime ne sadrži slovo A (u Oracle i SQL Serveru).

```
SELECT ime
FROM RADNIK
WHERE ime NOT LIKE '%a%';
```

Ime
Miloš

```
SELECT ime
FROM RADNIK
WHERE ime NOT LIKE '*a*';
```

Prvi slučaj je napisan za MS SQL Server, a drugi za MS Access. Korišćen je operator NOT LIKE, koji je negacija od operatora LIKE, pa su u rezultatu, prikazana su samo ona imena u kojima nema slova ***a***.

Primer 72. Prikazati ime, posao i broj odeljenja zaposlenih čije ime počinje slovom M, a drugo slovo u imenu je o ili i.

```
SELECT ime, posao, brod
FROM RADNIK
WHERE ime LIKE 'M[o, i]%';

SELECT ime, posao, brod
FROM RADNIK
WHERE ime LIKE 'M[o, i]*';
```

Ime	Posao	Brod
Mirjana	čistač	10
Miloš	direktor	40
Mitar	savetnik	20

Prvi slučaj je napisan za MS SQL Server, a drugi za MS Access. U ovom zadatku je dodat uslov da drugo slovo bude neko od navedenih u uglastim zagradama [ ].

Primer 73. Prikazati broj odeljenja, ime odeljenja i mesto za odeljenja čije ime počinje slovom P ili slovom D.

```
SELECT brod, imeod, mesto
FROM ODELJENJE
WHERE  imeod LIKE '[P, D]%';
```

Brod	Imeod	Mesto
20	Plan	Dorćol
30	Prodaja	Stari Grad
40	Direkcija	Banovo Brdo

Ovaj upit se može uraditi i na drugi način korišćenjem operatora OR.

```
SELECT brod, imeod, mesto
FROM ODELJENJE
WHERE  imeod LIKE 'P%' OR imeod LIKE 'D%';
```

Primer 74. Prikaži prezime i kvalifikaciju zaposlenih čija prezimena počinju slovom P.

a) bez uređivanja,
b) rezultate urediti po prezimenu u rastućem redosledu.

a)
```
SELECT R.prezime, R.kvalif
FROM Radnik R
WHERE R.prezime LIKE 'p%';
```

Prezime	Kvalif
Perić	KV
Petrović	KV
Perić	KV

Prezime	Kvalif

b) **SELECT** R.prezime, R.kvalif
FROM Radnik R
WHERE R.prezime **LIKE** *'p%*
ORDER BY R.prezime;

Perić	KV
Perić	KV
Petrović	KV

Rad sa Null vrednostima

Za testiranje vrednosti kolona koje sadrže Null vrednosti na raspolaganju su samo dve opcije IS NULL i IS NOT NULL, a moguće je koristiti i operatore poređenja.

NULL se ne može pojaviti u kolonama koje su definisane kao NOT NULL i kao primarni ključ (PRIMARY KEY). Ranije verzije nekih SUBP su Null vrednost predstavljale kao prazan niz znakova **“”**. Teorijski to nije isto i danas se ni jedna druga vrednost ne poistovećuje sa Null. Rad sa Null vrednostima je vrlo složen, jer sve operacije poređenja (izuzev IS NULL i IS NOT NULL) imaju nepoznat rezultat kada je jedan od argumenata Null. Isto važi i pri izračunavanju vrednosti logičkih izraza. Nepoznata istinitosna vrednost se može posmatrati kao *možda* (MAYBE). Pri izračunavanju aritmetičkih izraza ili funkcija neophodno je voditi računa o tome šta predstavlja rezultat funkcije ili izraza koji sadrže Null vrednosti. Ako se vrše izračunavanja sa kolonama koje sadrže vrednost Null, rezultat je Null, jer nije moguće izvršiti operaciju nad nepostojećom vrednošću.

Unarni operatori IS Null i IS NOT Null, tablice istinitosti.

<vrednost>	IS NULL	IS NOT NULL
T	F	T
F	F	T
Null	T	F

Tabele istinitosti iskaza i složenih iskaza u logici sa dve vrednosti (T, F)

Izrazi u kojima se pojavljuju atributi sa Null vrednostima mogu imati tri istinitosne vrednosti: (TRUE, T), netačna (FALSE, F) i nepostojeća (Null). Trovrednosnu (trovalentnu) logiku ([4], [13]) svaki proizvođač sistema za upravljanje bazama podataka realizuje različito i ne baš uvek u skladu sa matematičkom logikom. Microsoft se pridržava logike da rezultat svake operacije sa Null daje Null, ali ne baš dosledno (XOR).

NOT, negacija: suprotna vrednost od tačnog iskaza je netačan i obrnuto.

x	NOT
T	F
F	T
Null	Null

AND, i: Istovremeno zadovoljenje dva ili više uslova.

AND	T	F	Null
T	T	F	Null
F	F	F	Null
Null	Null	Null	Null

Access i SQL Server

AND	T	F	Null
T	T	F	Null
F	F	F	F
Null	Null	F	Null

Bulova algebra i Oracle

OR, ili: zadovoljenje bar jednog od dva ili više uslova.

OR	T	F	Null
T	T	T	Null
F	T	F	Null
Null	Null	Null	Null

Access i SQL Server

OR	T	F	Null
T	T	T	T
F	T	F	Null
Null	T	Null	Null

Bulova algebra i Oracle

XOR, isključivo ili: Zadovoljenje samo jednog od dva uslova.

XOR	T	F	Null
T	F	T	T
F	T	F	Null
Null	T	Null	Null

Access

XOR	T	F	Null
T	F	T	Null
F	T	F	Null
Null	Null	Null	Null

SQL Server i Oracle

Relacija jednakosti: =. Rezultujući izraz je tačan samo ako su svi iskazi koji ga čine iste istinitosne vrednosti (oba tačna ili oba netačna).

=	T	F	Null
T	T	F	Null
F	F	T	Null
Null	Null	Null	Null

Kod SQL Servera, ako je isključena opcija ANSI_NULLS, Null=Null daje vrednost True, Null=<bilo koja vrednost, T ili F> daje rezultat False.

Relacija nejednakosti: ≠ **(**<>, !=**)**. Rezultujući izraz je tačan samo ako su iskazi koji ga čine suprotne istinitosne vrednosti (jedan tačan a jedan netačan).

≠	T	F	Null
T	F	T	Null
F	T	F	Null
Null	Null	Null	Null

U radu sa Null vrednostima mogu se pojaviti neočekivani rezultati ukoliko korisnici i programeri nisu svesni da vrednost nekog svojstva može biti i nepostojeća. Zbog toga se treba truditi da bude što manje Null vrednosti ili da se njihovo prisustvo jasno naznači.

Primer 75. Prikaži ime, kvalifikaciju, platu i premiju zaposlenih koji:

a) imaju premiju.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
WHERE premija IS NOT Null;
```

b) nemaju premiju.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
WHERE premija IS Null;
```

Ime	Kvalif	Plata	Premija
Petar	KV	1300	1900
Aleksandar	KV	1000	800
Vanja	VKV	1200	1300
Jovan	KV	1000	500
Mirjana	KV	1000	0
Tomislav	KV	1000	1100
Andrija	KV	1100	1200
Slobodan	KV	900	1300
Jovan	KV	1100	0
Marko	VSS	1300	3000
Janko	VSS	3900	10
Ivan	VSS	1600	3200

a)

Ime	Kvalif	Plata	Premija
Janko	VSS	2400	NULL
Božidar	VSS	2200	NULL
Pavle	VSS	2800	NULL
Miloš	VSS	3000	NULL
Svetlana	VSS	2750	NULL
Mitar	VSS	2600	NULL
Luka	VSS	2000	NULL

b)

Napomena: Ovde treba uočiti razliku između objekata koji nemaju premiju (imaju Null vrednost) i objekata koji imaju premiju, a vrednost premije može biti i 0.

Rad sa Null vrednostima – dodela neutralne vrednosti pomoću funkcije

Funkcija **NVL**(atrib, vrednost]) služi da se u upitu dodeli neka poznata vrednost poljima u koloni koja imaju vrednost Null. Ta dodela je samo privremena i odnosi se na izvršavanje tog upita. Da bi se u izveštaj uključili i slogovi u kojima se javljaju Null vrednosti, potrebno je Null vrednostima dodeliti neutralnu vrednost u odnosu na operaciju koja se primenjuje. Na primer, neutralna vrednost za sabiranje je broj 0. U raznim SUBP u tu svrhu koriste se različite funkcije. U MS Access.u to je funkcija **NZ** (atrib[, vrednost]), u SQL Server-u **ISNULL** (atrib, vrednost]) a u MySQL-u **IFNULL** (atrib, vrednost]).

Primer 76. Za sve zaposlene prikazati njihove identifikacione brojeve, imena, broj odeljenja u kome rade i ukupna primanja.

Ukupna primanja zaposlenog dobijemo kada saberemo platu i premiju. Pošto polje PREMIJA može imati Null vrednost, da je zadatak urađen na sledeći način, bili bi isključeni svi zaposleni koji ne primaju premiju iako oni primaju platu:

```
SELECT idbr, ime, brod, plata+premija AS [ukupna primanja]
FROM RADNIK;
```

Idbr	Ime	Brod	Ukupna primanja
5367	Petar	20	3200
5497	Aleksandar	10	1800
5519	Vanja	10	2500
5652	Jovan	10	1500
5662	Janko	10	NULL
5696	Mirjana	10	1000
5780	Božidar	20	NULL
5786	Pavle	30	NULL
5842	Miloš	40	NULL
5867	Svetlana	40	NULL
5874	Tomislav	10	2100
5898	Andrija	30	2300
5900	Slobodan	20	2200

5932	Mitar	20	NULL
5953	Jovan	30	1100
6234	Marko	30	4300
6789	Janko	40	3910
7890	Ivan	20	4800
7892	Luka	NULL	NULL

Zbog toga je potrebno koristiti funkciju NVL( ) u Oracle, koja je u MS Accessu NZ( ), u MS SQL Serveru ISNULL( ). Tako funkcija NVL(PREMIJA, 0) vraća vrednost 0 u onim slučajevima kada je vrednost atributa PREMIJA jednaka Null. Ako je potrebno da se umesto vrednosti Null dobije neka druga vrednost, na primer 5, izraz bi izgledao NVL(PREMIJA, 5).

Zadatak urađen u SUBP Oracle-u:

SELECT idbr, ime, brod, plata+**NVL**(premija, 0) **AS** [Ukupna primanja]
FROM RADNIK;

Zadatak urađen u MS Access-u:

SELECT idbr, ime, brod, plata+**NZ**(premija) **AS** [Ukupna primanja]
FROM RADNIK;

Zadatak urađen u MS SQL Server-u:

SELECT idbr, ime, brod, plata+ **ISNULL**(premija, 0) **AS** [Ukupna primanja]
FROM RADNIK;

Zadatak urađen u MySQL-u:

SELECT idbr, ime, brod, plata+ **IFNULL**(premija, 0) **AS** [Ukupna primanja]
FROM RADNIK;

Ispravan rezultat prikazan je u sledećoj tabeli:

Idbr	Ime	Brod	Ukupna primanja
5367	Petar	20	3200
5497	Aleksandar	10	1800
5519	Vanja	10	2500
5652	Jovan	10	1500
5662	Janko	10	2400
5696	Mirjana	10	1000
5780	Božidar	20	2200
5786	Pavle	30	2800
5842	Miloš	40	3000
5867	Svetlana	40	2750
5874	Tomislav	10	2100
5898	Andrija	30	2300
5900	Slobodan	20	2200
5932	Mitar	20	2600
5953	Jovan	30	1100
6234	Marko	30	4300
6789	Janko	40	3910
7890	Ivan	20	4800
7892	Luka	NULL	2000

Opšti oblik funkcije **NZ** (atribut[, zamena]). Funkcija NZ za polja koja imaju vrednost Null vraća vrednost broj. Ako se izostavi parametar broj, funkcija vraća vrednost 0 i tada može

da stoji samo NZ(premija). U SQL Serveru ekvivalenta funkcija ISNULL traži dva parametra, pa mora da piše ISNULL(premija, 0). U MySQL ekvivalenta funkcija je IFNULL(premija, 0).

Primer 77. Prikazati ime, prezime i ukupna primanja za zaposlene čija je kvalifikacija KV. Rezultate urediti po ukupnim primanjima u opadajućem redosledu.

```
SELECT ime, prezime, plata+ISNULL(premija, 0) AS [ukupna primanja]
FROM RADNIK
WHERE kvalif='KV'
ORDER BY plata+ISNULL(premija, 0) DESC;
```

Ime	Prezime	Ukupna primanja
Petar	Vasić	3200
Andrija	Ristić	2300
Slobodan	Petrović	2200
Tomislav	Bogovac	2100
Aleksandar	Marić	1800
Jovan	Perić	1500
Jovan	Perić	1100
Mirjana	Dimić	1000

Napomena: U odredbi ORDER BY može se koristiti ceo izraz po kome se vrši uređivanje (kao u prethodnom slučaju) ili redni broj izraza u naredbi SELECT (naredni primer), i može se koristiti i natpis (engl. **Caption**) kolone (u ovom slučaju [Ukupna primanja]).

```
SELECT ime, prezime, plata+ISNULL(premija, 0) AS [ukupna primanja]
FROM RADNIK
WHERE kvalif='KV'
ORDER BY 3 DESC;
```

Odredba GROUP BY

Odredba **GROUP BY,** koju treba da sledi lista atributa, koristi se za grupisanje n-torki na osnovu nekog kriterijuma. Naime, naredba SELECT kao rezultat daje opet relaciju, pa n-torke nisu složene ni po kakvom redu, jer to, po definiciji relacije, nije ni potrebno. Naredbom GROUP BY n-torke u relaciji bivaju presložene tako da sve n-torke unutar grupe imaju jednake vrednosti atributa po kome se grupišu ([4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [15], [25], [27], [30], [32]).

Ako se navede više atributa po kojima treba vršiti grupisanje, prvo se ređaju n-torke sa jednakom vrednošću prvog atributa, zatim se unutar tih grupa preslažu n-torke prema vrednostima drugog atributa, itd. Grupni upiti vraćaju rezultate koji su karakteristika neke grupe slogova, a ne jednog sloga. Svi slogovi koji zadovoljavaju uslov selekcije predstavljaju grupu nad kojom se izračunava jedna ili više grupnih funkcija. Kada je navedena odredba GROUP BY, izrazi u odredbi SELECT mogu da budu:

- konstante,
- grupne funkcije,
- bilo koji od izraza iz odredbe GROUP BY (u nekim SUBP svi izrazi), i
- izraz koji je sačinjen od gornjih oblika i koji za svaki slog u grupi daje jednaku vrednost.

Svi slogovi za koje je neki određeni izraz za grupisanje jednak Null, smeštaju se u jednu grupu.

Primer 78. Prikaži ime, kvalifikaciju, platu i premiju:

a) grupisanu po kvalifikaciji, po plati i po premiji,

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
GROUP BY kvalif, plata, premija, ime;
```

b) uređenu po kvalifikaciji, plati i premiji u rastućem redosledu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY kvalif, plata, premija;
```

Napomena: Ova sličnost je samo prividna. Prava uloga odredbe GROUP BY biće opisana pri upotrebi sumarnih (agregatnih) funkcija, gde se određene funkcije (na primer srednja vrednost) primenjuju na grupe podataka sa nekim zajedničkim svojstvom (na primer po odeljenjima).

Napomena: Atributi po kojima se vrši grupisanje ne moraju biti navedeni i u SELECT naredbi, a mora se vršiti grupisanje po svim atributima navedenim u naredbi **SELECT**. Dakle, u prethodnom primeru grupisanje se mora izvršiti i po imenu, iako to nismo želeli (Access, MS SQL Server, Oracle).

Grupisanje, GROUP BY se koristi za izračunavanje zbirnih (SUM) i dobijanje graničnih (najmanjih MIN, najvećih MAX), srednjih (AVG) vrednosti u skupu, odnosno grupi podataka, ako i za prebrojavanje vrednosti u nekom skupu (COUNT).

Ime	Kvalif	Plata	Premija
Slobodan	KV	900	1300
Mirjana	KV	1000	0
Jovan	KV	1000	500
Aleksandar	KV	1000	800
Tomislav	KV	1000	1100
Jovan	KV	1100	0
Andrija	KV	1100	1200
Petar	KV	1300	1900
Vanja	VKV	1200	1300
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Luka	VSS	2000	NULL
Božidar	VSS	2200	NULL
Janko	VSS	2400	NULL
Mitar	VSS	2600	NULL
Svetlana	VSS	2750	NULL
Pavle	VSS	2800	NULL
Miloš	VSS	3000	NULL
Janko	VSS	3900	10

a) grupisanje podataka

Ime	Kvalif	Plata	Premija
Slobodan	KV	900	1300
Mirjana	KV	1000	0
Jovan	KV	1000	500
Aleksandar	KV	1000	800
Tomislav	KV	1000	1100
Jovan	KV	1100	0
Andrija	KV	1100	1200
Petar	KV	1300	1900
Vanja	VKV	1200	1300
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Luka	VSS	2000	NULL
Božidar	VSS	2200	NULL
Janko	VSS	2400	NULL
Mitar	VSS	2600	NULL
Svetlana	VSS	2750	NULL
Pavle	VSS	2800	NULL
Miloš	VSS	3000	NULL
Janko	VSS	3900	10

b) sortiranje podataka

Funkcije u SQL-u i upiti sa izračunavanjem novih vrednosti

Pretraživanje nekog informacionog sistema u svrhu dobijanja novih i relevantnih informacija je, svakako, najinteresantniji oblik primene analize baze podataka. Upravo u takvom pretraživanju je SQL pokazao svoja preimućstva nad drugim paketima. SQL ima ugrađen veliki broj gotovih funkcija za *dobijanje grupnih informacija* (**AVG**, **SUM**, **MIN**, **MAX** i manje poznata **COUNT**), za *obavljanje aritmetičkih operacija* i *uobličavanje rezultata* (**POWER**, **ROUND**, **TRUNC**, **ABS**, **SIGN**, **MOD**, **SQRT**), kao i za *rad na tekstu*, tj. *sa*

nizovima karaktera (**LENGHT**, **SUBSTR**, **INSTR**, **UPPER**, **LOWER**, **TO_NUM**, **TO_CHAR**, **NVL**, **DECODE**, itd.). Pri pretraživanju baze podataka, u naredbi SELECT se mogu kombinovati funkcije i aritmetičke operacije nad pojedinim atributima i grupisati njihove vrednosti po nekom kriterijumu (atributu).

Aritmetičke funkcije – funkcije za rad sa numeričkim podacima

Za rad sa brojčanim podacima većina SUBP koristi sledeće funkcije:

- **POWER**(broj, e) - izračunava **broj** na stepen **e,**
- **ROUND**(broj[,d]) - zaokružuje **broj** na **d** decimala,
- **TRUNC** (broj[,d]) - odseca **broj** na **d** decimala (nema u MS Access i MS SQL Server),
- **ABS**(broj) - apsolutna vrednost broja,
- **SIGN**(broj) - znak broja (1 za broj>0, 0 za 0, -1 za broj<0),
- **SQRT**(broj) - izračunava kvadratni koren od broja,
- **MOD**(broj1, broj2) - izračunava **broj1 Moduo broj2.**

SQL naredbe mogu imati aritmetičke izraze sastavljene od imena kolona, imena funkcija i konstanti povezanih operatorima (+, *, -,/).

Primer 79. Koji radnici zarađuju više od 10 novčanih jedinica po satu. Zaradu zaokružiti na dve decimale.

```
SELECT ime, ROUND (plata/(22*8), 2) AS [zarada po satu]
FROM RADNIK
WHERE plata/(22*8)>10;
```

Napomena: Ovaj primer je urađen u MS Access-u.

Ime	Zarada po satu
Janko	13,64
Božidar	12,5
Pavle	15,91
Miloš	17,05
Svetlana	15,62
Mitar	14,77
Janko	22,16
Luka	11,36

Funkcije za rad sa nizovima karaktera

Za rad na tekstualnim podacima svaki SUBP ima na raspolaganju veliki broj različitih funkcija i sve one navode se u odredbi SELECT. SUBP Oracle koristi sledeće:

- **CONCAT**(s1,s2) – spaja dva niza karaktera u jedan,
- string1 || string2 – spaja dva niza karaktera u jedan,
- **LENGHT**(string) – određuje dužinu stringa,
- **SUBSTR**(string, spos, [,len]) – daje **podstring** dužine **len** karaktera počev od pozicije **spos,**
- **INSTR**(string, sstring[, spos]) – traži podstring **sstring** u stringu, polazeći od pozicije **spos,**
- **UPPER**(string) – menja sva mala slova u velika,
- **LOWER**(string) – menja sva velika slova u mala,
- **INITCAP**(str) – pretvara prva slova reči u velika,
- **TO_NUM**(string) – pretvara niz numeričkih karaktera u broj,
- **TO_CHAR**(string) – pretvara broj u niz karaktera,
- **LPAD**(string, len[, char]) – popunjava levu stranu stringa **string** karakterima **char** u dužini **len**,
- **RPAD**(string, len[, char]) – popunjava desnu stranu stringa string karakterima **char** u dužini len,
- **NVL**(string1, string2) – ako je string1 Null, vraća string2,

- **DECODE**(string, cs1, rst1, …, dft) – daje rezultat rst1, ako je cs1= string, rst2, ako je cs2= string, u protivnom je dft,
- **SOUNDEX**(Str) – pronalazi reč koja slično zvuči.

Primer 80. Za radnike iz odeljenja 30 prikaži imena i prezimena radnika zajedno sa poslom koji obavljaju i razdvojiti ih razmakom i zapetom. Izveštaj urediti po poslovima.

```
SELECT (Ime+' '+prezime+', '+ posao)  AS Zaposleni
FROM RADNIK
WHERE brod=30
ORDER BY posao;
```

Zaposleni
Marko Nastić, analitičar
Jovan Perić, nabavljač
Andrija Ristić, nabavljač
Pavle Šotra, upravnik

Napomena: U SUBP Oracle za spajanje tekstova (konkatenacija) koristi se operator ||, pa bi isti zadatak urađen Oracle-u bio:

```
SELECT ime ||'  ' || prezime ||', ' || posao Zaposleni
FROM RADNIK
WHERE brod=30
ORDER BY posao;
```

Primer 81. Koliko su duga imena odeljenja?

```
SELECT imeod, LEN(imeod) dužina
FROM ODELJENJE;
```

Imeod	Dužina
Komercijala	11
Plan	4
Prodaja	7
Direkcija	9
Računski centar	15

Napomena: MS SQL Server koristi funkciju LEN za određivanje dužine stringa.

Ovaj primer urađen u SUBP Oracle izgleda ovako:

```
SELECT imeod, LENGHT(imeod) dužina
FROM ODELJENJE;
```

Primer 82. Prikazati prezime i ime radnika velikim slovima razdvojene razmakom, za radnike čija je kvalifikacija KV.

```
SELECT UPPER (prezime+' '+ime) AS [Prezime i ime]
FROM RADNIK
WHERE kvalif='KV';
```

Prezime i ime
VASIĆ PETAR
MARIĆ ALEKSANDAR
PERIĆ JOVAN
DIMIĆ MIRJANA
BOGOVAC TOMISLAV
RISTIĆ ANDRIJA
PETROVIĆ SLOBODAN
PERIĆ JOVAN

Primer 83. Prikaži imena radnika sa najdužim imenom.

```
SELECT ime, LEN(ime) AS [najduže ime]
FROM RADNIK
WHERE LEN(ime) = (SELECT MAX(LEN(ime))
                  FROM RADNIK);
```

Ime	Najduže ime
Aleksandar	10

Napomena: Funkcija **LEN** se koristi nakon popunjavanja baze podataka u cilju redefinicije dužine polja kako bi se uštedeo memorijski prostor. Naime, u fazi sa projektovanjem baze kod tekstualnih podataka, neophodno je proceniti najveću dužinu koja, po pravilu, nije poznata. Da ne bi imali problema pri unosu podataka, za dužinu polja se obično uzimaju veće vrednosti; na primer za ime i prezime 50 znakova. Posle punjenja baze stvarnim podacima, realna dužina podataka se odredi primenom funkcije **LEN**(**GHT**) i onda se primenom naredbi **ALTER**, **MODIFY** izmeni dužina atributa tako da je jednaka maksimalnoj dužini uvećanoj za, na primer, 20%.

Primer 84. Koristeći kolonu *Posao* formirati kolonu *Klasa* tako da posao analitičara bude klase 1, upravnika klase 3, klase direktora 5, a svih drugih klasa 2.

SELECT ime, posao, **DECODE**(Posao, '*analitičar*', 1, '*upravnik*', 3, '*direktor*', 5, 2) Klasa
FROM RADNIK;

Primer je urađen u SUBP Oracle, a odgovor sistema je:

Ime	Posao	Klasa
Petar	vozač	2
Aleksandar	električar	2
Vanja	prodavac	2
Jovan	električar	2
Janko	upravnik	3
Mirjana	čistač	2
Božidar	upravnik	3
Pavle	upravnik	3
Miloš	direktor	5
Svetlana	savetnik	2
Tomislav	električar	2
Andrija	nabavljač	2
Slobodan	vozač	2
Mitar	savetnik	2
Jovan	nabavljač	2
Marko	analitičar	1
Janko	upravnik	3
Ivan	analitičar	1
Luka	analitičar	1

Funkcije za rad sa datumima

Datumi su izuzetno važni za poslovne aplikacije, zbog toga svaki SUBP nudi veliki broj datumskih funkcija. SUBP Oracle koristi sledeće funkcije:

- **ADD_MONTHS**(D,N) - datum D plus N meseci,
- **GREATEST**(D1,D2) - LEAST(D1,D2) - posle većeg ili pre manjeg od dva datuma,
- **LAST_DAY**(D-mesec) – poslednji dan u mesecu u kom je datum,
- **MONTHS_BETWEEN** (SYS DATE, D) - Broj meseci između trenutnog datuma i datuma D,
- **NEXT_DAY**(D, 'Friday') - datum prvog petka posle datuma D,
- **TO_CHAR**(d, format) - pretvara datum u karakter po modelu,
- **TO_DATE**(d, format) - pretvara karakter po modelu u datum.

Primer 85. Prikazati ime i godinu zaposlenja za zaposlene koji obavljaju posao električara.

Ime	Godina

```
SELECT ime, YEAR(datzap) AS godina
FROM RADNIK
WHERE posao='električar';
```

Aleksandar	1990.
Jovan	1980.
Tomislav	1971.

Primer 86. Prikazati ime i godinu zaposlenja i radni staž za zaposlene u odeljenju 20.

```
SELECT ime, datzap, ROUND(CONVERT(INTEGER, (GETDATE()-datzap)) /365, 0) AS
       [Radni staž]
FROM RADNIK
WHERE brod=20;
```

Ime	Datzap	Radni staž
Petar	1978-01-01 00:00:00	27
Božidar	1984-08-11 00:00:00	20
Slobodan	2002-10-03 00:00:00	2
Mitar	2000-03-25 00:00:00	4
Ivan	2003-12-17 00:00:00	1

Napomena: U Oracle postoji slična funkcija pod nazivom **SYSDATE**.

Napomena: Radni staž je podatak koji je suštinski važan za poslovanje firme jer se na bazi njega izračunava plata zaposlenih. Taj podatak je vremenski stalno promenljiv i takav atribut nije pogodan za rad sa bazama podataka (zahtevao bi svakodnevno ažuriranje cele tabele). Ovaj podatak se izračunava pomoću datuma zaposlenja koji je vremenski nepromenljiv, kao razlika tekućeg datuma i datuma zaposlenja.

Funkcije za dobijanje grupnih informacija - Agregatne funkcije

Svaki sistem za upravljanje bazama podataka (SUBP) raspolaže velikim brojem ugrađenih funkcija kojima se dobijaju grupne, sumarne, agregatne informacije koje se odnose na sve redove u nekoj grupi (Aggregate Functions). "U velikim informacionim sistemima vrlo često postoji potreba da se preuzimaju podaci iz drugih IS razmenom datoteka. Tokom punjenja datoteka, treba proveravati i potvrđivati vrednosti podataka u odnosu na ograničenja definisana u bazi podataka. Ako se provera vrši na svakom zapisu pojedinačno, brzina preuzimanja datoteka se smanjuje."[17] "U realnim bazama podataka postoje transakcije koje se odnose na mnogo zapisa koje mogu biti odbijene ako agregatne vrednosti (ne pojedinačne vrednosti, već kao celina) u odgovarajućim poljima skupa zapisa prevaziđu neku graničnu vrednost."[18] Grupa može da se formira pomoću odredbe WHERE i tada se funkcije primenjuju na sve redove koji zadovoljavaju postavljeni uslov. Kada se u uslovu može pojaviti više pojedinačnih vrednosti, za neki atribut mogu se formirati grupe. Grupe se formiraju upotrebom odredbe GROUP BY i mogu se formirati i po više atributa.

Grupne funkcije služe za izračunavanje zbirnih informacija na osnovu podataka iz skupa zapisa u tabelama. Svaki sistem za upravljanje bazama podataka ima svoj skup funkcija i imena koja se koriste (funkcije koje obavljaju istu obradu u dva SUBP imaju različita imena).

[17] [32] S. Ilić, S. Obradović, "Comparison of Methods for Validation of Exceeded Limit Values in Batch Data Loading", 10th Anniversary International Scientific Conference Unitech'10, Technical University Of Gabrovo, Bulgaria, 19-20 November 2010, Pp. I-333-I-338, Issn 1313-230x.

[18] [29] S. Ilić, S. Obradović, "Efficient Validation of Exceeded Limit Values in Batch Data Loading by Database Trigger", International scientific conference UNITECH'09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 393-I 398, ISSN:1313-230X.

Grupne funkcije u MS SQL Serveru

Transact SQL (TSQL) podržava sledeće grupne funkcije:

AVG	Vraća prosečnu vrednost u grupi. Null vrednosti su ignorisane.
BINARY CHECKSUM	Vraća binarnu checksum vrednost proračunatu u okviru vrsta u tabeli ili u okviru liste izraza. BINARY_CHECKSUM može biti upotrebljena da detektuje promene u nekoj vrsti tabele.
CHECKSUM	Vraća checksum vrednost proračunatu u okviru vrsta u tabeli ili u okviru liste izraza.
CHECKSUM AGG	Vraća checksum vrednosti u grupi. Null vrednosti se ignorišu.
COUNT	Vraća broj stavki u grupi.
COUNT BIG	Vraća broj stavki u grupi.COUNT_BIG radi isto kao COUNT funkcija. Jedina razlika je u njihovoj vraćenoj vrednosti. COUNT_BIG uvek vraća BIGINT tip podatka, a COUNT vraća INTEGER.
MAX	Vraća maksimalnu vrednost u izrazu.
MIN	Vraća minimalnu vrednost u izrazu.
SUM	Vraća sumu svih vrednosti, ili samo različitih (DISTINCT) vrednosti u okviru izraza. SUM može biti korišćena samo nad numeričkim kolonama. Null vrednosti su ignorisane.
STDEV	Vraća statističku standardnu devijaciju svih vrednosti u datom izrazu.
STDEVP	Vraća statističku standardnu devijaciju za populaciju svih vrednosti u datom izrazu.
VAR	Vraća statističku varijansu svih vrednosti u datom izrazu.
VARP	Vraća statističku varijansu za populaciju svih vrednosti u datom izrazu.

Kompatibilnost agregatnih funkcija u Oracle-u i SQL serveru

Funkcija	*Oracle*	*SQL server*
Average	**AVG**	**AVG**
Count	**COUNT**	**COUNT**
Maximum	**MAX**	**MAX**
Minimum	**MIN**	**MIN**
Standard deviation	**STDDEV**	**STDEV**
Summation	**SUM**	**SUM**
Variance	**VARIANCE**	**VAR**

Agregatne funkcije u MySQL-u

AVG(*izraz)* - *vraća* prosečnu (average) vrednost izraza.

BIT_AND(*izraz*) - vraća bitwise i svih bita u izrazu. Proračun se obavlja sa 64-bitnom (bigint) preciznošću. Počevši od MySQL 4.0.17 verzije, funkcija vraća 18446744073709551615 ako ne postoje vrste koje se poklapaju. Pre ove verzije funkcija je vraćala 1.

BIT_OR(*izraz*) - vraća bitwise ILI svih bita u izrazu. Proračun se obavlja sa 64-bitnom (bigint) preciznošću. Funkcija vraća 0 ako ne postoje vrste koje se poklapaju.

BIT_XOR(*izraz*) - vraća bitwise XOR svih bita u izrazu. Proračun se obavlja sa 64-bitnom (bigint) preciznošću. Funkcija vraća 0 ako ne postoje vrste koje se poklapaju.

COUNT(*izraz*) - vraća broj Not Null vrednosti u vrstama koje vraća SELECT lista.

COUNT(*) - vraća broj vrsta koje su obuhvaćene upitom, bez obzira na to da li one sadrže ili ne sadrže Null vrednosti.

COUNT(DISTINCT *izraz*,[*izraz*...]) - vraća broj različitih ne-Null vrednosti.

GROUP_CONCAT(*izraz*) - vraća string konkateniranih vrednosti iz grupe. Mogu se eliminisati duplikati.

MIN(*izraz*) - vraća minimalnu vrednost izraza iz grupe.

MAX(*izraz*) - vraća maksimalnu vrednost izraza iz grupe.

STD(*izraz*) - vraća standardnu devijaciju izraza (kvadratni koren od VARIANCE()). Ova funkcija predstavlja proširenje u odnosu na standardni SQL.

STDDEV(*izraz*) - vraća isto što i STD(izraz), koristi se zbog kompatibilnosti sa Oracle-om.

SUM(*izraz*) - vraća sumu izraza u okviru grupe.

VARIANCE(*izraz*) - vraća standardnu varijansu izraza (posmatrajući vraćene vrste kao celu populaciju, ne kao uzorak, pa ima broj vrsta kao denominator). Ova funkcija predstavlja proširenje u odnosu na standardni SQL.

Skup agregatnih funkcija koje su na raspolaganju u MS Accessu dat je u sledećoj tabeli:

FUNKCIJA	OPIS	TIPOVI POLJA
Avg	Prosek vrednosti u jednoj koloni	Svi tipovi, osim: Text, Memo i OLE
Count	Ukupan broj vrednosti nekog polja koja su Not Null	Svi tipovi polja
First	Vrednosti nekog polja u prvom zapisu	Svi tipovi polja
Last	Vrednosti nekog polja u poslednjem zapisu	Svi tipovi polja
Max	Najveća vrednost nekog polja	Svi tipovi osim: Text, Memo i OLE
Min	Najmanja vrednost jednog polja	Svi tipovi osim: Text, Memo i OLE
StDev (); StDevP()	Statistička standardna devijacija vrednosti jednog polja	Svi tipovi osim: Text, Memo i OLE
Sum()	Ukupan zbir vrednosti jednog polja	Svi tipovi osim: Text, Memo i OLE
Var (); VarP ()	Statistička varijansa vrednosti jednog polja	Svi tipovi osim: Text, Memo i OLE

Grupne funkcije nad celom tabelom

Ukoliko ne postoji uslov za selekciju u WHERE odredbi, niti je izvršeno grupisanje po nekom od atributa, onda će se grupne funkcije izvršiti nad svim slogovima jedne tabele.

Primer 87. Prikazati srednju platu u celom preduzeću.

```
SELECT AVG(plata) AS [prosečna plata]
FROM RADNIK;
```

Prosečna plata
1797

Primer 88. Prikazati maksimalnu i minimalnu platu u celom preduzeću.

```
SELECT MIN(plata) AS [najmanja plata], MAX(plata) AS [najveća plata]
FROM RADNIK;
```

Najmanja plata	Najveća plata
900	3900

Primer 89. Prikazati ukupnu platu u celom preduzeću.

```
SELECT SUM(plata) AS [UKUPNA PLATA]
FROM RADNIK;
```

Ukupna plata
34150

Primer 90. Prikazati broj zaposlenih u celom preduzeću.

```
SELECT COUNT (*) AS [BROJ RADNIKA]
FROM RADNIK;

SELECT COUNT(idbr) AS [BROJ RADNIKA]
FROM RADNIK;
```

Broj radnika
19

Sledeća dva primera urađena su u MS Access-u.

Primer 91. Prikaži najmanju, najveću, srednju platu i broj zaposlenih u celom preduzeću.

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveća,
       AVG(plata) AS srednja, COUNT(*) AS [broj zaposlenih]
FROM RADNIK;
```

Najmanja	Najveća	Srednja	Broj zaposlenih
900	3900	1797,36842105263	19

Napomena: U ovom primeru srednja vrednost je izračunata sa velikim brojem decimala, pa je u drugim primerima korišćena funkcija za zaokruživanje (**ROUND**(atribut, n)) na n decimala.

Primer 92. Prikaži najmanju, najveću, srednju platu i broj zaposlenih u celom preduzeću, sa zaokruživanjem na dve decimale.

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveća,
       ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS [broj zaposlenih]
FROM RADNIK;
```

Najmanja	Najveća	Srednja	Broj zaposlenih
900	3900	1797,37	19

Napomena: Da je u ovom primeru srednja vrednost je izračunata korišćenjem funkcije za odsecanje (**TRUNC**(atribut, n)) na n decimala (n = 2), rezultat bi bio tabela:

Najmanja	Najveća	Srednja	Broj zaposlenih
900	3900	1797,36	19

Upotreba WHERE odredbe u grupnim funkcijama

Odredba WHERE se upotrebljava u grupnim funkcijama kada proračune vršimo samo nad n-torkama koji zadovoljavaju dati uslov. Redosled operacija u ovakvim upitima je takav da se najpre izdvoje n-torke koji zadovoljavaju dati uslov, a zatim se izvrši proračun nad izdvojenim n-torkama.

Primer 93. Prikazati srednju platu analitičara.

```
SELECT AVG(plata) AS [prosečna plata]
FROM RADNIK
WHERE posao='analitičar';
```

Primer 94. Prikazati minimalnu i maksimalnu platu zaposlenih sa kvalifikacijom VSS.

```
SELECT MIN(plata) AS [minimalna plata] , MAX(plata) AS  [maksimalna plata]
FROM RADNIK
WHERE kvalif='VSS';
```

Minimalna plata	Maksimalna plata
1300	3900

Primer 95. Prikazati ukupnu platu svih radnika koji ne rade u odeljenjima 30 i 40.

```
SELECT SUM(plata) AS [UKUPNA PLATA]
FROM RADNIK
WHERE brod NOT IN (30,40);
```

Ukupna plata
16200

Primer 96. Prikazati broj zaposlenih koji rade u odeljenju 40.

```
SELECT COUNT (*) AS [BROJ ZAPOSLENIH]
FROM RADNIK
WHERE brod=40;

SELECT COUNT (idbr) AS [BROJ ZAPOSLENIH]
FROM RADNIK
WHERE brod=40;
```

Broj zaposlenih
3

Grupisanje slogova upotrebom odredbe GROUP BY

Osim nad celom tabelom, grupne funkcije se mogu izvršiti i nad nekim atributom ili atributima. Ako agregatne funkcije izvršavamo po nekom atributu (atributima), onda se najpre mora izvršiti grupisanje po tom atributu (atributima). Atributi po kojima se vrši grupisanje moraju biti navedeni u odredbi SELECT, bez obzira na to da li želimo da budu prikazani u rezultatu ili ne (vidljivi ili nevidljivi) ([4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [15], [25], [27], [30], [32]).

Grupne funkcije izvršavaju proračun nad setom vrednosti i vraćaju jednu vrednost. Sa izuzetkom **COUNT**(*), sve druge agregatne funkcije ignorišu Null vrednosti i mogu da ispred argumenta imaju ključnu reč DISTINCT (sve različite vrednosti bez duplikata) ili ALL (sve vrednosti uključujući i duplikate). Ako ništa nije navedeno, podrazumeva se ALL. U slučaju da argument poprima samo vrednost Null, ili ni jedan slog ne zadovoljava uslov selekcije ili grupisanja, sve funkcije osim COUNT vraćaju Null, a COUNT vraća 0. Agregatne funkcije se najčešće koriste u kombinaciji sa GROUP BY odredbom. Sve agregatne funkcije su determinističke, vraćaju istu vrednost svaki put kada su pozvane, a prosleđeni su im isti parametri.

Grupne funkcije su dozvoljene u okviru SELECT liste (glavnog upita ili podupita) i u **HAVING** odredbi.

Primer 97. Prikazati brojeve odeljenja i srednju platu u svakom od njih.

```
SELECT brod, AVG (plata) AS [prosečna plata]
FROM RADNIK
GROUP BY brod;
```

Brod	Prosečna plata
NULL	2000
10	1266
20	1720
30	1575
40	3216

Primer 98. Prikazati za svaku kvalifikaciju minimalnu i maksimalnu platu.

```
SELECT kvalif, MIN(plata) AS [najmanja plata], MAX(plata) AS [najveća plata]
FROM RADNIK
GROUP BY kvalif;
```

Kvalif	Najmanja plata	Najveća plata
KV	900	1300
VKV	1200	1200
VSS	1300	3900

Primer 99. Prikazati za svaki posao ukupnu platu radnika koji ga obavljaju. Rezultate urediti po ukupnim primanjima u opadajućem redosledu.

```
SELECT posao, SUM (plata) AS [ukupna plata]
FROM RADNIK
GROUP BY posao
ORDER BY SUM (plata)  DESC;
```

Posao	Ukupna plata
upravnik	7400
savetnik	5350
analitičar	4900
upravnik	3900
direktor	3000
električar	3000
nabavljač	2200
vozač	2200
prodavac	1200
čistač	1000

Kao i kod agregacija nad celom tabelom, i kod agregacija nad nekim atributom možemo postaviti uslov za proračun. Redosled operacija kod ovakvih upita je sledeći: najpre se izdvoje zapisi koji zadovoljavaju kriterijum, zatim se izdvojeni zapisi grupišu po atributu (atributima) definisanim u GROUP BY odredbi, i na kraju se izvršava agregatna funkcija nad svakom od izdvojenih grupa.

Primer 100. Prikazati brojeve odeljenja i srednju platu u svakom od njih. Iz proračuna isključiti analitičare i upravnike. Rezultate urediti po prosečnim primanjima u rastućem redosledu.

```
SELECT brod, AVG (plata) AS [prosečna plata]
FROM RADNIK
WHERE posao NOT IN ('upravnik' , 'analitičar')
GROUP BY brod
ORDER BY AVG (plata);
```

Primer 101. Prikazati za svaku kvalifikaciju minimalnu i maksimalnu platu. U proračun uključiti samo radnike iz odeljenja 10 i 20.

SELECT kvalif, **MIN**(plata) **AS** [najmanja plata], **MAX(**plata) **AS** [najveća plata]
FROM RADNIK
WHERE brod **IN** (*10,20*)
GROUP BY kvalif;

Kvalif	Najmanja plata	Najveća plata
KV	900	1300
VKV	1200	1200
VSS	1600	2600

Primer 102. Prikazati za svaki posao ukupnu platu radnika koji ga obavljaju. U proračun uzeti u obzir električare, vozače i analitičare.

SELECT posao, **SUM** (plata) **AS** [ukupna plata]
FROM RADNIK
WHERE POSAO **IN** (*'električar', 'vozač', 'analitičar'*)
GROUP BY posao;

Posao	Ukupna plata
analitičar	4900
električar	3000
vozač	2200

Primer 103. Prikaži najmanju, najveću, srednju platu i broj zaposlenih u odeljenju 10.

SELECT MIN(plata) **AS** najmanja, **MAX(**plata) **AS** najveća,
ROUND(**AVG(**plata), 2) **AS** srednja, **COUNT**(*) **AS** [broj zaposlenih]
FROM RADNIK
WHERE brod=*10*;

Najmanja	Najveća	Srednja	Broj zaposlenih
1000	2400	1266,67	6

Napomena: U ovom primeru prikazana je mogućnost selektivnog grupisanja na bazi WHERE odredbe. Ako bismo hteli da saznamo najmanju, najveću, srednju platu i broj zaposlenih i u odeljenjima 20, 30, 40, bilo bi neophodno napisati identične upite. Samo bismo u odredbi WHERE menjali uslov, odnosno postojalo bi više identičnih upita sa različitim uslovima. To se može izbeći upravo korišćenjem GROUP BY odredbe umesto odredbe WHERE.

Primer 104. Prikaži najmanju, najveću, srednju platu i broj zaposlenih po odeljenjima.

SELECT brod, **MIN**(plata) **AS** najmanja, **MAX(**plata) **AS** najveća,
ROUND(**AVG(**plata), 2) **AS** srednja, **COUNT**(*) **AS** [broj zaposlenih]
FROM RADNIK
GROUP BY brod;

Brod	Najmanja	Najveća	Srednja	Broj zaposlenih
	2000	2000	2000	1
10	1000	2400	1266,67	6
20	900	2600	1720	5
30	1100	2800	1575	4
40	2750	3900	3216,67	3

Napomena: U navedenom primeru agregatne funkcije su primenjene na grupe podataka, po odeljenjima. Naime, GROUP BY omogućava dobijanje sumarne informacije za svaku različitu

vrednost kolone po kojoj se vrši grupisanje. Dakle, GROUP BY odredba zamenjuje višestruko pisanje SELECT naredbe sa različitim uslovima.

Grupisanje po više atributa

Grupisanje se može vršiti po više kolona i tada svaka različita kombinacija kolona predstavlja jednu grupu. U okviru dobijenih grupa mogu se uvoditi dodatni uslovi za selekciju primenom odredbe HAVING (*koji imaju*).

Primer 105. Prikazati kvalifikaciju, posao i ukupnu platu radnika sa istom kvalifikacijom koji obavljaju određeni posao. Rezultate urediti po kvalifikaciji u rastućem redosledu.

```
SELECT kvalif, posao, SUM (plata) AS [ukupna plata]
FROM RADNIK
GROUP BY kvalif, posao
ORDER BY kvalif;
```

Kvalif	Posao	Ukupna plata
KV	čistač	1000
KV	električar	3000
KV	nabavljač	2200
KV	vozač	2200
VKV	prodavac	1200
VSS	analitičar	4900
VSS	direktor	3000
VSS	savetnik	5350
VSS	upravnik	7400
VSS	upravnik	3900

Primer 106. Izračunaj broj zaposlenih koji obavljaju različite poslove unutar svakog odeljenja. Rezultate urediti po broju odeljenja u rastućem redosledu.

```
SELECT brod, posao, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK
GROUP BY brod, posao
ORDER BY brod;
```

Brod	Posao	Broj zaposlenih
NULL	analitičar	1
10	čistač	1
10	električar	3
10	prodavac	1
10	upravnik	1
20	analitičar	1
20	savetnik	1
20	upravnik	1
20	vozač	2
30	analitičar	1
30	nabavljač	2
30	upravnik	1
40	direktor	1
40	savetnik	1
40	upravnik	1

Grupne funkcije i upotreba odredbe HAVING (uslovi za grupe)

Da bismo postavili uslove koji se odnose na same grupe, upotrebićemo odredbu **HAVING** (*koji imaju*). Ova odredba koristi se jedino ako u upitu imamo GROUP BY. Dok se odredba WHERE primenjuje na sve redove (zapise, slogove) pre nego što oni i postanu deo grupe, odredba HAVING se primenjuje na već agregiranu vrednost za tu grupu. Naravno da brzina dobijanja odgovora nije ista, jer se pri primeni odredbe HAVING najpre formiraju grupe, uključujući u njih i one n-torke koje ne zadovoljavaju uslove za selekciju. Ukoliko se u upitu koriste sve tri odredbe, onda se to mora uraditi sledećim redosledom:

WHERE uslovi za selekciju n-torki,
GROUP BY ime atributa po kojima se vrši grupisanje, i na kraju
HAVING uslov za izdvajanje podgrupa iz formiranih grupa.

Napomena: Uslove za grupe nije moguće izvršiti odredbom WHERE!

Primer 107. Prikazati brojeve odeljenja i srednju platu u svakom od njih, samo za odeljenja u kojima je srednja plata veća od 2 000.

SELECT brod, **AVG(**plata) AS [srednja plata]
FROM RADNIK
GROUP BY brod
HAVING AVG(plata)>*2000*;

Brod	Srednja plata
40	3216

Primer 108. Prikazati za svaku kvalifikaciju minimalnu i maksimalnu platu, samo za kvalifikacije za koje je minimalna plata veća od 1 000.

SELECT kvalif, **MIN**(plata) **AS** [najmanja plata], **MAX(**plata) **AS** [najveća plata]
FROM RADNIK
GROUP BY kvalif
HAVING MIN(plata)>*1000*;

Kvalif	Najmanja plata	Najveća plata
VKV	1200	1200
VSS	1300	3900

Primer 109. Prikazati za svaki posao ukupnu platu radnika koji ga obavljaju, samo za poslove koje obavlja više od 2 radnika.

SELECT posao, **SUM** (plata) **AS** [ukupna plata]
FROM RADNIK
GROUP BY posao
HAVING COUNT (idbr) > 2;

Posao	Ukupna plata
analitičar	4900
električar	3000
upravnik	7400

Primer 110. Izlistaj šifre radnika i broj projekata na kojima rade za radnike koji rade na dva ili više projekata. Rezultat sortirati po šifri radnika u rastućem redosledu.

```
SELECT UCESCE.idbr, COUNT(idbr) AS [Broj projekata]
FROM UCESCE
GROUP BY UCESCE.idbr
HAVING COUNT(*)>=2
ORDER BY UCESCE.idbr;
```

Id. br	Broj projekata
5652	2
5696	2
5932	3
5953	2
6234	3

Primer 111. Prikaži koje poslove obavlja više od jednog radnika unutar svakog odeljenja.

```
SELECT brod, posao, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK
GROUP BY brod, posao
HAVING COUNT(*)>1;
```

Brod	Posao	Broj zaposlenih
10	električar	3
30	nabavljač	2
20	vozač	2

Primer 112. Prikazati šifru, ime, platu radnika, broj časova angažovanja na projektima i broj projekata na kojima rade za radnike koji rade na dva ili više projekta.

```
SELECT RADNIK.idbr, RADNIK.ime, RADNIK.plata, SUM(UCESCE.brsati)
      AS [broj sati], COUNT (*) AS [broj_projekata]
FROM RADNIK, UCESCE
WHERE RADNIK.idbr = UCESCE.idbr
GROUP BY RADNIK.idbr, RADNIK.ime, RADNIK.plata
HAVING COUNT(*)>=2;
```

Idbr	Ime	Plata	Broj sati	Broj projekata
5652	Jovan	1000	2000	2
5696	Mirjana	1000	4000	2
5932	Mitar	2600	2000	3
5953	Jovan	1100	2000	2
6234	Marko	1300	2000	3

SQL naredbe mogu sadržati aritmetičke izraze sastavljene od funkcija, imena kolona i konstanti povezanih aritmetičkim operatorima (+, -, * i /).

Odredbe GROUP BY i WHERE mogu se koristiti zajedno, pri tome *GROUP BY mora uvek biti iza WHERE odredbe*, jer najpre treba izvršiti selekciju (smanjiti broj n-torki), pa se one grupišu sa GROUP BY, a onda se dodatno izabiraju grupe odredbom **HAVING**. Ključnu reč HAVING mora slediti lista uslova, ali za razliku od WHERE**,** sada se iz rezultata *eliminišu sve one n-torke koje ne zadovoljavaju uslove* navedene u listi. Po pravilu, ova operacija izvodi se nad grupom podataka. Ako nije definisana grupa (opcijom GROUP BY), smatra se da je dobijeni rezultat jedna grupa. Odredba WHERE filtrira po vrednostima koje isključivo postoje u tabelama, a ne nakon primene agregacionih uslova. Na primer, *broj_projekata* ne postoji u tabelama, pa zato ne možemo da napišemo uslov where [broj_projekata] >=2. To ne važi za HAVING, jer on filtrira nakon što se u upitu definišu agregatne funkcije.

Primer 113. Odrediti srednju godišnju platu unutar svakog odeljenja ne uzimajući u obzir plate direktora i upravnika.

```
SELECT brod, AVG(plata)*12 AS  [prosek plata]
FROM RADNIK
WHERE posao NOT IN ('direktor', 'upravnik')
GROUP BY brod;
```

Brod	Prosek plata
	24000
10	12480
20	19200
30	14000
40	39900

Napomena: Srednja godišnja plata je računata kao prosečna mesečna plata pomnožena sa 12 (godina ima 12 meseci).

Upotreba funkcije za dodelu neutralne vrednosti nepostojećem podatku

Ako se grupna funkcija primeni na atribut koji ima Null vrednosti, u izveštaj neće biti uključeni oni slogovi kod kojih je vrednost tog atributa Null.

Kada u izrazima treba koristiti vrednosti kolone koja može imati Null vrednosti, onda treba koloni dodeliti neutralnu vrednost za željenu operaciju, npr. pri sabiranju neutralna vrednost je 0. Dodela neke vrednosti koloni koja sadrži Null vrednosti vrši se funkcijom ISNULL(atrib**,** vrednost) u SQL Serveru ili u Oracle funkcijom NVL(atrib**,** vrednost), a u MySQL je to funkcija IFNULL(atrib**,** vrednost). Tako se vrednost 0 u koloni *premija* u raznim SUBP dodeljuje zaposlenima koji nemaju premiju funkcijom: ISNULL(*premija*,0), NVL(*premija*,0), IFNULL(*premija*,0). U Access-u se u tu svrhu koristi funkcija NZ (*premija*).

Primer 114. Za svako odeljenje prikazati ukupan broj radnika koji primaju premiju.

I)
```
SELECT brod, COUNT (premija) AS [broj radnika sa premijom]
FROM RADNIK
GROUP BY brod;
```

II)
```
SELECT brod, COUNT(*) AS [broj radnika sa premijom]
FROM RADNIK
WHERE premija IS NOT Null
GROUP BY brod;
```

III)
```
SELECT brod, COUNT(idbr) AS [broj radnika sa premijom]
FROM RADNIK
WHERE premija IS NOT Null
GROUP BY brod;
```

Brod	Broj radnika sa premijom
10	5
20	3
30	3
40	1

Primer 115. Za svako odeljenje prikazati broj radnika i ukupna primanja.

Tačan odgovor:

```
SELECT brod, COUNT (idbr) AS [BROJ RADNIKA], SUM (plata+NVL(premija)) AS
       [UKUPNA PRIMANJA]
FROM RADNIK
GROUP BY brod;
```

Brod	Broj radnika	Ukupna primanja
NULL	1	2000
10	6	11300
20	5	15000
30	4	10500
40	3	9660

Netačan odgovor:

SELECT brod,**COUNT** (idbr) **AS** [BROJ RADNIKA], **SUM** (plata+premija) **AS** [UKUPNA PRIMANJA]
FROM RADNIK
GROUP BY brod;

Brod	Broj radnika	Ukupna primanja
NULL	1	NULL
10	6	8900
20	5	10200
30	4	7700
40	3	3910

Primer 116. Odrediti srednja godišnja primanja unutar svakog odeljenja ne uzimajući u obzir plate direktora i upravnika.

a) **SELECT** brod, **AVG(**plata +**ISNULL**(premija,0))*12 **AS** [prosek primanja], **COUNT**(*) **AS** [broj zaposlenih], **SUM**(plata + **ISNULL**(premija,0))*12 **AS** [ukupni prihod]
FROM RADNIK
WHERE posao **NOT IN** (*'direktor', 'upravnik'*)
GROUP BY brod;

Brod	Prosek primanja	Broj zaposlenih	Ukupni prihod
NULL	24000	1	24000
10	21360	5	106800
20	38400	4	153600
30	30792	3	92400
40	39960	2	79920

b) Nije ispravno uraditi zadatak na sledeći način:

SELECT brod, **AVG(**plata +premija)*12 **AS** [prosek primanja],
COUNT(*)**AS** [broj zaposlenih], **SUM**(plata + (premija)*12 **AS** [ukupni prihod],
COUNT(premija) **AS** [sa premijom]
FROM RADNIK
WHERE posao **NOT IN** (*'direktor', 'upravnik'*)
GROUP BY brod;

U prethodnom primeru a) proizvod prosečnih primanja i broja zaposlenih jednak je ukupnom prihodu, dok to nije slučaj u primeru b) kada je pri izračunavanju proseka uzet u obzir samo broj zaposlenih koji imaju premiju (poslednja kolona u izveštaju). Tačnije, pri izračunavanju prosečnih primanja ukupni prihod deljen je samo sa brojem onih koji imaju premiju, a ne sa ukupnim brojem zaposlenih u odeljenju, ne uzimajući u obzir direktora i upravnike.

Skupovni operatori

Dva ili više upita mogu se kombinovati u jednu naredbu upotrebom operatora za rad sa skupovima: **UNION – unija** (**UNION ALL –** unija svih), **INTERSECT - presek** i **MINUS – diferencija**, **razlika**. Da bi se upiti spajali ovim operatorima, liste atributa ili izraza u odredbama svih SELECT naredbi moraju da budu iste po broju i tipu (moraju da imaju iste domene, UNION kompatibilne). Rezultujuća tabela će imati iste nazive kolona kao što su nazivi kolona u prvom upitu. Pošto svi skupovni operatori imaju isti prioritet, najbolje je redosled izračunavanja odrediti upotrebom zagrada.

Operatori unije (engl. *relational union*), preseka (engl. *relational intersection*), razlike (engl. *relational difference*) i Dekartov proizvod (engl. *Cartesian product*) se zasnivaju na teoriji skupova, ali su donekle izmenjeni jer se primenjuju nad relacijama.

Da bi se primenile prve tri operacije, relacije moraju biti Union-kompatibilne, odnosno moraju imati isti broj atributa i odgovarajući atributi moraju imati iste korespondentne domene. Čak je dovoljno da su atributi istog tipa.

Unija je u suštini relaciona verzija sabiranja. Rezultat je relacija koja sadrži sve zapise jedne na koju su dodati zapisi druge relacije, ali su duplikati eliminisani. Unija omogućava dodavanje novih zapisa u relaciju.

Presek je operacija koja vraća zapise koji su zajednički u obe relacije i kojom se u stvari pronalaze duplikati.

Primer 117. Prikazati ime, broj odeljenja i platu za zaposlene koji rade u odeljenju 20 kao i za zaposlene koji imaju platu manju od 1 200 bez obzira u kom odeljenju rade.

Ovaj upit može se rešiti na dva načina, korišćenjem operatora UNION ili ranije spomenutog operatora OR.

```
SELECT ime, brod, plata
FROM RADNIK
WHERE brod=20
        UNION
SELECT ime, brod, plata
FROM RADNIK
WHERE plata<1200;
```

Ime	Brod	Plata
Aleksandar	10	1000
Andrija	30	1100
Božidar	20	2200
Ivan	20	1600
Jovan	10	1000
Jovan	30	1100
Mirjana	10	1000
Mitar	20	2600
Petar	20	1300
Slobodan	20	900
Tomislav	10	1000

Umesto UNION**,** može se koristiti logički operator OR, tako da se ovaj primer može uraditi i na sledeći način:

```
SELECT ime, brod, plata
FROM RADNIK
WHERE brod=20 OR  plata<1200;
```

Korišćenjem operatora **UNION ALL** dobijaju se duplirani zapisi za one slučajeve gde su ispunjena oba uslova, tako da u ovom slučaju postoje i duplikati:

```
SELECT ime, brod, plata
FROM RADNIK
WHERE brod=20
        UNION ALL
SELECT ime, brod, plata
FROM RADNIK
WHERE plata<1200;
```

Ime	Brod	Plata
Petar	20	1300
Božidar	20	2200
Slobodan	**20**	**900**
Mitar	20	2600
Ivan	20	1600
Aleksandar	10	1000
Jovan	10	1000
Mirjana	10	1000
Tomislav	10	1000
Andrija	30	1100
Slobodan	**20**	**900**
Jovan	30	1100

S obzirom na to da UNION ALL vraća i duplikate, operator OR se ne može koristiti umesto ovog operatora.

Napomena: Korišćenjem operatora UNION ALL u rezultatu smo dobili zapis o radniku Slobodanu dva puta jer on zadovoljava oba postavljena uslova, da je iz odeljenja 20 i da mu je plata manja od 1200.

Napomena: Neki SUBP ne podržavaju svaku od ovih funkcija. Tako, na primer, SQL server (starije verzije), MS Access imaju samo UNION, a INTERSECT i MINUS ne podržavaju. Ove operacije se mogu simulirati, odnosno napraviti pomoću operacije spajanja (JOIN). U Oracle-u i MySQL-u možemo napraviti primere za ove operacije, pri tome treba imati u vidu sa se za operaciju razlike (minus) u SQL server-u i MySQL koristi odredba **EXCEPT**. Sledeća dva primera urađena su u SUBP Oracle.

Primer 118. Prikazati imena zaposlenih i kvalifikacije radnika iz odeljenja 10, ali ne one čija je kvalifikacija KV.

```
SELECT ime, kvalif
FROM RADNIK
WHERE brod=10
        MINUS
SELECT ime, kvalif
FROM RADNIK
WHERE kvalif='KV';
```

Primer 119. Prikazati imena zaposlenih i kvalifikacije radnika iz odeljenja 10 i koji imaju kvalifikaciju KV.

```
SELECT ime, kvalif
FROM RADNIK
WHERE brod=10
        INTERSECT
SELECT ime, kvalif
FROM RADNIK
WHERE kvalif='KV';
```

Napomena: Isti rezultat može se dobiti i primenom logičkog operatora AND.

```
SELECT ime, kvalif
FROM RADNIK
WHERE brod=10 AND kvalif='KV';
```

Upiti nad jednom relacijom kao argument u drugom upitu i spajanje relacija

Zajednička karakteristika svih do sada prezentiranih primera bila je **postavljanje upita uvek samo nad jednom relacijom,** jer se odgovor uvek mogao naći unutar jedne relacije. Manipulisanje informacionim sistemom zahteva, međutim, često dobijanje odgovora za koje upit treba postaviti nad dve ili više relacija istovremeno. U tom slučaju moraju se koristiti ugnježđeni upiti ili se mora preduzeti operacija spajanja dobijenih tabela (najčešće prirodnog spajanja) sa nekim projekcijama i/ili restrikcijama nad njima.

Ugnježđeni upiti (podupiti)

Ugnježđeni upiti se koriste onda kada se sve informacije koje u upitu želimo da prikažemo nalaze u jednoj tabeli (SELECT lista je iz jedne tabele), ali su informacije preko kojih postavljamo uslove u istoj ili drugim tabelama. Ugnježđeni upit (podupit, sub query) je ugrađen u neki drugi upit. Rezultat podupita se koristi kao konkretna vrednost (ako se radi o upitu koji vraća jedan slog, single row sub query). Ako podupit vraća skup vrednosti, mora se koristiti odredba IN. Podupiti se mogu navesti i u odredbi WHERE i u odredbi FROM. Ako se podupit nalazi na levoj strani nekog relacionog operatora tipa: =, >=, !=, <=, <, > mora vraćati samo jedan slog. Tip kolone podupita mora biti isti kao tip izraza na levoj strani operatora. Podupit može da vrati više slogova ako se navede desno od operatora IN ili relacionih operatora iza koji sledi reč **ANY** ili **ALL** (na primer: = **ANY**, > **ALL**).

U podupitima se može koristiti i operator **EXISTS** (engl. postoji). To je unarni logički operator iza kojeg sledi podupit. Njegova vrednost je istinita ako podupit izdvaja bar jedan slog. Ovaj operator vrlo često se može zameniti operatorom **IN**, ali **EXSISTS**, po pravilu, daje rezultate od operatora **IN**.

Podupit se može navesti umesto naziva tabele iza odredbe **FROM**. Tada tabela koja predstavlja rezultat podupita, određuje virtuelnu tabelu iz koje se izdvajaju n-torke (slogovi). Nazivi kolona u spoljnom upitu moraju biti podskup skupa naziva kolona u unutrašnjem upitu.

Napomena: Ranije verzije MySQL nisu podržavale ugnježđene upite, dok novije verzije podržavaju i tu mogućnost.

Primer 120. Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

Potrebna je sledeća informacija: spisak imena svih zaposlenih u odeljenju koje je locirano na Dorćolu. To se postiže ulaganjem rezultata jednog upita u **WHERE** odredbu drugog upita. Prvi, unutrašnji upit, trebao bi da iz relacije ODELJENJE pruži odgovor na to koja je šifra odeljenja (brod) koje je locirano na *Dorćolu*. Kada dobijemo da je to odeljenje čija je šifra brod=*20*, onda drugim, spoljnim upitom, iz relacije RADNIK tražimo spisak imena zaposlenih u odeljenju gde je brod=*20*.

I) Ovaj upit vraća broj odeljenja koje je smešteno na Dorćolu:

```
SELECT brod
FROM ODELJENJE
WHERE mesto='Dorćol';
```

Dobili smo da je broj odeljenja koje je smešteno na Dorćolu 20.

II) Ovaj upit vraća imena zaposlenih koji rade u odeljenju 20:

```
SELECT ime, brod
FROM RADNIK
WHERE brod=20;
```

Naravno, ovo nije način kako se rešavaju zadaci postavljani na navedeni način. Ovaj zadatak se može rešiti korišćenjem ugnježdenih upita na sledeći način:

III) **SELECT** ime, brod
FROM RADNIK
WHERE RADNIK.brod = (**SELECT** ODELJENJE.brod
FROM ODELJENJE
WHERE ODELJENJE.mesto=*'Dorćol'*);

Ime	Brod
Petar	20
Božidar	20
Slobodan	20
Mitar	20
Ivan	20

Primer 121. Prikazati ime i posao svih radnika koji rade na Novom Beogradu.

SELECT ime, posao
FROM RADNIK
WHERE brod **IN** (**SELECT** brod
FROM ODELJENJE
WHERE mesto=*'Novi Beograd'*);

Napomena: Ako ugnježdeni upit vraća više od jednog zapisa, onda u uslovu klauzale WHERE treba napisati IN, a ne znak jednakosti (=). Znak jednakosti se koristi samo u slučajevima kada je izvesno da podupit vraća samo jedan zapis.

Primer 122. Prikazati ime, posao i kvalifikaciju svih radnika koji imaju istu kvalifikaciju kao Mitar.

SELECT ime, posao, kvalif
FROM RADNIK
WHERE kvalif **IN** (**SELECT** kvalif
FROM RADNIK
WHERE ime=*'Mitar'*);

Ime	Posao	Kvalif
Janko	upravnik	VSS
Božidar	upravnik	VSS
Pavle	upravnik	VSS
Miloš	direktor	VSS
Svetlana	savetnik	VSS
Mitar	savetnik	VSS
Marko	analitičar	VSS
Janko	upravnik	VSS
Ivan	analitičar	VSS
Luka	analitičar	VSS

Primer 123. Izlistaj ime, posao i platu zaposlenih u odeljenju *10*, koji imaju isti posao kao zaposleni u odeljenju *Plan*.

SELECT ime, posao, plata
FROM RADNIK
WHERE brod=*10* **AND** posao **IN** (**SELECT** posao
FROM RADNIK
WHERE brod **IN** (**SELECT** brod
FROM ODELJENJE
WHERE imeod=*'plan'*));

Ime	Posao	Plata
Janko	upravnik	2400

Primer 124. Prikazati ime i ukupna primanja svih zaposlenih koji imaju isti posao kao Slobodan.

```
SELECT ime, plata+ISNULL(premija, 0) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE  posao IN (SELECT posao
                 FROM RADNIK
                 WHERE IME='Slobodan');
```

Ime	Ukupna primanja
Petar	3200
Slobodan	2200

Napomena: U rezultatu upita dobijena su dva zapisa za radnike Petra i Slobodana koji obavljaju posao vozača, a taj uslov smo dobili iz postavljenog ugnježdenog upita (podupita). Ako je potrebno da izostavimo prikazivanje radnika Slobodana, već da budu prikazani samo oni radnici koji obavljau posao kao on, upit bi glasio:

```
SELECT ime, plata+ISNULL(premija, 0) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE  ime<>'Slobodan' AND posao  IN  (  SELECT posao
                                         FROM RADNIK
                                         WHERE IME='Slobodan');
```

Ime	Ukupna primanja
Petar	3200

Primer 125. Prikazati ime, datum zaposlenja i posao zaposlenih koji imaju kvalifikaciju kao Marko i zaposleni su 1990. godine (primer je urađen u Ms Access-u).

```
SELECT ime, datzap, posao
FROM RADNIK
WHERE   kvalif   IN  ( SELECT kvalif
                       FROM RADNIK
                       WHERE ime='Marko')
AND datzap BETWEEN #1/1/1990# AND #31/12/1990# ;
```

Napomena: Isti primer može se uraditi korišćenjem funkcije **YEAR** koja iz datuma izdvaja godinu.

```
SELECT ime, datzap, posao
FROM RADNIK
WHERE  kvalif  IN  ( SELECT kvalif
                     FROM RADNIK
                     WHERE ime='Marko')
AND YEAR(datzap)=1990;
```

Ime	Datzap	Posao
Marko	1990-12-17 00:00:00	analitičar

Primer 126. Prikazati ime, posao i ukupna primanja radnika koji rade na Novom Beogradu ne uzimajući u obzir prodavce i upravnike. Rezultate sortirati po ukupnim primanjima u opadajućem redosledu.

```
SELECT ime, posao, plata+ISNULL(premija, 0) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE brod IN (SELECT brod
                FROM ODELJENJE
                WHERE mesto='Novi Beograd')
AND posao NOT IN ('prodavac', 'upravnik')
ORDER BY plata+ISNULL(premija, 0) DESC;
```

Napomena: Ovaj primer može se uraditi u MS Access-u:

```
SELECT ime, posao, plata+NZ(premija) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE brod IN (SELECT brod
                FROM ODELJENJE
                WHERE mesto='Novi Beograd')
AND posao NOT IN ('prodavac','upravnik')
ORDER BY plata+NZ(premija) DESC;
```

Isti primer urađen u SUBP Oracle bio bi:

```
SELECT ime, posao, plata+NVL(premija) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE  brod  IN  ( SELECT brod
                   FROM ODELJENJE
                   WHERE mesto='Novi Beograd')
AND posao NOT IN ('prodavac','upravnik')
ORDER BY plata+NVL(premija, 0) DESC;
```

Primer 127. Prikazati ime i kvalifikaciju svih zaposlenih koji imaju istu platu kao Jovan. Rezultate urediti po imenu u opadajućem redosledu.

```
SELECT ime, kvalif
FROM RADNIK
WHERE plata IN (SELECT plata
                FROM RADNIK
                WHERE ime='Jovan')
ORDER BY ime DESC;
```

Ime	Kvalif.
Tomislav	KV
Mirjana	KV
Jovan	KV
Jovan	KV
Andrija	KV
Aleksandar	KV

Primer 128. Prikazati ime i kvalifikaciju svih radnika koji rade na istim projektima kao Marko.

```
SELECT ime, kvalif
FROM RADNIK
WHERE idbr IN (SELECT DISTINCT idbr
               FROM UCESCE
               WHERE brproj IN (SELECT brproj
                                FROM UCESCE
                                WHERE idbr IN (SELECT idbr
                                               FROM RADNIK
                                               WHERE ime='Marko')));
```

Primer 129. Prikazati idbr, ime, platu i kvalifikaciju zaposlenih koji imaju istu platu kao bilo koji zaposleni čija je kvalifikacija VSS.

```
SELECT idbr, ime, plata, kvalif
FROM RADNIK
WHERE plata = ANY (SELECT plata
                   FROM RADNIK
                   WHERE kvalif='VSS');
```

```
SELECT idbr, ime, plata, kvalif
FROM RADNIK
WHERE plata = SOME (SELECT plata
                    FROM RADNIK
                    WHERE kvalif='VSS');
```

Idbr	Ime	Plata	Kvalif
5367	Petar	1300	KV
5662	Janko	2400	VSS
5780	Božidar	2200	VSS
5786	Pavle	2800	VSS
5842	Miloš	3000	VSS
5867	Svetlana	2750	VSS
5932	Mitar	2600	VSS
6234	Marko	1300	VSS
6789	Janko	3900	VSS
7890	Ivan	1600	VSS
7892	Luka	2000	VSS

Napomena: U rezultatu ovog upita pojavio se zapis o radniku Petru, kvalifikacije KV, koji ima platu 1300 istu kao radnik Marko, kvalifikacije VSS.

Primer 130. Prikazati idbr, ime, platu i kvalifikaciju zaposlenih koji imaju platu manju od svih zaposlenih čija je kvalifikacija VSS.

```
SELECT idbr, ime, plata, kvalif
FROM RADNIK
 WHERE plata < ALL (SELECT plata
                    FROM RADNIK
                    WHERE kvalif='VSS');
```

Idbr	Ime	Plata	Kvalif
5497	Aleksandar	1000	KV
5519	Vanja	1200	VKV
5652	Jovan	1000	KV
5696	Mirjana	1000	KV
5874	Tomislav	1000	KV
5898	Andrija	1100	KV
5900	Slobodan	900	KV
5953	Jovan	1100	KV

Primer 131. Prikazati sve podatke o odeljenjima u kojima ima zaposlenih radnika.

```
SELECT *
FROM ODELJENJE
WHERE  EXISTS ( SELECT brod
                FROM radnik
                WHERE RADNIK.brod=ODELJENJE.brod);
```

Brod	Imeod	Mesto	Sefod
10	Komercijala	Novi Beograd	5662
20	Plan	Dorćol	5780
30	Prodaja	Stari Grad	5786
40	Direkcija	Banovo Brdo	5842

Napomena: Kao rezultat ovog upita nije dobijen zapis o odeljenju 50, jer u tom odeljenju nema zaposlenih.

Primer 132. Prikazati sve podatke o odeljenjima u kojima nema zaposlenih radnika.

```
SELECT *
FROM ODELJENJE
WHERE  NOT EXISTS ( SELECT brod
                    FROM radnik
                    WHERE RADNIK.brod=ODELJENJE.brod);
```

Brod	Imeod	Mesto	Sefod
50	Računski centar	Zemun	NULL

Primer 133. Prikazati sve podatke o zaposlenim koji rade na nekom projektu.

```
SELECT *
FROM RADNIK
WHERE EXISTS (SELECT idbr
              FROM UCESCE
              WHERE UCESCE.idbr=RADNIK.idbr);
```

Primer 134. Prikazati sve podatke o zaposlenim koji ne rade ni na jednom projektu.

```
SELECT *
FROM RADNIK
WHERE NOT EXISTS (SELECT idbr
                  FROM UCESCE
                  WHERE UCESCE.idbr=RADNIK.idbr);
```

Napomena: U ugnježđenim upitima mogu se koristiti funkcije i grupisanja po raznim kriterijumima.

Primer 135. Prikazati imena odeljenja u kojima su ukupna primanja svih radnika u odeljenju veća od 10000.

```
SELECT odeljenje.imeod
FROM ODELJENJE
WHERE  odeljenje.brod  IN  (SELECT brod
                            FROM RADNIK
                            GROUP BY brod
                            HAVING SUM (plata+ISNULL(premija, 0)) > 10000);
```

Imeod
Komercijala
Plan
Prodaja

Primer 136. Prikazati ime i primanja svih zaposlenih čija su primanja veća od prosečnog primanja u preduzeću.

```
SELECT ime, plata+ ISNULL(premija, 0) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE plata+ ISNULL(premija, 0)> (SELECT AVG (plata+ISNULL(premija, 0))
                                  FROM RADNIK);
```

Ime	Ukupna primanja
Petar	3200
Pavle	2800
Miloš	3000
Svetlana	2750
Mitar	2600
Marko	4300
Janko	3910
Ivan	4800

Primer 137. Prikazati imena svih radnika čija su ukupna primanja manja od prosečnih primanja u odeljenju čije je sedište u Starom Gradu.

```
SELECT ime
FROM RADNIK
WHERE plata+ISNULL(premija, 0)< (SELECT AVG(plata+ISNULL(premija, 0))
                                 FROM RADNIK
                                 WHERE brod IN (SELECT brod
                                                FROM ODELJENJE
                                                WHERE mesto='Stari Grad'));
```

Primer 138. Prikazati ime, datum zaposlenja i primanja zaposlenih koji su angažovani na dva projekta. Rezultate urediti po datumu zaposlenja u opadajućem redosledu.

```
SELECT ime, datzap, plata+ISNULL(premija, 0) AS [UKUPNA PRIMANJA]
FROM RADNIK
WHERE  idbr  IN (  SELECT idbr
                   FROM UCESCE
                   GROUP BY idbr
                   HAVING COUNT (idbr)>2)
ORDER BY datzap;
```

Ime	Datzap	Ukupna primanja
Marko	1990-12-17 00:00:00	4300
Mitar	2000-03-25 00:00:00	2600

Primer 139. Prikazati imena i posao radnika čija su ukupna primanja manja od najmanjih primanja u odeljenju smeštenom u Starom Gradu. Rezultate poređati po imenu u rastućem redosledu.

```
SELECT ime, posao
FROM RADNIK
WHERE plata + ISNULL(premija, 0) < (SELECT MIN (plata+ISNULL(premija, 0))
                                    FROM RADNIK
                                    WHERE brod IN (SELECT brod
                                                   FROM ODELJENJE
                                                   WHEREmesto='Stari Grad'))
ORDER BY 1;
```

Ime	Posao
Mirjana	čistač

Primer 140. Prikazati ime, datum zaposlenja i primanje zaposlenih koji su angažovani na dva projekata. Rezultate urediti po datumu zaposlenja u opadajućem redosledu.

```
SELECT ime, datzap, plata+ISNULL(premija, 0) AS primanje
FROM RADNIK
WHERE idbr IN (SELECT idbr
               FROM UCESCE
               GROUP BY idbr
               HAVING COUNT (idbr)=2)
ORDER BY datzap DESC;
```

Primer 141. Ko su najbolje plaćeni radnici u celom preduzeću?

```
SELECT ime, brod, posao, plata
FROM RADNIK
WHERE  plata  =  ( SELECT MAX(plata)
                   FROM RADNIK);
```

Ime	Brod	Posao	Plata
Janko	40	upravnik	3900

Napomena: Spoljašnji i unutrašnji upit mogu biti povezani po vrednostima više atributa. U tom slučaju, ako se upoređuju argumenti koji se sastoje od više atributa, oba argumenta moraju imati jednak broj atributa, a upoređuje se prvi sa prvim, drugi sa drugim itd. Konačno, napomenimo da atributi koji se upoređuju moraju biti istog ili kompatibilnog tipa podataka.

Primer 142. Ko su najbolje plaćeni radnici u svakom odeljenju.

Sledeći upit će samo slučajno dati tačan rezultat, jer u bazi ima malo podataka i nije se desio slučaj koji bi nam omogućio da na osnovu dobijenog rezultata zaključimo da je postavljeni upit netačan. Ako bi bilo kom radniku iz odeljenja 30 (na primer radniku sa idbr=5896) privremeno upisali platu 2600 (koliko iznosi i najveća plata u odeljenju 20) onda bi dobili odgovor na osnovu kojeg bi mogli da zaključimo da ovaj upit nije ispravan. Najveća plata u odeljenju 30 je 2800, a pojavio bi se i radnik u odeljenju 30 sa platom 2600.

```
SELECT ime, brod, plata
FROM RADNIK
WHERE  plata  IN  ( SELECT MAX(plata)
                    FROM RADNIK
                    GROUP BY brod );
```

Ime	Brod	Plata
Janko	10	2400
Pavle	30	2800
Andrija	**30**	**2600**
Mitar	20	2600
Janko	40	3900
Luka	NULL	2000

Da bi napravili ispravan upit, potrebno je izjednačiti vrednosti grupe atributa ili izraza u odredbi WHERE spoljnjeg upita i vrednosti grupe atributa ili izraza u odredbi SELECT unutrašnjeg upita. Ovo u mnogim SUBP nije moguće (Access, SQL Server, ...). Samo neki SUBP ovo podržavaju, na primer Oracle.[19]

[19] To naravno ne znači da se ova informacija ne može dobiti u Acces-u ili SQL Serveru. To je moguće korišćenjem upita nad upitima u odredbi FROM. Videti Primer 188.

Primer urađen u SUBP Oracle:

```
SELECT ime, brod, plata
FROM RADNIK
WHERE (brod, plata) IN (SELECT brod, MAX(plata)
                        FROM RADNIK
                        GROUP BY brod );
```

Ime	Brod	Plata
Janko	10	2400
Pavle	30	2800
Mitar	20	2600
Janko	40	3900
Luka	NULL	2000

Primer 143. Prikaži imena radnika koji su se poslednji zaposlili u svakom odeljenju.

Primer urađen u SUBP Oracle:

```
SELECT ime, datzap, brod
FROM RADNIK
WHERE  (datzap, brod)  IN  ( SELECT MAX(datzap), brod
                             FROM RADNIK
                             GROUP BY brod);
```

Ime	Datum zaposlenja	Brod
Janko	1993-08-12 00:00:00.000	10
Marko	1990-12-17 00:00:00.000	30
Janko	2003-12-23 00:00:00.000	40
Ivan	2003-12-17 00:00:00.000	20
Luka	2004-05-20 00:00:00.000	NULL

Složenija pretraživanja, kao što smo to videli u prethodnim primerima, koriste obavezno takozvana "potpretraživanja" to jest, pretraživanja rezultata prethodnog pretraživanja. Pre nego što započne izvršavanje spoljnjeg upita, unutrašnji upit je već izvršen i njegov rezultat je poznat i već su konkretne vrednosti smeštene u memoriju računara. Naredba SELECT daje, naime, kao rezultat relaciju (koja ne postoji i fizički, na disku, ali je dostupna za vreme izvođenja te naredbe u vidu relacije u memoriji računara) koja se može dalje pretraživati. Ovakav pristup je moguć samo u slučaju da se u konačnom izveštaju pojavljuju samo podaci iz jedne relacije, kao što je u svim prethodnim primerima bio slučaj. Dakle, povezivanje tabela dinamičkom zamenom rezultata jednog, unutrašnjeg upita u WHERE odredbu drugog, spoljnjeg upita, može se primeniti samo ako su svi podaci koji se prikazuju u spoljnjem upitu iz jedne tabele.

Ali u nekim slučajevima u rezultatu spoljnjeg upita kombinuju se podaci iz više tabela. U tom slučaju, mora se izvršiti spajanje (**JOIN**) dveju ili više tabela. Spajanje tabela vrši se korišćenjem zajedničkih atributa, tj. atributa koji su definisani nad istim domenima. Najčešće se koristi spajanje po jednakosti ili **EQUIJOIN**.

Spajanje relacija

Spajanje tabela se koristi kada se informacije koje želimo da prikažemo (lista atributa u SELECT odredbi) nalaze u više tabela. Pomoću spajanja se dobijaju informacije iz dve ili više tabela kao jedan skup rezultata (recordset, resultset). Skup rezultata predstavlja "virtuelnu" tabelu koja postoji samo u memoriji računara (ali ne i na disku). Kako JOIN spaja informacije iz više tabela u jednu, rezultat zavisi od toga kako zahtevamo da se spajaju podaci.

Postoje različite vrste spajanja. Na primer, ako su nam potrebni podaci iz tabele RADNIK i UČEŠĆE istovremeno, potrebno je da se uverimo da su njihova odgovarajuća polja

spojena (JOIN). Spajanje se najčešće vrši preko zajedničkih atributa, u ovom primeru to je atribut *idbr*. U slučaju da tabele u upitu nisu povezane bilo direktno ili indirektno, ne postoji način da DBMS utvrdi koji su zapisi iz jedne tabele povezani sa zapisima iz druge tabele, pa kao odgovor vraća sve kombinacije zapisa. Ovakav rezultat predstavlja Dekartov ili Kartezijev proizvod ("cross-product" ili "Cartesian product"). Na primer, ako bismo imali dve tabele sa po 10 zapisa, rezultat bi se sastojao iz 100 zapisa (10 x 10). To takođe znači da bi se upit dugo izvršavao i da bi vraćao rezultate bez ikakvog smisla.

Tabela se može spajati i sa samom sobom (SELF JOIN), kada unutar tabele postoji relacija između pojedinih n-torki. To je slučaj sa tabelom RADNIK jer su neki zaposleni rukovodioci nekim drugim radnicima. Kao što smo to već napomenuli, imena atributa mogu se pisati navođenjem i imena relacije kojoj taj atribut pripada (RADNIK.brod ili ODELJENJE.brod), a ovo je neophodno uraditi ako se odlučimo da u različitim relacijama jedne baze podataka koristimo ista imena atributa. Kako je uobičajeno da spoljni ključevi imaju ista imena kao odgovarajući primarni ključevi, pri spajanju tabela neophodno je navesti ime tabele i ime atributa.

Unutrašnje spajanje (INNER JOIN)

Ova vrsta spajanja je najčešće u upotrebi. Ona spaja zapise na osnovu jednog ili više zajedničkih polja (kao i većina naredbi JOIN), ali vraća samo zapise u kojima su jednake vrednosti jednog ili više polja preko kojeg (kojih) se vrši spajanje. Zapis je odgovarajući samo ako postoje identične vrednosti u poljima koja spajaju tabele. U većini slučajeva spajanje se vrši preko polja jedinstvenog primarnog ključa u jednoj tabeli i polja spoljnjeg ključa u drugoj tabeli (u relaciji jedan prema više). Ako za neku vrednost primarnog ključa jedne tabele u drugoj tabeli ne postoji ni jedan odgovarajući zapis, taj zapis se ne pojavljuje u rezultatu upita.

Primer 144. Prikazati za svakog radnika ime, posao, kao i broj odeljenja, naziv i mesto odeljenja u kome radi.

Pogledajmo šta će se desiti ako ne povežemo tabele:

```
SELECT R.ime, R.posao, O.brod, O.imeod, O.mesto
FROM ODELJENJE O, RADNIK R;
```

Napomena: Kada se ne uradi povezivanje tabela u upitu, kao rezultat upita dobije se Dekartov proizvod, što znači da bi kao rešenje prethodnog upita dobili 95 (19x5) zapisa, tj. povezao bi se svaki zapis iz tabele RADNIK sa svakim zapisom iz tabele ODELJENJE.

Primenimo sada **INNER JOIN** na ovaj upit kako bismo izbegli Dekartov proizvod:

```
SELECT RADNIK.ime, RADNIK.posao, ODELJENJE.brod, ODELJENJE.imeod,
       ODELJENJE.mesto
FROM RADNIK INNER JOIN ODELJENJE ON RADNIK.brod=ODELJENJE.brod;
```

Ime	Posao	Brod	Imeod	Mesto
Petar	vozač	20	Plan	Dorćol
Aleksandar	električar	10	Komercijala	Novi Beograd
Vanja	prodavac	10	Komercijala	Novi Beograd
Jovan	električar	10	Komercijala	Novi Beograd
Janko	upravnik	10	Komercijala	Novi Beograd
Mirjana	čistač	10	Komercijala	Novi Beograd
Božidar	upravnik	20	Plan	Dorćol
Pavle	upravnik	30	Prodaja	Stari Grad
Miloš	direktor	40	Direkcija	Banovo Brdo
Svetlana	savetnik	40	Direkcija	Banovo Brdo

Tomislav	električar	10	Komercijala	Novi Beograd
Andrija	nabavljač	30	Prodaja	Stari Grad
Slobodan	vozač	20	Plan	Dorćol
Mitar	savetnik	20	Plan	Dorćol
Jovan	nabavljač	30	Prodaja	Stari Grad
Marko	analitičar	30	Prodaja	Stari Grad
Janko	upravnik	40	Direkcija	Banovo Brdo
Ivan	analitičar	20	Plan	Dorćol

Napomena: Spajanjem tabela RADNIK i ODELJENJE preko atributa brod u ovom upitu dobili smo traženi rezultat. Kao rezultat dobijeno je 18 zapisa, iako imamo 19 zaposlenih u preduzeću. Podaci o radniku sa šifrom 7892 (Luka Bošković) nisu prikazani jer on još nije raspoređen u neko odeljenje.

Primer 145. Prikazati za svakog radnika ime, posao i sve informacije o projektu na kome radi.

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj,
       PROJEKAT.sredstva, PROJEKAT.rok
FROM RADNIK INNER JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr
            INNER JOIN PROJEKAT ON PROJEKAT.brproj=UCESCE.brproj;
```

Primer urađen u MS Accessu:

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj,
       PROJEKAT.sredstva, PROJEKAT.rok
FROM RADNIK INNER JOIN (PROJEKAT INNER JOIN UCESCE ON
PROJEKAT.brproj = UCESCE.brproj) ON RADNIK.idbr = UCESCE.idbr;
```

Napomena: U ovom primeru vršeno je povezivanje tri tabele, tabele RADNIK i UCESCE povezane su preko atributa *idbr*, a tabele PROJEKAT i UCESCE preko atributa *brproj*. **Ako bi izostavili zagrade u izrazu za spajanje izraz ne bi bio sintaksno ispravan.**

Alternativa za INNER JOIN (spajanje pomoću WHERE odredbe)

Pored upotrebe INNER JOIN-a tabele se mogu spojiti pomoću odredbe WHERE.

Primer 146. Uradimo sada Primer 144 upotrebivši odredbu WHERE umesto INNER JOIN. Prikazati za svakog radnika ime, posao, kao i broj odeljenja, naziv i mesto odeljenja u kome radi.

```
SELECT RADNIK.ime, RADNIK.posao, ODELJENJE.brod, ODELJENJE.imeod,
       ODELJENJE.mesto
FROM RADNIK, ODELJENJE
WHERE RADNIK.brod=ODELJENJE.brod;
```

Primer 147. Uradimo sada Primer 145 upotrebivši odredbu WHERE umesto INNER JOIN. Prikazati za svakog radnika ime, posao i sve informacije o projektu na kome radi.

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj,
       PROJEKAT.sredstva, PROJEKAT.rok
FROM RADNIK, UCESCE, PROJEKAT
WHERE RADNIK.idbr=UCESCE.idbr AND PROJEKAT.brproj=UCESCE.brproj;
```

Primer 148. Izlistaj spisak imena i prezimena zaposlenih koji rade na Dorćolu.

```
SELECT ime, prezime
FROM RADNIK INNER JOIN ODELJENJE ON RADNIK.brod= ODELJENJE.brod
WHERE mesto='Dorćol' ;

SELECT ime, prezime
FROM RADNIK, ODELJENJE
WHERE RADNIK.brod= ODELJENJE.brod AND mesto='Dorćol' ;
```

Ime	Prezime
Petar	Vasić
Božidar	Ristić
Slobodan	Petrović
Mitar	Vuković
Ivan	Buha

U prethodnim primerima u odredbi WHERE korišćeno je spajanje po jednakosti (ODELJENJE.brod= RADNIK.brod), ali moguće je spajanje po bilo kom operatoru poređenja. Ključnu reč WHERE**,** u naredbi SELECT moraju da slede uslovi koje treba da zadovolje podaci u n-torkama iz određene relacije, a koji se žele dobiti kao rezultat. Ako se ne navede nikakav uslov, onda naredba SELECT daje Kartezijev proizvod (**CARTESIAN JOIN**) relacija navedenih u listi relacija. To je iz matematike poznati Dekartov proizvod dva skupa. Tako, na primer, naredba:

SELECT * FROM A, B;

daje Kartezijev proizvod relacija A i B. Ako relacija A ima 1 000 n-torki i relacija B ima 1 000 n-torki onda nova relacija dobijena spajanjem ovih dveju relacija ima 1.000*1.000, tj. 1.000.000 n-torki. Dakle, nova relacija ima veoma veliki broj zapisa. Sada sledi pretraživanje tog skupa i izdvajanje samo onih n-torki koje zadovoljavaju uslov iz odredbe WHERE. Ovo je vrlo dugotrajan postupak, za razliku od tehnike ugnježdenih upita, gde se najpre izvrši selekcija n-torki koje zadovoljavaju uslov, a tek onda tako redukovan skup se proverava u sledećem upitu. Iako se ugnježdeni upiti uvek mogu realizovati putem operacije prirodnog spajanja, taj postupak se ne preporučuje zbog drastičnog povećanja vremena obrade upita.

Primer 149. Izlistaj ime i posao zaposlenih, broj, ime odeljenja i mesto gde rade zaposleni koji obavljaju posao analitičara.

```
SELECT ime,  posao, RADNIK.brod, imeod, mesto
FROM ODELJENJE,  RADNIK
WHERE ODELJENJE.brod= RADNIK.brod
AND posao =  'analitičar';
```

Ime	Posao	Brod	Imeod	Mesto
Ivan	analitičar	20	Plan	Dorćol
Marko	analitičar	30	Prodaja	Stari Grad

Zadatak urađen upotrebom odredbe **INNER JOIN**:

```
SELECT RADNIK.ime, RADNIK.posao, RADNIK.brod, ODELJENJE.imeod,
      ODELJENJE.mesto
FROM ODELJENJE INNER JOIN RADNIK ON ODELJENJE.brod= RADNIK.brod
WHERE posao='analitičar';
```

Primer 150. Prikazati *idbr*, ime i prezime zaposlenih koji rade na projektu 300.

```
SELECT RADNIK.idbr, ime, prezime
FROM RADNIK, UCESCE
WHERE RADNIK.idbr=UCESCE.idbr
AND brproj=300;
```

Idbr	Ime	Prezime
5652	Jovan	Perić
5662	Janko	Mančić
5696	Mirjana	Dimić
5932	Mitar	Vuković
5953	Jovan	Perić
6234	Marko	Nastić
7890	Ivan	Buha

Napomena: U ovom primeru, u delu naredbe SELECT**,** samo ispred identifikacionog broja (atribut idbr) stavljen je pun naziv koji podrazumeva ime tabele i odgovarajućeg atributa (RADNIK.*idbr*), a za ostale atribute (ime i prezime) stavljeno je samo ime atributa. Ovako napisan upit je ispravan jer jedino od pomenutih atributa, atribut *idbr* postoji u obe tabele, RADNIK i UCESCE, pa je neophodno navesti i ime tabele pre imena atributa. Ako atribut ne postoji u više od jedne pomenute tabele u upitu, nije neophodno navesti ime atributa pre imena tabele.

Zadatak je ispravno urađen i na neki od sledećih načina:

I)
```
SELECT RADNIK.idbr, RADNIK.ime, RADNIK.prezime
FROM RADNIK, UCESCE
WHERE RADNIK.idbr=UCESCE.idbr AND brproj=300;
```

II)
```
SELECT UCESCE.idbr, RADNIK.ime, RADNIK.prezime
FROM RADNIK, UCESCE
WHERE RADNIK.idbr=UCESCE.idbr AND brproj=300;
```

III)
```
SELECT UCESCE.idbr, ime, prezime
FROM RADNIK, UCESCE
WHERE RADNIK.idbr=UCESCE.idbr AND brproj=300;
```

Napomena: Pošto atribut *idbr* postoji u obe tabele, kod unutrašnjeg spajanja nije bitno da li će se navesti ime jedne ili druge tabele pre imena atributa, to jest: RADNIK.*idbr* ili UCESCE.*idbr*.

Primer 151. Prikazati ime radnika i mesto u kome rade za sve radnike čija je plata između 1500 i 2500 (uključujući i te vrednosti) i čije ime ne sadrži slovo *e*. Rezultate poređati po plati u opadajućem, a zatim po imenu u rastućem redosledu.

```
SELECT RADNIK.ime, ODELJENJE.mesto
FROM RADNIK, ODELJENJE
WHERE  RADNIK.brod=ODELJENJE.brod
          AND RADNIK.plata BETWEEN 1500 AND 2500
          AND RADNIK.ime NOT LIKE '%E%'
ORDER BY RADNIK.plata DESC, RADNIK.ime;
```

Ime	Mesto
Janko	Novi Beograd
Božidar	Dorćol
Ivan	Dorćol

Primer 152. Prikazati imena zaposlenih i imena projekata za zaposlene koji imaju funkciju nadzornika na projektu i čija je plata veća od srednje vrednosti plate zaposlenih u odeljenju 30.

```
SELECT RADNIK.ime, PROJEKAT.imeproj
FROM RADNIK, UCESCE, PROJEKAT
WHERE RADNIK.idbr= UCESCE.idbr
AND PROJEKAT.brproj=UCESCE.brproj
AND UCESCE.funkcija='nadzornik' AND RADNIK.plata > ( SELECT AVG(plata)
                                                     FROM RADNIK
                                                     WHERE brod=30);
```

Ime	Imeproj
Mitar	plasman

Napomena: U upitu se mogu kombinovati ugnježđeni upiti i prirodno spajanje.

Primer 153. Prikazati imena odeljenja i ime i prezime šefova odeljenja.

```
SELECT ODELJENJE.imeod, RADNIK.ime, RADNIK.prezime
FROM ODELJENJE, RADNIK
WHERE ODELJENJE.sefod=RADNIK.idbr;
```

Imeod	Ime	Prezime
Komercijala	Janko	Mančić
Plan	Božidar	Ristić
Prodaja	Pavle	Šotra
Direkcija	Miloš	Marković

Napomena: U ovom primeru tabele RADNIK i ODELJENJE povezane su preko identifikacionih brojeva zaposlenih. U tabeli ODELJENJE postoji informacija o šefovima odeljenja (atribut *sefod*) koja je u stvari, identifikacioni broj zaposlenog (*idbr*) u tabeli RADNIK, u kojoj su podaci o imenu i prezimenu zaposlenih.

Primer 154. Prikazati imena projekata i broj radnika na njima za sve projekte na kojima radi više od 3 radnika.

```
SELECT PROJEKAT.imeproj, COUNT (UCESCE.idbr) AS [broj radnika]
FROM PROJEKAT, UCESCE
WHERE PROJEKAT.brproj=UCESCE.brproj
GROUP BY PROJEKAT.imeproj
HAVING COUNT (UCESCE.idbr) > 3
```

Imeproj	Broj radnika
Izvoz	7
Plasman	7
Uvoz	7

Primer 155. Prikazati ime projekta i ukupan broj sati angažovanja svih zaposlenih za projekte na kojima je ukupan broj časova angažovanja između 5000 i 10000.

```
SELECT PROJEKAT.imeproj, SUM (UCESCE.brsati) AS [UKUPAN BROJ SATI]
FROM PROJEKAT, UCESCE
WHERE PROJEKAT.brproj=UCESCE.brproj
GROUP BY PROJEKAT.imeproj
HAVING SUM (UCESCE.brsati) BETWEEN 5000 AND 10000;
```

Imeproj	Ukupan broj sati
Uvoz	9000

Upotreba sinonima i skraćenica (alias) umesto imena tabela

Sinonimi su alternativni nazivi za isti objekat. Sinonimi se mogu pridružiti tabelama, pogledima, programima u bazi i drugim sinonimima. Sinonimi mogu biti privatni, tj. vlasništvo jedne određene šeme baze podataka,a kod nekih DBMS mogu se definisati na nivou DBMS-a i tada su javni. Sinonimi su pogodni za referenciranje na objekte iz drugih baza podataka ili drugih šema. Sinonimi se mogu koristiti u DML naredbama na gotovo svim mestima gde se inače navodi naziv objekta. Po pravilu, prave se i uništavaju naredbama CREATE i DROP (DML naredbe).[20]

Prilikom rada sa više tabela potrebno je, osim naziva atributa, navesti i naziv tabele u kojoj se taj atribut nalazi. Ovo je neophodno ako se u različitim tabelama nalaze atributi istog imena, a bar jedan se koristi u upitu. Kod pisanja kraćih upita navođenje imena tabele ispred atributa nije problematično, ali kod kompleksnijih upita lakše je pridružiti tabeli neko kraće ime, pa se umesto navođenja imena tabele u upitu pozivati na ime koje joj je dodeljeno.

Primer 156. Uradimo sada Primer 144 upotrebivši alijas R za tabelu RADNIK i O za tabelu ODELJENJE. Prikazati za svakog radnika ime, posao, kao i broj odeljenja, naziv i mesto odeljenja u kome radi.

```
SELECT R.ime, R.posao, O.brod, O.imeod, O.mesto
FROM ODELJENJE AS O INNER JOIN RADNIK AS R ON O.brod = R.brod;
```

```
SELECT R.ime, R.posao, O.brod, O.imeod, O.mesto
FROM ODELJENJE AS O , RADNIK AS R
WHERE R.brod=O.brod;
```

Napomena: Kod upotrebe skraćenica (alijas), sinonima potrebno je održati doslednost, to jest ako se u upitu koriste skraćenice za tabele, onda ih je neophodno koristiti na svim mestima gde treba navesti ime tabele. Samo na jednom mestu u upitu, u FROM odredbi navede se puno ime tabele i dodeli se skraćenica za tabelu koja se koristi u tom upitu.

Netačni su primeri I i II a ispravan je upit u slučaju III:

I)
```
SELECT ime, posao, brod, imeod, mesto
FROM ODELJENJE AS O, RADNIK AS R
WHERE R.brod=O.brod;
```

U ovom slučaju nisu korišćeni sinonimi ispred svih imena atributa u SELECT odredbi.

II)
```
SELECT R.ime, R.posao, O.brod, O.imeod, O.mesto
FROM ODELJENJE AS O , RADNIK AS R
WHERE brod=O.brod;
```

[20] U ovoj zbirci nećemo kreirati niti pokazivati primere za rad sa sinonimima, već ćemo samo u upitima koristiti sinonime u obliku naziva kolona i relacija zadtih u upitima kao i skraćenice.

U ovom slučaju nisu korišćeni sinonimi ispred imena atributa *brod* u WHERE odredbi.

III) **SELECT** R.ime, posao, O.brod, O.imeod, mesto
FROM ODELJENJE **AS** O , RADNIK **AS** R
WHERE R.brod=O.brod;

U ovom slučaju nisu korišćeni sinonimi ispred nekih imena atributa u SELECT odredbi. Ipak, SUBP može prepoznati *posao* kao atribut iz tabele RADNIK, a *mesto* kao atribut tabele ODELJENJE, tako da je upit sintaksno ispravan.

Primer 157. Uraditi sada Primer 145 upotrebivši alijas R za tabelu RADNIK, U za tabelu UCESCE i P za tabelu PROJEKAT. Prikazati za svakog radnika ime, posao i sve informacije o projektu na kome radi.

SELECT R.ime, R.posao, P.brproj, P.imeproj, P.sredstva, P.rok
FROM RADNIK R, UCESCE U , PROJEKAT **AS** P
WHERE R.idbr=U.idbr **AND** P.brproj=U.brproj;

SELECT ime, posao, p.brproj, imeproj, sredstva, rok
FROM RADNIK r **INNER JOIN (**PROJEKAT p **INNER JOIN** UCESCE u **ON**
p.BRPROJ = u.BRPROJ**)** **ON** r.IDBR = u.IDBR;

Napomena: U ovom primeru nije korišćeno AS, već je samo stavljen znak razmaka (space) iza imena tabele, a zatim navedena skraćenica za tabelu.

Primer 158. Prikazati identifikacioni broj i imena zaposlenih koji rade na 2 do 6 projekata (uključujući i te vrednosti) rezultate urediti po imenu zaposlenog u rastućem redosledu.

SELECT RADNIK.idbr, ime, COUNT(*) **AS** [broj projekata]
FROM RADNIK **INNER JOIN** UCESCE **ON** RADNIK.idbr=UCESCE.idbr
GROUP BY RADNIK.idbr, ime
HAVING COUNT(*) >= 2 **AND** COUNT (*) <= 6
ORDER BY ime;

Idbr	Ime	Broj projekata
5652	Jovan	2
5953	Jovan	2
6234	Marko	3
5696	Mirjana	2
5932	Mitar	3

Primer 159. Prikazati idbr, ime, ime projekta i ukupna primanja radnika koji ne učestvuju na projektu izvoz i čija su ukupna primanja manja od 1500. Rezultate urediti po ukupnim primanjima u opadajućem redosledu, pa po imenu projekta u rastućem redosledu.

a) Oracle

SELECT RADNIK.idbr, ime, imeproj, plata+**NVL**(premija,0) **AS** [ukupna primanja]
FROM PROJEKAT, RADNIK, UCESCE
ON PROJEKAT.brproj=UCESCE.brproj
WHERE RADNIK.idbr=UCESCE.idbr **AND** PROJEKAT.brproj=UCESCE.brproj
AND (imeproj **NOT IN** ('izvoz') **AND** plata+**NVL**(premija,0)<1500)
ORDER BY plata+**NVL**(premija, 0) **DESC** , imeproj;

b) Access

```
SELECT RADNIK.idbr, ime, imeproj, plata+NZ(premija) AS [ukupna primanja]
FROM PROJEKAT INNER JOIN
        (RADNIK INNER JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr)
        ON PROJEKAT.brproj=UCESCE.brproj
WHERE imeproj NOT IN ('izvoz') AND plata+NZ(premija)<1500
ORDER BY plata+NZ(premija) DESC , imeproj;
```

c) SQL Server

```
SELECT RADNIK.idbr, ime, imeproj, plata+ISNULL(premija,0) AS [ukupna primanja]
FROM PROJEKAT INNER JOIN
        (RADNIK INNER JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr)
        ON PROJEKAT.brproj=UCESCE.brproj
WHERE imeproj NOT IN ('izvoz') AND plata + ISNULL(premija, 0) < 1500
ORDER BY plata+ ISNULL (premija,0) DESC, imeproj;
```

Idbr	Ime	Imeproj	Ukupna primanja
5953	Jovan	plasman	1100
5953	Jovan	uvoz	1100
5696	Mirjana	plasman	1000

Spoljnje spajanje (OUTER JOIN)

Spoljnje spajanje (**OUTER JOIN**) koristimo da se u rezultat spajanja uključe i one n-torke koje ne zadovoljavaju uslov spajanja, tj. nemaju parnjaka u obe tabele, ali zadovoljavaju uslov iz WHERE odredbe. U bazi postoji jedan radnik koji još nije raspoređen ni u jedno odeljenje. Isto tako, postoji jedno odeljenje u kojem ne radi ni jedan radnik. Ako hoćemo da u odgovor uključimo i podatke iz druge tabele, koji nemaju parnjaka u prvoj tabeli, to se zove spoljnje spajanje (**OUTER JOIN**) i može biti desno spajanje (**RIGHT JOIN**), levo spajanje (**LEFT JOIN**) ili potpuno spajanje (**FULL JOIN**).

Ova vrsta spajanja koristi se kod održavanja baze podataka da bi se iz tabele uklonili zapisi "siročići" (zapisi koji nisu u relaciji) ili duplikati podataka, tako što se pravi nova tabela. Funkcioniše tako što prikazuje sve zapise iz jedne tabele koji zadovoljavaju postavljeni uslov i imaju parnjaka u drugoj tabeli (spajanje po jednakosti), i sve one zapise iz te tabele za koje u drugoj tabeli (članu spoja) ne postoje odgovarajući zapisi. Takav je na primer radnik Ivan, koji još nije raspoređen ni u jedno odeljenje. MS Access podržava dve vrste OUTER JOIN-a, LEFT i RIGHT. U jačim sistemima za upravljanje bazama podataka (kao što je na primer SQL server) postoji i takozvani FULL JOIN.

LEFT **(**OUTER) JOIN vraća sve zapise iz tabele koju u spoju proglasimo kao "LEVU", odnosno koja je prva navedena u izrazu za spajanje. Za zapise sa leve strane koji nemaju odgovarajući zapis sa desne, rezultat ima vrednost Null.

Kod kreiranja spoljašnjih spojeva izuzetno je značajno koju smo tabelu proglasili "levom", a koju "desnom". U SQL-u "leva" tabela predstavlja tabelu levo od ključne reči LEFT JOIN, a "desna" tabela desno od date ključne reči. U QBE-u "leva" tabela postaje ona od koje počinjemo da "prevlačimo" spoj.

RIGHT **(**OUTER) JOIN vraća sve zapise iz tabele koju u spoju proglasimo kao "DESNU", bez obzira na to da li se odgovarajući zapisi nalaze u "levoj" tabeli. Za zapise sa desne strane koji nemaju odgovarajući zapis sa leve, rezultat ima vrednost Null. Izrada ovakve vrste upita vrši se po analogiji sa prethodnom, osim što je potrebno kao svojstva spajanja (Join Properties) izabrati opciju za RIGHT OUTER JOIN.

Primer 160. Izlistaj sve podatke o odeljenjima i radnicima za radnike čija je premija >2000.

U SUBP Oracle:

```
SELECT *
FROM ODELJENJE,  RADNIK
WHERE ODELJENJE.[brod] (+) = RADNIK.[brod] AND  premija >2000;
```

U MS Access-u:

```
SELECT *
FROM ODELJENJE RIGHT JOIN RADNIK ON ODELJENJE.brod = RADNIK.brod
WHERE RADNIK.premija >2000;
```

Primer 161. Izlistaj sve podatke o odeljenjima i radnicima za odeljenja čija imena počinju slovima **d** ili **r**.

U SUBP Oracle:

```
SELECT *
FROM ODELJENJE,  RADNIK
WHERE  (ODELJENJE.[brod] = RADNIK.[brod] (+))  AND
       (imeod LIKE 'd%'  OR imeod LIKE 'r%');
```

U MS Accessu:

```
SELECT *
FROM ODELJENJE LEFT JOIN RADNIK ON ODELJENJE.brod = RADNIK.brod
WHERE ODELJENJE.imeod LIKE 'd*' OR ODELJENJE.imeod LIKE 'r*';
```

Primer 162. Prikazati nazive odeljenja, ime i posao svakog radnika koji u njima rade, uključujući i radnike koji nisu raspoređeni ni u jednom odeljenju.

I) Rešenje zadatka upotrebom desnog spajanja - **RIGHT JOIN** je:

```
SELECT ODELJENJE.imeod, RADNIK.ime, RADNIK.posao
FROM ODELJENJE RIGHT OUTER JOIN RADNIK ON
       ODELJENJE.brod = RADNIK.brod;
```

II) Rešenje zadatka upotrebom **LEFT JOIN**:

```
SELECT ODELJENJE.imeod, RADNIK.ime, RADNIK.posao
FROM RADNIK LEFT OUTER JOIN ODELJENJE ON
       RADNIK.brod = ODELJENJE.brod;
```

Ključnu reč **OUTER** možemo izostaviti iz upita, s obzirom na to da se prilikom upotrebe operacija LEFT ili RIGHT JOIN podrazumeva da je on spoljašnji (OUTER). Ovaj upit može da se reši i upotrebom sinonima ili skraćenica (aliasa):

```
SELECT O.imeod, R.ime, R.posao
FROM RADNIK R RIGHT OUTER JOIN ODELJENJE O ON R.brod=O.brod;
SELECT O.imeod, R.ime,R.posao
FROM ODELJENJE O LEFT OUTER JOIN RADNIK R ON R.brod=O.brod;
```

Imeod	Ime	Posao
Plan	Petar	vozač
Komercijala	Aleksandar	električar
Komercijala	Vanja	prodavac
Komercijala	Jovan	električar
Komercijala	Janko	upravnik
Komercijala	Mirjana	čistač
Plan	Božidar	upravnik
Prodaja	Pavle	upravnik
Direkcija	Miloš	direktor
Direkcija	Svetlana	savetnik
Komercijala	Tomislav	električar
Prodaja	Andrija	nabavljač
Plan	Slobodan	vozač
Plan	Mitar	savetnik
Prodaja	Jovan	nabavljač
Prodaja	Marko	analitičar
Direkcija	Janko	upravnik
Plan	Ivan	analitičar
NULL	Luka	analitičar

Primer 163. Prikazati nazive odeljenja, ime i posao radnika koji rade u njima, uključujući i odeljenja u kojima ne radi ni jedan radnik.

```
SELECT ODELJENJE.imeod, RADNIK.ime, RADNIK.posao
FROM ODELJENJE LEFT JOIN RADNIK ON ODELJENJE.brod = RADNIK.brod;
```

```
SELECT ODELJENJE.imeod, RADNIK.ime, RADNIK.posao
FROM RADNIK RIGHT JOIN ODELJENJE ON RADNIK.brod = ODELJENJE.brod;
```

Imeod	Ime	Posao
Komercijala	Aleksandar	električar
Komercijala	Vanja	prodavac
Komercijala	Jovan	električar
Komercijala	Janko	upravnik
Komercijala	Mirjana	čistač
Komercijala	Tomislav	električar
Plan	Petar	vozač
Plan	Božidar	upravnik
Plan	Slobodan	vozač
Plan	Mitar	savetnik
Plan	Ivan	analitičar
Prodaja	Pavle	upravnik
Prodaja	Andrija	nabavljač
Prodaja	Jovan	nabavljač
Prodaja	Marko	analitičar
Direkcija	Miloš	direktor
Direkcija	Svetlana	savetnik
Direkcija	Janko	upravnik
Računski centar	NULL	NULL

Primer 164. Prikazati imena i poslove radnika, kao i broj i imena projekata na kojima rade uključujući i projekte na kojima ne radi ni jedan radnik.

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj
FROM RADNIK RIGHT JOIN UCESCE ON RADNIK.IDBR=UCESCE.IDBR
            RIGHT JOIN PROJEKAT ON PROJEKAT.brproj=UCESCE.brproj;
```

Primer 165. Prikazati imena i poslove radnika, kao i broj i imena projekata na kojima rade, uključujući i radnike koji ne rade ni na jednom projektu.

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj
FROM RADNIK LEFT JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr
            LEFT JOIN PROJEKAT ON PROJEKAT.brproj=UCESCE.brproj;
```

Primer 166. Prikazati imena i poslove radnika, kao i broj i imena projekata samo za projekte na kojima ne radi ni jedan radnik.

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj
FROM RADNIK RIGHT JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr
            RIGHT JOIN PROJEKAT ON PROJEKAT.brproj=UCESCE.brproj
WHERE RADNIK.ime IS NULL;
```

Ime	Posao	Brproj	Imeproj
NULL	NULL	500	izgradnja

Primer 167. Prikazati imena i poslove radnika, kao i broj i imena projekata na kojima rade samo za radnike koji ne rade ni na jednom projektu.

```
SELECT RADNIK.ime, RADNIK.posao, PROJEKAT.brproj, PROJEKAT.imeproj
FROM RADNIK LEFT JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr
            LEFT JOIN PROJEKAT ON PROJEKAT.BRPROJ=UCESCE.BRPROJ
WHERE PROJEKAT.brproj IS NULL;
```

Ime	Posao	Brproj	Imeproj
Petar	vozač	NULL	NULL
Vanja	prodavac	NULL	NULL
Tomislav	električar	NULL	NULL
Luka	analitičar	NULL	NULL

Primer 168. Prikazati nazive odeljenja, ime i posao svakog radnika koji u njima rade uključujući i radnike koji nisu raspoređeni ni u jednom odeljenju, kao i odeljenja u kojima ne radi ni jedan radnik.

```
SELECT O.imeod,R.ime,R.posao
FROM RADNIK R FULL OUTER JOIN ODELJENJE O ON R.brod=O.brod;
```

Napomena: MS Access ne podržava **full join**!

Primer 169. Prikazati imena radnika i imena projekata na kojima rade uključujući i projekte na kojima ne radi ni jedan radnik, kao i radnike koji ne rade ni na jednom projektu.

```
SELECT RADNIK.ime, PROJEKAT.imeproj
FROM RADNIK FULL JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr
            FULL JOIN PROJEKAT ON PROJEKAT.brproj=UCESCE.brproj;
```

Primer 170. Prikazati imena radnika koji ne rade ni na jednom projektu i imena projekata na kojima ne radi ni jedan radnik.

```
SELECT RADNIK.ime, PROJEKAT.imeproj
FROM RADNIK FULL JOIN UCESCE ON RADNIK.idbr=UCESCE.idbr
            FULL JOIN PROJEKAT ON PROJEKAT.brproj=UCESCE.brproj
WHERE RADNIK.idbr IS NULL OR  PROJEKAT.brproj IS NULL;
```

Ime	Imeproj
Petar	NULL
Vanja	NULL
Tomislav	NULL
Luka	NULL
NULL	izgradnja

Spajanje tabela sa samom sobom (SELF JOIN)

Tabela se može spajati i sa samom sobom (**SELF JOIN**), kada unutar tabele postoji relacija između pojedinih n-torki. To je slučaj sa tabelom RADNIK jer su neki zaposleni rukovodioci nekim drugim radnicima.

Ova vrsta spajanja odnosi se na podatke u jednoj tabeli kada unutar objekata jedne tabele postoji relacija 1:1 ili 1:N. U MS Accessu se spoj tabele sa samom sobom pravi tako što se upitu dodaje duplikat tabele (MS Access u ovom slučaju sam dodeljuje pseudonim-ime za duplikat) i zatim se spajaju polja iz originala i kopije tabele.

Primer 171. Prikazati imena radnika i imena njihovih neposrednih rukovodilaca.

```
SELECT R.ime AS IME_RADNIKA, R1.ime AS IME_RUKOVODIOCA
FROM RADNIK R INNER JOIN RADNIK R1 ON R.rukovodilac=R1.idbr;
```

Primer 172. Prikazati imena radnika i imena njihovih neposrednih rukovodilaca uključujući i radnike koji nemaju neposrednog rukovodioca.

```
SELECT R.ime AS IME_RADNIKA,R1.ime AS IME_RUKOVODIOCA
FROM RADNIK R LEFT JOIN RADNIK R1 ON R.rukovodilac=R1.idbr;
```

Primer 173. Prikazati imena radnika i imena njihovih neposrednih rukovodilaca samo za radnike koji nemaju neposrednog rukovodioca.

```
SELECT R.ime, R1.ime
FROM RADNIK R LEFT JOIN RADNIK R1 ON R.rukovodilac=R1.idbr
WHERE R1.idbr IS NULL;
```

Primer 174. Prikazati ime i zaradu radnika i njihovih neposrednih rukovodilaca za one radnike koji zarađuju više od svojih neposrednih rukovodilaca.

```
SELECT R.ime AS IME_RADNIKA, R.plata+ ISNULL(R.premija,0) AS
ZARADA_RADNIKA, R1.ime AS IME_RUK, R1.plata + ISNULL(R1.premija,0)  AS
ZARADA_RUK
FROM RADNIK R INNER JOIN RADNIK R1 ON R.rukovodilac=R1.idbr
WHERE R.plata+ ISNULL(R.premija,0) > R1.PLATA + ISNULL(R1.premija,0);
```

IME_RADNIKA	ZARADA_RADNIKA	IME_RUK	ZARADA_RUK
Petar	3200	Božidar	2200
Vanja	2500	Janko	2400
Marko	4300	Svetlana	2750
Janko	3910	Miloš	3000
Ivan	4800	Svetlana	2750

Primer 175. Prikazati imena radnika koji imaju istu kvalifikaciju kao Mitar.

a) pomoću unutrašnjeg spajanja - SELF JOIN

```
SELECT R.ime AS IME_RADNIKA
FROM RADNIK R INNER JOIN RADNIK R1 ON R.kvalif=R1.kvalif
WHERE R1.ime='Mitar';
```

b) pomoću ugnježdenog upita

```
SELECT ime
FROM RADNIK
WHERE  kvalif  IN (  SELECT kvalif
                     FROM RADNIK
                     WHERE IME='Mitar');
```

Primer 176. Prikaži imena, posao i broj odeljenja radnika kojima je rukovodilac *Pavle*.

```
SELECT R.ime AS [ime radnika], R.posao, R.brod, ŠEF.ime AS [ime šefa]
FROM RADNIK R,  RADNIK ŠEF
WHERE ŠEF.idbr = R.rukovodilac AND ŠEF.ime='Pavle';
```

Ime radnika	Posao	Brod	Ime šefa
Andrija	nabavljač	30	Pavle
Jovan	nabavljač	30	Pavle

Primer 177. Izlistaj šifru radnika, broj sati angažovanja i funkciju na projektu za radnike koji rade na projektu *uvoz*.

a) prirodnim spajanjem tabela

```
SELECT UCESCE.idbr, UCESCE.brsati, UCESCE.funkcija
FROM PROJEKAT, UCESCE
WHERE PROJEKAT.brproj = UCESCE.brproj
AND PROJEKAT.imeproj = 'uvoz';
```

b) ugnježdenim upitom

```
SELECT UCESCE.idbr, UCESCE.brsati, UCESCE.funkcija
FROM UCESCE
WHERE UCESCE.brproj  =  ( SELECT PROJEKAT.brproj
                          FROM PROJEKAT
                          WHERE PROJEKAT.imeproj = 'uvoz');
```

Idbr	Brsati	Funkcija
5652	1000	IZVRŠILAC
5786	2000	KONSULTANT
5842	2000	ŠEF
5900	2000	IZVRŠILAC
5932	500	KONSULTANT
5953	1000	IZVRŠILAC
6234	500	NADZORNIK

Primer 178. Izlistaj šifru odeljenja i najveću platu u svakom odeljenju.

```
SELECT RADNIK.brod AS BROD, Max(RADNIK.plata) as [Najveća plata]
FROM Radnik
GROUP BY RADNIK.BROD;
```

Brod	Najveća plata
NULL	2000
10	2400
20	2600
30	2800
40	3900

Napomena: U ovom primeru nije poštovana konvencija da se ključne reči SQL jezika pišu velikim slovima. To, naravno, nije dovelo do pojave grešaka jer SQL nije osetljiv na velika i mala slova. Dakle **as**, **As** i **AS** su ista reč, kao što je **COUNT** isto što i **Count**, a **Max**, **max** i **MAX** označavaju funkciju **najveći**. Isto važi i za imena tabela i atributa: **radnik**, **Radnik** i **RADNIK** označavaju tabelu u kojoj se skladište podaci o zaposlenima, a **brod**, **Brod** i **BROD** označavaju ime jednog te istog atributa u tabeli RADNIK (ovo ne važi pod operativnim sistemima Unix/Linux).

Primer 179. Izlistaj šifru odeljenja i najveću platu u svakom odeljenju. Rezultat urediti po broju odeljenja u opadajućem redosledu.

```
SELECT RADNIK.brod AS BROD, MAX(RADNIK.plata) AS [Najveća plata]
FROM RADNIK
GROUP BY RADNIK.brod
ORDER BY brod DESC;
```

Brod	Najveća plata
40	3900
30	2800
20	2600
10	2400
NULL	2000

Primer 180. Izlistaj šifru i ime odeljenja, kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odeljenju.

Primer je urađen tako što je izvršeno spajanje tabele RADNIK i podupita koji izdvaja broj odeljenja i najveću platu u njemu (podupit stavljen u FROM odredbu prethodnom upita):

```
SELECT O.brod AS brod, idbr, ime, posao, plata
FROM  (SELECT brod, Max(plata) AS  [mplata]
       FROM RADNIK
       GROUP BY brod) AS  O, RADNIK
WHERE RADNIK.brod = O.brod AND RADNIK.plata = O.mplata
ORDER BY 1;
```

Realizacija preseka i razlike relacija preko spoljnjeg spajanja

Presek i razlika se u Accessu i SQL Serveru realizuje pomoću spoljnjeg spajanja. Neka postoji tabela RADNIK20 koja sadrži zapise o zaposlenima u odjeljenju 20, sa izmenjenim podacima o radniku čiji *idbr* je 7890. Takođe, izmenjen je primarni ključ radnika Luke Boškovića sada je 7891.

RADNIK20									
Idbr	Ime	Prezime	Posao	Kvalif.	Rukovodilac	Datzap	Premija	Plata	Brod
5367	Petar	Vasić	vozač	KV	5780	1/1/1978	1900	1300	20
5780	Božidar	Ristić	upravnik	VSS	6789	8/11/1984		2200	20
5900	Slobodan	Petrović	vozač	KV	5780	10/3/2002	1300	900	20
5932	Mitar	Vuković	savetnik	VSS	5842	3/25/2000		2600	20
7890	Ivana	Buha	analitičar	VSS	5867	12/17/2003	3200	1600	
7891	Luka	Bošković	analitičar	VSS	5867	5/20/2004		2000	

Presek projekcija (idbr, ime, brod) relacija RADNIK i RADNIK20 se realizuje kao levo spajanje iz kojeg su isključeni oni zapisi kod kojih je vrednost *radnik20.idbr* jednaka Null.

Primer 181. Kreirati presek projekcija (*idbr*, *ime*, *brod*) relacija RADNIK i RADNIK20.

```
SELECT R.IDBR, R.IME, R.BROD, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON R.idbr = radnik20.idbr
WHERE radnik20.idbr IS NOT NULL;
```

Radnik.idbr	Radnik.ime	Radnik.brod	Radnik20.idbr	Radnik20.ime	Radnik20.brod
5367	Petar	20	5367	Petar	20
5780	Božidar	20	5780	Božidar	20
5900	Slobodan	20	5900	Slobodan	20
5932	Mitar	20	5932	Mitar	20
7890	Ivan	20	7890	Ivana	

Presek dveju relacija

Odgovor na ovaj upit (prikazan na prethodnoj slici) sadrži sve zapise iz relacije RADNIK20, bez obzira što vrednosti neprimarnih atributa nisu iste u obe relacije. Odgovor ne sadrži zapis o radniku Luki Boškoviću jer on ima drugačiji primarni ključ u tabelama radnik (7892) i Radnik20 (7891), pa su to dva različita zapisa.

Ako u prethodnom upitu koristimo naredbu SELECT R.*IDBR*, R.*IME*, R.*BROD* onda je to RADNIK∩RADNIK20 a dobija se leva polovina prethodne tabele (prve tri kolone).

Ako koristimo SELECT radnik20.*idbr*, radnik20.*ime*, radnik20.*brod* onda je u pitanju RADNIK20∩RADNIK a dobija se desna polovina prethodne tabele (poslednje tri kolone).

Primer 182. Kreirati razliku projekcija (idbr, ime, brod) relacija RADNIK i RADNIK20.

Razlika relacija sadrži one zapise iz prve relacije koji ne postoje u drugoj relaciji. Razlika relacija RADNIK i RADNIK20 se realizuje kao levo spajanje iz kojeg su isključeni oni zapisi kod kojih je vrednost *radnik20.idbr* jednaka Null. Operator spoljnjeg spajanja učitava sve zapise iz obe tabele i postavlja Null vrednosti u polja koja nemaju odgovarajuće parnjake u drugoj (ovog puta levoj) relaciji (slika b). Operator IS NULL u odredbi WHERE izdvaja iz ovog skupa samo one zapise u kojima su Null vrednosti (odnosno zapise bez parnjaka).

```
SELECT R.IDBR, R.IME, R.BROD, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON R.idbr = radnik20.idbr
WHERE radnik20.idbr IS NULL;
```

R.idbr	R.ime	R.brod	Radnik20.idbr	Radnik20.ime	Radnik20.brod
5367	Petar	20	5367	Petar	20
5497	Aleksandar	10			
5519	Vanja	10			
5652	Jovan	10			
5662	Janko	10			
5696	Mirjana	10			
5780	Božidar	20	5780	Božidar	20
5786	Pavle	30			
5842	Miloš	40			
5867	Svetlana	40			
5874	Tomislav	10			
5898	Andrija	30			
5900	Slobodan	20	5900	Slobodan	20
5932	Mitar	20	5932	Mitar	20
5953	Jovan	30			
6234	Marko	30			
6789	Janko	40			
7890	Ivan	20	7890	Ivana	
7892	Luka				

a) Levo spajanje (upit bez WHERE odredbe)

R.idbr	R.ime	R.brod	Radnik20.idbr	Radnik20.ime	Radnik20.brod
5497	Aleksandar	10			
5519	Vanja	10			
5652	Jovan	10			
5662	Janko	10			
5696	Mirjana	10			
5786	Pavle	30			
5842	Miloš	40			
5867	Svetlana	40			
5874	Tomislav	10			
5898	Andrija	30			
5953	Jovan	30			
6234	Marko	30			
6789	Janko	40			
7892	Luka				

b) Razlika dveju relacija

Napomena: Dobijene tabele pokazuju da su dva zapisa ista ako su im jednaki primarni ključevi, a vrednosti ostalih atributa mogu biti različite. Imena odgovarajućih atributa ne moraju biti ista, ali su njihovi tipovi isti ili kompatibilni. Zapis (7890, Ivan, 20) i zapis (7890, Ivana, Null) tretiraju se kao jednaki jer je primarni ključ u oba zapisa isti (7890).

Sledeći primeri bi bili trivijalni kada bi bila podržana operacija preseka u Access i starijim verzijama SQL Server a to nije slučaj. Zadaci se moraju rešavati na neki drugi način, preko upita nad upitom, grupisanja, spajanja ili ugnježdenih upita.

Primer 183. Prikazati šifre projekata na kojima rade i radnik 6234 i radnik 5696 (zajedno).

```
SELECT brproj
FROM (SELECT brproj FROM ucesce WHERE idbr=6234)
WHERE brproj IN (SELECT brproj FROM ucesce WHERE idbr=5696);
```

ili

```
SELECT DISTINCT u1.brproj
FROM ucesce AS u1, ucesce AS u2
WHERE u1.brproj=u2.brproj AND u1.idbr=6234 AND u2.idbr=5696;
```

ili

```
SELECT BRPROJ
FROM UCESCE
WHERE IDBR IN (5696,6234)
GROUP BY brproj
HAVING COUNT(*)>1;
```

Brproj
200
300

Primer 184. Prikazati imena projekata na kojima rade i Marko i Mirjana (zajedno).

```
SELECT IMEPROJ
FROM PROJEKAT
WHERE   brproj  IN  ( SELECT DISTINCT u1.brproj
                      FROM ucesce AS u1, ucesce AS u2
                      WHERE u1.brproj = u2.brproj
                      AND  u1.idbr = ( SELECT idbr
                                       FROM radnik
                                       WHERE ime='Marko')
                      AND  u2.idbr = ( SELECT idbr
                                       FROM radnik
                                       WHERE ime='Mirjana'));
```

ili

```
SELECT IMEPROJ
FROM PROJEKAT
WHERE brproj IN (SELECT brproj
                 FROM ucesce
                 WHERE brproj IN (SELECT brproj
                                  FROM ucesce
                                  WHERE idbr = (SELECT idbr
                                                FROM radnik
                                                WHERE ime='marko'))
                                    AND  idbr = (SELECT idbr
                                                 FROM radnik
                                                 WHERE ime='mirjana'));
```

Imeproj
izvoz
plasman

Korelisani podupiti

Korelisani podupit je takav podupit, čiji rezultat zavisi od neke promenljive. Ta promenljiva dobija svoju vrednost u nekom spoljašnjem upitu. Obrada takvog podupita se ponavlja za svaku vrednost promenljive u upitu, a ne izvršava se svaki put.

Primer 185. Prikazati imena projekata na kojima rade radnici čija je funkcija organizator.

```
SELECT p.IMEPROJ
FROM PROJEKAT p
WHERE  "organizator"  IN  ( SELECT funkcija
                            FROM ucesce u
                            WHERE u.brproj = p.brproj);
```

p.Imeproj
izvoz

Objašnjenje: Ovaj primer se razlikuje od prethodnih sa ugnježđenim upitima (podupitima) po tome što se unutrašnji podupit ne izvršava pre spoljašnjeg, zato što taj unutrašnji upit zavisi od promenljive p.*brproj*, čije se značenje menja u odnosu na to kako sistem proverava različite redove tabele PROJEKAT. Upit se izračunava na sledeći način:

a) Sistem proverava prvi red tabele PROJEKAT i pronalazi da je *brproj* jednak 100. Tako imamo:

```
(SELECT funkcija
FROM ucesce u
WHERE u.brproj=100);
```

Kao rezultat dobija se skup (‘‘IZVRŠILAC‘‘, ‘‘KONSULTANT’‘, ‘‘ŠEF‘‘, ‘‘NADZORNIK‘‘). Sada se može izvršiti obrada spoljnjeg upita za projekat 100. Vrednost ORGANIZATOR ne pripada skupu rezultata podupita. Prelazi se na sledeći red tabele PROJEKAT i pronalazi da je *brproj* jednak 200. Tada imamo podupit:

```
(SELECT funkcija
FROM ucesce u
WHERE u.brproj=200);
```

Kao rezultat dobija se skup (‘‘ŠEF‘‘, ‘‘ORGANIZATOR‘, ‘‘KONSULTANT’‘, ‘‘IZVRŠILAC‘‘). Sada se može izvršiti obrada spoljnjeg upita za projekat 200. Kao rezultat toga upita dobija se ime projekta *Izvoz* jer u skupu postoji vrednost ORGANIZATOR.

Onda se prelazi na sledeći red tabele PROJEKAT. Postupak se ponavlja sve do poslednjeg reda tabele PROJEKAT.

Primer 186. Prikazati imena radnika koji na projektima imaju više od 2000 sati angažovanja .

```
SELECT ime
FROM RADNIK AS r
WHERE  2000 <  ( SELECT SUM(brsati)
                 FROM ucesce
                 WHERE  idbr = r.idbr);
```

Ime
Mirjana

Za svaku moguću vrednost šifre radnika *idbr* u tabeli **r** izvršava se sledeće:

- izračunati podupit i dobiti vrednost (ili skup vrednosti u prethodnom primeru) za ukupan broj sati na svim projektima na kojima radi taj radnik, na primer 2000;
- izdvojiti, u odnosu na rezultujuću Vrednost ili skup, vrednost r.*idbr*, ako i samo ako je izračunata vrednost veća od 2000, a to je slučaj kod radnika sa *idbr* = 5696, čije ime je Mirjana. Kada se radi sa skupovima onda se izdvajaju vrednost koja pripada skupu (odnosno za koju je uslov u WHERE odredbi spoljnjeg upita tačan).

Napomena: Simbol ***r*** označava i baznu tabelu (fizičku tabelu na disku računara), a takođe, i promenljivu (r.*idbr*), koja se odnosi na skup zapisa te bazne tabele. Uvođenje sinonima nikada nije greška, a često je i neophodnost (kod spajanja tabele sa samom sobom, SELF JOIN).

Upiti nad upitima

U odredbi FROM osim relacija (tabela) može se koristiti i pogled, imenovani upit ili virtuelna tabela koja je rezultat SQL upita koji je u odredbi FROM napisan u obliku teksta.

Neka u bazi podataka postoji pogled ili upit, zapamćen pod imenom **maxplata** koji kao rezultat daje najveće plate u svakom odeljenju. To je upit:

```
SELECT Brod,  MAX(Plata) AS Max_plata
FROM RADNIK
GROUP BY  Brod;
```

Odgovor sistema na ovaj upit se nalazi u radnoj memoriji računara i bio bi:

Brod	Max_plata
	2000
10	2400
20	2600
30	2800
40	3900

Primer 187. Prikazati iznose najvećih plata u svakom odeljenju.

```
SELECT *
FROM maxplata;
```

Brod	Max_plata
	2000
10	2400
20	2600
30	2800
40	3900

Primer 188. Ko su najbolje plaćeni radnici u svakom odeljenju (Primer 142).

I) Rešenje korišćenjem pogleda ili imenovanog upita ***maxplata*** kao argument u odredbi FROM:

```
SELECT Ime, Posao, Plata, Radnik.brod
FROM radnik, maxplata
WHERE plata=max_plata AND radnik.brod=maxplata.brod
```

II) Rešenje korišćenjem SQL upita koji je napisan kao argument u odredbi FROM:

```
SELECT Ime, Posao, Plata, Radnik.brod
FROM radnik, (SELECT MAX(plata) AS max_plata, brod
              FROM radnik
              GROUP BY brod) AS maxplata
WHERE plata=max_plata AND radnik.brod=maxplata.brod
```

U oba slučaja rezultat je isti, i prikazan je tabelom:

Ime	Posao	Plata	Brod
Janko	upravnik	2400	10
Pavle	upravnik	2600	20
Mitar	savetnik	2800	30
Janko	upravnik	3900	40

Upotreba brzog if polja

Često je u upitu potrebno u istoj koloni ubaciti jednu vrednost pod jednim uslovom odnosno drugu vrednost pod drugim uslovom. U tome nam pomažu brze if naredbe.

U MS Access to se postže upotrebom iif naredbe sa sintaksom: ***iif*** (*uslov, vrednost kada je uslov tačan, vrednost kada je uslov netačan*). Ukoliko imamo više uslova onda se to kombinuje sa više iif naredbi (npr. ***iif*** (*uslov1, iif(uslov2, vrednost kada je uslov2 tačan, vrednost kada je uslov2 netačan*), *vrednost kada je uslov1 netačan)).*

U SUBP MS SQL Server, MySQL i Oracle postoje CASE naredbe sa sledećom sintaksom:

CASE WHEN uslov*1* **THEN** vrednost*1*
WHEN uslov*2* **THEN** vrednost*2*
. . .
ELSE uslov*N*
END

Tako, na primer, možemo formirati kolonu *brdana* izvedenu iz kolone *brsati* u tabeli UCESCE tako što ćemo kolonu *brsati* podeliti sa 10 ako zaposleni ima 1000 i više sati rada na projektu, odnosno sa 8 ako zaposleni ima manji broj sati rada na projektu.

Primer 189. Izlistati zaposlene (idbr, ime i prezime), projekat (brproj i imeproj), broj sati, broj radnih dana po pomenutom uslovu i funkciju za sve zaposlene koji učestvuju na projektima. Komandu napisati za MS Access, a zatim i za MS SQL

```
SELECT r.idbr, r.prezime & ' ' & r.ime AS zaposleni, p.brproj, p.imeproj, u.brsati,
  IIF(u.brsati >= 1000, u.brsati/10, u.brsati/8) AS brdana, u.funkcija
FROM radnik r, ucesce u, projekat p
WHERE r.idbr = u.idbr and u.brproj = p.brproj
ORDER BY p.brproj, u.funkcija, r.idbr
```

```
SELECT r.idbr, r.prezime + N' ' + r.ime AS zaposleni, p.brproj, p.imeproj, u.brsati,
  CASE WHEN u.brsati >= 1000 THEN brsati/10 ELSE brsati/8 END AS brdana, u.funkcija
FROM radnik r, ucesce u, projekat p
WHERE r.idbr = u.idbr and u.brproj = p.brproj
ORDER BY p.brproj, u.funkcija, r.idbr
```

idbr	zaposleni	brproj	imeproj	brsati	brdana	funkcija
5652	Perić Jovan	100	uvoz	1000	100	IZVRŠILAC
5900	Petrović Slobodan	100	uvoz	2000	200	IZVRŠILAC
5953	Perić Jovan	100	uvoz	1000	100	IZVRŠILAC
5786	Šotra Pavle	100	uvoz	2000	200	KONSULTANT
5932	Vuković Mitar	100	uvoz	**500**	**62**	KONSULTANT
6234	Nastić Marko	100	uvoz	**500**	**62**	NADZORNIK
5842	Marković Miloš	100	uvoz	2000	200	ŠEF
5898	Ristić Andrija	200	izvoz	2000	200	IZVRŠILAC
6234	Nastić Marko	200	izvoz	1200	120	IZVRŠILAC
...	...	...	...	...	...	...

Na prethodnoj slici su zamašćenim slovima pokazane vrednosti za kolone brsati i brdana za zaposlene čiji je broj sati ispod 1000 i čiji se broj dana dobija deljenjem broja sati sa 8.

Isto tako, možemo formirati kolonu *naknada* izvedenu iz kolone *brsati* u tabeli UCESCE i kolone *plata* i *premija* iz tabele RADNIK, tako što ćemo kolonu *naknada* definisati kao zbir kolona *plata* i *premija* ako zaposleni ima 1000 i više sati rada na projektu, odnosno samo kao vrednost kolone *plata* ako zaposleni ima manji broj sati rada na projektu.

Primer 190. Izlistati zaposlene koje rade na projektu 100 (idbr, ime i prezime), njihovu platu, premiju, naknadu po pomenutom uslovu i broj radnih sati. Komandu napisati za MySQL, a zatim i za Oracle

```
SELECT r.idbr,
  CONCAT(r.prezime,  ' ', r.ime) AS zaposleni, r.plata,
  IFNULL(r.premija, 0.00) AS premija,
  CASE WHEN u.brsati >= 1000 THEN r.plata + IFNULL(r.premija, 0.00) ELSE r.plata
        END AS  naknada, u.brsati
FROM radnik r, ucesce u
WHERE r.idbr = u.idbr and u.brproj = 100
ORDER BY r.idbr
```

```
SELECT r.idbr,
  r.prezime || ' ' || r.ime AS zaposleni, r.plata,
  NVL(r.premija, 0.00) AS premija,
  CASE WHEN u.brsati >= 1000 THEN r.plata + NVL(r.premija, 0.00) ELSE r.plata
        END AS  naknada, u.brsati
FROM radnik r, ucesce u
WHERE r.idbr = u.idbr and u.brproj = 100
ORDER BY r.idbr
```

Ažuriranje baze podataka

Dodavanje, izmena (ažuriranje u užem smislu) i brisanje podataka vrši se naredbama **INSERT**, **UPDATE** i **DELETE**. Kod primene ovih naredbi ne garantuje se očuvanje integriteta baze podataka, pa se zato njihovo direktno korišćenje ne preporučuje, jer tada o integritetu mora da brine sam korisnik. Ove naredbe koristi neposredno samo administrator baze podataka. Normalno ažuriranje podataka vrši se preko aplikacija za interaktivno ažuriranje u koje su ugrađene procedure za zaštitu integriteta, a sastavni deo ovih procedura su naredbe INSERT, UPDATE i DELETE.

Dodavanje novih n-torki

U postojeće relacije je veoma jednostavno dodavanje novih slogova, a izvodi se naredbom INSERT. Opšti oblik glasi:

INSERT INTO ime relacije **[**lista atributa**] VALUES (**lista *vrednosti* **)**

uz napomenu da se ovom naredbom odjednom u relaciju mogu uneti vrednosti atributa samo jedne n-torke. Podrazumeva se dodavanje na kraju tabele, jer redosled navođenja n-torki u relaciji nije bitan.

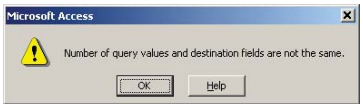
Postoje tri tipa ovih naredbi:

1. za ubacivanje vrednosti SVIH atributa jedne n-torke,
2. za ubacivanje vrednosti NEKIH atributa jedne n-torke,
3. za ubacivanje podataka iz jedne tabele u drugu.

Primer 191. Dodati podatke o novom projektu broj 600, imena Obrazovanje za koji se izdvajaju sredstva od 2,000.000.[21]

INSERT INTO PROJEKAT
VALUES (*600, 'Obrazovanje', 2000000*);

Odgovor sistema je:



Odgovor je očekivan zbog toga što nismo naveli vrednosti za sve atribute i nedostaje podataka za atribut *rok*. Dodati da je rok za ovaj projekat 22.10.2005.

INSERT INTO PROJEKAT
VALUES (*600, 'Obrazovanje', 2000000, '22.10.2005'*);

Odgovor sistema je:

[21] Svi primeri za ažuriranje podataka urađeni su u MS Accessu, nijedan upit nije izvršen do kraja kako bi sadržaj baze ostao nepromenjen i kako bi svi primeri bili urađeni nad istim tabelama

Pokretanjem ovog upita dodaće se jedan zapis o novom projektu sa zadatim vrednostima u tabelu PROJEKAT. U datom slučaju, iza imena tabele PROJEKAT nisu navedena imena atributa, ali su u delu VALUES navedene vrednosti svih atributa onim redosledom kojim su ti atributi predstavljeni u tabeli.

Za ubacivanje vrednosti svih atributa jedne n-torke nije potrebno specificirati nazive atributa. Primetite da je za ime projekta *imeproj* vrednost atributa ('Obrazovanje') stavljena sa apostrofima, jer je taj atribut tekstualnog tipa, kao i vrednost za datum.

Upit se može uraditi i na sledeći način:

INSERT INTO PROJEKAT (brproj, imeproj, sredstva, rok)
VALUES (*600, 'Obrazovanje', 2000000, '22.10.2005'*);

ili

INSERT INTO PROJEKAT (imeproj, brproj, rok, sredstva)
VALUES (*'Obrazovanje', 600, '22.10.2005', 2000000*);

U ova dva rešenja navedena su iza imena tabele imena atributa (u prvom slučaju su navedena imena atributa onim redosledom koji je u tabeli, a u drugom proizvoljnim), a onda istim tim redosledom i odgovarajuće vrednosti atributa. Za ubacivanje vrednosti NEKIH ili SVIH atributa n-torki, iz jedne tabele u drugu potrebno je da su tabele identično definisane ili se moraju specificirati nazivi atributa i njihove vrednosti (pomoću naredbe SELECT) i to u identičnom poretku.

Primer 192. Dodati podatke o novom projektu broj 200, imena Obrazovanje za koji se izdvajaju sredstva od 2 000 000, a rok je 23.10.2005.

INSERT INTO PROJEKAT (brproj, imeproj, sredstva, rok)
VALUES (*200, 'Obrazovanje', 2000000, '22.10.2005'*);

Ako bi pokušali da izvršite ovaj upit u tabelu PROJEKAT, ne bi bio dodat ni jedan zapis. Zadatak je sličan kao prethodni, ali sa različitim brojem projekta, pa ovaj upit ne bi uradio ono što na prvi pogled izgleda da bi se desilo. Razlog je u tome što se ovde za novi projekat traži da broj projekta bude 200, a u tabeli PROJEKAT već postoji projekat sa tim brojem. Problem je zbog toga što je atribut *BrProj* primarni ključ u tabeli PROJEKAT, znači da ne može postojati više projekata sa istim imenom. Ako su ime ili sredstva za novi projekat ista kao za neki od već postojećih, tada nema problem jer ta polja nisu primarni ključevi, osim u slučaju ako neko od tih polja nije primarni ključ, ali ima svojstvo **UNIQUE** (tj. ne dozvoljava duplikate).

Primer 193. Dodati podatke o novom odeljenju čije je ime *Računovodstvo*, a broj odeljenja 60.

INSERT INTO ODELJENJE (brod, imeod, mesto, sefod)
VALUES (*60, 'Računovodstvo', Null, Null*);

ili

INSERT INTO ODELJENJE (brod, imeod)
VALUES (*60, 'Računovodstvo'*);

Ovaj zadatak je urađen na dva načina, u prvom slučaju je ovaj zadatak rešen navođenjem svih atributa, tako jer kako nije poznata vrednost za atribut *mesto*, to polje ostaje prazno, tj. dodeljena mu je vrednost Null, kao i atribut *sefod*. U drugom slučaju, navedeni su samo oni atributi koji se dodaju.

Pokretanjem ovog upita, u tabelu ODELJENJE biće dodat jedan zapis o novom odeljenju. To se neće desiti u slučaju ako je u strukturi tabele ODELJENJE postavljeno da atribut *mesto* ne može imati Null vrednost, tj. da to polje ne može ostati prazno.

Za ubacivanje vrednosti NEKIH atributa jedne n-torke potrebno je specificirati nazive atributa i njihove vrednosti i to u identičnom poretku.

Primer 194. Dodati podatke o novom odeljenju čije je ime Marketing.

INSERT INTO ODELJENJE (imeod)
VALUES (*'Marketing'*);

Navedeni zadatak bi se mogao rešiti na taj način u slučaju da tabela ODELJENJE nema primarni ključ ili da je primarni ključ atribut *brod* tipa autonumber, odnosno da mu je dodeljen tip podatka koji se sam povećava za neku vrednost pri dodavanju sledećeg zapisa u tabelu.

INSERT INTO ODELJENJE (brod, imeod, mesto)
VALUES (*Null, 'Marketing', Null*);

Ovaj zadatak bi se mogao rešiti na taj način u slučaju da tabela ODELJENJE nema primarni ključ, a da atributi *brod* i *mesto* mogu imati Null vrednost.

INSERT INTO ODELJENJE (brod, imeod, mesto)
VALUES (70, *'Marketing', Null*);

ili

INSERT INTO ODELJENJE (brod, imeod)
VALUES (70, *'Marketing'*);

Na ovakav način treba uraditi zadatak kada je atribut *brod* primarni ključ u tabeli ODELJENJE, a nije tipa "Auto Increment", pa mu je potrebno dodeliti neku vrednost, koja već ne postoji u tabeli. Naravno, da vrednost za atribut *mesto* nije neophodno uneti.

Primer 195. U relaciju RADNIK dodaj podatke o novom zaposlenom licu koji ima šifru 7891**,** ime mu je Mirko, prezime Vuković**,** radi na poslovima analitičara, ima visoku stručnu spremu (VSS), još nema rukovodioca (NULL), počeo je da radi 29-jan-05, ima premiju 3000, platu 2000 i još nije raspoređen ni u jedno odeljenje (NULL).

INSERT INTO RADNIK **VALUES**
(*7891, 'Mirko', 'Vuković', 'analitičar', 'VSS'*, Null, '29-*jun*-02', *3000, 2000*, Null);

Napomena: Ukoliko podatak za neki od atributa nije poznat, kao podatak se unosi vrednost Null. Redosled podataka u listi VALUES mora biti isti kao u definiciji tabele.

Primer 196. U relaciju RADNIK dodaj podatke o novom zaposlenom, ime mu je "Mirko**"**, koji ima šifru 7891**,** visoku stručnu spremu (VSS), počeo je da radi na dan 29-jun-02.

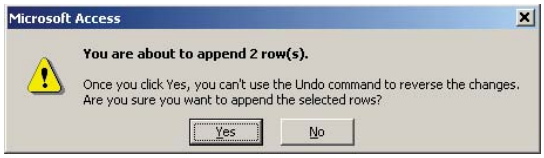
INSERT INTO RADNIK (ime, idbr, kvalif, datzap)
VALUES *('Mirko', 7891, 'VSS', '29-jun-02');*

Napomena: Ovaj upit ne radi i javlja grešku. Uzrok može biti: što se vrednost atributa (koja je primarni ključ) ponavlja, ponavlja se neka vrednost atributa koja treba da ima jedinstvenu vrednost ili nije uneta vrednost nekog atributa koja ne sme imati vrednost Null (na primer *prezime*).

Primer 197. U već postojeću tabelu PENZIONISANI, sa svim istim atributima kao i tabela RADNIK, prebaciti sve podatke o zaposlenim pre 20.10.1971.

```
INSERT INTO PENZIONISANI
SELECT *
FROM RADNIK
WHERE datzap<'20.10.1971';
```

Odgovor sistema je:



U slučaju kada podaci koje treba dodati u neku tabelu već postoje, koristi se sintaksa kao u ovom primeru, za razliku od prethodnih slučajeva u kojima se dodaju novi zapisi koji nisu nigde uneti. Tabele RADNIK i PENZIONISANI imaju sve iste atribute pa ih zato nije neophodno dodatno navoditi. Primetite da je ovde datum napisan '1.2.1971', a ako se koristi MS Access pisalo bi #1.2.1971#.

Primer 198. U tabelu UPRAVNIK, koja ima atribute *idbr*, *ime*, *datzap*, *brod* dodati podatke o svim zaposlenim koji imaju posao upravnika.

```
INSERT INTO UPRAVNIK (idbr, ime, datzap, brod)
SELECT idbr, ime, datzap, brod
FROM RADNIK
WHERE posao='upravnik';
```

```
INSERT INTO UPRAVNIK
SELECT idbr, ime, datzap, brod
FROM RADNIK
WHERE posao='upravnik';
```

```
INSERT INTO UPRAVNIK (brod, idbr, ime, datzap)
SELECT brod, idbr, ime, datzap
FROM RADNIK
WHERE posao='upravnik';
```

U ovom slučaju potrebno je navesti imena atributa i to istim redosledom u obe tabele, i onu u koju se dodaju podaci i onu iz koje se čitaju.

Primer 199. Kreirati tabelu ZAPOSLENI sa svim istim atributima kao i u tabeli RADNIK i u nju dodati sve zaposlene.

```
SELECT * INTO ZAPOSLENI
FROM RADNIK;
```

Ovim upitom se kreira tabela ZAPOSLENI sa istim atributima kao i u tabeli RADNIK i u nju se dodaju svi zapisi iz tabele RADNIK.

Primer 200. U nepostojeću tabelu KVALIF_VKV sa atributima *idbr*, *ime*, *brod* i u nju smestiti sve zaposlene koji imaju kvalifikaciju VKV.

SELECT idbr, ime, brod **INTO** KVALIF_VKV
FROM RADNIK
WHERE kvalif=*'VKV'*;

Napomena: Posle izvršenja ovog upita u bazi se kreira nova tabela KVALIF_VKV, sa atributima: *idbr*, *ime*, *brod*, i u nju su dodati odgovarajući zapisi o radnicima čija je kvalifikacija VKV.

Primer 201. Dodati zaposlenog čije je *idbr* 6234 na projekat Izgradnja.

INSERT INTO UCESCE
SELECT RADNIK.idbr, PROJEKAT.brproj
FROM RADNIK, PROJEKAT
WHERE RADNIK.idbr=*6234* **AND** PROJEKAT.imeproj=*'Izgradnja'*;

Primer 202. Dodati zaposlenog čije je ime *Marko* na projekat Izgradnja.

INSERT INTO UCESCE(idbr, brproj)
SELECT idbr, brproj
FROM RADNIK, PROJEKAT
WHERE Ime=*'Marko'* **AND** imeproj=*'Izgradnja'*;

Rezultat i odgovor sistema na prethodna dva upita je isti (slučajno, jer postoji samo jedan *Marko* i njegov idbr je *6234*). U tabelu Ucesce bi bio dodat jedan red:

Primer 203. U relaciju POVIŠICA <idbr, povišica, datzap> dodati podatke o analitičarima i vozačima i dati im povećanje plate od 15%.

INSERT INTO POVIŠICA (idbr, povišica, datzap)
SELECT idbr, plata*1.15, datzap
FROM RADNIK
WHERE posao **IN** (‘*analitičar’*, ‘*vozač’*);

Napomena: Naredba INSERT ubacuje u tabelu POVIŠICA podatke o svim analitičarima i vozačima iz tabele RADNIK, pri čemu platu povećava za 15%. Naravno, originalni podaci o platama radnika u tabeli RADNIK ostali su nepromenjeni.

Primer 204. Istovremeno kreirati tabelu ODELJENJE30 i u nju dodati idbr, ime, prezime, plata i brod iz tabele RADNIK, za zaposlene u odeljenju 30.

SELECT idbr, ime, prezime, plata, brod **INTO** ODELJENJE30
FROM RADNIK
WHERE BROD=*30*;

Odgovor sistema je:

Napomena: Posle izvršenja ovog upita u bazi se kreira nova tabela ODELJENJE30, sa atributima: *idbr*, *ime*, *prezime*, *plata*, *brod*, i u nju su dodati odgovarajući zapisi o radnicima iz odeljenja 30.

U SUBP Oracle se ovaj primer radi na sledeći način:

```
CREATE TABLE ODELJENJE30 AS
SELECT idbr, ime, prezime, plata, brod
FROM RADNIK
WHERE BROD=30;
```

Primer 205. Istovremeno kreirati tabelu PROJEKAT200 i u nju dodati ime i prezime zaposlenih, broj projekta na kome rade i funkciju koju obavljaju na tom projektu, za zaposlene koji su angažovani na projektu 200.

```
SELECT ime, prezime, brproj, funkcija INTO PROJEKAT200
FROM RADNIK, UCESCE
WHERE RADNIK.idbr=UCESCE.idbr AND brproj=200;
```

Odgovor sistema je:

Brisanje slogova

Brisanje slogova u relaciji može se izvesti pojedinačno ili grupno. Komandom **DELETE** uvek se briše cela n-torka, a ne samo pojedina vrednost nekog atributa. Sintaksa naredbe u opštem slučaju glasi:

DELETE FROM ime relacije **[WHERE** lista uslova **]**

Primer 206. Izbrisati sve podatke iz tabele KVALIF_VKV.

I) **DELETE** * **FROM** KVALIF_VKV;

II) **DELETE** KVALIF_VKV.* **FROM** KVALIF_VKV;

III) **DELETE FROM** KVALIF_VKV;

Napomena: Ako nije naveden uslov (WHERE), brišu se svi podaci iz tabele. Daleko efikasnija od naredbe DELETE za brisanje svih podataka iz tabele je naredba **TRUNCATE**. Ova naredba briše sve podatke iz tabele, ali ne i definiciju tabele.

Primer 207. U relaciji RADNIK obriši sve podatke o radniku *idbr = 5953* koji je otišao u penziju.

```
DELETE FROM RADNIK
WHERE idbr = 5953;
```

Odgovor sistema je:

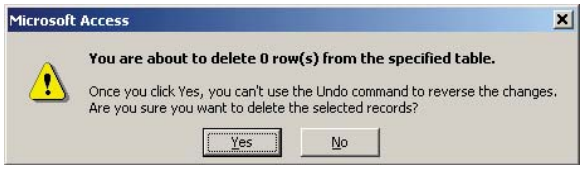
Primer 208. Izbrisati podatke o radnicima koji su se zaposlili pre 17-feb-90.

```
DELETE FROM RADNIK
WHERE RADNIK.datzap < '17-feb-90 ';
```

Primer 209. Izbrisati podatke o svim radnicima koji rade u odeljenju *priprema*.

```
DELETE FROM RADNIK
WHERE RADNIK.brod = (SELECT brod
                     FROM ODELJENJE
                     WHERE imeod= 'priprema');
```

Odgovor sistema je:



U ovom primeru neće biti izbrisan ni jedan zapis jer ne postoji odeljenje sa imenom *priprema*.

Primer 210. Zaposlene čiji je rukovodilac Pavle izbaciti sa svih projekata.

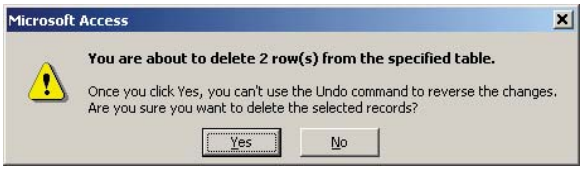
```
DELETE *
FROM UCESCE
WHERE idbr IN (SELECT idbr
               FROM RADNIK
               WHERE rukovodilac IN (SELECT idbr
                                     FROM RADNIK
                                     WHERE ime='Pavle'));
```

Odgovor sistema je:

Primer 211. Radnika sa najmanjim primanjima u preduzeću izbaciti sa svih projekata.

```
DELETE *
FROM UCESCE
WHERE idbr IN (SELECT idbr
               FROM RADNIK
               WHERE plata+NZ(premija,0) IN
                                     (SELECT MIN (plata+NZ(premija, 0))
                                      FROM RADNIK));
```

Odgovor sistema je:



Primer 212. Sa projekta Izgradnja ukloniti zaposlenog čije je ime *Marko*.

```
DELETE
FROM UCESCE
WHERE  idbr = (SELECT idbr FROM radnik WHERE ime='Marko')
   AND brproj = (SELECT brproj FROM projekat WHERE imeproj='izgradnja')
```

Primer 213. Obrisati sa svih projekata radnike čija je premija 0.

```
DELETE *
FROM UCESCE
WHERE idbr IN (SELECT idbr
                    FROM RADNIK
                    WHERE premija=0);
```

Napomena! Ukoliko se obrati pažnja na izgled upita za brisanje može se uočiti velika sličnost sa klasičnim SELECT upitom. Upravo to navodi na jedan logičan redosled koraka pri brisanju podataka (posebno velikih grupa). Ako SUBP ne podržava transakcije, onda je to nepovratan postupak Ukoliko se izbrišu podaci koje nismo želeli, gotovo nemoguće ih je vratiti. Zbog toga je pre naredbe DELETE pogodno najpre pomoću SELECT upita izdvojiti podatke koje treba obrisati. Dobijeni skup podataka pažljivo pregledati i analizirati. Ukoliko se utvrdi da su to upravo podaci koje treba obrisati, onda jednostavno umesto odredbe SELECT u upitu upisati odredbu DELETE i izvršiti operaciju brisanja.

U prethodnom primeru, najpre bi izdvojili podateke pomoću upita:

```
SELECT *
FROM UCESCE
WHERE idbr IN (SELECT idbr
                    FROM RADNIK
                    WHERE premija=0);
```

Na monitoru se pojavljuje odgovor sistema u obliku tabele:

Idbr	Brproj	Brisati	Funkcija
5696	200	2000	ŠEF
5696	300	2000	IZVRŠILAC
5953	100	1000	IZVRŠILAC
5953	300	1000	IZVRŠILAC

Kada se analizom podataka utvrdi da je to upravo grupa podataka koje treba obrisati onda se to može bezbedno učiniti.

Menjanje postojećih podataka

Modifikacija postojećih podataka (ažuriranje u užem smislu) izvodi se komandom **UPDATE - SET**. Opšti oblik naredbe glasi:

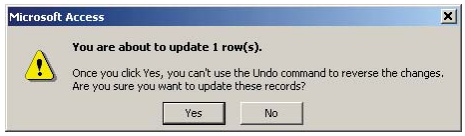
UPDATE ime relacije
SET atribut1=vrednost1[, atribut2=vrednost2, **...**]
WHERE [lista uslova]

Naredbom UPDATE - SET može se menjati vrednost samo jednog ili više podataka unutar jedne n-torke.

Primer 214. Zaposlenom čija je šifra *idbr = 5932* dodeliti rukovodioca čija je šifra *5842*.

UPDATE RADNIK SET rukovodilac = 5842
WHERE idbr = 5932;

Odgovor sistema je:



Naredbom UPDATE - SET može se menjati i vrednost jednog atributa unutar više n-torki.

Primer 215. Prebaci sve zaposlene iz odeljenja 30 u odeljenje 20, a naredba glasi:

UPDATE RADNIK **SET** brod = 20
WHERE brod **= 50;**

Odgovor sistema je:

Primer 216. Povećati platu za 10% svim zaposlenim u odeljenju 10.

UPDATE RADNIK **SET** plata=plata*1.1
WHERE brod = *10*;

Naredbom UPDATE se menjaju vrednosti atributa koji se navedu. U ovom slučaju postojeća vrednost plate se množi sa 1.1 da se dobije plata uvećana za 10% i to postaje nova vrednost plate za zaposlene iz odeljenja 10.

Primer 217. Odeljenje Plan premestiti na Voždovac.

UPDATE ODELJENJE **SET** MESTO=*'Voždovac'*
WHERE IMEOD=*'Plan'*;

Primer 218. Zaposlenima u odeljenju smeštenom na Novom Beogradu povećati platu 30%.

```
UPDATE RADNIK
SET plata=plata*1.3
WHERE brod IN ( SELECT brod
                FROM ODELJENJE
                WHERE mesto='Novi Beograd');
```

Odgovor sistema je:

Primer 219. Zaposlenima u odeljenju smeštenom na Dorćolu dodeliti premiju u iznosu 40% od plate, ne uzimajući u obzir direktore i upravnike.

```
UPDATE RADNIK
SET premija=plata*0.4
WHERE brod IN ( SELECT brod
                FROM ODELJENJE
                WHERE mesto='Dorćol')
                AND posao NOT IN ('direktor', 'upravnik');
```

Primer 220. Svim konsultantima smanjiti platu za 25% i ukinuti premiju.

```
UPDATE RADNIK
SET plata= plata*0.75, premija=NULL
WHERE idbr IN (SELECT idbr
               FROM UCESCE
               WHERE funkcija='konsultant');
```

Odgovor sistema je:

Napomena: Prilikom upotrebe naredbi za ažuriranje neophodno je biti veoma oprezan jer one menjaju stanje baze, tj. vrednosti podataka. Zbog toga se preporučuje da se najpre pomoću obične SELECT naredbe izdvoje n-torke na koje bi se primenile naredbe za ažuriranje, pa kada posle analize rezultata nepobitno utvrdimo da se radi o željenoj grupi n-torki, onda kreirati odgovarajuću naredbu UPDATE ili DELETE. MS Access-u u tu svrhu ima takozvane akcione upite pomoću kojih se mogu ne samo ažurirati podaci već i kreirati pogledi i tabele.

Napomena: U SUBP koje imaju transakcije ove naredbe nemaju nikakvo dejstvo na podatke u fizičkim tabelama (na disku). Odnose se samo na kopije podataka u radnoj memoriji računara dok se njihovo dejstvo ne prenese na fizičke podatke kao transakcija.

Inače, možda je najbolje rešenje bar za početnike u radu da se umesto brisanja (DELETE) podaci samo učine nedostupnim. Ovo se jednostavno može postići dodavanjem jedne dodatne kolone u svaku tabelu, na primer sa nazivom *Obrisan*. U ovu kolonu sistem automatski upisuje neku podrazumevanu (DEFAULT) vrednost, na primer: NE. Kada želimo

da “obrišemo“ neke podatke, onda umesto DELETE upita uradimo UPDATE i za izabrane slogove u kolonu *Obrisan* upišemo vrednost DA. Tada bi upit: Obrisati sa svih projekata radnike čija je premija 0 (Primer 213) imao oblik:

```
UPDATE  UCESCE
SET Obrisan = “DA“
WHERE idbr IN (SELECT idbr
                FROM RADNIK
                WHERE premija=0);
```

U svim upitima za rad sa podacima bi postojala odredba WHERE i u njoj uslov Obrisan = “NE“.

Ako već postoji neki uslov za selekciju redova onda se na njega dodaje AND Obrisan = “NE“.

Primer 221. Prikazati podatke o radnicima iz odeljenja 20 koji imaju premiju 0. (U tabelu RADNIK prethodno je dodata kolona *Obrisan*).

```
SELECT *
FROM RADNIK
WHERE (brod = 20 AND premija=0)      Komentar: Ovo je stvarni uslov zadatka
AND Obrisan = “NE“.                  Komentar: Ovo je dopunski uslov
```

Kreiranje i upotreba okidača (Trigger)

Okidač (**trigger**) je imenovani blok naredbi koji se izvršava onda kada se u sistemu desi događaj kom je pridružen okidač. Ti događaji se najčešće odnose na izvršavanje naredbi za ažuriranje INSERT, UPDATE ili DELETE. Okidači su, u stvari, specijalna vrsta zapamćenih procedura koje se pokreću posle promene vrednosti ili neposredno pre ili posle prenosa dejstva (naredbom COMMIT WORK) ovih operacija. Ako izvršenje okidača ne uspe, podaci se ne ažuriraju i aplikacija dobija poruku o grešci. Sem ovih događaja, okidače mogu pokrenuti i upiti (pre ili posle prikaza rezultujućih slogova), pri pokretanju ili zatvaranju aplikacije. Dakle, okidači se mogu definisati na nivou polja, bloka ili aplikacije. "Trigeri se mogu u bazama podataka efikasno koristiti za proveru dostignutih ograničenja u paketnoj obradi u više-korisničkom okruženju. Prednost imaju DBMS koji mogu koristiti čitav set upisanih ili izbrisanih zapisa u jednom pozivu okidačke funkcije (pomoću SQL naredbe) u odnosu na DBMS koji dopuštaju samo red po red upis/brisanje (za svaki red posebno) ".[22]

Okidači se koriste da bi se obezbedilo poštovanje poslovnih pravila u bazi podataka. Koriste se u slučajevima kada ne mogu da se primene standardne tehnike ograničenja ili obezbeđivanja deklarativnog referencijalnog integriteta definisanog nad tabelama. Koriste se kod aplikacija koje izvršavaju mnogo kaskadnih operacija nad drugim tabelama i slogovima.

Okidači se najčešće koriste za automatsko održavanje vrednosti u izvedenim kolonama, proveru pravila sigurnosti, sprečavanje nedozvoljenih operacija, žurnalsko praćenje događaja u sistemu, ali i za sprovođenje složenih pravila poslovanja.

Svaki SUBP ima svoju sintaksu kako treba kreirati okidač i te sintakse su međusobno slične.

Sintakse za kreiranje okidača

Sintaksa za kreiranje okidača u SUBP **MS SQL Server** je sledeća:

```
CREATE TRIGGER [ ime_šeme . ]ime_okidača
ON naziv_tabele
[ WITH <dml_opcije_okidača> [ ...,n ] ]
AFTER | INSTEAD OF
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS {
  sql_naredbe [ ...n ]
}
```

gde je:

[ime_šeme .]ime_okidača	Definiše opciono ime_šeme i obavezan naziv okidača (npr. dbo.trgUpamtiVremeIzvrsenja)

[22] [29] S. Ilić, S. Obradović, "Efficient Validation of Exceeded Limit Values in Batch Data Loading by Database Trigger", International scientific conference UNITECH'09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 393-I 398, ISSN:1313-230X.].

naziv_tabele	Naziv tablice nad kojom pratimo ažuriranje podataka (npr. RADNIK)
dml_opcije_okidača	Omogućavaju navođenje šifriranja sadržaja okidača ili sigurnosnih pravila pod kojim će okidač biti izvršen
AFTER | INSTEAD OF	Ključna reč AFTER (eng. nakon, posle) označava da se okidač aktivira nakon što se završilo ažuriranje podataka u tablici nad kojom je okidač postavljen usled delovanja jedne ili više SQL naredbi iz skupa {INSERT, UPDATE i DELETE} Ključna reč INSTEAD OF (eng. umesto) označava da se u ovom slučaju samo okidač izvršava umesto da se izvrši ažuriranje podataka naredbom koja je aktivirala okidač. Ova vrsta okidača se može primenjivati i nad tablicama i nad pogledima (view). U telu okidača se obično vrši validacija (provera) mogućih promena izazvanih naredbama iz skupa {INSERT, UPDATE i DELETE} pre nego što se explicitno navede u telu okidača ta ista naredba koja je aktivirala okidač.
[INSERT] [,] [UPDATE] [,] [DELETE]	Definišu nakon kojih od navedenih SQL operacija okidač treba da bude izvršen. Moguće je navesti više događaja koji aktiviraju okidač odvajajući ih zapetom
NOT FOR REPLICATION	Ova opcija se navodi u slučaju replikacije (kopiranja sadržaja baze podataka sa jednog servera na drugi) da se prilikom kopiranja podataka okidač ne izvrši
sql_naredbe [...n]	Skup od jedne ili više SQL naredbi koje treba da se izvrše prilikom aktiviranja okidača

Ako korisnik želi da izmeni jednu ili više naredbi u telu okidača (sql_naredbe [...n]) ne mora da prvo briše postojeći okidač (naredbom DROP trigger [ime_šeme.]ime_okidača) i nanovo kreira novi, već to može uraditi naredbom:

```
ALTER TRIGGER [ ime_šeme . ]ime_okidača
ON naziv_tabele
[ WITH <dml_opcije_okidača> [ ...,n ] ]
AFTER | INSTEAD OF
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS {
  sql_naredbe [ ...n ]
}
```

Jednom kreirani okidač može da se privremeno de-aktivira naredbom:

DISABLE TRIGGER [ime_šeme .]ime_okidača
ON naziv_tabele

i kasnije aktivira naredbom:

ENABLE TRIGGER [ime_šeme .]ime_okidača
ON naziv_tabele

Sintaksa za kreiranja okidača u **SUBP MySQL** ima sledeći oblik:

```
CREATE
  [DEFINER = { db_korisnik | CURRENT_USER }]
  TRIGGER     ime_okidača     vreme_okidanja
              događaj_okidanja
  ON naziv_tabele
FOR EACH ROW telo_okidača
```

gde je:

db_korisnik	Korisničko ime korisnika koji je kreirao okidač, određuje korisnička prava i proverava privilegije korisnika u momentu kada se aktivira okidač, U SUBP MySQL samo korisnik sa SUPER privilegijama može da kreira okidač
ime_okidača	Naziv pod kojim se čuva okidač u SUBP
vreme_okidanja	Može biti jedna od dve ključne reči: BEFORE i AFTER koje označava kada se aktivira okidač: pre (BEFORE) ili nakon (AFTER) što se ažurira svaki red ponaosob
događaj_okidanja	Predstavlja naredbu koja aktivira okidač i ona pripada skupu {INSERT, UPDATE, DELETE}
telo_okidača	Skup SQL naredbi koje se izvršavaju kada se okidač aktivira

U SUBP MySQL ažuriranje SQL naredbi koje pripadaju telu okidača nije moguće, tako da se okidač mora prvo izbrisati naredbom DROP TRIGGER ime_okidača i ponovo kreirati naredbom CREATE TRIGGER.

Isto tako jedan okidač se ne može aktivirati sa više događaja okidanja u istom telu okidača, kao što je to slučaj kod MS SQL Servera gde se isti okidač može aktivirati kada se sadržaj tabele ažurira bilo kojom od naredbi: INSERT, UPDATE ili DELETE. Tada, za svaku naredbu koja ažurira tabelu moramo kreirati zaseban okidač sa potpuno istim naredbama u telu okidača. Za ovaj SUBP ne postoje komande za eksplicitno de-aktiviranje i nanovo aktiviranje okidača, mada uvek postoji „zaobilazni način“ (work-around) da se to uradi.

Sintaksa za kreiranja okidača u **SUBP Oracle** ima sledeći oblik:

```
CREATE [OR REPLACE] TRIGGER
   [ ime_šeme . ]ime_okidača
BEFORE | AFTER | INSTEAD OF
   dml_događaj ON naziv_tabele
[ON EACH ROW]
[REFERENCING OLD as naziv_starog_reda,
 NEW as naziv_novog_reda]
[WHEN (uslov)]
```

gde je:

OR REPLACE	Pokazuje da se okidač sa navedenim imenom može zameniti novim ukoliko već postoji
[ime_šeme .]ime_okidača	Definiše opciono ime_šeme i obavezan naziv okidača
BEFORE | AFTER | INSTEAD OF	Pokazuje kada i kako se aktivira okidač - pre (BEFORE), nakon (AFTER) i umesto (INSTEAD OF) ažuriranja tablice (ako se ne navode ključne reči ON EACH ROW) ili reda u tablici (ako se navode ključne reči ON EACH ROW)
dml_događaj	To je događaj koji aktivira okidač i on pripada skupu {INSERT, UPDATE, DELETE}. SUBP Oracle dozvoljava da se navedu više događaja unutar istog okidača i oni se razdvajaju ključnom rečju OR (npr. INSERT OR UPDATE)
[ON EACH ROW]	Ako su navedene ove ključne reči onda se okidač aktivira pri ažuriranju svakog reda, a ako nisu onda se okidač aktivira pri ažuriranju cele tablice
[REFERENCING OLD as naziv_starog_reda, NEW as naziv_novog_reda]	Podrazumevani nazivi starog (pre ažuriranja) i novog (nakon ažuriranja) reda u tabeli su **new** i **old**. Ovim ključnim rečima moguće je promeniti podrazumevane nazive redova novim imenima.
[WHEN (uslov)]	To je uslov koji je opisan SQL naredbom koji mora da se ispuni kako bi se aktivirao okidač

I u SUBP Oracle jednom kreirani okidač može da se privremeno de-aktivira naredbom:

ALTER TRIGGER [ime_šeme .]ime_okidača **DISABLE**

i kasnije aktivira naredbom:

ALTER TRIGGER [ime_šeme .]ime_okidača **ENABLE**

SUBP SQL Server i Oracle imaju i okidače koji se aktiviraju na tzv. DDL događaje (mi smo ovde naveli samo okidače koji se aktiviraju na DML događaje – komande koje ažuriraju podatke u tabelama), kao što su na primer: kreiranje/menjanje/brisanje indeksa nad tabelom, kreiranje/menjanje-brisanje tabele i druge, ali ove okidače nećemo opisivati u ovoj knjizi.

Primena okidača

U cilju demonstriranja upotrebe okidača (trigera) kreiraćemo tabelu ISTORIJA_RMESTA u kojoj će se voditi evidencija o promeni posla za svakog zaposlenog sa periodima trajanja tog posla. Tabela se sastoji od sledećih polja: *idbr*, *pocetak*, *kraj* i *posao*. U poljima *pocetak* i *kraj* se pamte datumi kada je zaposleni započeo, odnosno prestao da radi posao koji je naveden u tabeli RADNIK. U slučaju da zaposleni još uvek radi isti posao, u polju *kraj* se nalazi prazna (NULL) vrednost.

Primer 222. Kreirati tabelu ISTORIJA_RMESTA sa njenim ograničenjima

MS SQL:

```
CREATE TABLE istorija_rmesta(
  idbr int NOT NULL,
  pocetak SMALLDATETIME NOT NULL,
  kraj SMALLDATETIME,
  posao NVARCHAR(20) NOT NULL,
  PRIMARY KEY (idbr, pocetak)
) ;

ALTER TABLE istorija_rmesta
ADD CONSTRAINT fk_istrm_radnik
FOREIGN KEY (idbr) REFERENCES radnik(idbr);
```

MySQL:

```
CREATE TABLE istorija_rmesta(
  idbr int NOT NULL,
  pocetak DATE NOT NULL,
  kraj DATE ,
  posao VARCHAR(20) NOT NULL,
  PRIMARY KEY (idbr, pocetak)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;

ALTER TABLE istorija_rmesta
ADD CONSTRAINT fk_istrm_radnik
FOREIGN KEY (idbr) REFERENCES radnik(idbr);
```

```
Oracle:
CREATE TABLE istorija_rmesta(
  idbr NUMBER(6,0),
  pocetak DATE NOT NULL,
  kraj DATE ,
  posao VARCHAR2(20) NOT NULL,
  CONSTRAINT pk_istrm PRIMARY KEY(idbr, pocetak)
);

ALTER TABLE istorija_rmesta
ADD CONSTRAINT fk_istrm_radnik
FOREIGN KEY (idbr) REFERENCES radnik(idbr);
```

Tabelu ISTORIJA_RMESTA ćemo inicijalno popuniti trenutnim poslovima zaposlenih naredbom:

Primer 223. Ubaciti u tabelu ISTORIJA_RMESTA vrednosti iz polja: *idbr*, *datzap* i *posao* u odgovarajuća polja u tabeli ISTORIJA_RMESTA.

```
MS SQL, MySQL, Oracle:
INSERT INTO istorija_rmesta (idbr, pocetak, posao)
  SELECT idbr, datzap, posao
  FROM radnik;
```

Na ovaj način će se tabela popuniti sa onoliko slogova koliko ima zaposlenih u tabeli RADNIK, kolona *pocetak* će biti popunjena datumima zapošljavanja zaposlenih, a kolona *kraj* će imati praznu (NULL) vrednost za sve zaposlene. Sadržaj prvih desetak redova tabele ISTORIJA_RMESTA dat je na slici.

idbr	pocetak	kraj	posao
5367	01.01.1978	NULL	vozač
5497	17.02.1990	NULL	električar
5519	07.11.1991	NULL	prodavac
5652	31.05.1980	NULL	električar
5662	12.08.1993	NULL	upravnik
5696	30.09.1991	NULL	čistač
5780	11.08.1984	NULL	upravnik
5786	22.05.1983	NULL	upravnik
5842	15.12.1981	NULL	direktor
...	...	...	...

Kreiraćemo okidač *trgInsRadnik* nad tabelom RADNIK koji će imati zadatak da nakon ubacivanja novog zapisa, odnosno podataka o novom zaposlenom u tabeli RADNIK istovremeno u tabeli ISTORIJA_RMESTA ubaci novi slog.

Primer 224. Kreirati okidač trgInsRadnik koji će se aktivirati nakon ubacivanja (INSERT) novog sloga u tabelu RADNIK tako što će ubaciti u tabelu ISTORIJA_RMESTA odgovarajuće novo-unete vrednosti za polja *idbr*, *pocetak* i *posao*

MS SQL:

```
CREATE TRIGGER trgInsRadnik ON radnik
AFTER INSERT
AS
BEGIN
      insert into istorija_rmesta (idbr, pocetak, posao)
      SELECT idbr, getdate(), posao
      FROM INSERTED
END
```

MySQL:

```
DELIMITER $$
CREATE TRIGGER trgInsRadnik
AFTER INSERT ON radnik
FOR EACH ROW
BEGIN
      insert into istorija_rmesta (idbr, pocetak, posao)
      values(new.idbr, current_date, new.posao);
END$$
DELIMITER ;
```

Oracle:

```
CREATE or REPLACE TRIGGER trgRadnikIns
AFTER INSERT
    ON RADNIK
    FOR EACH ROW
BEGIN
    INSERT INTO istorija_rmesta (idbr, pocetak, posao)
    VALUES (:new.idbr, CURRENT_DATE, :new.posao);
END;
```

Napomena: u SUBP MySQL običan korisnik ne može kreirati okidač, već samo korisnik sa Super privilegijama (administrator/root).

Kao što se može videti iz programskog koda za svaki SUBP, prilikom kreiranja okidača u SUBP Oracle i MySQL postoji deo naredbe „FOR EACH ROW“ koji nije prisutan u sintaksi kod MS SQL servera. To je posledica interne implementacije (primene) okidača kod pomenutih SUBP.

Kod SUBP Oracle i MySQL u telu okidača (u naredbama koje se izvršavaju kada se okidač aktivira) se pristupa svakom pojedinačnom redu (slogu) koji se ubacuje (insert) u tabelu nad kojom se postavlja okidač poslednjom INSERT naredbom i koji se simbolički naziva „new“. Vrednostima polja u tom redu se pristupa tako što se iza naziva reda doda tačka i naziv polja kojem se pristupa (npr. new.idbr, new.posao, itd.)

U SUBP MS SQL Server se u okidaču formira tabela INSERTED u kojoj se nalaze svi redovi (slogovi) koji se ubacuju (insertuju) u tabelu nad kojom se postavlja okidač poslednjom INSERT naredbom, pa se u telu okidača mora voditi računa da tabela INSERTED može imati više od jednog reda. Izvršenje okidača ćemo demonstrirati ubacivanjem novog zaposlenog.

Primer 225. Ubaciti u tabelu RADNIK novog zaposlenog: Milanku Crnogorac sa idbr=3887 koja je počela sa radom 1. Januara 2012. godine na radnom mestu analitičara sa platom od 2000 novčanih jedinica

MS SQL, MySQL (*u ovom SUBP ukloniti N ispred '*) :

```
INSERT INTO radnik(idbr, ime, prezime, posao, kvalif,
rukovodilac, datzap, premija, plata, brod)
values(3887, N'Milanka', N'Crnogorac', N'analitičar', N'VSS',
NULL, '2012-01-01', NULL, 2000, NULL);
```

Oracle:

```
INSERT INTO radnik(idbr, ime, prezime, posao, kvalif,
rukovodilac, datzap, premija, plata, brod)
values(3887, 'Milanka', 'Crnogorac', 'analitičar', 'VSS', NULL,
to date('01012012', 'MMDDYYYY'), NULL, 2000, NULL);
```

po ubacivanju radnika sa idbr=3887 u telu okidača, u SUBP Oracle i MySQL će se formirati red **new** (MySQL) odnosno **:new** (Oracle) kao na slici ispod sa konkretnim vrednostima u svakom polju. Sivom bojom je zasenčena vrednost polja new.posao (MySQL) odnosno :new.posao (Oracle).

new

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	Premija	Plata	Brod
3887	Milanka	Crnogorac	analitičar	VSS	NULL	25.06.2012	NULL	2000	NULL

U MS SQL serveru će se formirati tabela INSERTED

INSERTED									
idbr	**Ime**	**prezime**	**posao**	**Kvalif**	**rukovodilac**	**datzap**	**premija**	**Plata**	**brod**
3887	Milanka	Crnogorac	analitičar	VSS	NULL	25.06.2012	NULL	2000	NULL

Možemo primetiti da zbog razlike objekata „new“ (:new) i INSERTED koji postoje u okidačima, imamo i drugačije programske kodove u telima okidača u SUBP MySQL i Oracle u odnosu na MS SQL server. U MySQL i Oracle komanda INSERT se u telu okidača izvršava sa delom VALUES koji pokazuje da se radi o ubacivanju jednog reda sa konkretnim vrednostima, dok se u MS SQL serveru u telu okidača komanda INSERT izvršava preko nastavka SELECT – to znači da se u tabelu ISTORIJA_RMESTA ubacuje više redova sa određenim kolonama.

Rezultat izvršenja naredbe ubacivanja podataka za novog radnika (Milanke Crnogorac sa primera 218, čiji je idbr=3887) ogleda se u ubacivanju podataka za novog zaposlenog u tabelu RADNIK, a zatim se pod dejstvom okidača, koji se aktivira nakon ubacivanja ovog sloga u tabeli RADNIK, izvršavaju naredbe u telu okidača – ubacivanje novog sloga u tabeli

ISTORIJA_RMESTA što se može videti na sledećoj slici. Masnim slovima je obeležen novi slog.

*SELECT * FROM istorija_rmesta*

Idbr	pocetak	kraj	posao
3887	**23.07.2012**	**NULL**	**analitičar**
5367	01.01.1978	NULL	vozač
5497	17.02.1990	NULL	električar
5519	07.11.1991	NULL	prodavac
5652	31.05.1980	NULL	električar
5662	12.08.1993	NULL	Upravnik
5696	30.09.1991	NULL	Čistač
5780	11.08.1984	NULL	Upravnik
5786	22.05.1983	NULL	Upravnik
5842	15.12.1981	NULL	Director
...	...	...	...

Ako nekom od zaposlenih promenimo vrednost kolone *posao* u tabeli RADNIK (to je u suštini isto što i brisanje postojećeg reda tog zaposlenog i dodavanje novog reda sa istim podacima kao u izbrisanom redu samo sa promenjenom vrednošću u koloni posao), u tabeli ISTORIJA_RMESTA trebalo bi da se izmeni postojeći zapis i doda novi. Da bi se to postiglo treba:

1) pronaći slog u kojem je polje *kraj* prazno (NULL) za *idbr* tog zaposlenog i čija je vrednost polja *posao* jednaka staroj vrednosti posla,
2) popuniti vrednost polja *kraj* današnjim datumom i
3) dodati novi slog čija je vrednost polja *pocetak* današnji datum, vrednost polja *kraj* prazna (NULL) i vrednost polja *posao* jednaka novom poslu zaposlenog iz tabele RADNIK.

Kako okidač treba da se aktivira nakon promene vrednosti u polju *posao* u tabeli RADNIK, to ćemo kreirati AFTER UPDATE okidač nad tabelom RADNIK.

Primer 226. Kreirati okidač trgRadnikUpdate koji će se aktivirati nakon ažuriranja (UPDATE) vrednosti u koloni *posao* tabele RADNIK i koji će postaviti vrednost u koloni *kraj* u tabeli ISTORIJA_RMESTA na današnji datum (za zaposlenog čiji se posao menja) i nakon toga ubaciti novi red sa novom vrednošću posla u istu tabelu

MS SQL:

```
CREATE TRIGGER trgRadnikUpdate on radnik
AFTER UPDATE
AS
BEGIN
  DECLARE @zavrsetak SMALLDATETIME;
  SET @zavrsetak = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');
```

MS SQL:

```
  -- nastavak sa prethodne strane
  IF UPDATE(posao)
  BEGIN
      UPDATE istorija_rmesta
      SET kraj = @zavrsetak
      FROM istorija_rmesta irm, inserted i, deleted d
      WHERE irm.idbr = i.IDBR AND i.idbr = d.idbr AND irm.kraj is NULL
            AND irm.posao = d.posao AND i.posao != d.posao;

            INSERT INTO istorija_rmesta (idbr, pocetak, posao)
            SELECT i.idbr, @zavrsetak, i.posao
            FROM INSERTED i, DELETED d
            WHERE i.idbr = d.idbr AND i.posao != d.posao
      END
END
```

MySQL:

```
DELIMITER $$
CREATE TRIGGER trgRadnikUpdate
AFTER UPDATE ON radnik
    FOR EACH ROW
BEGIN
  DECLARE zavrsetak DATE;
  SET zavrsetak = CURRENT_DATE;
  if (new.posao <> old.posao) then
      UPDATE istorija_rmesta
      SET kraj = zavrsetak
      WHERE idbr = new.idbr AND kraj is NULL AND posao = old.posao;

      INSERT INTO istorija_rmesta (idbr, pocetak, posao)
      VALUES (new.idbr, zavrsetak, new.posao);
  end if;
END$$
DELIMITER ;
```

U navedenim primerima kreiranja okidača koji se izvršavaju nakon ažuriranja (operacija UPDATE) slogova u tabeli RADNIK možemo primetiti da se naredbe UPDATE i INSERT unutar tela okidača izvršavaju nakon provere da li se u tabeli RADNIK menja vrednost u polju *posao*. Za svaki pokazani SUBP realizacija provere je različita:

- U MS SQL serveru u telu okidača je postavljena naredba IF UPDATE(posao) kojom se proverava da li je došlo do promene vrednosti u polju *posao*. Ako jeste izvršavaju se UPDATE i INSERT naredbe koje se nalaze u telu okidača;
- U MySQL serveru u telu okidača proverava se da li je stara vrednost polja posao u tabeli RADNIK različita od nove vrednosti (IF new.posao <> old.posao) i u zavisnosti od toga izvršavaju se UPDATE i INSERT naredbe koje se nalaze u telu okidača;
- U Oracle serveru se u samoj definiciji kreiranja okidača podešava da se on okida samo u slučaju da je izvršena promena vrednosti u polju *posao*.

Oracle:

```
CREATE or REPLACE TRIGGER trgRadnikUpdate
AFTER UPDATE of posao
    ON RADNIK
    FOR EACH ROW
DECLARE
  localIDBR RADNIK.IDBR%TYPE;
  zavrsetak DATE;

BEGIN
  localIDBR := :new.idbr;
  SELECT CURRENT_DATE into zavrsetak FROM DUAL;

  UPDATE istorija_rmesta SET kraj = zavrsetak
  WHERE idbr = localIDBR AND kraj is NULL AND posao = :old.posao;

  INSERT INTO istorija_rmesta (idbr, pocetak, posao)
  VALUES (localIDBR, zavrsetak, :new.posao);
END;
```

Iz navedenih programskih kodova može se isto tako videti da se promenljive unutar tela okidača deklarišu nakon ključne reči BEGIN, kojom se započinje skup naredbi u telu okidača, za MS SQL i MySQL server i to ključnom reči DECLARE, a da se kod Oracle servera one deklarišu ispred ključne reči BEGIN. Isto tako, u prva dva SUBP-a za svaku promenljivu mora da se navede tačno određeni tip podataka, dok u SUBP Oracle to može da se uradi dvojako: definisanjem tačno određenog tipa (NUMBER, VARCHAR2, itd.) ili preko tipa koji je već dodeljen nekom polju u tabeli (RADNIK.IDBR%TYPE – što znači: isti tip kao u polju *idbr* iz tabele RADNIK).

Tekući datum (datum koji se odnosi na tekući dan) u različitim SUBP se različito dobija. U MS SQL se to radi preko funkcije GET_DATE(), a u MySQL-u i Oracle-u preko sinonima CURRENT_DATE. MS SQL funkcija GET_DATE() vraća ne samo tekući datum, već i tekuće vreme, pa je iz te vrednosti potrebno odstraniti vremenski deo (tj. zadržati datumski deo) funkcijom Convert(varchar(10), getdate(), 102) kojom se dobija datum u obliku YYYY.MM.DD (2012.08.23). Naredbom REPLACE, u primeru, zamenjujemo tačku (.) crticom(-), pri čemu dobijamo standardni datumski string: YYYY-MM-DD (2012-08-23).

Rezultat rada okidača kojeg smo kreirali možemo pratiti ako izvršimo SQL komandu UPDATE nad tabelom RADNIK.

Primer 227. Zameniti posao u tabeli RADNIK vrednošću "šef smene" za zaposlenog sa idbr=3887

MS SQL, MySQL, Oracle (*napomena: za Oracle i MySQL ukloniti N ispred '*):

```
UPDATE radnik SET posao = N'šef smene' WHERE idbr = 3887;
```

Izvršenjem navedene UPDATE komande u SUBP MySQL i Oracle će se kreirati dva reda u okidaču: **old** i **new** (MySQL) odnosno **:old** i **:new** (Oracle) :

old

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	Plata	Brod
3887	Milanka	Crnogorac	analitičar	VSS	NULL	25.06.2012	NULL	2000	NULL

new

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	Plata	Brod
3887	Milanka	Crnogorac	šef smene	VSS	NULL	25.06.2012	NULL	2000	NULL

a u MS SQL serveru dve tabele: INSERTED i DELETED:

DELETED

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	brod
3887	Milanka	Crnogorac	analitičar	VSS	NULL	25.06.2012	NULL	2000	NULL

INSERTED

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	brod
3887	Milanka	Crnogorac	šef smene	VSS	NULL	25.06.2012	NULL	2000	NULL

Rezultat delovanja okidača koji se pokreće nakon izmene podatka (Update) nad tabelom RADNIK vidi se u tabeli ISTORIJA_RMESTA. Masnim slovima su naglašene promene – ažurirana vrednost u koloni *kraj* za posao „analitičar“ i ubačen novi red za *posao* „šef smene“. Voditi računa da je autor knjige ovaj upit izvršio 23.08.2012, pa se zato pojavljuje ovaj datum kao tekući.

*SELECT * FROM istorija_rmesta*
WHERE idbr = 3887

idbr	pocetak	kraj	posao
3887	23.07.2012	**23.08.2012**	analitičar
3887	**23.08.2012**	**NULL**	**šef smene**

Kako se u telima okidača u SUBP Oracle i MySQL vidi po jedan red sa starim i novim vrednostima (UPDATE = DELETE + INSERT), to je lako izvršiti UPDATE u tabeli ISTORIJA_RMESTA u odnosu na vrednosti iz pomenutih redova. U telu okidača u MS SQL serveru ažuriranje i ubacivanje novih slogova u tabelu ISTORIJA_RMESTA je nešto složenije (preko složene naredbe UPDATE i INSERT into/SELECT) i biće bolje objašnjeno na narednom primeru.

SQL naredbom kojom se ažurira više redova u tabeli RADNIK pokazaćemo kako se izvršava okidač.

Primer 228. U tabeli RADNIK zameniti posao "načelnik" u "upravnik"

```
MS SQL, MySQL, Oracle (napomena: za Oracle i MySQL ukloniti N ispred '):
UPDATE radnik SET posao = N'načelnik' WHERE posao = N'upravnik';
```

Komandom iz prethodnog primera će se u tabeli RADNIK ažurirati 4 reda jer se menja posao kod 4 zaposlena radnika.

Napomena: U MySQL serveru, ova naredba će možda prikazati grešku da nije dozvoljeno ažurirati tabelu ako se u WHERE uslovu ne postavi uslov po primarnom ključu. U tom slučaju treba pokrenuti naredbu:

SET SQL_SAFE_UPDATES=0;

pre izvršenja UPDATE naredbe.

U SUBP Oracle i MySQL okidač će se izvršiti (aktivirati) četiri puta – za svaki ažurirani red po jednom, pa će sadržaji redova **new** i **old** u telu okidača biti:

old - 1

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
5662	Janko	Mančić	upravnik	VSS	6789	12.08.1993	NULL	2400	10

new - 1

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
5662	Janko	Mančić	načelnik	VSS	6789	12.08.1993	NULL	2400	10

old – 2

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
5780	Božidar	Ristić	upravnik	VSS	6789	11.08.1984	NULL	2200	20

new – 2

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
5780	Božidar	Ristić	načelnik	VSS	6789	11.08.1984	NULL	2200	20

old – 3

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
5786	Pavle	Šotra	upravnik	VSS	6789	22.05.1983	NULL	2800	30

new – 3

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
5786	Pavle	Šotra	načelnik	VSS	6789	22.05.1983	NULL	2800	30

old – 4

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
6789	Janko	Simić	upravnik	VSS	5842	23.12.2003	10	3900	40

new – 4

idbr	ime	prezime	posao	Kvalif	rukovodilac	datzap	premija	plata	Brod
6789	Janko	Simić	načelnik	VSS	5842	23.12.2003	10	3900	40

U SUBP MS SQL server se formiraju samo dve tabele i okidač se izvršava jednom. Tabele INSERTED i DELETED koje se formiraju tada date su na sledećoj slici.

DELETED									
idbr	**ime**	**prezime**	**Posao**	**Kvalif**	**rukovodilac**	**datzap**	**premija**	**plata**	**brod**
5662	Janko	Mančić	upravnik	VSS	6789	12.08.1993	NULL	2400	10
5780	Božidar	Ristić	upravnik	VSS	6789	11.08.1984	NULL	2200	20
5786	Pavle	Šotra	upravnik	VSS	6789	22.05.1983	NULL	2800	30
6789	Janko	Simić	upravnik	VSS	5842	23.12.2003	10	3900	40

INSERTED									
idbr	**ime**	**prezime**	**posao**	**Kvalif**	**rukovodilac**	**datzap**	**premija**	**plata**	**brod**
5662	Janko	Mančić	načelnik	VSS	6789	12.08.1993	NULL	2400	10
5780	Božidar	Ristić	načelnik	VSS	6789	11.08.1984	NULL	2200	20
5786	Pavle	Šotra	načelnik	VSS	6789	22.05.1983	NULL	2800	30
6789	Janko	Simić	načelnik	VSS	5842	23.12.2003	10	3900	40

U UPDATE naredbi okidača trgRadnikUpdate u SUBP MS SQL, prvo se povezuje tabela ISTORIJA_RMESTA sa tabelom INSERTED po polju *idbr* i vrednosti polja *kraj* is NULL, tabela INSERTED sa tabelom DELETED po polju *idbr* i uslovu da je vrednost polja *posao* u te dve tabele različita, a tabela DELETED sa tabelom ISTORIJA_RMESTA po polju *posao*. Kada se izvrši povezivanje (join) tada se u polju *kraj* (koje je bilo prazno) upisuje vrednost tekućeg datuma. Na sledećoj slici podvučenim (i.idbr, d.idbr, irm.idbr), zadebljanim (d.posao, irm.posao) i kosim (kraj) slovima su pokazane vrednosti u poljima po kojima se vrši spajanje tabela INSERTED, DELETED i ISTORIJA_RMESTA.

INSERTED join DELETED											ISTORIJA_RMESTA			
i.idbr d.idbr	ime	prezime	d.posao	i.posao	kvalif	rukovo-dilac	datzap	premija	plata	brod	irm.idbr	pocetak	kraj	irm.posao
5662	Janko	Mančić	**upravnik**	načelnik	VSS	6789	12.08.1993	NULL	2400	10	5662	12.08.1993	*NULL*	**upravnik**
5780	Božidar	Ristić	**upravnik**	načelnik	VSS	6789	11.08.1984	NULL	2200	20	5780	11.08.1984	*NULL*	**upravnik**
5786	Pavle	Šotra	**upravnik**	načelnik	VSS	6789	22.05.1983	NULL	2800	30	5786	22.05.1983	*NULL*	**upravnik**
6789	Janko	Simić	**upravnik**	načelnik	VSS	5842	23.12.2003	10	3900	40	6789	23.12.2003	*NULL*	**upravnik**

Nakon što se NULL vrednosti u izdvojenim redovima u tabeli ISTORIJA_RMESTA zamene tekućim datumom, naredbom INSERT into / SELECT, u istu tabelu se dodaju slogovi koji su rezultat promene/razlike između tabela INSERTED i DELETED. Tako na kraju imamo sledeće promene u tabeli ISTORIJA_RMESTA kao na sledećoj slici. Ponovo voditi računa da je autor knjige ovaj upit izvršio 23.08.2012, pa se zato pojavljuje ovaj datum kao tekući.

Kako UPDATE okidač treba da izvršava naredbe u telu okidača samo u slučaju da se ažurira vrednost polja *posao* u tabeli RADNIK, pokušaćemo da ažuriramo vrednost nekog drugog polja kako bi se uverili da tada okidač ne menja sadržaj tabele ISTORIJA_RMESTA. U tu svrhu izmenićemo datum zapošljavanja (vrednost u koloni datzap) u tablici RADNIK zaposlenom sa idbr = 5696 kao što je to pokazano u Primer 229.

*SELECT * FROM istorija_rmesta*
WHERE idbr in (SELECT idbr from radnik
WHERE posao = 'načelnik') order by idbr

idbr	pocetak	kraj	posao
5662	12.08.1993	**23.08.2012**	upravnik
5662	**23.08.2012**	**NULL**	**načelnik**
5780	11.08.1984	**23.08.2012**	upravnik
5780	**23.08.2012**	**NULL**	**načelnik**
5786	22.05.1983	**23.08.2012**	upravnik
5786	**23.08.2012**	**NULL**	**načelnik**
6789	23.12.2003	**23.08.2012**	upravnik
6789	**23.08.2012**	**NULL**	**načelnik**

Primer 229. Promeniti vrednost datuma zapošljavanja zaposlenog sa idbr=5696 na 29. septembar 1991. godine

MS SQL, MySQL:

```
UPDATE radnik SET datzap = '1991-09-29' WHERE idbr = 5696;
```

Oracle:

```
UPDATE radnik SET datzap = TO_DATE('19910929','YYYYMMDD')
WHERE idbr = 5696;
```

Možemo da se uverimo da se ništa nije promenilo u tabeli ISTORIJA_RMESTA za radnika sa idbr=5696, što se može videti na sledećoj slici.

*SELECT * FROM istorija_rmesta*
WHERE idbr = 5696

idbr	pocetak	kraj	posao
5696	30.09.1991	NULL	čistač

Zapamćene, uskladištene procedure (STORED PROCEDURE)

Zapamćene, uskladištene procedure (**stored procedure**) predstavljaju imenovane blokove naredbi sa opcionim nizom ulaznih i izlaznih parametara. One se izvršavaju na serveru. Imenovani skup procedura, kursora i promenljivih nazivaju se paket (PACKAGE). Procedure i funkcije se izvršavaju eksplicitnim pozivanjem. Zapamćene procedure se mogu pozvati iz neke aplikacije (napisane u nekom programskom jeziku: C, Visual Basic, i sl.), ili ih mogu pozivati pravila ili okidači koji obezbeđuju integritet podataka.

Izvršavaju se na lokalnom ili udaljenom serveru. Zapamćene procedure se izvršavaju vrlo brzo. To je najčešće posledica činjenice da su serveri vrlo moćne mašine, a sem toga svi potrebni podaci se fizički, po pravilu, nalaze na istoj mašini. Povećavaju bezbednost jer predstavljaju odlično sredstvo da se programski kontrolišu, pre svega, operacije dodavanja promene i brisanja podataka. Zapamćene procedure se prevode prvi put kada se izvrše i čuvaju se u sistemskoj tabeli. Pri prevođenju se i optimizuje u pogledu izbora najboljeg puta pristupa podacima.

Procedurama se mogu proslediti vrednosti nekih parametara (tada su ti parametri ulazni) a vrednosti nekih parametara se mogu odrediti (izračunati) u samoj proceduri i dobiti na njenom izlazu (tada su ti parametri izlazni). Najčešće služi da se skupom SQL naredbi koje se nalaze unutar tela procedure izvrši jedna celovita operacija.

Sintakse za kreiranje procedure

Sintaksa kreiranja procedure u **SUBP MS SQL** je:

```
CREATE { PROC | PROCEDURE }
   [ime_šeme.] ime_procedure
 [ { @parametar [ ime_šeme. ] tip_promenljive }
       [ = podrazumevana_vrednost ]
       [ OUT |   OUTPUT ] [READONLY]
] [ ,...n ]
[ WITH <opcije_procedure> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_naredbe [;] [ ...n ] [ END ] }
[;]
```

gde je:

CREATE { PROC | PROCEDURE }	U opštem slučaju kreiranje procedure počinje ključnim rečima CREATE PROCEDURE ali se ovo može i "skratiti" sintaksom CREATE PROC.
[ime_šeme.] ime_procedure	Definiše opciono ime_šeme i obavezan naziv procedure

@parametar [ime_šeme.] tip_promenljive	označava da se promenljive u procedurama moraju uvek označavati sa znakom @ na početku naziva svake promenljive a tip promenljive označava jedan iz skupa {bigint, bit, decimal, int, money, smallint, tinyint, numeric, smallmoney, float, real, date, smalldatetime, char, varchar, nchar, nvarchar, text, binary, etc} (npr. @plata money)
[= podrazumevana_vrednost]	Predstavlja podrazumevanu vrednost parametra unutar tela procedure ukoliko korisnik ne prosledi vrednost tog parametra proceduri prilikom pozivanja procedure (npr. @danasnjidatum smalldatetime = GetDate())
[OUT | OUTPUT]	Ako se vrednost parametra određuje unutar procedure i rezultat šalje nazad korisniku koji je proceduru pozvao onda se iza naziva parametra i tipa piše ključna reč OUTPUT ili "skraćeno" OUT (npr. @ukupnaplata money OUT)
[WITH <opcije_procedure> [,...n]]	<opcije_procedure> mogu biti [ENCRYPTION] (kada se telo procedure šifrira tako da administrator ne može da vidi listu SQL naredbi), [RECOMPILE] (obično, prilikom kreiranja procedure SUBP kreira optimalni plan izvršenja SQL naredbi u telu procedure, ovom naredbom se naređuje proceduri da kreira taj plan nanovo svaki put kada se procedura izvršava) i [EXECUTE AS] (ovde se navodi pod čijim korisničkim nalogom treba izvršiti proceduru)
[FOR REPLICATION]	Ove ključne reči navode da se procedura kreira za replikaciju
sql_naredbe [...n]	Skup od jedne ili više SQL naredbi koje treba da se izvrše unutar procedure

Sintaksa kreiranja procedure u **SUBP MySQL** je:

```
CREATE
   [DEFINER = { db_korisnik | CURRENT_USER }]
   PROCEDURE ime_procedure
([ IN | OUT | INOUT ] ime_parametra
  tip_parametra)
   [ COMMENT 'string' | LANGUAGE SQL |
[NOT] DETERMINISTIC | { CONTAINS SQL | NO
SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }]
telo_procedure
```

gde je:

db_korisnik	Korisničko ime korisnika koji kreira proceduru, određuje korisnička prava i proverava privilegije korisnika u momentu kada se izvršava procedura
[IN | OUT | INOUT] ime_parametra tip_parametra	Definiše parametar koji može biti "ulazni" (IN), "izlazni" ili "ulazno/izlazni" (isti parametar na ulazu u proceduru može imati jednu vrednost a na izlazu drugu) (npr. plata OUT decimal(17,2))
COMMENT 'string'	Predstavlja komentar koji ne utiče na izvršenje iste, ali autoru daje informaciju šta radi procedura (npr. Procedura služi da ažurira plate zaposlenih)
LANGUAGE	Pokazuje koji programski jezici su dozvoljeni unutar tela procedure - dozvoljena vrednost je SQL
DETERMINISTIC	Pokazuje da je procedura deterministička tj. da za skup istih ulaznih parametara uvek daje isti rezultat. Suprotno je NOT DETERMINISTIC što se može izostaviti u sintaksi.
CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA	To su ključne reči koje definišu "ponašanje" naredbi unutar tela procedure - CONTAINS SQL pokazuje da procedura ne sadrži naredbe koje čitaju ili upisuju podatke (npr. SET @x = 1), NO SQL pokazuje da procedura ne sadrži SQL naredbe, READS SQL DATA pokazuje da procedura sadrži naredbe koje čitaju podatke (npr SELECT naredba), ali ne i naredbe koje upisuju podatke i MODIFIES SQL DATA pokazuje da procedura sadrži naredbe koje mogu menjati neke podatke (npr. INSERT ili DELETE)
SQL SECURITY	Ova karakteristika može da sadrži ključne reči DEFINER (onaj koji je definisao proceduru) ili INVOKER (onaj koji je pokrenuo proceduru) kako bi se naznačilo da li će se procedura izvršiti pod privilegijama korisničkog naloga iza reči DEFINER ili naloga korisnika koji ju je pokrenuo (INVOKER)
sql_naredbe [...n]	Predstavlja jednu ili više naredbi u telu procedure koje se izvršava kada se pokrene procedura

Sintaksa kreiranja procedure u **SUBP Oracle** je:

```
CREATE [ OR REPLACE ] PROCEDURE
[ime_šeme.] ime_procedure
  [ (parametar [ { IN | OUT | IN OUT } ]
    [ NOCOPY ] tip_podatka [ DEFAULT izraz ]
  [, parametar [ { IN | OUT | IN OUT } ]
    [ NOCOPY ] ip_podatka [ DEFAULT izraz ]
      ].
    )
  ]
  [ AUTHID { CURRENT_USER | DEFINER } ]
  { IS | AS }
  { telo procedure } ;
```

gde je:

[ime_šeme .]ime_procedure	Definiše opciono ime_šeme i obavezan naziv procedure
(parametar [{ IN | OUT | IN OUT }] [NOCOPY] tip_podatka [DEFAULT vrednost])	Definiše parametar čija se vrednost: prosleđuje proceduri prilikom poziva (IN), čita kada se procedura završi (OUT) ili prvo prosleđuje prilikom poziva a zatim čita kada se procedura završi (IN OUT). Podrazumevana vrednost parametra (ako se ne naznači) je IN. Ključna reč NOCOPY je instrukcija bazi da prenese parametar proceduri što pre. Ako je parametar tipa IN onda se NOCOPY ne mora navoditi jer se podrazumeva. Tip podatka se navodi bez "veličine" podatka. Tako npr. VARCHAR2 je pravilan tip a VARCHAR2(50) nije. Podrazumevana vrednost za parametar se unosi iza ključne reči DEFAULT i obično je praćena znakom dodele vrednosti (:=)
[AUTHID { CURRENT_USER | DEFINER }]	Ovim ključnim rečima se navodi da li procedura treba da se izvrši sa ovlašćenjima i privilegijama korisnika koji ju je pokrenuo (CURRENT_USER) ili korisnika koji ju je kreirao (DEFINER).
IS | AS	Nakon definisanja imena, parametara i ovlašćenja procedure sledi ključna reč IS ili AS. Sasvim je svejedno koju ćemo ključnu reč upotrebiti
sql_naredbe [...n]	Predstavlja jednu ili više naredbi u telu procedure koje se izvršava kada se pokrene procedura

Primena procedura

Primer 230. Kreirati proceduru procUnesiOdeljenje čiji će zadatak biti da ubaci novo odeljenje u tabelu ODELJENJE

MS SQL:

```
CREATE PROCEDURE procUnesiOdeljenje(
  @parBrod int,
  @parImeod nvarchar(50),
  @parMesto nvarchar(50),
  @parSefod int)
AS
BEGIN
  INSERT INTO odeljenje(brod, imeod, mesto, sefod)
  VALUES (@parBrod, @parImeod, @parMesto, @parSefod);
END
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procUnesiOdeljenje(
  IN parBrod int,
  IN parImeod varchar(50),
  parMesto varchar(50),
  parSefod int)
BEGIN
  INSERT INTO odeljenje(brod, imeod, mesto, sefod)
  VALUES (parBrod, parImeod, parMesto, parSefod);
  commit;
END$$
DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procUnesiOdeljenje(
  parBrod IN ODELJENJE.BROD%TYPE,
  parImeod IN ODELJENJE.IMEOD%TYPE,
  parMesto VARCHAR2,
  parSefod ODELJENJE.MESTO%TYPE)
IS
BEGIN
  INSERT INTO odeljenje(brod, imeod, mesto, sefod)
  VALUES (parBrod, parImeod, parMesto, parSefod);
  commit;
END procUnesiOdeljenje;
```

U programskim kodovima kreiranja procedura sa Primer 230 možemo uočiti da:

- kod SUBP MS SQL ne pišemo za ulazne promenljive ključnu reč INPUT kako bi naznačili da su one promenljive tipa „ulaz“;
- kod SUBP MS SQL promenljive koje se koriste u proceduri se definišu pomoću specijalnog karaktera @ (npr. @parBrod);
- kod SUBP MySQL promenljive koje se koriste u procedure se mogu definisati pomoću karaktera @ ili bez njega;
- kod SUBP MySQL za ulazne promenljive možemo, ali i ne moramo, da koristimo ključnu reč IN da bi naznačili da su promenljive koje se koriste za procedure tipa “ulaz” (obratiti pažnju da se ključna reč IN prilikom korišćenja piše ispred naziva promenljive);
- kod SUBP Oracle u deklaraciji promenljivih koje se koriste u procedurama se ne deklariše veličina promenljive (npr. VARCHAR2(20) ili NUMBER(6)) već samo tip promenljive (npr. NUMBER, VARCHAR2); isto tako, umesto konkretnog tipa promenljive može se navesti i sintaksa “tip kao kod promenljive” (npr. ODELJENJE.IMEOD%TYPE – tip promenljive je isti kao tip polja imeod iz tabele ODELJENJE);

Kod SUBP Oracle za ulazne promenljive možemo ali i ne moramo da koristimo ključnu reč IN da bi naznačili da su promenljive koje se koriste u procedure tipa “ulaz” (obratiti pažnju da se ključna reč IN prilikom korišćenja piše iza naziva promenljive).

Proceduru izvršavamo pomoću ključne reči „execute“ (za SUBP MS SQL i Oracle) odnosno „call“ (za SUBP MySQL) i prosleđivanjem vrednosti za ulazne parametre onim redosledom kako su navedeni u deklaraciji procedure.

Primer 231. Izvršiti proceduru procUnesiOdeljenje uz prosleđivanje sledećih parametara: parBrod=70, parImeod='Vozni park', parMesto='Banovo Brdo', parSefod=5367

```
MS SQL
execute procUnesiOdeljenje 70, N'Vozni park', N'Banovo Brdo', 5367;
```

```
MySQL
call procUnesiOdeljenje( 70, 'Vozni park', 'Banovo Brdo', 5367);
```

```
Oracle
execute procUnesiOdeljenje( 70, 'Vozni park', 'Banovo Brdo', 5367);
```

Iz naredbi pozivanja procedura možemo primetiti da u SUBP MS SQL server nakon ključne reči „execute“ i naziva funkcije navodimo vrednosti parametara bez upotrebe zagrada, dok kod SUBP MySQL i Oracle iza naziva procedure otvaramo zagradu i iza poslednjeg parametra je zatvaramo.

U cilju demonstriranja mogućnosti korišćenja procedura kreiraćemo tabele PLATE i PLATE_STAVKE u kojima ćemo pamtiti naknade koje zaposleni dobijaju u toku vremena (npr. prva polovina naknade za mesec januar 2012. godine, druga polovina naknade za mesec januar 2012. godine, itd.).

Primer 232. Kreirati tabelu PLATE sa sledećim poljima: *autoid*, *rbpl*, *godina*, *mesec* i *datum* i tabelu PLATE_STAVKE sa sledećim poljima: *autoid*, *idbr*, *iznos*. Postaviti da je polje *autoid* u tabeli PLATE primarni ključ, a kombinacija polja: *rbpl*, *godina* i *mesec* jedinstveni ključ. Za tabelu PLATE_STAVKE postaviti da polja *autoid* i *idbr* čine kompozitni primarni ključ. Isto tako postaviti odgovarajuće zavisne spoljašnje ključeve

MS SQL:

```
CREATE TABLE plate (
  autoid int NOT NULL IDENTITY(1,1),
  rbpl tinyint NOT NULL,
  godina smallint NOT NULL,
  mesec tinyint NOT NULL,
  datum SMALLDATETIME NOT NULL,
  PRIMARY KEY (autoid)
);

CREATE UNIQUE INDEX ux_plate ON plate (rbpl, godina, mesec);

CREATE TABLE plate_stavke (
  autoid int NOT NULL,
  idbr int NOT NULL,
  iznos money,
  PRIMARY KEY (autoid, idbr)
);

ALTER TABLE plate_stavke ADD CONSTRAINT fk_platest_radnik
  FOREIGN KEY (idbr) REFERENCES radnik(idbr);
ALTER TABLE plate_stavke ADD CONSTRAINT fk_platest_plate
  FOREIGN KEY (autoid) REFERENCES plate(autoid);
```

MySQL:

```
CREATE TABLE plate (
  autoid int NOT NULL AUTO_INCREMENT,
  rbpl tinyint NOT NULL,
  godina smallint NOT NULL,
  mesec tinyint NOT NULL,
  datum DATE NOT NULL,
  PRIMARY KEY (autoid)
)ENGINE=InnoDB  DEFAULT CHARSET=utf8;

CREATE UNIQUE INDEX ux_plate ON plate (rbpl, godina, mesec);

CREATE TABLE plate_stavke (
  autoid int NOT NULL,
  idbr int NOT NULL,
  iznos decimal(17, 2),
  PRIMARY KEY (autoid, idbr)
)ENGINE=InnoDB  DEFAULT CHARSET=utf8;
```

MySQL:

```
-- nastavak sa prethodne strane

ALTER TABLE plate_stavke
  ADD CONSTRAINT fk_platest_radnik
  FOREIGN KEY (idbr) REFERENCES radnik(idbr);
ALTER TABLE plate_stavke
  ADD CONSTRAINT fk_platest_plate
  FOREIGN KEY (autoid) REFERENCES plate(autoid);
```

Oracle:

```
CREATE TABLE plate (
  autoid NUMBER(6,0) PRIMARY KEY,
  rbpl NUMBER(2,0) NOT NULL,
  godina NUMBER(4, 0) NOT NULL,
  mesec NUMBER(2, 0) NOT NULL,
  datum DATE NOT NULL
);

CREATE UNIQUE INDEX ux_plate ON plate (rbpl, godina, mesec);

CREATE TABLE plate_stavke(
  autoid NUMBER(6,0) NOT NULL,
  idbr NUMBER(6, 0) NOT NULL,
  iznos NUMBER(17, 2)
);

ALTER TABLE plate_stavke
  ADD CONSTRAINT fk_platest_radnik
  FOREIGN KEY (idbr) REFERENCES radnik(idbr);
ALTER TABLE plate_stavke
  ADD CONSTRAINT fk_platest_plate
  FOREIGN KEY (autoid) REFERENCES plate(autoid);
```

U tabeli RADNIK postoje polja *plata* i *premija* u kojima su definisana maksimalna mesečna primanja po svakom zaposlenom a u novim tabelama će se voditi evidencija primanja naknada u vremenu. U prvoj tabeli će se pamtiti polja vezana za obračun naknade i voditi računa da ne može dva puta da se upiše isti podatak (primiti naknada za isti period, odnosno isti deo plate u toku meseca i godine - *rbpl*), a u drugoj će se pamtiti iznos naknade po svakom zaposlenom za svaki obračun. U drugoj tabeli treba da vodimo računa da zaposleni u toku meseca ne može primiti naknadu veću od zbira vrednosti plate i premije iz tabele RADNIK.

U tabeli PLATE postoji polje *autoid* čija vrednost treba za svaki uneseni slog da se poveća za jedan i na taj način osigura jedinstvenost primarnog ključa. Kao što se može videti iz prethodnih programskih kodova za SUBP MS SQL i MySQL to se rešava tako što se u naredbi CREATE TABLE pored naziva polja piše ključna reč IDENTITY(početna vrednost, korak) (za MS SQL), odnosno AUTO_INCREMENT (za MySQL).

Kreiranjem jedinstvenog indeksa (unique index) ux_plate nametnuli smo ograničenje da se u toku jednog meseca (polje *mesec*) i godine (polje *godina*) ne može upisati isti, već

različiti deo plate (polje *rbpl*). Tako npr. u toku januara 2012. godine možemo radniku obračunati plate sa rednim brojevima 1 i 2 (prvi i drugi deo plate za januar 2012), ali isto tako u februaru 2012. godine možemo ponoviti plate sa rednim brojevima 1 i 2 i dodati obračun sa rednim brojem 3 (prvi, drugi i treći deo plate za februar 2012). Iako se u opštem slučaju kompozitni ključ (ključ koji se sastoji iz više polja) može upotrebiti kao primarni ključ, mi smo se odlučili da zbog jednostavnosti kreiramo primarni ključ nad jednim poljem i pri tom izbegli da prilikom povezivanja (join) tabele PLATE sa tabelom PLATE_STAVKE izjednačavamo vrednosti više polja.

U SUBP Oracle se polje u kome se automatski vrednost povećava ubacivanjem novog sloga rešava preko sekvence (SEQUENCE) koja ima za zadatak da generiše brojeve po zadatom pravilu inkrementiranja (slično kao IDENTITY u MS SQL).

Sekvence

Sekvence (Sequence) ili brojači su objekti koji po pozivu generišu celobrojnu vrednost saglasno sa unapred zadatim pravilom. Generisane vrednosti mogu biti rastuće ili opadajuće i svaka sledeća vrednost se od prethodne razlikuje za unapred definisanu vrednost (increment).

Sintaksa generisanja sekvence u SUBP Oracle je:

```
CREATE SEQUENCE [ ime_šeme. ] ime_sekvence
  [ { INCREMENT BY | START WITH } integer
  | { MAXVALUE integer | NOMAXVALUE }
  | { MINVALUE integer | NOMINVALUE }
  | { CYCLE | NOCYCLE }
  | { CACHE integer | NOCACHE }
  | { ORDER | NOORDER }
  ]...
;
```

gde je:

[ime_šeme.] ime_sekvence	Definiše opciono ime_šeme i obavezan naziv sekvence. Ako napišemo samo CREATE SEQUENCE naziv_šeme.naziv_sekvence bez ijedne dodatne ključne reči, onda će se kreirati rastuća sekvenca koja startuje brojem 1 i u svakom sledećem pozivu se povećava za 1 bez gornje granice
INCREMENT BY | START WITH INCREMENT BY	INCREMENT BY definiše interval između dva uzastopna broja koja generiše brojač - ova vrednost može biti bilo koja pozitivna ili negativna celobrojna vrednost, ali ne može biti jednaka nuli. START WITH definiše prvu vrednost koja će se generisati u brojaču.

{ MAXVALUE integer | NOMAXVALUE } { MINVALUE integer | NOMINVALUE }	MAXVALUE i MINVALUE predstavljaju maksimalnu odnosno minimalnu vrednost koju brojač može generisati. MAXVALUE se obično upotrebljava kada se generiše rastući niz brojeva, a MINVALUE kada se generiše opadajući. Ključne reči NOMAXVALUE i NOMINVALUE pokazuju da ne treba postavljati gornju i donju vrednost za generisane brojeve
CYCLE | NOCYCLE	Definišu da kada vrednost brojača dostigne maksimalnu ili minimalnu vrednost nastavlja sa generisanjem brojeva počevši od minimalne odnosno maksimalne vrednosti respektivno. Ključna reč NOCYCLE pokazuje da se generisanje brojeva zaustavlja kada se dostigne maksimalna odnosno minimalna vrednost.
CACHE integer | NOCACHE	Ključna reč CACHE definiše koliko vrednosti u bazi podataka treba unapred generisati i držati u memoriji radi bržeg dodeljivanja istih. Ključna reč NOCACHE pokazuje da ne treba unapred generisati brojeve
ORDER | NOORDER	Ključna reč ORDER se koristi kako bi se garantovalo kreiranje brojeva po redosledu dobijanja zahteva za generisanjem brojeva. Ovo je važno ako se brojevi generisani u sekvenci koriste za vremensko označavanje trenutaka određenih događaja, jer je tada bitno utvrditi koji se događaj desio pre ili posle kog događaja. Ključna reč NOORDER ne garantuje kreiranje brojeva po redosledu dobijanja zahteva za generisanjem brojeva

U cilju generisanja rastućih celobrojnih vrednosti za polje *autoid* u tablici PLATE generisaćemo prvo sekvencu seq_plate.

Primer 233. Kreirati sekvencu seq_plate kojom se generišu brojevi počevši od broja 1 sa inkrementom (korakom) 1

Oracle

```
CREATE SEQUENCE seq_plate
    START WITH 1
    INCREMENT BY 1;
```

Kreiraćemo proceduru „procIzracunajPlatu“ koja će imati za zadatak da na osnovu ulaznih parametara popuni tablicu PLATE i PLATE_STAVKE.

Primer 234. Kreirati proceduru procIzracunajPlatu koja će na osnovu vrednosti ulaznih parametara: parRbpl (redni broj plate), parGodina i parMesec popuniti odgovarajuća polja u tabeli PLATE, a na osnovu vrednosti parametara parPostotakPlate i parPostotakPremije za svakog zaposlenog (polje *idbr* iz tabele RADNIK) izračunati naknadu kao zbir dela plate (parPostotakPlate pomnožen sa vrednošću plate zaposlenog iz tabele RADNIK) i dela premije (parPostotakPremije pomnožen sa vrednošću premije zaposlenog iz tabele RADNIK) i uneti u tabelu PLATE_STAVKE

MS SQL:

```
CREATE PROCEDURE procIzracunajPlatu(
  @parRbpl tinyint,
  @parGodina smallint,
  @parMesec  tinyint,
  @parPostotakPlate decimal(6,2),
  @parPostotakPremije decimal(6,2))
AS
BEGIN
  DECLARE @danas SMALLDATETIME, @autoid int;
  SET @danas = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');

  INSERT INTO plate(rbpl, godina, mesec, datum)
  values(@parRbpl, @parGodina, @parMesec, @danas)

  SELECT @autoid = @@IDENTITY

  INSERT into plate_stavke(autoid, idbr, iznos)
  SELECT @autoid, idbr, @parPostotakPlate/100* plata +
    @parPostotakPremije/100*ISNULL(premija, 0.00)
  FROM radnik;
END;
```

MySQL:

```
DELIMITER $$

create PROCEDURE procIzracunajPlatu(
  parRbpl tinyint,
  parGodina smallint,
  parMesec  tinyint,
  parPostotakPlate decimal(6,2),
  parPostotakPremije decimal(6,2))
BEGIN
  DECLARE local_autoid int;
  INSERT INTO plate(rbpl, godina, mesec, datum)
  Values (parRbpl, parGodina, parMesec, current_date);

  SET local_autoid = LAST_INSERT_ID();
```

MySQL:

```
  -- Nastavak sa prethodne strane

   INSERT INTO plate_stavke(autoid, idbr, iznos)
   SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
parPostotakPremije/100*IFNULL(premija, 0.00)
   FROM radnik;

commit;
END$$

DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procIzracunajPlatu(
  parRbpl PLATE.RBPL%type,
  parGodina PLATE.GODINA%type,
  parMesec  PLATE.MESEC%type,
  parPostotakPlate NUMBER,
  parPostotakPremije NUMBER)
IS
  local_autoid NUMBER(6);
BEGIN
  SELECT seq_plate.nextval into local_autoid FROM dual;

  INSERT INTO plate(autoid, rbpl, godina, mesec, datum)
  VALUES (local_autoid, parRbpl, parGodina, parMesec, CURRENT_DATE);

  INSERT INTO plate_stavke (autoid, idbr, iznos)
  SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
    parPostotakPremije/100*nvl(premija, 0.00)
  FROM radnik;

  commit;
END procIzracunajPlatu;
```

Izvršenje procedure procIzracunajPlatu proverićemo kreiranjem obračuna prvog dela plate za januar 2012. godine.

Primer 235. Izvršiti proceduru procIzracunajPlatu kojom će se kreirati obračun prvog dela plate za januar 2012 koja se sastoji od pola mesečne plate i pola mesečne premije.

MS SQL

```
execute procIzracunajPlatu 1, 2012, 1, 50.0, 50.0;
```

MySQL

```
call procIzracunajPlatu(1, 2012, 1, 50.0, 50.0);
```

Oracle

```
execute procIzracunajPlatu(1, 2012, 1, 50.0, 50.0);
```

Kao rezultat izvršenja procedura nastaju sledeći slogovi u tabelama PLATE i PLATE_STAVKE kao što je pokazano na sledećoj slici.

*SELECT * FROM plate*

autoid	rbpl	godina	mesec	datum
1	1	2012	1	23.08.2012

*SELECT * FROM plate_stavke*

autoid	idbr	Iznos
1	3887	1000
1	5367	1600
1	5497	900
1	5519	1250
1	...	...

Ako pokušamo da izvršimo istu proceduru još jednom, MS SQL će javiti dve greške:

1) Cannot insert duplicate key row in object 'dbo.plate' with unique index 'ux_plate' i
2) Cannot insert the value NULL into column 'autoid', table 'student.dbo.plate_stavke'

preko kojih možemo zaključiti da prva INSERT naredba u proceduri nije uspela zbog narušavanja jedinstvenog ključa ux_plate pri čemu je vraćena vrednost NULL kao @@identity i kopirana u lokalnu promenljivu @autoid, pa je neuspeo i pokušaj da se drugom INSERT naredbom ubace u tabelu PLATE_STAVKE slogovi sa vrednošću polja autoid = NULL.

Kod SUBP MySQLse ovom prilikom javlja greška Duplicate entry '1-2012-1' for key 2 kojom nas SUBP obaveštava da je pokušano narušavanje jedinstvenog ključa u tabeli PLATE. Druga INSERT naredba u proceduri nije ni izvršena jer se izvršenje procedure zaustavilo nailaskom na prvu grešku.

U SUBP Oracle se ovom prilikom javlja greška: unique constraint (STUDENT.UX_PLATE) violated.

Obrada grešaka u procedurama

Sva tri SUBP-a omogućavaju obradu grešaka i postavljanje svojih opisa grešaka. Na sledećim primerima će biti pokazano kako za pokušaj narušavanja jedinstvenog ključa možemo korisniku vratiti informaciju (poruku) da je nastala greška pri ažuriranju podataka u prethodnim procedurama: “Greška: Dupli ključ!”, a za ostale greške možemo vratiti opštu poruku: “Desila se greška!”.

Primer 236. Modifikovati proceduru procIzracunajPlatu, tako što će za pokušaj narušavanja primarnog ključa javiti poruku „Greška, dupli ključ!“, a za ostale greške javiti poruku „Desila se greška!“, i snimiti je pod nazivom procIzracunajPlatu1.

MS SQL:

```
CREATE PROCEDURE procIzracunajPlatu1(
  @parRbpl tinyint,
  @parGodina smallint,
  @parMesec  tinyint,
  @parPostotakPlate decimal(6,2),
  @parPostotakPremije decimal(6,2))
AS
BEGIN
  DECLARE @danas SMALLDATETIME, @autoid int, @error int;
  SET @danas = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');

  INSERT INTO plate(rbpl, godina, mesec, datum)
    values(@parRbpl, @parGodina, @parMesec, @danas)
  SET @error = @@error
  IF @error = 0
  BEGIN
    SELECT @autoid = @@IDENTITY

    INSERT into plate_stavke(autoid, idbr, iznos)
    SELECT @autoid, idbr, @parPostotakPlate/100* plata +
    @parPostotakPremije/100*ISNULL(premija, 0.00)
    FROM radnik;
    SET @error = @@error;
  END
  IF @error > 0
  BEGIN
    IF @error = 2627 or @error = 2601
      RaisError('Greška: dupli ključ!', 16, 1);
    ELSE
      RaisError('Desila se greška!', 16, 1);
  END
END;
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procIzracunajPlatu1(
  parRbpl tinyint,
  parGodina smallint,
  parMesec  tinyint,
  parPostotakPlate decimal(6,2),
  parPostotakPremije decimal (6,2),
  out parPoruka varchar(255))
BEGIN
```

MySQL:

```
-- nastavak sa prethodne strane

DECLARE local_autoid int;
  DECLARE EXIT handler for 1062
    set parPoruka = 'Greška: dupli ključ!';
  DECLARE EXIT handler for sqlexception
    set parPoruka = 'Desila se greška!' ;

  SET parPoruka = 'OK';

  INSERT INTO plate(rbpl, godina, mesec, datum)
  values (parRbpl, parGodina, parMesec, current_date);

  IF length(parPoruka) = 2 then
    SET local_autoid = LAST_INSERT_ID();

    INSERT INTO plate_stavke(autoid, idbr, iznos)
    SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
      parPostotakPremije/100*IFNULL(premija, 0.00)
    FROM radnik;
  END IF;
  IF length(parPoruka) = 2 then
    Commit;
  END IF;
END $$

DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procIzracunajPlatu1(
  parRbpl PLATE.RBPL%type,
  parGodina PLATE.GODINA%type,
  parMesec  PLATE.MESEC%type,
  parPostotakPlate NUMBER,
  parPostotakPremije NUMBER)
IS
  local_autoid NUMBER(6);
BEGIN
  SELECT seq_plate.nextval into local_autoid FROM dual;

  INSERT INTO plate(autoid, rbpl, godina, mesec, datum)
  VALUES (local_autoid, parRbpl, parGodina, parMesec, CURRENT_DATE);

  INSERT INTO plate_stavke (autoid, idbr, iznos)
  SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
    parPostotakPremije/100*nvl(premija, 0.00)
  FROM radnik;

  commit;
```

```
Oracle:

-- nastavak sa prethodne strane

  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      raise_application_error(-20205, 'Greška: dupli ključ!');
    WHEN OTHERS THEN
      raise_application_error(-20205, 'Desila se greška!');

END procIzracunajPlatu1;
```

U programskim kodovima za sve SUBP **masnim slovima** je naznačena razlika u naredbama u odnosu na prethodni primer.

Kao što možemo da vidimo iz programskih redova u procedurama, u MS SQL serveru postoji globalna promenljiva @@error čija je vrednost jednaka nuli ako se nije dogodila greška i veća od nule ukoliko se desila greška. Poželjno je odmah iza naredbe trenutnu vrednost globalne promenljive kopirati u lokalnu i ispitati njenu vrednost. U zavisnosti od vrednosti te promenljive dozvoljavamo da procedura nastavi sa daljim izvršenjem ostalih naredbi ili javljamo našu grešku naredbom Raiserror.

Kod SUBP MySQL potrebno je definisati tzv. EXIT handler, odnosno skup akcija koje treba izvršiti ako prilikom izvršenja procedure dođe do greške. U ovom primeru mi smo definisali da se u slučaju greške dodeli određeni tekst greške promenljivoj parPoruka. Kada se desi greška, sistem na osnovu rednog broja greške dodeljuje određeni tekst promenljivoj parPoruka. Pomoću trenutne vrednosti promenljive parPoruka ispitujemo u telu procedure da li možemo da nastavimo dalje sa ostalim naredbama ili zaustavljamo izvršenje procedure (IF length(parPoruka) = 2).

U SUBP Oracle ne moramo iza svake naredbe da ispitujemo da li je došlo do greške, jer na kraju procedure možemo postaviti ključnu reč EXCEPTION. Ako se desi greška u bilo kojoj naredbi, program „skače“ na deo EXCEPTION i u zavisnosti od rednog broja greške (u ovom slučaju kodovana vrednost greške) sistem javlja opis greške kako mi to definišemo.

Rezultat pokazanih obrada grešaka možemo videti nakon što izvršimo procedure ponavljanjem istog obračuna naknade.

Primer 237. Izvršiti proceduru procIzracunajPlatu1 sa istim parametrima kao na Primer 235.

```
MS SQL

execute procIzracunajPlatu1 1, 2012, 1, 50.0, 50.0;
```

```
MySQL

call procIzracunajPlatu1(1, 2012, 1, 50, 50, @a);
SELECT @a;
```

```
Oracle

execute procIzracunajPlatu1(1, 2012, 1, 50.0, 50.0);
```

Prilikom izvršenja procedure iz primera 230 sva tri SUBP-a će prijaviti grešku: „Greška: dupli ključ!“. Kako unapred znamo da ne možemo u tabeli PLATE da unesemo isti obračun plate dva puta, mogli bi u proceduri da izvršimo proveru da li taj obračun plate već postoji pre nego što pokušamo da izvršimo ponovno ubacivanje iste plate, pa ako postoji da odmah objavimo grešku. Da bi to postigli, prethodni programski kod treba izmeniti, odnosno dopuniti odgovarajućim delom u kojem se vrši ta provera pre pokušaja upisa novog sloga.

Primer 238. Modifikovati proceduru procIzracunajPlatu1, tako što će se prvo izvršiti provera da li već postoji kombinacija vrednosti ulaznih parametara za polja *godina*, *mesec* i *rbpl* u tabeli PLATE i zatim, ako ne postoji, ubaciti odgovarajući slog u tabeli PLATE i odgovarajuće slogove u tabeli PLATE_STAVKE. U suprotnom – ako ta kombinacija postoji javiti grešku: „Navedeni obračun plate već postoji u bazi!“.

```
MS SQL:

CREATE PROCEDURE procIzracunajPlatu2(
  @parRbpl tinyint,
  @parGodina smallint,
  @parMesec  tinyint,
  @parPostotakPlate decimal(6,2),
  @parPostotakPremije decimal(6,2))
AS
BEGIN
  DECLARE @danas SMALLDATETIME, @autoid int, @error int, @brSlogova int;
  SET @danas = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');

  SELECT @brSlogova = count(*) FROM plate
  WHERE godina = @parGODINA and mesec = @parMESEC and rbpl = @parRBPL;

  IF @brSlogova = 0
  BEGIN
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values(@parRbpl, @parGodina, @parMesec, @danas)
    SET @error = @@error
    IF @error = 0
    BEGIN
      SELECT @autoid = @@IDENTITY

      INSERT into plate_stavke(autoid, idbr, iznos)
      SELECT @autoid, idbr, @parPostotakPlate/100* plata +
        @parPostotakPremije/100*ISNULL(premija, 0.00)
      FROM radnik;
      SET @error = @@error;
    END
    IF @error > 0
      RaisError('Desila se greška!', 16, 1);
  END
```

MS SQL:

```
  -- nastavak sa prethodne strane

  END
  ELSE
    RaisError('Navedeni obračun plate već postoji u bazi!', 16, 1);
END;
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procIzracunajPlatu2(
  parRbpl tinyint,
  parGodina smallint,
  parMesec  tinyint,
  parPostotakPlate decimal(6,2),
  parPostotakPremije decimal (6,2),
  out parPoruka varchar(255))
BEGIN
  DECLARE brSlogova int;
  DECLARE local_autoid int;
  DECLARE EXIT handler for 1062
    set parPoruka = 'Greška: dupli ključ!';
  DECLARE EXIT handler for sqlexception
    set parPoruka = 'Desila se greška!' ;

  SET parPoruka = 'OK';

  SELECT count(*) into brSlogova FROM plate
  WHERE godina = parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 Then
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values (parRbpl, parGodina, parMesec, current_date);

    IF length(parPoruka) = 2 then
      SET local_autoid = LAST_INSERT_ID();

      INSERT INTO plate_stavke(autoid, idbr, iznos)
      SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
        parPostotakPremije/100*IFNULL(premija, 0.00)
      FROM radnik;
    END IF;
    IF length(parPoruka) = 2 then
      Commit;
  END IF;
  ELSE
    SET parPoruka ='Navedeni obračun plate već postoji u bazi!';
  END IF;
END $$
```

Oracle:

```
create or replace
PROCEDURE procIzracunajPlatu2(
  parRbpl PLATE.RBPL%type,
  parGodina PLATE.GODINA%type,
  parMesec  PLATE.MESEC%type,
  parPostotakPlate NUMBER,
  parPostotakPremije NUMBER)
IS
  local_autoid NUMBER(6);
  brSlogova NUMBER(4);
  DUPLA_PLATA EXCEPTION;
BEGIN

  SELECT count(*) into brSlogova FROM PLATE
  WHERE godina= parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 THEN
    SELECT seq_plate.nextval into local_autoid FROM dual;

    INSERT INTO plate(autoid, rbpl, godina, mesec, datum)
    VALUES (local_autoid, parRbpl, parGodina, parMesec, CURRENT_DATE);

    INSERT INTO plate_stavke (autoid, idbr, iznos)
    SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
      parPostotakPremije/100*nvl(premija, 0.00)
    FROM radnik;

    commit;
  ELSE
    RAISE DUPLA_PLATA;
  END IF;

  EXCEPTION
    WHEN DUPLA_PLATA THEN
      raise_application_error(-20205, 'Navedeni obračun plate već postoji
u bazi!');
    WHEN DUP_VAL_ON_INDEX THEN
      raise_application_error(-20205, 'Greška: dupli ključ!');
    WHEN OTHERS THEN
      raise_application_error(-20205, 'Desila se greška!');
END procIzracunajPlatu2;
```

Rezultat novih obrada grešaka možemo videti nakon što izvršimo procedure ponavljanjem istog obračuna naknade.

Primer 239. Izvršiti proceduru procIzracunajPlatu2 sa istim parametrima kao u Primer 235

MS SQL

```
execute procIzracunajPlatu2 1, 2012, 1, 50.0, 50.0;
```

MySQL

```
call procIzracunajPlatu2(1, 2012, 1, 50, 50, @a);
SELECT @a;
```

Oracle

```
execute procIzracunajPlatu2(1, 2012, 1, 50.0, 50.0);
```

Poslovno pravilo da ne možemo da unesemo duple obračune plata i obradu grešaka koje javljamo u slučaju da probamo da prekršimo ovo pravilo pokazali smo prethodnim primerima. Probajmo da primenimo i drugo pravilo - da zabranimo unos naknade po radniku u periodu od mesec dana koji premašuje zbir mesečnih iznosa plate i premije za tog radnika definisanim u tabeli RADNIK.

To znači da ako radniku sa brojem 5367 – Petru Vasiću, koji u toku mesec dana može maksimalno da primi 1300 novčanih jedinica plate i 1900 novčanih jedinica premije (ukupno 3200 jedinica), u prvom delu ukupnih primanja obračunamo iznos od 1600 jedinica, on u drugom delu ne sme da primi više od preostalih 1600 jedinica.

Probaćemo da obračunamo drugi deo plate za mesec januar sa iznosom parametara parPostotakPlate i parPostotakPremije većim od 50% čime svakom radniku dodeljujemo ukupnu mesečnu naknadu koja je veća od 100% (jer smo u prethodnim procedurama već obračunali 50% od ukupne naknade).

Primer 240. Izvršiti proceduru procIzracunajPlatu2, tako što će se proslediti sledeći parametri: parRbpl=2, parGodina=2012, parMesec=1, parPostotakPlate=55 i parPostotakPremije=50

MS SQL

```
execute procIzracunajPlatu2 2, 2012, 1, 55.0, 50.0;
```

MySQL

```
call procIzracunajPlatu2(2, 2012, 1, 55, 50, @a);
SELECT @a;
```

Oracle

```
execute procIzracunajPlatu2(2, 2012, 1, 55.0, 50.0);
```

Kao rezultat izvršenja procedura nastaju sledeći slogovi u tabelama PLATE i PLATE_STAVKE

*SELECT * FROM plate*

autoid	Rbpl	godina	mesec	datum
1	1	2012	1	23.08.2012
2	2	2012	1	23.08.2012

SELECT p.mesec, p.godina, ps.idbr, r.plata+isnull(r.premija, 0.0) as naknadaRadnik, sum(ps.iznos) as naknada
FROM plate p, plate_stavke ps, radnik r
WHERE p.autoid = ps.autoid and ps.idbr = r.idbr
GROUP BY p.mesec, p.godina, ps.idbr, r.plata+isnull(r.premija, 0.0)

mesec	godina	idbr	Naknada Radnik	naknada
1	2012	3887	2000	2100
1	2012	5367	3200	3265
1	2012	5497	1800	1850
1	2012	…	…	…

Kao što vidimo, u tabeli PLATE_STAVKE možemo da unesemo veće iznose za naknadu radnika (kolona: naknada) od propisane (kolona: Naknada Radnik) kao maksimalne. U tom cilju, ograničenje da u toku meseca zaposleni ne može da primi sumu naknada (delova plate) veću od sume plate i premije definisane u tabeli RADNIK, možemo da primenimo kreiranjem okidača.

Taj okidač će pri ubacivanju ili ažuriranju slogova u tabelu PLATE_STAVKE javiti grešku u slučaju narušavanja ovog ograničenja.

Primer 241. Kreirati okidač nad tablicom PLATE_STAVKE koji će u slučaju da je suma naknada (kolona *iznos*) u toku jednog meseca za bilo kog zaposlenog veća od sume plate i premije u tablici RADNIK za istog zaposlenog, javiti grešku

MS SQL:

```
CREATE TRIGGER trgPlataProvera on plate_stavke
AFTER INSERT, UPDATE
AS
BEGIN
  DECLARE @koefStaz decimal(4,2)= 1.0;
  IF EXISTS(
   SELECT ps.idbr, p.mesec, p.godina,
     (r.plata*@koefStaz + isnull(r.premija, 0.00))
   FROM plate p, plate_stavke ps, inserted i, radnik r, plate p1
   WHERE p.autoid = ps.autoid and i.autoid = p1.autoid AND
     p.godina = p1.godina AND p.mesec = p1.mesec and
     ps.idbr = i.idbr and r.idbr = i.idbr
   GROUP BY ps.idbr, p.mesec, p.godina,
     (r.plata*@koefStaz + isnull(r.premija, 0.00))
   HAVING sum(ps.iznos) - (r.plata*@koefStaz + isnull(r.premija, 0.00))
     > 0)
  RaisError('Ukupni iznos plate i premije po radniku je veći od
            dozvoljenog mesečnog limita', 16, 1);
END;
```

MySQL:

```
DELIMITER $$

CREATE TRIGGER trgPlataProveraIns [trgPlataProveraUpd]
AFTER INSERT [UPDATE] ON plate_stavke
    FOR EACH ROW
BEGIN
  DECLARE lMesec tinyint; DECLARE lGodina smallint;
  DECLARE lkoefStaz decimal(4,2);
  DECLARE lMesecniIznos decimal(17,2); DECLARE lIznos decimal (17, 2);
  SET lkoefStaz = 1.0;
  SELECT mesec, godina into lMesec, lGodina
  FROM plate
  WHERE autoid = new.autoid;

  SELECT sum(ps.iznos) into lIznos
  FROM plate_stavke ps, plate p
  WHERE ps.autoid = p.autoid AND p.godina = lGodina
    AND p.mesec = lMesec AND ps.idbr = new.idbr;

  SELECT plata*lkoefStaz + IFNULL(premija, 0.00) into lMesecniIznos
  FROM radnik
  WHERE idbr = new.idbr;

  if lIznos > lMesecniIznos then
    SELECT ime into lIznos FROM radnik WHERE idbr=new.idbr;
  end if;
END $$
DELIMITER ;
```

Oracle:

```
CREATE GLOBAL TEMPORARY TABLE temp_plate_stavke
ON COMMIT DELETE ROWS
AS
SELECT ps.*, p.mesec, p.godina, ps.iznos as mesecniIznos
FROM plate_stavke ps, plate p
WHERE p.autoid = ps.autoid and 1=0;
--------------------------

CREATE OR REPLACE
TRIGGER trgPlataProvera
AFTER INSERT OR UPDATE ON plate_stavke
    FOR EACH ROW
DECLARE
  lMesec plate.mesec%TYPE;
  lGodina plate.godina%TYPE;
  lMesecniIznos radnik.plata%TYPE; lkoefStaz NUMBER(4,2) := 1.0;
```

Oracle:

```
-- nastavak sa prethodne strane

BEGIN
  SELECT mesec, godina into lMesec, lGodina
  FROM plate
  WHERE autoid = :new.autoid;

  SELECT plata*lkoefStaz + nvl(premija, 0.00) into lMesecniIznos
  FROM radnik where idbr = :new.idbr;

  insert into temp_plate_stavke(autoid, idbr, iznos, mesec, godina,
     mesecniiznos)
  values(:new.autoid, :new.idbr, :new.iznos, lMesec, lGodina,
     lMesecniIznos);
END trgPlataProvera;
----------------------------------------

CREATE OR REPLACE
TRIGGER trgPlataProveraStmt
AFTER INSERT OR UPDATE ON plate_stavke
DECLARE
  lBrSlogova NUMBER;
BEGIN
  INSERT into temp_plate_stavke(autoid, idbr, mesec, godina,
        mesecniiznos, iznos)
  SELECT -1, i.idbr, i.mesec, i.godina, i.mesecniiznos, sum(ps.iznos)
  FROM temp_plate_stavke i, plate_stavke ps, plate p
  WHERE p.autoid = ps.autoid and
      p.godina = i.godina AND p.mesec = i.mesec and
      ps.idbr = i.idbr
  GROUP BY -1, i.idbr, i.mesec, i.godina, i.mesecniiznos
  HAVING sum(ps.iznos) - i.mesecniiznos > 0;

  SELECT count(*) into lBrSlogova
  FROM temp_plate_stavke
  WHERE autoid = -1;

  DELETE FROM temp_plate_stavke;

  IF lBrSlogova > 0 THEN
    raise_application_error(-20205, 'Ukupni iznos plate i premije
           po radniku je veći od  dozvoljenog mesečnog limita ');
  end if;
END trgPlataProveraStmt;
```

Kao što može da se vidi iz Primer 241, kod svakog SUBP kreiranje okidača ima drugačiju koncepciju.

U SUBP MS SQL Server je dozvoljno kreiranje jednog okidača za više SQL komandi (AFTER INSERT, UPDATE).U telu okidača može da se navede naredba EXISTS (SELECT

...) koja vraća vrednost TRUE ako SELECT naredba u zagradi vraća bar jedan slog, odnosno FALSE ako ista naredba ne vraća ni jedan slog. SELECT naredba na osnovu tabele INSERTED kreira slogove kod kojih je suma iznosa po zaposlenom u toku jednog meseca u godini veća od zbira plate i premije iz tabele RADNIK. Ako postoji bar jedan takav slog javlja se greška (naredbom RaisError).

U SUBP MySQL nije dozvoljeno kreiranje jednog okidača za više SQL komandi, pa je tako potrebno kreirati jedan okidač za naredbu INSERT i jedan okidač za naredbu UPDATE. Za naredbu INSERT nazvaćemo okidač trgPlataProveraIns a za naredbu UPDATE trgPlataProveraUpd. Kako oba okidača imaju iste naredbe u telu okidača, da ne bi ponavljali identične blokove naredbi, uglastim zagradama su označene alternative. FOR EACH ROW okidač u MySQL omogućava pristup svakom polju iz reda **new** pa tako možemo u lokalne promenljive lMesec i lGodina zapamtiti vrednost meseca i godine iz odgovarajućeg sloga u tabeli PLATE, u lokalnu promenljivu lIznos zapamtiti sumu iznosa koju je zaposleni (new.idbr) primio u toku meseca - lMesec i godine – lGodina i u lokalnu promenljivu lMesecniIznos zapamtiti zbir plate i naknade iz tabele RADNIK. Ako je vrednost ukupno primljene naknade (lIznos) veća od propisane mesečne vrednosti (lMesecniIznos) tada okidač mora javiti grešku čime se automatski SQL komande INSERT ili UPDATE koje su pokrenule okidač poništavaju (rollback). U SUBP MySQL do verzije 5.5 nije postojala mogućnost definisanja greške od strane korisnika (SIGNAL SQLSTATE), pa je u telu okidača inicirana proizvoljna greška (npr. kopiranje vrednosti stringa u promenljivu tipa decimal) koja se kasnije može detektovati. U novijim verzijama treba upotrebiti naredbu SIGNAL SQLSTATE.

U SUBP Oracle je dozvoljno kreiranje jednog okidača za više SQL komandi (AFTER INSERT OR UPDATE). U FOR EACH ROW okidaču postoji ograničenje da tabela nad kojom se vrši okidanje ne može i da se čita (tzv. Mutating table) jer je tabela u procesu ažuriranja. To znači da sadržaj tabele PLATE_STAVKE nad kojom je postavljen okidač ne možemo da čitamo u telu FOR EACH ROW okidača i time saberemo sve iznose koje je zaposleni u toku meseca i godine primio. Međutim, ako se u definiciji okidača izostave ključne reči FOR EACH ROW, okidač se pokreće nakon što se potpuno završi izvršenje SQL komande i postaje „statement“ okidač, i u njemu je nakon završetka procesa INSERT, UPDATE ili DELETE moguće pročitati sadržaj tabele nad kojom je postavljen okidač. Ali u „statement“ okidaču nemamo pristup redovima **:new** pa tako ne znamo za koje slogove treba kontrolisati traženo ograničenje.

Intuitivno rešenje bi bilo da se sadržaji jednog po jednog reda **:new** kopiraju negde iz FOR EACH ROW okidača (u privremenu tabelu), a da se u „statement“ okidaču taj kopirani sadržaj iskoristi i uporedi sa tabelom PLATE_STAVKE.

To se može odraditi tako što ćemo prvo kreirati privremenu tabelu kako bi sačuvali listu slogova koji se menjaju naredbama INSERT ili UPDATE i koje vidimo u redovima **:new**, kako bi kasnije u „statement“ okidaču mogli da im pridjemo i povežemo ih sa tabelom PLATE_STAVKE. Privremenu tabelu kreiramo naredbom CREATE GLOBAL TEMPORARY TABLE. Ova tabela se ne kreira kao standardne tabele definisanjem naziva polja i njihovih tipova, već na osnovu SELECT upita. Uslov WHERE 1=0 smo postavili u upitu kako bi generisali praznu tabelu. Prilikom generisanja dva okidača (sa i bez ključnih reči FOR EACH ROW) vodili smo računa da se prvo izvršava FOR EACH ROW okidač, a zatim „statement“ (bez FOR EACH ROW) okidač.

U FOR EACH okidaču, na osnovu vrednosti polja iz reda **:new** možemo da pronađemo odgovarajući mesec i godinu iz tabele PLATE, kao i zbir mesečne plate i premije iz tabele RADNIK. Ove vrednosti kopiramo u lokalne promenljive i njihov sadržaj, kao i sadržaj polja iz reda **:new** ubacujemo u privremenu tabelu. U „statement“ okidaču na osnovu postojećih

slogova u privremenoj tabeli kreiramo nove slogove, podešavajući vrednost polja *autoid* na -1 (kako bi ta vrednost bila drugačija od ostalih vrednosti polja autoid koje su kopirane iz reda :new) i polja *iznos* na sumu naknada upisanih u tabeli PLATE_STAVKE (jer sada možemo da čitamo ovu tabelu u "statement" okidaču) za odgovarajućeg zaposlenog u godini i mesecu koji odgovaraju postojećim slogovima u privremenoj tabeli. Ostala polja prepisujemo iz postojećih slogova. Novi slogovi se u privremenu tabelu upisuju samo ako je ukupni iznos koji je zaposleni dobio u odgovarajućem mesecu i godini veći od dozvoljenog mesečnog primanja (HAVING sum(ps.iznos) - i.mesecniiznos > 0) što detektujemo preko naredbe „SELECT count(*) into lBrSlogova FROM temp_plate_stavke WHERE autoid = -1;". Ako postoji bar jedan takav slog (lBrSlogova > 0) onda okidač javlja grešku (naredbom raise_application_error).

Kako bi objasnili kako ovi okidači rade promenićemo iznos naknade u tabeli PLATE_STAVKE u obračunu sa rbpl=2 za mesec januar 2012. godine i ažuriraćemo svim zaposlenima iznos na 800 novčanih jedinica.

Primer 242. Ažurirati vrednost u koloni *iznos* u tabeli PLATE_STAVKE na 800 novčanih jedinica u svim slogovima koji su vezani za *rbpl* = 2, *mesec* = 1 i *godina* = 2012 iz tabele PLATE

```
 MS SQL, MySQL, Oracle:

UPDATE plate_stavke SET iznos = 800 where autoid in
(SELECT autoid FROM plate WHERE rbpl=2 and mesec=1 and godina=2012)
```

U okidaču trgPlataProvera u SUBP MS SQL, SELECT upit u delu IF EXISTS(SELECT...) vratiće sledeće slogove kao na sledećoj slici (u koloni desno postavljen je uslovu delu HAVING u SELECT naredbi kojim detektujemo slogove koji su premašili iznos mesečne naknade):

SELECT ps.idbr, p.mesec, p.godina…				*sum(ps.iznos) - (r.plata + isnull(r.premija, 0.00))*
idbr	**mesec**	**godina**	**No column name**	
5652	1	2012	1500	50
5696	1	2012	1000	300
5953	1	2012	1100	250

Iz pokazane tabele se vidi da postoje redovi koji zadovoljavaju HAVING uslov, pa okidač javlja poruku „Ukupni iznos plate i premije po radniku je veći od dozvoljenog mesečnog limita".

U okidaču trgPlataProveraUpd u SUBP MySQL za sve ažurirane slogove iz prethodne tabele imamo vrednosti lMesec=1 i lGodina = 2012. Kada je u okidaču prisutan red **new** sa sledećim podacima: new.autoid=2, new.iznos=800 i new.idbr=5652 ili new.idbr=5696 ili new.idbr=5953 dobija se: lIznos=1550 i lMesecniIznos=1500 (za new.idbr=5652), lIznos=1300 i lMesecniIznos=1000 (za new.idbr=5696) i lIznos=1350 i lMesecniIznos = 1100 (za new.idbr=5953). Za sve ove slučajeve je lMesecniIznos manji od lIznosa pa okidač javlja grešku „incorrect decimal value..." .

U okidaču trgPlataProvera u SUBP Oracle, koji je FOR EACH ROW okidač, za svaki **:new** red na osnovu :new.autoid nalazi se mesec i godina iz tabele PLATE i kopira se u lokalne promenljive lMesec i lGodina, a na osnovu :new.idbr iz tabele RADNIK se nalazi ukupna mesečna naknada i kopira u lokalnu promenljivu lMesecniIznos. Na kraju se sva polja iz reda :new, kao i promenljive lMesec, lGodina i lMesecniIznos ubacuju u privremenu tabelu TEMP_PLATE_STAVKE.

U okidaču trgPlataProveraStmt koji je okidač tipa „statement“ se nalaze sledeći sadržaj tabele TEMP_PLATE_STAVKE nakon INSERT into temp_plate_stavke naredbe kao što je to pokazano na sledećoj slici.

*SELECT * from temp_plate_stavke*

autoid	idbr	iznos	mesec	godina	mesecnilznos
2	3887	800	1	2012	2000
2	5367	800	1	2012	3200
2	5497	800	1	2012	1800
2	5519	800	1	2012	2500
2	5652	800	1	2012	1500
2	...	800	1	2012	...
2	...	800	1	2012	...
-1	**5652**	**1550**	**1**	**2012**	**1500**
-1	**5696**	**1300**	**1**	**2012**	**1000**
-1	**5953**	**1350**	**1**	**2012**	**1100**

Kao što se iz tabele može videti (zamašćeni slogovi) postoje tri reda čiji je autoid = -1 (lBrSlogova > 0) pa zbog toga okidač javlja grešku „Ukupni iznos plate i premije po radniku je veći od dozvoljenog mesečnog limita“.

Transakcije

Transakcija je logička jedinica posla koja treba da se izvrši. Ona sadrži jednu ili više SQL naredni. Transakcija se izvršava potpuno, ili se neće izvršiti ni jedan njen deo (atomičnost transakcije). Ako transakcija ima više SQL naredbi, njihovo dejstvo se prihvata u celosti ili se sve poništavaju [29], [32]. Ako treba preneti novac s jednog računa na drugi, onda se jednom naredbom novac skida s nekog računa a drugom naredbom se stavlja na ciljni račun. Ako prva naredba uspe, a druga naredba ne uspe, mora se poništiti i dejstvo prve naredbe. Ako se transakcija prekine, sistem se mora vratiti u pre početka transakcije (konzistentnost). Transakcija mora biti zasebna celina, čije izvršenje ne zavisi od drugih transakcija. Ova osobina se zove izolovanost ili serijalnost. Kada se transakcija uspešno završi, ne postoji ni jedan razlog da se ona poništi (trajnost). U nekim SUBP transakcije ne postoje, odnosno dejstvo naredbi za ažuriranje je automatsko. Kod većine SUBP može se definisati da se transakcije automatski započnu kada se izvršavaju određene operacije (**IMPLICITE TRANSACTION**). Ipak je bolje da se transakcije eksplicitno deklarišu. Neki postupci se ne mogu izvesti u okviru transakcija. To se, pre svega, odnosi na naredbe za definisanje baze (**ALTER** i **CREATE DATABASE**) i nekih objekata (**CREATE INDEX**, **TABLE** i isl.).

Sve promene nad bazom podataka izazvane SQL naredbama najčešće se odražavaju samo na stanje podataka u operativnoj memoriji korisnikovog računara ili servera. Ako želimo da se izmene odraze na stvarne podatke na disku (server), potrebno je potvrditi ove promene, transakcije, naredbom **COMMIT WORK** ili odustati od njih naredbom **ROLLBACK**

WORK. Naime, transakcija baze podataka je logička jedinica posla koja se izvršava do kraja ili se poništava u celini. Neke transakcije mogu trajati vrlo dugo (izmena velikog broja podataka).

Ako u proceduri postoji više od jedne SQL komandi kao što su INSERT, UPDATE, DELETE kojima se menja sadržaj u tabelama baze podataka, najčešće je poželjno da se one izvrše po principu „sve ili ništa“. U našim procedurama procIzracunajPlatu, procIzracunajPlatu1 i procIzracunajPlatu2 može da se desi da se komanda INSERT INTO PLATE izvrši bez grešaka ali u sledećoj komandi INSERT INTO PLATE_STAVKE može da se desi greška. U tom slučaju bi imali unet slog sa autoid, rbpl, mesecom i godinom u tabeli PLATE i ni jedan slog u tabeli PLATE_STAVKE. Logično je da prilikom pojave greške u bilo kojoj komandi baza mora da vrati sadržaj tabela u poslednje konzistentno stanje naredbom rollback, a ukoliko sve prođe u redu da izvrši commit i upiše novi sadržaj tabela na magnetni, trajni medijum.

U SUBP MS SQL se početak transakcije označava naredbom BEGIN TRANSACTION a završava naredbom COMMIT TRANSACTION u slučaju uspeha ili ROLLBACK TRANSACTION u slučaju greške.

U SUBP MySQL početak transakcije se označava naredbom START TRANSACTION, a završava naredbom COMMIT u slučaju uspeha ili ROLLBACK u slučaju greške.

U SUBP Oracle ne postoji naredba koja označava početak transakcije, podrazumeva se da je poslednja naredba commit u stvari početak narednih transakcija. Ovde postoje samo naredbe COMMIT i ROLLBACK.

Kreiraćemo proceduru procIzracunakPlatu3 na taj način što ćemo izvršenje dve INSERT naredbe postaviti unutar jedne transakcije.

Primer 243. Kreirati proceduru procIzracunajPlatu3 tako što će se blok od dve INSERT naredbe iz procedure procIzracunajPlatu2 postaviti unutar transakcije. Ukoliko se bilo koja od dve INSERT naredbe izvrši sa greškom, javiti grešku i odraditi ROLLBACK.

MS SQL:

```
CREATE PROCEDURE procIzracunajPlatu3(
  @parRbpl tinyint,
  @parGodina smallint,
  @parMesec  tinyint,
  @parPostotakPlate decimal(6,2),
  @parPostotakPremije decimal(6,2))
AS
BEGIN
  DECLARE @danas SMALLDATETIME, @autoid int, @error int, @brSlogova int;
  SET @danas = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');

  SELECT @brSlogova = count(*) FROM plate
  WHERE godina = @parGODINA and mesec = @parMESEC and rbpl = @parRBPL;
```

MS SQL:

```
-- nastavak sa prethodne strane

  IF @brSlogova = 0
  BEGIN
    BEGIN TRANSACTION
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values(@parRbpl, @parGodina, @parMesec, @danas)
    SET @error = @@error
    IF @error = 0
    BEGIN
      SELECT @autoid = @@IDENTITY

      INSERT into plate_stavke(autoid, idbr, iznos)
      SELECT @autoid, idbr, @parPostotakPlate/100* plata +
        @parPostotakPremije/100*ISNULL(premija, 0.00)
      FROM radnik;
      SET @error = @@error;
    END
    IF @error > 0
    BEGIN
      ROLLBACK TRANSACTION
      RaisError('Desila se greška!', 16, 1);
    END
    ELSE
      COMMIT TRANSACTION
  END
  ELSE
    RaisError('Navedeni obračun plate već postoji u bazi!', 16, 1);
END;
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procIzracunajPlatu3(
  parRbpl tinyint,
  parGodina smallint,
  parMesec  tinyint,
  parPostotakPlate decimal(6,2),
  parPostotakPremije decimal (6,2),
  out parPoruka varchar(255))
BEGIN
  DECLARE brSlogova int;
  DECLARE local_autoid int;
```

MySQL:

```
-- nastavak sa prethodne strane

  DECLARE EXIT handler for 1062
  BEGIN
    ROLLBACK;
    set parPoruka = 'Greška: dupli ključ!';
  END;
  DECLARE EXIT handler for sqlexception
  BEGIN
    ROLLBACK;
    set parPoruka = 'Desila se greška!' ;
  END;

  SET parPoruka = 'OK';

  SELECT count(*) into brSlogova FROM plate
  WHERE godina = parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 Then
    START TRANSACTION;
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values (parRbpl, parGodina, parMesec, current_date);

    IF length(parPoruka) = 2 then
      SET local_autoid = LAST_INSERT_ID();

      INSERT INTO plate_stavke(autoid, idbr, iznos)
      SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
        parPostotakPremije/100*IFNULL(premija, 0.00)
      FROM radnik;
    END IF;
    IF length(parPoruka) = 2 then
      Commit;
    END IF;
  ELSE
    SET parPoruka = 'Navedeni obračun plate već postoji u bazi!';
  END IF;
END $$

DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procIzracunajPlatu3(
  parRbpl PLATE.RBPL%type,
  parGodina PLATE.GODINA%type,
  parMesec  PLATE.MESEC%type,
  parPostotakPlate NUMBER,
  parPostotakPremije NUMBER)
IS
  local_autoid NUMBER(6);
  brSlogova NUMBER(4);
  DUPLA_PLATA EXCEPTION;

BEGIN
  SELECT count(*) into brSlogova FROM PLATE
  WHERE godina= parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 THEN
    SELECT seq_plate.nextval into local_autoid FROM dual;

    INSERT INTO plate(autoid, rbpl, godina, mesec, datum)
    VALUES (local_autoid, parRbpl, parGodina, parMesec, CURRENT_DATE);

    INSERT INTO plate_stavke (autoid, idbr, iznos)
    SELECT local_autoid, idbr, parPostotakPlate/100 * plata +
      parPostotakPremije/100*nvl(premija, 0.00)
    FROM radnik;

    commit;
  ELSE
    RAISE DUPLA_PLATA;
  END IF;

  EXCEPTION
    WHEN DUPLA_PLATA THEN
      raise_application_error(-20205, 'Navedeni obračun plate
                              već postoji u bazi!');
    WHEN DUP_VAL_ON_INDEX THEN
      rollback;
      raise_application_error(-20205, 'Greška: dupli ključ!');
    WHEN OTHERS THEN
      rollback;
      raise_application_error(-20205, 'Desila se greška!');

END procIzracunajPlatu3;
```

U primeru izvršavanja procedure procIzracunajPlatu2 (primer Primer 240) uneli smo drugi deo naknade za mesec januar. Da bi smo pokazali kako radi izvršenje transakcije po principu „sve ili ništa“ izbrisaćemo iz tabela PLATE i PLATE_STAVKE uneti drugi deo naknade za januar i probaćemo ponovo da ga unesemo novom procedurom.

Primer 244. Iz tabela PLATE i PLATE_STAVKE izbrisati sve slogove koji su povezani sa drugim delom plate za januar 2012. godine.

MS SQL, MySQL, Oracle (*za MS SQL i MySQL ignorisati commit*):

```
DELETE FROM plate_stavke WHERE autoid in
(SELECT autoid FROM plate WHERE rbpl=2 and mesec=1 and godina = 2012);
DELETE FROM plate WHERE rbpl=2 and mesec=1 and godina = 2012;
COMMIT;
```

Unećemo drugi deo plate za januar sa 20% mesečne naknade. Ovaj unos dela naknade ne premašuje 100% mesečnog iznosa jer je u unosu prvog dela naknade za januar uneto 50% mesečnog iznosa.

Primer 245. Izvršiti proceduru procIzracunajPlatu3 sa sledećim parametrima: parRbpl=2, parGodina=2012, parMesec=1, parPostotakPlate=20 i parPostotakPremije=20

MS SQL

```
execute procIzracunajPlatu3 2, 2012, 1, 20.0, 20.0;
```

MySQL

```
call procIzracunajPlatu3(2, 2012, 1, 20, 20, @a);
SELECT @a;
```

Oracle

```
execute procIzracunajPlatu3(2, 2012, 1, 20.0, 20.0);
```

Izvršenje procedure iz prethodnog primera će proći bez problema i nijedan SUBP neće javiti grešku. Pokušaćemo da unesemo i treći deo naknade za mesec januar u iznosu od 40% mesečnog iznosa.

Primer 246. Izvršiti proceduru procIzracunajPlatu3 sa sledećim parametrima: parRbpl=3, parGodina=2012, parMesec=1, parPostotakPlate=40 i parPostotakPremije=40

MS SQL

```
execute procIzracunajPlatu3 3, 2012, 1, 40.0, 40.0;
```

MySQL

```
call procIzracunajPlatu3(3, 2012, 1, 40, 40, @a);
SELECT @a;
```

```
Oracle
execute procIzracunajPlatu3(3, 2012, 1, 40.0, 40.0);
```

U proceduri procIzracunajPlatu3 izvršiće se komanda „INSERT into plate...“ bez grešaka i upisaće se odgovarajući slog u tabeli PLATE. Međutim, prilikom upisa slogova u tabelu PLATE_STAVKE komandom „INSERT into plate_stavke...“ premašiće se dozvoljeno mesečno primanje za mesec januar (50% u prvom delu + 20% u drugom + 40% u trećem) tako da će okidač nad tabelom PLATE_STAVKE javiti grešku. Ta greška se detektuje u proceduri i ona pokreće naredbu rollback čime se briše i prethodno upisani slog iz tabele PLATE.

Nakon izvršenja procedure procIzracunajPlatu3 proverićemo da li u tabeli PLATE imamo tri ili dva obračuna naknada.

Primer 247. Izlistati sadržaj tabele PLATE

```
MS SQL, MySQL, Oracle:
SELECT * FROM PLATE;
```

Korišćenje procedura za izvršavanje upita

Procedure se uopšte mogu primeniti ne samo u svrhu menjanja podataka u tabelama naredbama INSERT, UPDATE i DELETE već i za izvršenje upita, a naročito ako je u pitanju pretraga po određenim parametrima. U tu svrhu kreiraćemo proceduru koja vraća rezultate upita koji pretražuje tabelu RADNIK po bilo kom ili kombinacijom sledećih parametara: ime, prezime i posao.

Primer 248. Kreirati proceduru procListajRadnike koja će imati tri ulazna parametra: pIme, pPrezime i pPosao i koja će izlistati sve zaposlene iz tablice RADNIK čije ime počinje stringom iz parametra pIme, prezime počinje stringom iz parametra pPrezime i posao počinje stringom iz parametra pPosao. Uzeti u obzir da parametri mogu biti prazne vrednosti (NULL)

```
MS SQL
CREATE PROCEDURE procListajRadnike(
      @pIme NVARCHAR(20)=NULL,
      @pPrezime NVARCHAR(40)=NULL,
      @pPosao NVARCHAR(20)=NULL)
AS
BEGIN
SELECT *
FROM radnik
WHERE (@pIme is NULL OR ime like @pIme+N'%') AND
      (@pPrezime is NULL OR prezime like @pPrezime+N'%') AND
      (@pPosao is NULL OR posao like @pPosao+N'%');

END
```

MySQL

```
DELIMITER $$
CREATE PROCEDURE procListajRadnike(
      pIme VARCHAR(20),
      pPrezime VARCHAR(40),
      pPosao VARCHAR(20))
BEGIN
SELECT *
FROM radnik
WHERE (pIme is NULL OR ime likeconcat(pIme,'%')) AND
      (pPrezime is NULL OR prezime likeconcat(pPrezime,'%')) AND
      (pPosao is NULL OR posao likeconcat(pPosao,'%'));
END$$

DELIMITER ;
```

Oracle

```
CREATE OR REPLACE PACKAGE nas_paket
AS
TYPE t_cursor is REF CURSOR;
PROCEDURE  procListajRadnike(pIme  IN  VARCHAR2,  pPrezime  IN
VARCHAR2,
   pPosao IN VARCHAR2, pRezultat OUT T_CURSOR);
END nas_paket;
--------------------

create or replace
PACKAGE BODY nas_paket
AS
PROCEDURE procListajRadnike(
  pIme IN VARCHAR2,
  pPrezime IN VARCHAR2,
  pPosao IN VARCHAR2,
  pRezultat OUT T_CURSOR)
AS
BEGIN
  OPEN pRezultat FOR
  SELECT *
  FROM RADNIK
  WHERE (pIme is NULL OR ime like pIme || '%') AND
      (pPrezime is NULL OR prezime = pPrezime || '%') AND
      (pPosao is NULL OR posao = pPosao || '%');
END procListajRadnike;

END nas_paket;
```

SUBP MS SQL i MySQL mogu direktno da koriste proceduru u svrhu vraćanja rezultata izvršenja upita, dok je kod SUBP Oracle to moguće uz upotrebu paketa (package) koji sadrži objekat tipa T_CURSOR. Kao što može da se vidi iz programskog koda, u deklaraciji paketa se navodi lista objekata i procedura, a u telu paketa (package body) se navodi kako se izvršava procedura (ili u opštem slučaju skup procedura).

Primer 249. Izvršiti proceduru procListajRadnike za vrednost ulaznog parametra pPrezime = 'Pe'

MS SQL

```
execute procListajRadnike @pPrezime=N'Pe';
```

MySQL

```
call procListajRadnike (NULL, 'Pe', NULL);
```

Oracle

```
variable skup_podataka REFCURSOR;
execute nas_paket.procListajRadnike(NULL, 'Pe', NULL, :skup_podataka);

PRINT :skup_podataka;
```

Rezultat izvršenja ovih procedura je pokazan na sledećoj slici:

idbr	ime	prezime	posao	kvalif	rukovodilac	datzap	premija	plata	brod
5652	Jovan	**Pe**rić	električar	KV	5662	1980-05-31	500	1000	10
5900	Slobodan	**Pe**trović	rukovodilac vozila	KV	5780	2002-10-03	1300	900	20
5953	Jovan	**Pe**rić	nabavljač	KV	5786	1979-01-12	0	1100	30

Funkcije definisane od strane korisnika

Sva tri SUBP-a omogućavaju kreiranje korisničkih funkcija kao što je to moguće u bilo kom programskom jeziku. Funkcija je potprogram koji na osnovu liste ulaznih parametara izračunava izlazni parameter (procedura za razliku od funkcije ne vraća ni jednu vrednost). U opštem slučaju možemo kreirati funkciju koja na izlazu daje rezultat samo na osnovu ulaznih parametara, ali isto tako možemo da kreiramo i funkciju koja na osnovu ulaznih parametara i pristupom nad određenim tabelama rezultat dobija kombinacijom ulaznih parametara i sadržaja tabela.

Sintakse za kreiranje funkcija

Sintaksa kreiranja funkcije u **SUBP MS SQL** je:

```
CREATE FUNCTION [ ime_šeme ] ime_funkcije
( [ { @parametar [ AS ][ ime_šeme ]
tip_promenljive
   [ = podrazumevana_vrednost ] [ READONLY ]
}
   [ ,...n ]
 ]
)
RETURNS tip_promenljive_koja_se_vraća
   [ WITH <opcije_funkcija> [ ,...n ] ]
   [ AS ]
   BEGIN
      telo_funkcije
      RETURN skalarni_izraz
   END
[ ; ]
```

gde je:

[ime_šeme.] ime_procedure	Definiše opciono ime_šeme i obavezan naziv funkcije
([{ @parametar [AS][ime_šeme] tip_promenljive [= podrazumevana_vrednost] [READONLY] }	Definiše naziv parametra koji se prosleđuje funkciji kao i tip podatka tog parametra. Za svaki parametar se može definisati i njegova podrazumevana vrednost. Ključna reč READONLY pokazuje da se parametru ne može promeniti vrednost unutar funkcije
tip_promenljive_koja_se_vraća	Definiše kog je tipa vrednost promenljive koja se vraća na izlazu funkcije

<opcije_funkcija>::=
{
[ENCRYPTION]
| [SCHEMABINDING]
| [RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT]
| [EXECUTE_AS]
}
	ENCRYPTION definiše da se telo funkcije može šifrirati tako da ga administratori ne mogu videti,
	SCHEMABINDING pokazuje da je funkcija vezana za objekte baze podataka u šemi na koje se naredbe referenciraju. Kada se ova ključna reč upotrebi, objekti koji se nalaze u funkciji se ne mogu promeniti jer bi onda funkcija bila neispravna. U tom slučaju funkcija bi morala da se izbriše (naredbom DROP FUNCTION) i ponovo kreira.
	RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT RETURNS NULL ON NULL INPUT definiše način funkcionisanja funkcije, da ako bilo koji od parametara ima NULL vrednost, funkcija automatski vraća istu (NULL) vrednost bez izvršavanja komandi koje se nalaze unutar tela funkcije. Ako se ova opcija ne definiše, onda se podrazumeva opcija CALLED ON NULL INPUT gde se komande u telu funkcije izvršavaju iako je nekom od parametara prosleđena NULL vrednost.
	EXECUTE_AS definiše pod čijim se korisničkim imenom i privilegijama izvršava funkcija

telo_funkcije [...n]	Skup od jedne ili više SQL naredbi koje treba da se izvrše unutar procedure
skalarni_izraz	Vrednost koju vraća fukcija

U SUBP MS SQL funkcija može vratiti i tabelu ukoliko se umesto dela RETURNS tip_promenljive_koja_se_vraća napiše RETURNS TABLE i umesto RETURN skalarni_izraz napiše SELECT komanda. Sintaksa tada ima oblik:

```
CREATE FUNCTION [ ime_šeme ] ime_funkcije
( [ { @parametar [ AS ][ ime_šeme ]
tip_promenljive
  [ = podrazumevana_vrednost ] [ READONLY ]
}
  [ ,...n ]
 ]
)
RETURNS TABLE
  [ WITH <opcije_funkcija> [ ,...n ] ]
  [ AS ]
  RETURN [ ( ] select_komanda [ ) ]
[ ; ]
```

Sintaksa kreiranja funkcije u **SUBP MySQL** je:

```
CREATE
    [DEFINER = { db_korisnik | CURRENT_USER }]
    FUNCTION ime_funkcije ([ime_parametra
tip_parametra[,...]])
    RETURNS tip_parametra_koji_se_vraća
    [characteristic ...] telo_funkcije

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL
DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

gde je:

db_korisnik	Korisničko ime korisnika koji kreira funkciju, određuje korisnička prava i proverava privilegije korisnika u momentu kada se izvršava funkcija
ime_parametra tip_parametra	Definiše parametar koji se prosleđuje funkciji (npr. plata decimal(17,2))
tip_promenljive_koja_se_vraća	Definiše kog je tipa vrednost promenljive koja se vraća na izlazu funkcije
COMMENT 'string'	Predstavlja komentar koji ne utiče na izvršenje funkcije, ali autoru daje informaciju šta radi funkcija (npr. Funkcija služi da izračuna mesečnu platu zaposlenih)
LANGUAGE	Pokazuje koji programski jezici su dozvoljeni unutar tela funkcije - dozvoljena vrednost je SQL
DETERMINISTIC	pokazuje da je funkcija deterministička tj. da za skup istih ulaznih parametara uvek daje isti rezultat. Suprotno je NOT DETERMINISTIC što se može izostaviti u sintaksi.
CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA	Ključne reči koje definišu "ponašanje" naredbi unutar tela funkcije - CONTAINS SQL pokazuje da funkcija ne sadrži naredbe koje čitaju ili upisuju podatke (npr. SET @x = 1), NO SQL pokazuje da funkcija ne sadrži SQL naredbe, READS SQL DATA pokazuje da funkcija sadrži naredbe koje čitaju podatke (npr SELECT naredba), ali ne i naredbe koje upisuju podatke i MODIFIES SQL DATA pokazuje da funkcija sadrži naredbe koje mogu menjati neke podatke (npr. INSERT ili DELETE)

SQL SECURITY	Ova karakteristika može da sadrži ključne reči DEFINER (onaj koji je definisao funkciju) ili INVOKER (onaj koji je pokrenuo funkciju) kako bi se naznačilo da li će se funkcija izvršiti pod privilegijama korisničkog naloga iza reči DEFINER ili naloga korisnika koji ju je pokrenuo (INVOKER)
telo_funkcije	Predstavlja jednu ili više naredbi u telu funkcije koje se izvršavaju kada se pokrene funkcija

Sintaksa kreiranja funkcije u **SUBP Oracle** je:

```
CREATE [ OR REPLACE ] FUNCTION [ ime_šeme.
]ime_funkcije
 [ (parametar [ IN | OUT | IN OUT ]
   [ NOCOPY ] tip_podatka
    [, parametar [ IN | OUT | IN OUT ]
      [ NOCOPY ] tip_podatka
    ]...
  )
 ]
 RETURN tip_podatka_koji_se_vraća
 [ { AUTHID { CURRENT_USER | DEFINER }
  | DETERMINISTIC
  | parallel_enable_clause
  }
   [ AUTHID { CURRENT_USER | DEFINER }
   | DETERMINISTIC
   | parallel_enable_clause
   ]...
 ]
 { { AGGREGATE | PIPELINED }
  USING [ ime_šeme. ]tip_implementacije
 | [ PIPELINED ]
  { IS | AS }
  { telo_funkcije | specifikacije_poziva }
 } ;
```

gde je:

[ime_šeme .]ime_funkcije	Definiše opciono ime_šeme i obavezan naziv funkcije

(parametar [{ IN | OUT | IN OUT } [NOCOPY]] tip_podatka	Definiše parametar čija se vrednost: prosleđuje funkciji prilikom poziva (IN), čita kada se funkcija završi (OUT) ili prvo prosleđuje prilikom poziva a zatim čita kada se funkcija završi (IN OUT). Ključna reč NOCOPY je instrukcija bazi da prenese parametar funkciji što pre. Ako je parametar tipa IN onda se NOCOPY ne mora navoditi jer se podrazumeva. Tip podatka se navodi bez "veličine" podatka. Tako npr. VARCHAR2 je pravilan tip a VARCHAR2(50) nije.
[AUTHID { CURRENT_USER | DEFINER }]	Ovim ključnim rečima se navodi da li funkcija treba da se izvrši sa ovlašćenjima i privilegijama korisnika koji ju je pokrenuo (CURRENT_USER) ili korisnika koji ju je kreirao (DEFINER).
IS | AS	Nakon definisanja imena, parametara i ovlašćenja funkcije sledi ključna reč IS ili AS. Sasvim je svejedno koju ćemo ključnu reč upotrebiti
parallel_enable_clause	To je optimizaciona opcija koja pokazuje da funkcija može da bude izvršena od strane paralelnog servera na kome se paralelno izvršava upit.
AGGREGATE	Služi da pokaže da se funkcija koristi (izračunava) kao agregaciona - tj. iz grupe slogova vraća jedan slog. Na ovaj način se definiše korisnička agregaciona funkcija (slično ugrađenim agregacionim funkcijama kao što su: MIN, MAX, AVG, COUNT, SUM). U SELECT naredbi ova funkcija se može upotrebiti u listi kolona u SELECT delu, HAVING delu i ORDER BY delu.
PIPELINED	To je opcija koja navodi Oracle bazu da vrati rezultate funkcijske tabele iterativno. U tom slučaju se u upitu navodi ključna reč TABLE pre imena funkcije u FROM delu upita (npr. SELECT * FROM TABLE(fn_prikazi_platu(...))
ime_šeme.]tip_implementacije	To je tip objekta koji sadrži implementaciju ODCIAggregate rutina
telo_funkcije	Sastoji se od skupa naredbi koje se izvršavaju kada se pokrene funkcija

Primena funkcija

Tako, na primer, funkciji možemo da prosledimo parametre za izračunavanje mesečne naknade a na izlazu možemo dobiti izračunatu naknadu po zadatoj formuli.

Primer 250. Kreirati funkciju fnIzracunajPlatu koja na osnovu ulaznih parametara plata_in, premija_in, datzap_in, posto_plata_in i posto_premija_in vraća izračunatu vrednost naknade tako što proizvod vrednosti plata_in i posto_plata_in uvećan za 0.4% po godini staža zaposlenog sabere sa vrednošću proizvoda premija_in i posto_premija_in.

MS SQL

```
CREATE FUNCTION fnIzracunajPlatu (
  @plata_in money,
  @premija_in money,
  @datzap_in SMALLDATETIME,
  @posto_plata_in decimal(6,2),
  @posto_premija_in decimal(6,2))
RETURNS money
AS
BEGIN
  DECLARE @lGodinaStaza smallint;
  DECLARE @lNaknada money;

  SET @lGodinaStaza=ISNULL(DATEDIFF(year, @datzap_in, GETDATE()), 0);
  SET @lNaknada = @plata_in * @posto_plata_in/100 *
             (1 + @lGodinaStaza*0.004) + ISNULL(@premija_in, 0.0)
              * @posto_premija_in/100;
  RETURN @lNaknada;
END
```

MySQL

```
DELIMITER $$

Create Function fnIzracunajPlatu(
  plata_in decimal(17,2),
  premija_in decimal(17,2),
  datzap_in DATE,
  posto_plata_in decimal(6,2),
  posto_premija_in decimal(6,2))
RETURNS decimal(17,2)
  BEGIN
    DECLARE lGodinaStaza smallint;
    DECLARE lNaknada decimal(17,2);

    SET lGodinaStaza = IFNULL(FLOOR(DATEDIFF(CURRENT_DATE,
                       datzap_in)/365), 0.0);
    SET lNaknada = plata_in*posto_plata_in/100 *
              (1 + lGodinaStaza*0.004) +
              IFNULL(premija_in, 0.0) * posto_premija_in/100;
    RETURN lNaknada;
END$$
DELIMITER ;
```

Oracle

```
create or replace
Function fnIzracunajPlatu(
  plata_in IN NUMBER,
  premija_in IN NUMBER,
  datzap_in IN DATE,
  posto_plata_in IN NUMBER,
  posto_premija_in IN NUMBER)
RETURN NUMBER
IS
  lGodinaStaza NUMBER;
  lNaknada NUMBER;
BEGIN
  lGodinaStaza := nvl(FLOOR((CURRENT_DATE - datzap_in)/365), 0.0);
  lNaknada := plata_in*posto_plata_in/100*(1 + lGODINASTAZA*0.004) +
              nvl(premija_in, 0.0) * posto_premija_in/100;
  RETURN lNaknada;
END;
```

Primer 251. Izvršiti funkciju fnIzracunajPlatu za sledeće vrednosti parametara: plata_in=2000, premija_in=1000, datzap_in = 22. april 2002., posto_plata_in=25 i posto_premija_in=25

MS SQL

```
SELECT dbo.fnIzracunajPlatu (2000, 1000, '2002-04-22',
                             25.0, 25.0);
```

MySQL

```
SELECT fnIzracunajPlatu (2000, 1000, '2002-04-22', 25.0, 25.0);
```

Oracle

```
SELECT fnIzracunajPlatu(2000, 1000,
                    to_date('22042002','DDMMYYYY'), 25.0, 25.0)
FROM dual;
```

Isto tako možemo da kreiramo funkciju koja će na osnovu ulaznog parametra rednog broja radnika iz tabele RADNIK (idbr) vratiti maksimalno mesečno primanje koje taj radnik može da ima. Ova funkcija pristupa tabeli RADNIK i iz nje izvlači podatke. Namerno smo u ovoj funkciji vrednost iz polja *plata* u tabeli RADNIK uvećali za 14% (0.4% x 40 godina) što je maksimalno uvećanje za godine radnog staža i tu vrednost ćemo koristiti u narednim primerima. U skladu sa time treba promeniti inicijalnu vrednost promenljive *lkoefStaz* u okidaču trgPlataProvera na 1.14.

Primer 252. Kreirati funkciju fnIzracunajMaxMesecnuPlatu koja na osnovu ulaznog parametara idbr_in u tabeli RADNIK nalazi odgovarajuće vrednosti iz kolona *plata* i *premija* i sabira vrednost plate uvećanu za 14% sa vrednošću premije za zaposlenog sa datim idbr

MS SQL

```
CREATE FUNCTION fnIzracunajMaxMesecnuPlatu(@idbr_in int)
RETURNS money
AS
BEGIN
  DECLARE @MesecniIznos money;

  SELECT @MesecniIznos = plata * 1.14 + ISNULL(premija, 0.00)
  FROM radnik
  WHERE idbr = @idbr_in

  RETURN isnull(@MesecniIznos, 0.00);
END
```

MySQL

```
DELIMITER $$

CREATE FUNCTION fnIzracunajMaxMesecnuPlatu(idbr_in int)
RETURNS decimal(17,2)
BEGIN
  DECLARE lMesecniIznos decimal(17,2);
  SELECT plata * 1.14 + IFNULL(premija, 0.00) into lMesecniIznos
  FROM radnik
  WHERE idbr = idbr_in;

  RETURN ifnull(lMesecniIznos, 0.00);
END$$
DELIMITER ;
```

Oracle

```
create or replace
Function fnIzracunajMaxMesecnuPlatu(idbr_in IN NUMBER)
RETURN NUMBER
IS
  lMesecniIznos NUMBER;
BEGIN
  SELECT plata * 1.14 + nvl(premija, 0.00) into lMesecniIznos
  FROM radnik
  WHERE idbr = idbr_in;

  RETURN nvl(lMesecniIznos, 0.00);
END;
```

Primer 253. Izvršiti funkciju fnIzracunajMaxMesecnuPlatu za parametrom idbr_in=3887

MS SQL

```
SELECT dbo.fnIzracunajMaxMesecnuPlatu(3887);
```

MySQL

```
SELECT fnIzracunajMaxMesecnuPlatu(3887);
```

Oracle

```
SELECT fnIzracunajMaxMesecnuPlatu(3887) FROM Dual;
```

Vrednost koja se dobija primenom izvršenja funkcije pokazane na prethodnom primeru je 2280 novčanih jedinica.

Kursori (Cursor)

Tipične SQL naredbe rade sa skupovima slogova, redova. Kursori omogućavaju rad sa pojedinačnim redovima. Kursor je u suštini pokazivač, definisan nad jednim skupom slogova. Taj pokazivač redom pokazuje na svaki slog (red) i na taj način omogućava pojedinačnu obradu svakog sloga.

Kursori se mogu koristiti na klijentskoj strani, ali se to najčešće ne preporučuje: kada se obrađuje velika količina podataka (dolazi do zagušenja mreže), troše mnogo resursa, i mnogi SUBP maju problem sa zaključavanjem podataka. Mnogo značajniji i češći u upotrebi su pozadinski ili serverski kursori.

Serverski kursori omogućavaju da se SQL naredbe izvršavaju nad skupom podataka koji ima jedan red (granularnost). Time se dobija mogućnost da se nad podacima obavljaju različite operacije zavisno od vrednosti podataka. Dakle, uslovna logička operacija obavlja se nad jednim redom nezavisno od drugih redova u istom skupu.

Rad sa kursorima je efikasniji od običnog ažuriranja. Ažuriranje može biti takvo da se isti red više puta ažurira, na primer nad istim veoma velikim skupom podataka se obavlja nekoliko uskladištenih procedura. Mnogo je efikasnije ako se sve procedure odjednom izvrše nad jednim redom nego da se isti red više puta pribavlja i obrađuje. Kursori po pravilu koriste manje resursa nego operacije sa skupovima. Kontrola transakcija je bolja jer se može odrediti postupak za jedan red nezavisno od ostalih redova.

Da bi obrada dobijenog skupa podataka bila efikasnija, može se koristiti više kursora. To su ugnježđeni kursori.

Da bi se koristio, svaki kursor se najpre mora deklarisati (**DECLARE**). Skup podataka na koji se odnosi neki kursor dobija se **SELECT** upitom, koji se navodi pri deklaraciji kursora. Kada je deklarisan, kursor se može otvoriti (**OPEN**) i tada se izvršava upit koji je naveden u deklaraciji kursora. Rezultujući skup podataka ostaje povezan sa kursorom i oni se mogu čitati (**FETCH**). Nakon čitanja sloga, po pravilu kursor se automatski pomera susedni slog (ako on postoji). Po završetku rada kursor se zatvara (**CLOSE**), a memoriju treba osloboditi (**DEALLOCATE**).

Dohvatanje podataka, odnosno pristup podacima može biti sekvencijalan (od početka do kraja skupa ili obrnuto, **FIRST**, **NEXT**, **LAST**, **PREVIOUS** (**PRIOR**)) ali i direktan u odnosu na početak ili kraj skupa ili u odnosu na tekući red. Kursori pamte položaj tekućeg sloga u bazi i imaju pokazivače na sledeći i prethodni red.

Sintakse za kreiranje kursora

Sintaksa kreiranja kursora u **SUBP MS SQL** je:

```
DECLARE ime_kursora [ INSENSITIVE ] [ SCROLL
] CURSOR
   FOR select_naredba
   [ FOR { READ ONLY | UPDATE [ OF
naziv_kolone [ ,...n ] ] } ]
[;]
```

gde je:

[INSENSITIVE]	Ključna reč koja definiše da kursor čuva podatke u privremenoj kopiji. Svi zahtevi prema kursoru se čitaju iz privremene tabele u šemi tempdb; pa se tako ažuriranja podataka koja se u međuvremenu (u toku izvršenja operacija nad kursorom) dešavaju nad tabelama od kojih se upit za kursor sastoji, ne reflektuju na podatke koji se čitaju iz kursora. Ako se ova ključna reč ne upotrebljava onda svako sledeće očitavanje podataka iz kursora vraća i nove podatke koji su se u međuvremenu promenili u tabelama iz kojih se upit kursora sastoji.
[SCROLL]	Ova ključna reč definiše da se pokazivač na red u kursoru može pomerati u svim pravcima: FIRST (prvi red), LAST (poslednji red), PRIOR (prethodni red), NEXT (sledeći red), RELATIVE (relativni broj reda u odnosu na), ABSOLUTE (redni broj reda u odnosu na prvi red). Ako se ova ključna reč preskoči tada je NEXT jedina komanda pomeranja pokazivača na red u kursoru.
select_naredba	To je upit koji vraća rezultantnu tabelu nad kojom se formira kursor
READ ONLY	Ova ključna reč se upotrebljava da bi onemogućila promenu podataka u tabelama kroz kursor
UPDATE [OF naziv_kolone [,...n]]	Definiše kolone koje se mogu ažurirati preko kursora. Ako se izostave nazivi kolona i zadrži samo ključna reč UPDATE onda se sve kolone u kursoru mogu ažurirati

Sintaksa kreiranja kursora u **SUBP MySQL** je:

```
DECLARE ime_kursora CURSOR
FOR select_naredba
```

gde je:

select_naredba	To je upit koji vraća rezultantnu tabelu nad kojom se formira kursor

Jedna od sintaksi kreiranja kursora u **SUBP Oracle** koju ćemo koristiti u našim primerima je:

```
CURSOR ime_kursora
 [(parametar [ IN] tip_podatka [{:= | DEFAULT }
vrednost]
 [, parametar [ IN] tip_podatka [{:= | DEFAULT }
vrednost...)]
 [ RETURN rowtype] IS select_naredba;
```

gde je:

parametar [IN] tip_podatka [{:= | DEFAULT }	Parametri u kursoru su opcioni i ako se koriste, onda se oni nalaze u delu select_naredba, najčešće u WHERE ili HAVING delu
RETURN rowtype	Ova ključna reč se upotrebljava da bi se podaci koje vraća upit u kursoru formatirali u formatima kolona u nekoj tablici (npr. radnici%ROWTYPE)
select_naredba	To je upit koji vraća rezultantnu tabelu nad kojom se formira kursor

Primena kursora

U cilju demonstriranja svrhe postojanja objekta CURSOR kreiraćemo tabelu PRIMANJA.

Tabela PRIMANJA treba da ima onoliko slogova koliko ima i tabela RADNIK i u nju se upisuju ukupna primanja svakog zaposlenog. To je redudantna tabela u odnosu na tabele PLATE i PLATE_STAVKE. Tako, ako radnik sa idbr 3887 koji ima mesečnu platu 2000 novčanih jedinica primi sledeće obračune: 1000 jedinica kao prvi deo plate za januar, 200 novčanih jedinica kao drugi deo za januar i 2000 jedinica kao jedini deo za februar, u tabeli PRIMANJA njegova ukupna primanja (polje *iznos*) treba da budu 1000+200+2000=3200 novčanih jedinica. Na početku, prilikom uvođenja, tabela PRIMANJA je prazna.Pri obračunu plate treba da se upiše obračun plata u tabelama PLATE i PLATE_STAVKE (recimo prvi deo plate za februar za zaposlenog sa idbr 3887 u iznosu od 2000 novčanih jedinica) i da se ukupna suma upiše u tabelu PRIMANJA (3200 novčanih jedinica za idbr 3887) kako bi se pratilo koliko je ukupno para isplaćeno svakom radniku.

Međutim, prilikom sledećeg obračuna plate u tabeli PRIMANJA potrebno je samo povećati (izmeniti, ažurirati) prethodno izračunati ukupni iznos za iznos novo-dodate naknade. Isto tako, može se desiti da ubacimo novog zaposlenog u tabeli RADNIK nakon što smo obračunali nekoliko plata ostalim radnicima (recimo zaposlimo novog radnika i prvi put mu obračunamo platu za mart 2012. godine, dok u tabelama postoje obračuni plata i za januar i za februar za ostale radnike), tako da novi zaposleni još uvek nema ni jednu evidentiranu platu niti slog u tabeli PRIMANJA. Tada prilikom obračunavanja sledeće plate, ažuriramo postojeći iznos u tabeli PRIMANJA za ostale radnike, a za novog zaposlenog po prvi put ubacujemo (insertujemo) ukupan iznos plate. Dakle, zadatak procedure je da nakon ubacivanja odgovarajućih slogova u tabele PLATE i PLATE_STAVKE za svakog zaposlenog ispita da li za njega postoji slog u tabeli PRIMANJA – ako postoji, da iznos ažurira, a ako ne postoji da

ubaci novi slog sa ukupnim iznosom plata za tog zaposlenog. Vidimo da postoje dve različite obrade podataka – jedna za slučaj da odgovarajući idbr postoji u tabeli PRIMANJA i drugi kada odgovarajući idbr ne postoji.

Primer 254. Kreirati tabelu PRIMANJA koja ima dve kolone: idbr (PK) i iznos. Kreirati i relaciju između kolona idbr u tabelama PRIMANJA i RADNIK

MS SQL

```
CREATE TABLE primanja(
  idbr int NOT NULL ,
  iznos money,
  PRIMARY KEY (idbr));
ALTER TABLE primanja ADD CONSTRAINT
fk_primanja_radnik FOREIGN KEY (idbr) REFERENCES radnik(idbr);
```

MySQL

```
CREATE TABLE primanja(
  idbr int NOT NULL ,
  iznos decimal(17,2),
  PRIMARY KEY (idbr)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;
ALTER TABLE primanja ADD CONSTRAINT
fk_primanja_radnik FOREIGN KEY (idbr) REFERENCES radnik(idbr);
```

Oracle

```
CREATE TABLE primanja(
  idbr NUMBER(6) PRIMARY KEY,
  iznos NUMBER(17,2),

  CONSTRAINT fk_primanja_radnik
  FOREIGN KEY (idbr) REFERENCES radnik(idbr)
);
```

Primer 255. Kreirati proceduru procIzracunajPlatu4, sličnu proceduri procIzracunajPlatu3, u kojoj će se osim ubacivanja novih slogova u tabele PLATE i PLATE_STAVKE ažurirati i tabela PRIMANJA. Tabelu PRIMANJA ažurirati za svakog zaposlenog tako da ako u njoj postoji slog sa odgovarajućim idbr, onda treba povećati vrednost u polju iznos za vrednost koja se ubacuje u tabelu PLATE_STAVKE, u suprotnom (ako ne postoji idbr) ubaciti novi slog sa iznosom koji je ubačen u tabeli PLATE_STAVKE

MS SQL:

```
CREATE PROCEDURE procIzracunajPlatu4(
  @parRbpl tinyint,
  @parGodina smallint,
  @parMesec  tinyint,
  @parPostotakPlate decimal(6,2),
  @parPostotakPremije decimal(6,2))
AS
BEGIN
-- 1. korak definisanje promenljivih
  DECLARE @brSlogova1 int;
  DECLARE @v_idbr int;
  DECLARE @v_iznos money;
  DECLARE @danas SMALLDATETIME, @autoid int, @error int,@brSlogova int;
  SET @danas = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');

-- 2. korak deklarisanje kurzora
  DECLARE nas_kurzor CURSOR FOR
  SELECT ps.idbr, ps.iznos
  FROM plate p, plate_stavke ps
  WHERE p.autoid = ps.autoid and p.rbpl=@parRbpl and
        p.godina=@parGodina and p.mesec=@parMesec
  ORDER BY ps.idbr;

  SELECT @brSlogova = count(*) FROM plate
  WHERE godina = @parGODINA and mesec = @parMESEC and rbpl = @parRBPL;

  IF @brSlogova = 0
  BEGIN
    BEGIN TRANSACTION
      INSERT INTO plate(rbpl, godina, mesec, datum)
      values(@parRbpl, @parGodina, @parMesec, @danas)
      SET @error = @@error
      SELECT @autoid = @@IDENTITY

      IF @error != 0
        GOTO Prijava_Greske;

      INSERT into plate_stavke(autoid, idbr, iznos)
      SELECT @autoid, idbr, @parPostotakPlate/100* plata +
        @parPostotakPremije/100*ISNULL(premija, 0.00)
      FROM radnik;
      SET @error = @@error;

      IF @error != 0
        GOTO Prijava_Greske;

    -- 3. korak otvaramo kurzor
      OPEN nas_kurzor;
```

MS SQL:

```
    -- nastavak sa prethodne strane
    -- 4. korak: preuzimamo red iz kurzora u lokalne promenljive
      FETCH NEXT FROM nas_kurzor
      INTO @v_idbr, @v_iznos;

      WHILE @@FETCH_STATUS = 0
      BEGIN
        -- 5. korak aktiviramo naredbu/naredbe
        SELECT @brSlogova1 = count(*)
        FROM primanja
        WHERE idbr = @v_idbr;

        if @brSlogova1 > 0
        UPDATE primanja SET iznos = iznos + @v_iznos
        WHERE idbr = @v_idbr;
        else
        INSERT INTO primanja(idbr, iznos)
        VALUES(@v_idbr, @v_iznos);

        SET @Error = @@ERROR;
        IF @Error != 0
        Goto Prijava_Greske;

        FETCH NEXT FROM nas_kurzor
        INTO @v_idbr, @v_iznos;
      END;

    -- 6. korak: zatvaramo kurzor
    CLOSE nas_kurzor;
    DEALLOCATE nas_kurzor;

    COMMIT TRANSACTION
  END
  ELSE
    RaisError('Navedeni obračun plate već postoji u bazi!', 16, 1);
  Return;

Prijava_Greske:
  ROLLBACK TRANSACTION
RaisError('Desila se greška!', 16, 1);

END;
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procIzracunajPlatu4(
  parRbpl tinyint,
  parGodina smallint,
  parMesec  tinyint,
  parPostotakPlate decimal(6,2),
  parPostotakPremije decimal (6,2),
  out parPoruka varchar(255))
BEGIN

  -- 1. korak definisanje promenljivih
  DECLARE done INT DEFAULT FALSE;
  DECLARE brSlogova1 int;
  DECLARE v_idbr int;
  DECLARE v_iznos decimal(17,2);
  DECLARE brSlogova int;
  DECLARE local_autoid int;

  -- 2. korak deklarisanje kurzora
  DECLARE nas_kurzor CURSOR FOR
  SELECT ps.idbr, ps.iznos
  FROM plate p, plate_stavke ps
  WHERE p.autoid=ps.autoid AND p.rbpl=parRBPL AND
        p.godina=parGodina AND p.mesec=parMesec
  ORDER BY ps.idbr;

  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET done = TRUE;
  DECLARE EXIT handler for sqlexception
  begin
      rollback;
      set parPoruka = 'Desila se greška!' ;
  end;
  SET parPoruka = 'OK';

  SELECT count(*) into brSlogova FROM plate
  WHERE godina = parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 Then
    START TRANSACTION;
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values (parRbpl, parGodina, parMesec, current_date);

    SET local_autoid = LAST_INSERT_ID();

    INSERT INTO plate_stavke(autoid, idbr, iznos)
    SELECT local_autoid, idbr, fnIzracunajPlatu(plata, premija, datzap,
      parPostotakPlate, parPostotakPremije)
    FROM radnik;

    -- 3. korak otvaramo kurzor
    OPEN nas_kurzor;
```

MySQL:

```
    -- nastavak sa prethodne strane

    startuj_petlju: LOOP

      -- 4. korak: preuzimamo red iz kurzora u lokalne promenljive
      FETCH nas_kurzor
      INTO v_idbr, v_iznos;

      -- izlazimo iz kurzora kada vise ne mozemo da fetchujemo
      --  ni jedan red, tada je done = 1)
      IF done THEN
        LEAVE startuj_petlju;
      END IF;

      -- 5. korak aktiviramo naredbu/naredbe
      SELECT count(*) into brSlogova1
      FROM primanja
      WHERE idbr = v_idbr;

      if brSlogova1 > 0 then
        UPDATE primanja SET iznos = iznos + v_iznos
        WHERE idbr = v_idbr;
      else
        INSERT INTO primanja(idbr, iznos)
        VALUES(v_idbr, v_iznos);
      end if;

    END LOOP;

    -- 6. korak: zatvaramo kurzor
    CLOSE nas_kurzor;

    Commit;
  ELSE
    SET parPoruka = 'Navedeni obračun plate već postoji u bazi!';
  END IF;
END $$

DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procIzracunajPlatu4(
  parRbpl PLATE.RBPL%type,
  parGodina PLATE.GODINA%type,
  parMesec  PLATE.MESEC%type,
  parPostotakPlate NUMBER,
  parPostotakPremije NUMBER)
IS
  local_autoid NUMBER(6);
  brSlogova NUMBER(4);
  DUPLA_PLATA EXCEPTION;

  -- 1. korak definisanje promenljivih
  v_idbr NUMBER(6);
  v_iznos NUMBER(17,2);
  brSlogova1 NUMBER(4);

-- 2. korak deklarisanje kurzora
  CURSOR nas_kurzor IS
  SELECT ps.idbr, ps.iznos
  FROM plate p, plate_stavke ps
  WHERE p.autoid=ps.autoid and p.rbpl=parRbpl and
        p.godina=parGodina and p.mesec=parMesec
  ORDER BY idbr;

BEGIN
  SELECT count(*) into brSlogova FROM plate
  WHERE godina= parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 THEN
    SELECT seq_plate.nextval into local_autoid FROM dual;

    INSERT INTO plate(autoid, rbpl, godina, mesec, datum)
    VALUES (local_autoid, parRbpl, parGodina, parMesec, CURRENT_DATE);

    INSERT INTO plate_stavke (autoid, idbr, iznos)
    SELECT local_autoid, idbr, fnIzracunajPlatu(plata, premija, datzap,
      parPostotakPlate, parPostotakPremije)
    FROM radnik;

    -- 3. korak otvaramo kurzor
    OPEN nas_kurzor;
    LOOP
      -- 4. korak: preuzimamo red iz kurzora u lokalne promenljive
      FETCH nas_kurzor
      INTO v_idbr, v_iznos;

      -- NOTFOUND (= true kada ne mozemo da fetchujemo nove redove)
      EXIT WHEN nas_kurzor%NOTFOUND;
```

```
Oracle:
```

-- nastavak sa prethodne strane

```
      -- 5. korak aktiviramo naredbu/naredbe
      SELECT count(*) into brSlogova1
      FROM primanja
      WHERE idbr = v_idbr;

      if brSlogova1 > 0 then
        UPDATE primanja SET iznos = iznos + v_iznos
        WHERE idbr = v_idbr;
      else
        INSERT INTO primanja(idbr, iznos)
        VALUES(v_idbr, v_iznos);
      end if;
    END LOOP;

    -- 6. korak: zatvaramo kurzor
    CLOSE nas_kurzor;

commit;
  ELSE
    RAISE DUPLA_PLATA;
  END IF;

  EXCEPTION
    WHEN DUPLA_PLATA THEN
      raise_application_error(-20205, 'Navedeni obračun plate već postoji
                              u bazi!');
    WHEN OTHERS THEN
      rollback;
      raise_application_error(-20205, 'Desila se greška!');

END procIzracunajPlatu4;
```

Opšti princip rada kursora kod sva tri SUBP-a je isti. Postoji nekoliko standardnih koraka:

1. Definišu se promenljive u kojima će se kopirati sadržaj polja rezultata upita kursora,
2. Deklariše se kursor kao skup podataka koji će se formirati nad rezultatom SELECT upita,
3. Otvara se kursor i definiše se petlja kroz koju se prolazi onoliko puta koliko ima slogova u izvršenom SELECT upitu kursora,
4. Pri svakom prolasku u petlji pokazivač se premešta na sledeći red u zamišljenoj tabeli koja se dobija kada se izvrši SELECT naredba u kursoru i vrednosti polja u tom redu se kopiraju u promenljive definisane u koraku 1,

5. Nakon preuzimanja vrednosti u lokalne promenljive izvršavaju se određene naredbe, obično iza uslovnog skoka (IF naredba),
6. Nakon završetka petlje zatvara se kursor.

Za svaki slog u našem kursoru koji se sastoji iz rednog broja zaposlenog (polje idbr) i naknade (polje iznos) i koji se upisuje u tabelu STAVKE_PLATE vrši se provera da li za tog zaposlenog (polje idbr) postoji odgovarajući slog u tabeli PRIMANJA. Ako postoji (IF brSlogova1 > 0) povećava se ukupna vrednost primljene naknade za iznos poslednje naknade naredbom UPDATE, a ako ne postoji (deo ELSE) ubacuje (insertuje) se novi slog u tabelu PRIMANJA.

Pre nego što pokrenemo proceduru procIzracunajPlatu4 inicijalno ćemo popuniti tabelu PRIMANJA ukupnim primanjima za svakog zaposlenog.

Primer 256. Inicijalno popuniti tabelu PRIMANJA sumom iznosa za svakog zaposlenog (idbr) iz tabele PLATE_STAVKE

MS SQL, MySQL, Oracle

```
INSERT into primanja (idbr, iznos)
SELECT idbr, sum(iznos)
FROM plate_stavke
GROUP BY idbr
ORDER BY idbr;
```

Primer 257. Izvršiti proceduru procIzracunajPlatu4 sa sledećim parametrima: parRbpl=1, parGodina=2012, parMesec=2, parPostotakplate=50, parPostotakpremije=50

MS SQL

```
execute procIzracunajPlatu4 1, 2012, 2, 50.0, 50.0;
```

MySQL

```
call procIzracunajPlatu4(1, 2012, 2, 50, 50, @a);
SELECT @a;
```

Oracle

```
execute procIzracunajPlatu4(1, 2012, 2, 50.0, 50.0);
```

U pokazanom primeru, u kursoru će se izvršavati samo naredba UPDATE primanja jer za svakog zaposlenog već postoji odgovarajući slog u tabeli PRIMANJA.

Međutim, ako ubacimo novog zaposlenog u tabelu RADNIK, a kasnije obračunamo novu platu, pri čemu se u obračunu prvi put računa plata za novog zaposlenog, u proceduri procIzracunajPlatu4 će se za tog zaposlenog izvršiti komanda INSERT into primanja... jer njegov idbr ne postoji u tabeli PRIMANJA.

Primer 258. U tabelu RADNIK ubaciti novog zaposlenog: Nebojšu Mudrinića, zaposlenog 14. Februara 2012. sa visokom školskom spremom na poziciji „prevodilac“ sa platom od 1000 i premijom od 2000 novčanih jedinica

MS SQL, MySQL (*ukloniti N ispred '*):

```
INSERT INTO radnik(idbr, ime, prezime, posao, kvalif,
                   rukovodilac, datzap, premija, plata, brod)
values(8357, N'Nebojša', N'Mudrinić', N'prevodilac', N'VSS',
        NULL, '2012-02-14', 1000, 2000, NULL);
```

Oracle:

```
INSERT INTO radnik(idbr, ime, prezime, posao, kvalif,
                   rukovodilac, datzap, premija, plata, brod)
values(8357, 'Nebojša', 'Mudrinić', 'prevodilac', 'VSS', NULL,
      to_date('02142012', 'MMDDYYYY'), 1000, 2000, NULL);
```

Primer 259. Izvršiti proceduru procIzracunajPlatu4 sa parametrima: parRbpl=2, parGodina=2012, parMesec=2, parPostotakPlate=50 i parPostotakPremije=50

MS SQL

```
execute procIzracunajPlatu4 2, 2012, 2, 50.0, 50.0;
```

MySQL

```
call procIzracunajPlatu4(2, 2012, 2, 50, 50, @a);
SELECT @a;
```

Oracle

```
execute procIzracunajPlatu4(2, 2012, 2, 50.0, 50.0);
```

Primetićemo da je u tabeli PRIMANJA dodat slog za idbr=8357 i iznosom u vrednosti pola plate za tog zaposlenog. To znači da je za tog zaposlenog izvršena naredba „INSERT into primanja“ koja se nalazi u ELSE bloku u IF naredbi u proceduri procIzracunajPlatu4.

Isti rezultat možemo dobiti ako izvršimo proceduru procIzracunajPlatu5 u kojoj objekat CURSOR zamenimo složenijom UPDATE naredbom praćenu INSERT naredbom.

Primer 260. Kreirati proceduru procIzracunajPlatu5 sličnu proceduri procIzracunajPlatu4 u kojoj će se upotreba kursor objekta zameniti prvo jednom UPDATE i jednom INSERT naredbom. UPDATE naredba treba da poveća u tabeli PRIMANJA vrednosti u koloni iznos za odgovarajuće vrednosti iznosa zaposlenih iz poslednjeg obračuna u tabeli PLATE_STAVKE, a zatim dodati iznose iz tabele PLATE_STAVKE u tabelu PRIMANJA za one zaposlene koji se ne nalaze u tabeli PRIMANJA

MS SQL:

```
CREATE PROCEDURE procIzracunajPlatu5(
  @parRbpl tinyint,
  @parGodina smallint,
  @parMesec  tinyint,
  @parPostotakPlatedecimal(6,2),
  @parPostotakPremije decimal(6,2))
AS
BEGIN
  DECLARE @danas SMALLDATETIME, @autoid int, @error int,@brSlogova int;
  SET @danas = REPLACE(Convert(varchar(10), getdate(), 102), '.', '-');

  SELECT @brSlogova = count(*) FROM plate
  WHERE godina = @parGODINA and mesec = @parMESEC and rbpl = @parRBPL;

  IF @brSlogova = 0
  BEGIN
    BEGIN TRANSACTION
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values(@parRbpl, @parGodina, @parMesec, @danas)
    SET @error = @@error
    IF @error != 0
      GOTO Prijava_Greske;

    SELECT @autoid = @@IDENTITY

    INSERT into plate_stavke(autoid, idbr, iznos)
    SELECT @autoid, idbr, @parPostotakPlate/100* plata +
      @parPostotakPremije/100*ISNULL(premija, 0.00)
    FROM radnik;
    SET @error = @@error;

    IF @error != 0
      GOTO Prijava_Greske;
```

MS SQL:

```
-- nastavak sa prethodne strane

    -- izvršimo ažuriranje svih iznosa za zaposlene koji se nalaze u
    -- tabeli PRIMANJA
    UPDATE primanja SET iznos = pr.iznos+ps.iznos
    FROM plate pl, plate_stavke ps, primanja pr
    WHERE pl.autoid = ps.autoid and ps.idbr =pr.idbr AND
pl.mesec=@parMesec
      AND pl.godina = @parGodina AND pl.rbpl = @parRbpl;

    SET @Error = @@ERROR;
    IF @Error != 0
      Goto Prijava_Greske;

    -- naknade zaposlenih koji se ne nalaze u tabeli primanja se unose
    --  ovde po prvi put
    INSERT INTO primanja(idbr, iznos)
    SELECT ps.idbr, ps.iznos
    FROM plate p, plate_stavke ps
    WHERE ps.autoid = p.autoid and ps.idbr
      NOT IN (SELECT idbr FROM primanja) and
      p.mesec=@parMesec AND p.godina = @parGodina AND p.rbpl = @parRbpl

    SET @Error = @@Error;
    IF @Error != 0
      Goto Prijava_Greske;

    commit transaction;
  END
  ELSE
    RaisError('Već postoji plata za navedeni mesec i godinu!', 16, 1);
  RETURN;

Prijava_Greske:
  BEGIN
    ROLLBACK TRANSACTION;
    RaisError('Desila se greška!', 16, 1);
  END

END
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procIzracunajPlatu5(
  parRbpl tinyint,
  parGodina smallint,
  parMesec  tinyint,
  parPostotakPlate decimal(6,2),
  parPostotakPremije decimal (6,2),
  out parPoruka varchar(255))
BEGIN
  DECLARE brSlogova int;
  DECLARE local_autoid int;

  DECLARE EXIT handler for sqlexception
  begin
      rollback;
      set parPoruka = 'Desila se greška!' ;
  end;
  SET parPoruka = 'OK';

  SELECT count(*) into brSlogova FROM plate
  WHERE godina = parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 Then
    START TRANSACTION;
    INSERT INTO plate(rbpl, godina, mesec, datum)
    values (parRbpl, parGodina, parMesec, current_date);

    SET local_autoid = LAST_INSERT_ID();

    INSERT INTO plate_stavke(autoid, idbr, iznos)
    SELECT local_autoid, idbr, fnIzracunajPlatu(plata, premija, datzap,
      parPostotakPlate, parPostotakPremije)
    FROM radnik;

    -- izvršimo ažuriranje svih iznosa za zaposlene koji se nalaze u
    -- tabeli PRIMANJA
    UPDATE plate pl, plate_stavke ps, primanja pr
    SET pr.iznos = pr.iznos+ps.iznos
    WHERE pl.autoid = ps.autoid and ps.idbr =pr.idbr AND pl.mesec=parMesec
          AND pl.godina = parGodina AND pl.rbpl = parRbpl;
    -- naknade zaposlenih koji se ne nalaze u tabeli primanja se unose
    -- ovde
    INSERT INTO primanja(idbr, iznos)
    SELECT ps.idbr, ps.iznos
    FROM plate p, plate_stavke ps
    WHERE ps.autoid = p.autoid and ps.idbr
      NOT IN (SELECT idbr FROM primanja) and
      p.mesec=parMesec AND p.godina = parGodina AND p.rbpl = parRbpl;
    commit;
  ELSE
    SET parPoruka = 'Već postoji plata za navedeni mesec i godinu!';
  END IF;
END $$
DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procIzracunajPlatu5(
  parRbpl PLATE.RBPL%type,
  parGodina PLATE.GODINA%type,
  parMesec  PLATE.MESEC%type,
  parPostotakPlate NUMBER,
  parPostotakPremije NUMBER)
IS
  local_autoid NUMBER(6);
  brSlogova NUMBER(4);
  DUPLA_PLATA EXCEPTION;
BEGIN
  SELECT count(*) into brSlogova FROM plate
  WHERE godina= parGodina and mesec = parMesec and rbpl = parRbpl;

  IF brSlogova = 0 THEN
    SELECT seq_plate.nextval into local_autoid FROM dual;

    INSERT INTO plate(autoid, rbpl, godina, mesec, datum)
    VALUES (local_autoid, parRbpl, parGodina, parMesec, CURRENT_DATE);

    INSERT INTO plate_stavke (autoid, idbr, iznos)
    SELECT local_autoid, idbr, fnIzracunajPlatu(plata, premija, datzap,
      parPostotakPlate, parPostotakPremije)
    FROM radnik;

    -- izvršimo ažuriranje svih iznosa za zaposlene koji se nalaze u
    -- tabeli PRIMANJA
    UPDATE
    (
    SELECT ps.idbr as idbr1, ps.iznos as iznosPlate, pr.idbr as idbr2,
      pr.iznos as iznosPrimanja
    FROM plate_stavke ps, plate pl, PRIMANJA pr
    WHERE  pl.autoid=ps.autoid  and  ps.idbr=pr.idbr  AND  pl.mesec=parMesec
      AND pl.godina = parGodina
    ) SET iznosPrimanja = iznosPrimanja + iznosPlate;

  -- naknade zaposlenih koji se ne nalaze u tabeli primanja se unose ovde
    INSERT INTO primanja(idbr, iznos)
    SELECT ps.idbr, ps.iznos
    FROM plate p, plate_stavke ps
    WHERE ps.autoid = p.autoid and ps.idbr
      NOT IN (SELECT idbr FROM primanja) and
      p.mesec = parMesec AND p.godina = parGodina AND p.rbpl = parRbpl;
    commit;
  ELSE
    RAISE DUPLA_PLATA;
  END IF;

  EXCEPTION
  WHEN DUPLA_PLATA THEN
    RAISE_APPLICATION_ERROR (-20205,
        'Vec postoji plata za navedeni mesec i godinu');
  WHEN OTHERS THEN
    rollback;
    raise_application_error(-20205, 'Greska: Greska nepoznata!' );
END procIzracunajPlatu5;
```

U ovoj proceduri možemo jednom naredbom da ažuriramo vrednost polja *iznos* u tabeli PRIMANJA kao zbir prethodne vrednosti u istom polju sa iznosom novog dela plate iz tabele PLATE_STAVKE ali samo u slučaju da slog sa vrednošću *idbr* već postoji u tabeli PRIMANJA. Ukoliko ne postoji taj zaposleni u tabeli PRIMANJA onda se naknada za tog zaposlenog ubacuje (insert) nakon UPDATE naredbe.

Ova zamena kursora, složenijom UPDATE i INSERT naredbama ima smisla samo ako je broj kombinacija u IF naredbi iza koraka 5 u opisu koraka kursora mali (npr. ako imamo jednu IF i par ELSE IF naredbi). Međutim, ako je broj uslovnog grananja veliki (nekoliko ELSE IF naredbi), tada je bolje koristiti kursor objekat.

Kreiranje i korišćenje pogleda (VIEW)

Koncept baza podataka daje mogućnost da više različitih aplikacija, pa samim tim i korisnika, obrađuje iste podatke i da iz njih dobija informacije koje su relevantne za onu vrstu posla koja je svakom od njih bitna. Korisnik ne mora da zna sve detalje projekta čitave baze podataka jer njega zanimaju samo neki objekti (tabele) i samo neka njihova svojstva (atributi).

E. F. Codd je u svome pionirskom radu o relacionim bazama podataka definisao više tipova relacija, a među njima i takozvani pogled ili **VIEW.** Osnovna razlika između bazne relacije i relacije VIEW je u tome što bazne relacije postoje na memorijskom medijumu računara (disku), dok je VIEW fiktivna, virtuelna relacija koja ne postoji na disku računara, ali se može formirati uvek ako nam zatreba, odnosno ako se pozovemo na nju. Za korisnika je onda, bar što se pretraživanja tiče, otvorena ista mogućnost korišćenja ovakve relacije kao i u slučaju pretraživanja neke bazne relacije.

VIEW je, prema tome, virtuelna imenovana relacija koja se takođe kreira naredbom SELECT**,** ali se u bazi podataka memoriše na specifičan način. Definicija pogleda (naziv pogleda, nazive kolona i upit) čuva se u bazi podataka u prevedenom obliku, što obezbeđuje veliku brzinu rada sa pogledima.

Svaki put kada korisniku zatreba taj "pogled" na informacioni sistem, on poziva i time kreira tu virtuelnu relaciju. Pri izvršavanju upita nad pogledom, SUBP kombinuje taj upit sa upitom koji definiše pogled i pravi se novi upit koji se izvršava nad baznim tabelama koje čuvaju podatke (fizičke tabele na disku).

Kako ažuriranje (dakle izmena podataka) ovakve relacije najčešće nije moguće, posebno ako je pogled kreiran nad više tabela (što je najčešće slučaj), to se rad sa pogledima nad relacijama koristi često i kao način zaštite integriteta podataka. Značajno je odmah napomenuti da mnogi SUBP nemaju mogućnost rada sa pogledima (na primer Access). Naredba za kreiranje pogleda – **CREATE VIEW** glasi:

CREATE VIEW ime_pogleda **[** atribut1, atribut2, ….. **] AS**
SELECT ……..

gde je **ime_pogleda** naziv novoformirane virtuelne relacije VIEW sa navedenim atributima. Nije potrebno ni naglašavati da u sistemu ne sme da postoji još neka bazna (ili virtuelna) relacija sa istim imenom.

Na primer, za rukovodioca odeljenja 20 nisu značajni podaci o svim radnicima, već samo onima iz njegovog odeljenja. U tu svrhu, iz relacije ***RADNIK < idbr#, ime, ...*** > treba izdvojiti samo one zaposlene koji su od interesa.

Primer 261. Kreirati pogled ODELJENJE20 koje sadrži podatke o odeljenju i ime, posao, kvalifikaciju i platu zaposlenih u odeljenju 20.

CREATE VIEW ODELJENJE20 **AS**
SELECT O.imeod, R. ime, R.posao, R.kvalif, R.plata
FROM RADNIK R, ODELJENJE O
WHERE O.brod=R.brod **AND** O.brod=*20*

Napomena: U ovome primeru nisu uvedeni novi atributi, iako ih pogled može imati. Imena atributa u pogledu i osnovnim relacijama iz kojih je izveden, u ovome primeru su ostala ista.

Napomena: U MS SQL Serveru na kraju upita kojim se kreira pogled ne stavlja se **;** (tačka-zarez).

Svaki put kada neko iz odeljenja 20 hoće da obrađuje podatke o zaposlenima u svom odeljenju on jednostavno koristi ovaj pogled.

Primer 262. Prikaži ime, posao i kvalifikaciju za zaposlene sa visokom stručnom spremom u odeljenju 20:

a)
```
SELECT O20.ime, O20.posao, O20.kvalif
FROM ODELJENJE20  O20
WHERE O20.kvalif='VSS';
```

b)
```
SELECT ime, posao, kvalif
 FROM ODELJENJE O INNER JOIN RADNIK R ON O.brod = R.brod
WHERE RADNIK.kvalif='VSS' AND RADNIK.brod=20;
```

Napomena: Ažuriranje baze podataka preko pogleda ima brojna ograničenja posebno ako je pogled definisan nad više tabela. Najkraće rečeno, da bismo mogli da ažuriramo podatke (u fizičkoj tabeli) preko pogleda, pogledi moraju biti definisani nad jednom tabelom i u definiciju moraju biti uključene sve NOT NULL kolone te tabele.

Napomena: Pogledi obezbeđuju:

- *jednostavnost korišćenja*, uprošćavaju se upiti (upit a je znatno jednostavniji nego b),
- *tajnost*, mehanizam za kontrolu pristupa podacima (korisnik vidi samo neke podatke),
- *performanse*, definicija pogleda se čuva u kompajliranom, prevedenom obliku,
- *nezavisnost podataka*, menjaju se definicije pogleda, a ne aplikacioni programi koji koriste podatke iz baze podataka preko pogleda.

Primer 263. Kreirati pogled IZVOZ koji sadrži podatke o identifikacionom broju radnika, imenu, prezimenu, broju sati angažovanja i funkciji koju imaju na projektu, samo za radnike koji su angažovani na projektu *izvoz*.

```
CREATE  VIEW IZVOZ  AS
SELECT R.idbr, R.ime, R.prezime, U.brsati, U.funkcija
FROM  RADNIK R, UCESCE U
WHERE  R.idbr = U.idbr  AND  U.brproj  =  (  SELECT P.brproj
                                             FROM PROJEKAT P
                                             WHERE P.imeproj='izvoz' );
```

Imena kolona u pogledu ne moraju biti ista kao imena kolona u tabelama iz kojih se pogled izvodi. Pogled se može izbaciti iz baze podataka naredbom **DROP VIEW** ime_pogleda. Ova komanda uklanja pogled i iz relacione šeme i iz rečnika podataka baze podataka.

Možda izgleda čudno, ali poglede treba povremeno izbacivati iz šeme i odmah ih kreirati. Razlog je vrlo prost. Pri kreiranju pogleda vrši se optimizacija njegovog izvršavanja. Naravno, optimalno izvršavanje zavisi od trenutnih vrednosti u tabelama. Nakon nekog vremena, tabele koje se često ažuriraju u toj meri promene stanje da pogled (binarni kod) više nema dovoljnu brzinu. Zbog toga se pogled mora izbaciti iz šeme baze podataka (DROP) i odmah iznova napraviti (CREATE). Prilikom ponovnog kreiranja novodobijeni binarni kod će biti optimalan za novo stanje podataka u bazi.

Zamena za CREATE VIEW u ACCESS-u

Kako je ranije navedeno, u SUBP MS Access nije moguće pokrenuti upit koji kreira pogled. Ipak, postoji mogućnost da se ovaj nedostatak eliminiše ili bar ublaži kreiranjem upita nad upitom ili korišćenjem imenovanih upita. Pogledajte sledeći primer:

Primer 264. Prikaži ime, posao i kvalifikaciju za zaposlene sa visokom stručnom spremom u odeljenju 20.

```
SELECT O20.ime, O20.posao, O20.kvalif
FROM  ( SELECT O.imeod, R.ime, R.posao, R.kvalif, R.plata
          FROM RADNIK AS R, ODELJENJE AS O
          WHERE O.brod = R.brod  AND O.brod = 20) AS O20
WHERE O20.kvalif = 'VSS' ;
```

Ovo je modifikacija zadatka (Primer 262) koji se oslanjao na prethodno kreiran pogled (Primer 261). Pošto u Access-u nemamo mogućnost da kreiramo pogled i da se na njega jednostavno pozovemo navođenjem njegovog imena u odredbi FROM, ovde smo kôd upita prepisali u odredbu FROM "glavnog" upita i tako u suštini kreirali pogled u nekom smislu. Naravno, ovaj "pogled", zapravo podupit, nema niti jednu od gore navedenih dobrih osobina pravog pogleda (jednostavnost korišćenja, tajnost, performanse, nezavisnost podataka) ali može poslužiti u nekim slučajevima, pa ga zato treba imati u vidu.

Primer 265. Izračunati koliko u preduzeću ima različitih kvalifikacija.

```
SELECT Count(*) AS [Broj kvalifikacija]
FROM  ( SELECT DISTINCT  kvalif  FROM  RADNIK );
```

S obzirom na to da **SELECT** count (**DISTINCT** kvalif) ... ili nešto slično nije od pomoći iako izgleda kao logično rešenje zadatka, ovde se možemo poslužiti podupitom koji nam vraća pogled/tabelu koja sadrži samo jednu kolonu u kojoj su navedene sve kvalifikacije koje u glavnom upitu prebrojavamo.

Primer 266. Izlistaj šifru odeljenja, kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odeljenju (slično kao Primer 142).

```
SELECT O.brod AS brod, idbr, ime, posao, plata
FROM  ( SELECT brod, Max(plata) AS  [mplata]
          FROM RADNIK
          GROUP BY brod) AS O, RADNIK
WHERE RADNIK.brod = O.brod AND RADNIK.plata = O.mplata
ORDER BY 1;
```

Brod	Idbr	Ime	Posao	Plata
10	5662	Janko	upravnik	2400
20	5932	Mitar	savetnik	2600
30	5786	Pavle	upravnik	2800
40	6789	Janko	upravnik	3900

Ono što predstavlja problem pri rešavanju ovog zadatka jeste veza između najveće plate u jednom odeljenju i ostalih podataka o radniku koji ima tu platu. Ako bi podupit vratio samo najveće plate po odeljenjima, a ne i iz kog su odeljenja, onda bi moglo da dođe do situacije u kojoj radnik iz jednog odeljenja koji ima platu koja je najveća u drugom odeljenju biva izlistan iako u njegovom odeljenju ima neko sa većom platom. Zbog toga podupit ima dva atributa: *brod* i *mplata* koji se koriste za povezivanje sa tabelom RADNIK, tako da se prethodni problem ne može javiti. Ipak, ono što može da se desi je da u jednom odeljenju bude više zaposlenih sa najvećom platom i oni će svi biti izlistani.

Treba uočiti da nije dobijena potpuna informacija jer nedosteje zapis koja je najveća plata radnika koji nisu rapoređeni ni u jedno odeljenje (NULL, 7892, Luka, analitičar, 2000). Kod SQL Servera, ako je isključena opcija ANSI_NULLS imali bi i tu informaciju jer je Null=Null tačno (True). Da bi dobili i tu informaciju treba dopuniti uslov i uzeti u obzir i radnike kod kojih RADNIK.*brod* (ili O,*brod*) ima nepostojeću vrednost. Tada upit u svim SUBP ima sledeći oblik:

```
SELECT O.brod AS brod, idbr, ime, posao, plata
FROM  ( SELECT brod, Max(plata) AS  [mplata]
          FROM RADNIK
          GROUP BY brod) AS O, RADNIK R
WHERE (R.brod = O.brod OR  R.brod IS NULL) AND R.plata = O.mplata
ORDER BY 1;
```

Brod	Idbr	Ime	Posao	Plata
	7892	Luka	analitičar	2000
10	5662	Janko	upravnik	2400
20	5932	Mitar	savetnik	2600
30	5786	Pavle	upravnik	2800
40	6789	Janko	upravnik	3900

Sledeći primer je skoro identičan prethodnom, ali dodavanje imena odeljenja u spisak potrebnih kolona nameće potrebu da se u podupitu izvrši spajanje tabela.

Primer 267. Izlistaj šifru i ime odeljenja, kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odeljenju.

```
SELECT O.brod AS brod, O.imeod AS imeod, idbr, ime, posao, plata
FROM (SELECT RADNIK.brod, ODELJENJE.imeod, Max(plata) as [mplata]
      FROM RADNIK, ODELJENJE
      WHERE RADNIK.brod = ODELJENJE.brod
      GROUP BY RADNIK.brod, ODELJENJE.imeod) AS O, RADNIK
WHERE RADNIK.brod = O.brod AND RADNIK.plata = O.mplata
ORDER BY 1 ;
```

Primer 268. Prikazati šifre projekata na kojima zajedno rade i radnik 6234 i radnik 5696 (Primer 183).

```
SELECT brproj
FROM  (SELECT brproj FROM ucesce WHERE idbr = 6234)
WHERE brproj IN ( SELECT brproj
                  FROM ucesce
                  WHERE idbr = 5696);
```

Brproj
200
300

Primena pogleda u MS SQL, MySQL i Oracle

U cilju demonstriranja pogleda za složenije potrebe, možemo kreirati upit koji nam pokazuje obračunate plate iz tabela PLATE i PLATE_STAVKE grupisane po svakom mesecu i upoređuje sa maksimalnom mesečnom po svakom zaposlenom.

Primer 269. Kreirati pogled vwMesecniIzvestajPlata koji po svakom zaposlenom pokazuje primljenu mesečnu zaradu, maksimalnu zaradu koju zaposleni može da primi, kao i procenat primljene zarade u odnosu na maksimalnu mesečnu. U pogledu koristiti funkciju fnIzracunajMaxMesecnuPlatu za izračunavanje maksimalne mesečne zarade

MS SQL:

```
CREATE VIEW vwMesecniIzvestajPlata
AS
SELECT p.mesec, p.godina, ps.idbr, r.ime + N' ' + r.prezime as zaposleni,
  sum(ps.iznos) as IznosPlata,
  dbo.fnIzracunajMaxMesecnuPlatu(ps.idbr) as MaxMesPlata,
  sum(ps.iznos)/dbo.fnIzracunajMaxMesecnuPlatu(ps.idbr) * 100 as procenat
FROM plate p, plate_stavke ps, radnik r
WHERE p.autoid = ps.autoid AND r.idbr = ps.idbr
GROUP BY p.mesec, p.godina, ps.idbr, r.ime + N' ' + r.prezime,
  dbo.fnIzracunajMaxMesecnuPlatu(ps.idbr)
```

MySQL:

```
CREATE VIEW vwMesecniIzvestajPlata
AS
SELECT pl.mesec, pl.godina, ps.idbr, concat(r.ime, ' ', r.prezime) as
  zaposleni, sum(ps.iznos) as IznosPlata,
  fnIzracunajMaxMesecnuPlatu(ps.idbr) as MaxMesPlata,
  sum(ps.iznos)/fnIzracunajMaxMesecnuPlatu(ps.idbr) * 100 as procenat
FROM plate pl, plate_stavke ps, radnik r
WHERE pl.autoid = ps.autoid AND r.idbr = ps.idbr
GROUP BY pl.mesec, pl.godina, ps.idbr, concat(r.ime, ' ', r.prezime),
  fnIzracunajMaxMesecnuPlatu(ps.idbr);
ORDER BY ps.idbr, pl.godina, pl.mesec;
```

Oracle:

```
CREATE VIEW vwMesecniIzvestajPlata
AS
SELECT pl.mesec, pl.godina, ps.idbr, r.ime || ' ' || r.prezime as
  zaposleni, sum(ps.iznos) as IznosPlata,
  fnIzracunajMaxMesecnuPlatu(ps.idbr) as MaxMesPlata,
  sum(ps.iznos)/fnIzracunajMaxMesecnuPlatu(ps.idbr) * 100 as procenat
FROM plate pl, plate_stavke ps, radnik r
WHERE pl.autoid = ps.autoid AND r.idbr = ps.idbr
GROUP BY pl.mesec, pl.godina, ps.idbr, r.ime || ' ' || r.prezime,
  fnIzracunajMaxMesecnuPlatu(ps.idbr);
ORDER BY ps.idbr, pl.godina, pl.mesec;
```

Primer 270. Prikazati mesečne zarade zaposlenih korišćenjem pogleda vwMesecniIzvestajPlata

MS SQL, MySQL, Oracle

```
SELECT * FROM vwMesecniIzvestajPlata;
```

Primer 271. Kreirati pogled vwRazlikaPlatePrimanja koji pokazuje kolika je razlika između sume zarada evidentirane u tabeli PLATE_STAVKE po svakom zaposlenom i evidencije iste u tabeli PRIMANJA

MS SQL:

```
CREATE VIEW vwRazlikaPlatePrimanja
AS
SELECT isnull(pr.idbr, ps.idbr) as idbr, isnull(pr.iznos, 0.0) as
  iznosPrimanja, Sum(ps.iznos) as iznosPlata, isnull(pr.iznos, 0.0) -
  Sum(ps.iznos) as razlika
FROM plate_stavke ps full join Primanja pr ON
  ps.idbr = pr.idbr
GROUP BY  isnull(pr.idbr, ps.idbr), isnull(pr.iznos, 0.0)
HAVING isnull(pr.iznos, 0.0) - Sum(ps.iznos) != 0;
```

MySQL:

```
CREATE VIEW vwRazlikaPlatePrimanjaStavke
AS
SELECT ps.idbr, IFNULL(pr.iznos, 0.0) as iznosPrimanja,
  ps.iznos as iznosPlata
FROM plate_stavke ps LEFT outer join primanja pr ON ps.idbr = pr.idbr
UNION
SELECT pr.idbr, IFNULL(pr.iznos, 0.0) as iznosPrimanja,
  IFNULL(ps.iznos, 0.0) as iznosPlata
FROM plate_stavke ps RIGHT join primanja pr ON
  ps.idbr = pr.idbr
WHERE ps.idbr is NULL;
----------

CREATE VIEW vwRazlikaPlatePrimanja
AS
SELECT idbr, iznosPrimanja, Sum(iznosPlata) as iznosPlata,
  iznosPrimanja - Sum(iznosPlata) as razlika
FROM vwRazlikaPlatePrimanjaStavke
GROUP BY  idbr, iznosPrimanja
HAVING iznosPrimanja - Sum(iznosPlata)<> 0
ORDER BY idbr;
```

Oracle:

```
CREATE OR REPLACE VIEW vwRazlikaPlatePrimanja
AS
SELECT nvl(pr.idbr, ps.idbr) as idbr, nvl(pr.iznos, 0.0) as iznosPrimanja,
  Sum(ps.iznos) as iznosPlata, nvl(pr.iznos, 0.0) - Sum(ps.iznos)
  as razlika
FROM plate_stavke ps full join primanja pr ON
  ps.idbr = pr.idbr
GROUP BY nvl(pr.idbr, ps.idbr), nvl(pr.iznos, 0.0)
HAVING nvl(pr.iznos, 0.0) - Sum(ps.iznos) != 0
ORDER BY nvl(pr.idbr, ps.idbr);
```

U ovom upitu upotrebljen je full join kako bi pokazao i slogove koji postoje u tabeli PRIMANJA a nema ih u tabeli PLATE_STAVKE i obrnuto: postoje u tabeli

PLATE_STAVKE a ne postoje u tabeli PRIMANJA. Na ovaj način se mogu videti nepoklapanja između evidencija plata u ovim tabelama.

U SUBP MySQL nije dozvoljeno da se unutar pogleda kreira pod-upit niti je dozvoljena upotreba obostranog spoljnjeg spajanja (full-join). Full join je u našem primeru rešen kao unija dva spoljnja spajanja: LEFT i RIGHT , a podupit je rešen tako što je prvo napravljen pomoćni (podupit) pogled (vwRazlikaPlatePrimanjaStavke) i taj pogled je kasnije upotrebljen kao objekat u odredbi FROM u drugom pogledu, tj. u vwRazlikaPlatePrimanja.

Da bi videli da se evidencija ukupnih plata iz tabele PLATE_STAVKE ne slaže sa evidencijom ukupnih plata iz tabele PRIMANJA, možemo pokrenuti prvo naredbu kojom umanjujemo ukupne plate svih zaposlenih za 100 novčanih jedinica.

Primer 272. U tabeli primanja umanjiti iznos za sve zaposlene za 800 novčanih jedinica

```
MS SQL, MySQL, Oracle
UPDATE primanja SET iznos = iznos - 100;
```

Nakon promene podataka u tabeli PRIMANJA pokazaće se razlike u evidencijama isplate zarada između tabela PLATE_STAVKE i PRIMANJA.

Primer 273. Pokazati razlike između evidencija isplate zarada zaposlenih korišćenjem upita vwRazlikaPlatePrimanja

```
MS SQL, MySQL, Oracle
SELECT * FROM vwRazlikaPlatePrimanja;
```

Prvih nekoliko slogova rezultata upita je dato na sledećoj slici:

idbr	iznosPrimanja	iznosPlata	razlika
8357	1400	1500	-100
5696	1600	1700	-100
5953	1770	1870	-100
5652	2450	2550	-100
5497	2960	3060	-100
3887	3300	3400	-100
7892	3300	3400	-100
5874	3470	3570	-100
...	...	...	...

Kada se ove evidencije razlikuju možemo napraviti proceduru kojom vrednosti u tabeli PRIMANJA ažuriramo na osnovu vrednosti iz tabele PLATE_STAVKE.

Primer 274. Kreirati proceduru procIspraviPrimanja koja će u koloni iznos u tabeli PRIMANJA upisati sumu zarada iz tabele PLATE_STAVKE za svakog zaposlenog koji je evidentiran u tabeli PRIMANJA a zatim dodati sumu zarada izračunatu nad tabelom PLATE_STAVKE za zaposlene koji nisu evidentirani u tabeli PRIMANJA. Upotrebiti transakciju za navedene naredbe.

MS SQL:

```
CREATE PROCEDURE procIspraviPrimanja
AS
BEGIN
  DECLARE @error int

  BEGIN TRANSACTION
  UPDATE primanja set iznos = t.sumaiznos
  FROM primanja pr, (
    SELECT idbr, sum(iznos) as sumaiznos
    FROM plate_stavke
    GROUP BY idbr) t
  WHERE pr.idbr = t.idbr AND pr.iznos != t.sumaiznos

  SET @error = @@error
  if @error != 0
      GOTO prijava_greske

  INSERT INTO primanja(idbr, iznos)
  SELECT idbr, sum(iznos) as sumaiznos
  FROM plate_stavke
  WHERE idbr not in (SELECT idbr from primanja)
  GROUP BY idbr

  SET @error = @@error
  if @error != 0
      GOTO prijava_greske

  COMMIT TRANSACTION
RETURN
prijava_greske:
  RaisError('Greska!', 16, 1)
    ROLLBACK TRANSACTION
END
```

MySQL:

```
DELIMITER $$

CREATE PROCEDURE procIspraviPrimanja(
  out parPoruka varchar(255))
BEGIN
  DECLARE EXIT handler for sqlexception
  begin
    rollback;
    set parPoruka = 'Desila se greška!' ;
  end;
  SET parPoruka = 'OK';

  START TRANSACTION;
```

MySQL:

```
  -- nastavak sa prethodne strane

  UPDATE primanja pr, (
    SELECT idbr, sum(iznos) as sumaiznos
    FROM plate_stavke
    GROUP BY idbr) t
  SET pr.iznos = t.sumaiznos
  WHERE pr.idbr = t.idbr AND pr.iznos <> t.sumaiznos;

  INSERT INTO primanja(idbr, iznos)
  SELECT idbr, sum(iznos) as sumaiznos
  FROM plate_stavke
  WHERE idbr not in (SELECT idbr from primanja)
  GROUP BY idbr;

  COMMIT;
END$$
DELIMITER ;
```

Oracle:

```
create or replace
PROCEDURE procIspraviPrimanja
AS
BEGIN

  UPDATE primanja p
  SET iznos = (SELECT sum(iznos) from plate_stavke ps
               WHERE ps.idbr = p.idbr)
  WHERE EXISTS (
    SELECT ps.idbr, sum(ps.iznos)
    from plate_stavke ps
    where ps.idbr = p.idbr
    group by ps.idbr
    HAVING sum(ps.iznos) <> p.iznos);

  INSERT INTO primanja(idbr, iznos)
  SELECT idbr, sum(iznos) as sumaiznos
  FROM plate_stavke
  WHERE idbr not in (SELECT idbr from primanja)
  GROUP BY idbr;

  COMMIT;

  EXCEPTION
  WHEN OTHERS THEN
    rollback;
    raise_application_error(-20205, 'Greska: Greska nepoznata!' );

END procIspraviPrimanja;
```

Ako izvršimo proceduru procIspraviPrimanja, a zatim pogledamo sadržaj pogleda vwRazlikaPlatePrimanja, videćemo da je pogled prazan – tj. procedura je „izjednačila“ evidenciju naknada u tabeli PRIMANJA sa evidencijom u tabeli PLATE_STAVKE.

Primer 275. Izvršiti proceduru procIspraviPrimanja

```
MS SQL, Oracle
execute procIspraviPrimanja;
```

```
MySQL
call procIspraviPrimanja;
```

Ako ponovimo Primer 273 videćemo da je pogled prazan.

Upravljačke naredbe

Upotreba podataka od strane više korisnika (posebno u mrežnom okruženju) unosi dodatne opasnosti po sigurnost podataka i njihov integritet. Zbog toga su razvijeni brojni i moćni postupci zaštite podataka od slučajnog, ali i od neovlašćenog i zlonamernog korišćenja, izmene ili uništenja. Zaštita podataka može biti razmatrana sa aspekta organizacije poslovanja, vrste obrade i sa aspekta zaštite celovitosti podataka. "Bezbedna i sigurna razmena podataka i obrada nisu moguće bez vrlo organizovanog pristupa i širokog opsega mera zaštite na svim nivoima i fazama obrade. Rukovodioci su odgovorni za implementaciju mera bezbednosti u svim segmentima: obrada podataka, korisnici, nadzor. Kontrola i zaštita moraju biti tako projektovane da minimiziraju gubitke. Ključnu ulogu u projektovanju i provođenju koncepta bezbednosti imaju profesionalci-kontrolori. Provera i kontrola su neophodni u svim fazama obrade podataka".[23] "Napadi na podatke i gubici mogu da se dese bilo gde. Samim tim, upravljačka funkcija i razvoj sistema i alata zaštite podataka postaje sve više važan. Postojeći trendovi pokazuju da će oni u budućnosti biti vrlo značajni. Mora se istaći da su se koncept i struktura, operacije, organizacija i obrada podataka danas značajno promenili". [24]

Zaštita baze podataka se posmatra sa dva aspekta:

- integritet - zaštita od slučajnog i/ili pogrešnog ažuriranja, i
- sigurnost - zaštita od neovlašćenog ažuriranja i korišćenja podataka.

Termin *integritet* ovde se koristi da označi tačnost, korektnost i zaštitu od nepravilnog unosa podataka. Narušavanje integriteta baze podataka može da nastane prilikom izvršavanja paralelnih transakcija, međutim savremeni softveri za upravljanje bazama podataka veoma dobro rešavaju ovaj problem, tako da će se ovde razmatrati samo problem zaštite integriteta prilikom izvođenja jedne transakcije.

Prilikom izrade modela podataka potrebno je definisati koje uslove podaci u bazi podataka treba da zadovolje, kada se vrši provera i koje akcije treba preduzeti kada definisani uslovi nisu ispunjeni. O ovome se vodi računa prilikom definisanja tipova podataka i njihovih domena pri kreiranju tabela i relacija između njih. Pravila integriteta se definišu za operacije ažuriranja baze podataka: **INSERT**, **UPDATE** i **DELETE**. Kako se ova pravila menjaju od slučaja do slučaja, ona moraju biti podržana i u samim aplikacijama.

Pravila integriteta se dele na dve klase:

- pravila integriteta domena (integritet entiteta) i
- pravila integriteta relacija (referencijalni integritet).

Prilikom izrade modela podataka za implementaciju baze, moraju se za svaki atribut definisati domeni, a takođe i uslovi koji moraju biti zadovoljeni prilikom izvođenja operacija nad objektima.

Sigurnost podataka – dodela prava korisnicima

Termin ”sigurnost podataka” odnosi se na mehanizme zaštite baze podataka od neovlašćenog korišćenja. Sigurnost podataka ima mnogo aspekata (kriptografija, zaštita pri

[23] [33] S. Obradović, D. Tesić, M. Mijalković, V. Marković, "The influence of the type of processing on computer crime protection", International scientific conference UNITECH’09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 343-I 350, ISSN:1313-230X.

[24] [28] S. Obradović, D. Tesić, S. Ilić, V. Marković, "Responsibility of management related to data security", International scientific conference UNITECH’09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 337-I 342, ISSN:1313-230X.],

prenosu, fizičko obezbeđenje, itd.), od kojih će biti opisana samo zaštita od neovlašćenog korišćenja koju pružaju softveri za upravljanje bazama podataka.

Najrasprostranjeniji princip sigurnosti podataka je dodela, tj. ograničavanje prava pristupa i korišćenja. Obično se svakom korisniku dodeljuju odgovarajuće privilegije koje određuju koje operacije korisnik može da izvrši nad bazom podataka i njenim objektima (tabelama).

Tabela koju kreira neki korisnik je njegova tabela, on je njen vlasnik. Drugi korisnik je načelno ne može koristiti ukoliko mu vlasnik eksplicitno ne dodeli prava korišćenja pomoću naredbe **GRANT**. Opšti oblik naredbe **GRANT** jeste:

GRANT {ALL | [ALTER, DELETE, INDEX, SELECT, UPDATE(atr**)]}**
ON [kreator**.]{**tabela|pogled**}**
TO {PUBLIC | korisnik1**[,** korisnik2 **...]}**
[WITH GRANT OPTION-].

Primer 276. Dodeliti korisniku korisničkog imena Luka pravo da kreira tabele u bazi PREDUZECE.

USE PREDUZECE
GRANT CREATE TABLE TO *Luka*;

Primer 277. Dodeliti sva prava korisniku korisničkog imena Miloš nad bazom PREDUZECE.

USE PREDUZECE
GRANT ALL TO *Miloš*;

Primer 278. Dodeliti SELECT pravo korisniku korisničkog imena Janko nad tabelom RADNIK baze PREDUZECE.

USE PREDUZECE
GRANT SELECT ON RADNIK **TO** *Janko*;

Primer 279. Dodeliti SELECT pravo korisniku korisničkog imena Janko nad atributima brod i imeod tabele ODELJENJE u bazi PREDUZECE.

USE PREDUZECE
GRANT SELECT (brod, imeod) **ON** ODELJENJE **TO** *Janko*;

Primer 280. Dodeliti pravo brisanja korisniku korisničkog imena Janko nad tabelom UCESCE u bazi PREDUZECE.

USE PREDUZECE
GRANT DELETE ON UCESCE **TO** *Janko*;

Napomena: **DELETE** pravo ne može biti dodeljeno nad nekim atributima tabele, već samo nad celom tabelom.

Primer 281. Dodeliti pravo korisniku korisničkog imena Janko da može da ažurira polja plata i premija u tabeli RADNIK baze PREDUZECE.

USE PREDUZECE
GRANT UPDATE (plata, premija) **ON** RADNIK **TO** *Janko*;

Primer 282. Dodeliti pravo korisniku korisničkog imena Janko da može da dodaje zapise u tabelu RADNIK baze PREDUZECE.

USE PREDUZECE
GRANT INSERT ON RADNIK **TO** *Janko*;

Primer 283. Dodeliti pravo korisniku korisničkog imena boza sva prava nad kreiranim pogledom ODELJENJE20.

USE PREDUZECE
GRANT ALL ON ODELJENJE20 **TO** *boza*;

Ako je drugi korisnik dobio od vlasnika i opciju [**WITH GRANT OPTION**-], onda on može davati drugim korisnicima prava korišćenja tabele, ali samo ista ili manja od onih koja je on dobio od vlasnika. Tako, prethodni primer mozemo izmeniti i dati korisniku **boza** pravo da pravo nad pogledom ODELJENJE20 može on da dodeli drugom korisniku.

Primer 284. Dodeliti pravo korisniku korisničkog imena boza sva prava nad kreiranim pogledom ODELJENJE20 i dati mu i pravo da prenese ovlašćenje nad istim drugom korisniku.

USE PREDUZECE
GRANT ALL ON ODELJENJE20 **TO** *boza* **WITH GRANT OPTION**;

Isto tako možemo i korisniku Janko da dodelimo ista prava kao i korisniku boza.

Primer 285. Dodeliti pravo korisniku korisničkog imena Janko sva prava nad kreiranim pogledom ODELJENJE20 i dati mu i pravo da prenese ovlašćenje nad istim drugom korisniku.

USE PREDUZECE
GRANT ALL ON ODELJENJE20 **TO** *Janko* **WITH GRANT OPTION**;

Imati na umu da nad objektom (pogledom) ODELJENJE20 ista prava imaju i Janko i boza. Recimo da je Janko preneo svoja prava korisniku Miloš, istom naredbom kao na Primer 283 izuzev da se naredba završava sa: **TO** *Miloš*. Neka je i korisnik **boza** to isto učinio, ne znajući da je Janko već preneo svoja prava Milošu. Miloš je sada dobio prava i od korisnika Janka i od korisnika boza.

Drugim korisnicima vlasnik može dodeliti sva prava (**ALL**) ili samo ona iz liste. Ako dozvoljavamo korisniku samo da gleda podatke, treba mu dodeliti pravo **SELECT**, a ako može i da ih briše, onda i pravo **DELETE**. Promena podataka (ažuriranje) može se ograničiti samo na neke atribute izabranih tabela ili pogleda.

Oduzimanje prava vrši se naredbom REVOKE, čiji je opšti oblik:

REVOKE {[GRANT OPTION FOR] | ALL |
[ALTER, DELETE, INDEX, SELECT, UPDATE(atr**)]}**
ON [kreator**.]{**tabela|pogled**}**
FROM {PUBLIC | korisnik1**[,** korisnik2 **...]}.**

Primer 286. Oduzeti korisniku korisničkog imena Luka pravo da kreira tabele u bazi PREDUZECE.

USE PREDUZECE
REVOKE CREATE TABLE TO *Luka*;

Primer 287. Oduzeti SELECT pravo korisniku korisničkog imena Janko nad tabelom RADNIK baze PREDUZECE.

USE PREDUZECE
REVOKE SELECT ON RADNIK **TO** *Janko*;

Primer 288. Oduzeti pravo korisniku korisničkog imena Janko da može da ažurira polja plata i premija u tabeli RADNIK baze PREDUZECE.

USE PREDUZECE
REVOKE UPDATE (plata, premija) **ON** RADNIK **TO** *Janko*;

Ovde se slično prenosu ovlašćenja kod naredbe WITH GRANT OPTION može oduzeti pravo da korisnik prenosi svoja ovlašćenja, bez oduzimanja ovlašćenja pristupa objektu. Tako možemo korisniku Janko oduzeti ovlašćenje da može dalje prenositi ovlašćenja nad objektom (pogledom) ODELJENJE20.

Primer 289. Zabraniti pravo prenosa ovlašćenja nad pogledom ODELJENJE 20 korisniku Janko.

USE PREDUZECE
REVOKE GRANT OPTION FOR ALL ON ODELJENJE20 **TO** *Janko*;

Korisnik Janko ovim više nema prava da prenosi svoja ovlašćenja nad pogledom ODELJENJE20, ali ovlašćenje koje je prethodno dodelio korisniku Miloš i dalje važi. Međutim, ako korisniku Janko oduzmemo sva ovlašćenja nad pogledom ODELJENJE20, onda korisnik Miloš nasleđuje oduzimanje ovlašćenja.

Primer 290. Zabraniti pravo svih ovlašćenja nad pogledom ODELJENJE 20 korisniku Janko.

USE PREDUZECE
REVOKE ALL ON ODELJENJE20 **TO** *Janko*;

Međutim, u ovom posebnom slučaju, korisnik Miloš neće izgubiti prava nad pogledom ODELJENJE20 jer je to pravo dobio i od korisnika boza koji pravo pristupa tom objektu još uvek ima.

MS SQL Server ima i naredbu **DENY** kojom se grupi ili korisniku zabranjuju neka prava.

DENY {ALL | [ALTER, DELETE, INDEX, SELECT, UPDATE(atr**)]}**
ON [kreator**.]{**tabela|pogled**}**
FROM {PUBLIC | korisnik1**[,** korisnik2 **...]}.**

Primer 291. Zabraniti pravo brisanja korisniku korisničkog imena boza nad tabelom UCESCE u bazi PREDUZECE.

USE PREDUZECE
DENY DELETE ON UCESCE **TO** *boza*;

Zaštita integriteta podataka

Koncept baze podataka daje prave efekte onda kada se radi u mrežnom okruženju, kada veliki broj korisnika istovremeno pristupa podacima iz jedne baze. U tom slučaju postoji realna opasnost da dva ili više korisnika – klijenata pristupe istom ili istim podacima u cilju čitanja ali i izmene podataka. U tom slučaju može doći do pojave pogrešnih rezultata, te i ažurnost i integritet podataka mogu biti ugroženi. Da bi se sprečile štetne posledice do kojih može doći kada više korisnika istovremeno pristupa istim podacima, većina sistema za upravljanje bazama podataka koristi razne tehnike zaključavanja podataka (**Data Locks**). Dakle, kada jedan korisnik pokuša da izvede neku operaciju sa podacima, DBMS te podatke automatski zaključava, naravno samo ako je u pitanju operacija ažuriranja (**UPDATE** ili **DELETE**). Nema potrebe zaključavati podatke kada ih neki upit samo čita, odnosno upit ne sme da blokira ažuriranje.

Postoji više strategija zaključavanja, od vrlo pesimističkih gde se zaključava čitava tabela (**table-level locking**) ili blokovi-stranice podataka (**page-level locking**), preko zaključavanja samo onih n-torki koje se ažuriraju (**row-level locking**), do optimističkih da do izmene istih polja neće ni doći (bez zaključavanja). Naravno, zbog mogućih problema sistemi za upravljanje bazama podataka moraju imati ugrađene mehanizme čuvanja prethodnih verzija podataka, pravljenje rezervnih kopija (**BACKUP**), a takođe vode se i dnevnici transakcija.

Na bezbednost podataka i brzinu rada mogu imati veliki uticaj i replikacija i distribucija podataka i obrade (distribuirane baze podataka) [17], [20], [22], 23].

Replikacija predstavlja kopiranje dela podataka iz centralne baze na neki udaljeni server sa ciljem povećanja brzine odziva i smanjenja komunikacionih troškova. Najveći deo obrade podataka se ostvaruje na lokalnom serveru (replici) pa je brzina velika i nema potrebe za komunikacijom na daljinu. Neophodno je uspostaviti i mehanizme za održavanje ažurnosti podataka u centralnoj bazi u slučaju promene podataka u repliciranoj grupi podataka. Osim podataka potrebno je održavati i sistemske kataloge koji takođe mogu biti replicirani. ”Centralizovani sistemski katalog ima određenih prednosti koje imaju svoju cenu. Šema sa repliciranim katalogom nije efikasna u slučaju velikog broja replika sistemskih kataloga i komunikacioni sistem može biti preopterećen”.[25]

Kod distribuiranih baza treba voditi računa da se mogu distribuirati podaci i/ili obrada. Dakle postoje četiri moguće kombinacije. Baza i obrada nisu distribuirani (centralizovan sistem). Distribuirani podaci a obrada centralizovana, centralizovani podaci a distribuirana obrada i distribuirani i podaci i obrada. "Sistem za podršku distribuiranog razvoja softvera obično koristi centralizovanu klijent-server arhitekturu. Ovakav pristup ima određenih nedostataka kao što je veliko kašnjenje (veliki komunikacioni troškovi) ako su korisnici na lokacijama koje su vrlo udaljene od servera. Pear-to-Pear sistemi baza podataka koji koriste lokalne kataloge nemaju prednost jer moraju da pretražuju sve lokalne kataloge".[26]

[25] [23] A. Donchev, S. Obradović, F. Abdiwi, "Cost Measurement of a Distributed Database System with Normalized Workload", International scientific conference UNITECH’07, TU-Gabrovo, 23-24 November 2007, Gabrovo, proceedings, vol.I, pp. I 478-I 482, ISSN 1313-230X.

[26] [20] A. Donchev, S. Obradović, " *System Catalogue Selection in a Peer-to-Peer* Database", International scientific conference UNITECH’06, TU-Gabrovo, 24-25 November 2006, Gabrovo, proceedings, vol.I, pp. I 345-I 349, ISBN 10: 954-683-351-7, ISBN 13: 978-954-683-351-8.

Način distribucije i particionisanje baze imaju veliki uticaj na karakteristike sistema. "... moguće je odrediti optimalan broj fragmenata za zadatu tabelu relacione baze podataka. Optimalan broj fragmenata zavisi od načina pristupa određenim zapisima u fragmentu". [27]

Danas se u relacionim bazama podataka vrlo često koriste i klasteri (više servera baze podataka), upravo da bi se postigle bolje performanse. Naravno realizacija klastera, posebno za Web aplikacije, ima smisla samo u slučaju kada postoji vrlo veliko opterećenje mreže. Takođe, "Web aplikacije moraju biti posebno projektovane sa ciljem da rade na klasteru. Kako klaster nameće dodatno opterećenje kao rezultat se dobija veće kašnjenje, ali se višestruko povećava propusni opseg. Web aplikacije moraju imati stvarne potrebe da rade u klasteru ... ".[28]

Oporavak baze podataka na osnovu rezervnih kopija

Oporavak baze podataka (**RECOVERY**) predstavlja proces vraćanja baze podataka u stanje za koje se zna da je korektno, a posle nekog softverskog ili hardverskog otkaza sistema. Uzroci otkaza mogu da budu različiti: greške u programiranju, greške u operativnom sistemu i samom softveru za upravljanje bazama podataka, padanje glava diska, nestanak napajanja, sabotaža itd.

Princip na kojem se zasniva oporavak baze podataka je *redundanca podataka*, odnosno postojanje više primeraka-kopija jednog te istog podatka na nekom od memorijskih uređaja (disku ili traci). Proces oporavka može se opisati na sledeći način:

- periodično se cela baza podataka **kopira** (*dump, backup*) na neku arhivsku memoriju,
- za svaku promenu u bazi u takozvani ***log file*** (*žurnal*) zapisuju se stara (*before image*) i nova vrednost (*after image*) sloga baze podataka,
- posle otkaza sistema, ukoliko je baza podataka oštećena, rekonstruiše se ispravno stanje baze na osnovu poslednje arhivske kopije, a ukoliko je baza samo dovedena u nekonzistentno stanje, poništavaju se sve nekorektne promene, a same transakcije se ponove.

Na kraju, možemo istaći da SQL ima znatno veće mogućnosti od onih koje pruža relaciona algebra i ima snagu koju obezbeđuje relacioni račun.

SQL omogućuje da se međurezultati smeštaju u privremene relacije i da se zadaju, odnosno kodiraju proizvoljni izrazi relacione algebre.

SQL pruža velike mogućnosti za obradu informacija i u potpunosti zadovoljava sve veće potebe koje mogu da se ostvaruju u informacionim sistemima.

Mnoge SQL implementacije omogućavaju da SQL upiti budu deo programa napisanih u jezicima opšte namene kao što su C, C++, Java, PL/1, Pascal, Cobol ili noviji vizuelni jezici Visual Basic ili Visual C++. Ovo proširuje mogućnosti programera da manipulišu bazama podataka.

[27] [17] A. Donchev, S. Obradović, "Analitical *determination of the horizontal partitions in a distributed database"*, International scientific conference UNITECH'04, TU-Gabrovo, 18-19 November 2004, Gabrovo, proceedings, vol.I, pp. I 342-I 347, ISBN 954-683-303-7.

[28] [22] Dragan Simić, Srećko Ristić, Slobodan Obradović *"Measurement of the Achieved Performance Levels of the Web Applications with Distributed Relational Database"*, Facta Universitatis Ser : Electonics &. Energetics Vol. 20, no 1, April 2007, pp 31-43, ISSN 0353-3670, COBISS.SR-ID 12826626

Prilog 1

U Prilogu 1 su dati programski redovi kojima kreiramo tabele šeme Preduzeće i popunjavamo ih odgovarajućim sadržajem kako bi rezultati primene upita bili isti kao u primerima pokazanim u knjizi.

Programski redovi dati u ovom prilogu podeljeni su u dva dela – prvi deo se izvršava od strane administratora i ima za cilj da kreira praznu šemu i ovlašćenja za korisnika koji će moći da kreira objekte u bazi i popuni ih odgovarajućim sadržajem i drugi deo gde korisnik kreira objekte (tabele i relacije) i popunjava ih datim sadržajem.

Kreiranje šeme Preduzeće u SUBP SQL Server

korisnik: sa ili administrator

```
CREATE DATABASE preduzece;

CREATE LOGIN student WITH PASSWORD = 'student'
      ,DEFAULT_DATABASE = [preduzece]
GO

      -- Sada kreiramo korisnika student za korisnički nalog student

USE preduzece;
CREATE USER student FOR LOGIN student;
GO

EXEC sp_addrolemember 'db_owner', 'student'
```

---

korisnik: student

```
CREATE TABLE odeljenje(
  brod int PRIMARY KEY,
  imeod NVARCHAR(50) NOT NULL,
  mesto NVARCHAR(50),
  sefod int
);

CREATE TABLE radnik(
  idbr int PRIMARY KEY,
  ime NVARCHAR (20) NOT NULL,
  prezime NVARCHAR (40) NOT NULL,
  posao NVARCHAR(20),
  kvalif NVARCHAR(10),
  rukovodilac int,
  datzap SMALLDATETIME,
  premija MONEY ,
  plata MONEY DEFAULT 0.00,
  brod int,
  CONSTRAINT fk_radnik_odeljenje
    FOREIGN KEY (brod) REFERENCES odeljenje(brod),
  CONSTRAINT fk_radnik_rukovodilac
    FOREIGN KEY (rukovodilac) REFERENCES radnik (idbr)
);
```

```
CREATE TABLE projekat(
  brproj INT PRIMARY KEY,
  imeproj NVARCHAR(20) NOT NULL,
  sredstva money DEFAULT 0.00,
  rok SMALLDATETIME
  );

CREATE TABLE ucesce(
  idbr INT NOT NULL,
  brproj INT NOT NULL,
  brsati SMALLINT NOT NULL,
  funkcija NVARCHAR(20),
  CONSTRAINT ucesce_pk PRIMARY KEY (idbr, brproj),
  CONSTRAINT fk_ucesce_radnik
    FOREIGN KEY (idbr) REFERENCES radnik(idbr),
  CONSTRAINT fk_ucesce_projekat
    FOREIGN KEY (brproj) REFERENCES projekat(brproj)
  );

INSERT INTO odeljenje (brod, imeod, mesto)
  values (10, N'Komercijala', N'Novi Beograd');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (20, N'Plan', N'Dorćol');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (30, N'Prodaja', N'Stari Grad');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (40, N'Direkcija', N'Banovo Brdo');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (50, N'Računski centar', N'Zemun');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (60, N'Nabavka', N'Rakovica');

INSERT INTO radnik(idbr, ime, prezime, posao, kvalif, rukovodilac,
                   datzap, premija, plata, brod)
values (5367, N'Petar', N'Vasić',   N'vozač', 'KV', NULL,
'1978-01-01', 1900.00, 1300.00,     20),
(5497, N'Aleksandar', N'Marić', N'električar', 'KV', NULL,
  '1990-02-17', 800.0, 1000.0, 10),
(5519, N'Vanja', N'Kondić', N'prodavac', 'VKV', NULL,
  '1991-11-07', 1300.0, 1200.0, 10),
(5652, N'Jovan', N'Perić', N'električar', 'KV', NULL,
  '1980-05-31', 500.0, 1000.0, 10),
(5662, N'Janko', N'Mančić',   N'upravnik', 'VSS', NULL,
  '1993-08-12', NULL, 2400, 10),
(5696, N'Mirjana', N'Dimić', N'čistač', 'KV', NULL,
  '1991-09-30', 0, 1000, 10),
(5780, N'Božidar', N'Ristić', N'upravnik', 'VSS', NULL,
  '1984-08-11', NULL, 2200, 20),
(5786, N'Pavle', N'Šotra', N'upravnik', 'VSS', NULL,
  '1983-05-22', NULL, 2800, 30),
(5842, N'Miloš', N'Marković', N'direktor', 'VSS', NULL,
  '1981-12-15', NULL, 3000,   40),
(5867, N'Svetlana', N'Grubač', N'savetnik', 'VSS', NULL,
  '1970-08-08', NULL, 2750, 40),
(5874, N'Tomislav', N'Bogovac', N'električar', 'KV', NULL,
  '1971-04-19', 1100, 1000, 10),
(5898, N'Andrija',      N'Ristić', N'nabavljač', 'KV', NULL,
  '1980-01-20', 1200, 1100, 30),
(5900, N'Slobodan', N'Petrović', N'vozač', 'KV', NULL,
  '2002-10-03', 1300, 900, 20),
(5932, N'Mitar', N'Vuković', N'savetnik', 'VSS', NULL,
```

```
  '2000-03-25', NULL, 2600, 20),
(5953, N'Jovan', N'Perić', N'nabavljač', 'KV', NULL,
  '1979-01-12', 0, 1100, 30),
(6234, N'Marko', N'Nastić', N'analitičar', 'VSS', NULL,
  '1990-12-17', 3000, 1300, 30),
(6789, N'Janko', N'Simić', N'upravnik', 'VSS', NULL,
  '2003-12-23', 10, 3900, 40),
(7890, N'Ivan', N'Buha', N'analitičar', 'VSS', NULL,
  '2003-12-17', 3200, 1600, 20),
(7892, N'Luka', N'Bošković', N'analitičar', 'VSS', NULL,
  '2004-05-20', NULL, 2000, NULL);

update radnik set rukovodilac = 5780 where idbr = 5367;
update radnik set rukovodilac = 5662 where idbr = 5497;
update radnik set rukovodilac = 5662 where idbr = 5519;
update radnik set rukovodilac = 5662 where idbr = 5652;
update radnik set rukovodilac = 6789 where idbr = 5662;
update radnik set rukovodilac = 5662 where idbr = 5696;
update radnik set rukovodilac = 6789 where idbr = 5780;
update radnik set rukovodilac = 6789 where idbr = 5786;
update radnik set rukovodilac = 5842 where idbr = 5867;
update radnik set rukovodilac = 5662 where idbr = 5874;
update radnik set rukovodilac = 5786 where idbr = 5898;
update radnik set rukovodilac = 5780 where idbr = 5900;
update radnik set rukovodilac = 5842 where idbr = 5932;
update radnik set rukovodilac = 5786 where idbr = 5953;
update radnik set rukovodilac = 5867 where idbr = 6234;
update radnik set rukovodilac = 5842 where idbr = 6789;
update radnik set rukovodilac = 5867 where idbr = 7890;
update radnik set rukovodilac = 5867 where idbr = 7892;

update odeljenje set sefod = 5662 where brod = 10;
update odeljenje set sefod = 5780 where brod = 20;
update odeljenje set sefod = 5786 where brod = 30;
update odeljenje set sefod = 5842 where brod = 40;

insert into projekat
  values(100, N'uvoz', 3000000.00, '2004-05-05');
insert into projekat
  values(200, N'izvoz', 2000000.00, '2005-08-22');
insert into projekat
  values(300, N'plasman', 6000000.00, '2004-12-02');
insert into projekat
  values(400, N'projektovanje', 5000000.00, '2005-04-14');
insert into projekat
  values(500, N'izgradnja', 0.00, '2005-08-22');

insert into ucesce values(5497, 400, 2000, N'IZVRŠILAC');
insert into ucesce values(5652, 100, 1000, N'IZVRŠILAC');
insert into ucesce values(5652, 300, 1000, N'IZVRŠILAC');
insert into ucesce values(5662, 300, 2000, N'ŠEF');
insert into ucesce values(5696, 200, 2000, N'ŠEF');
insert into ucesce values(5696, 300, 2000, N'IZVRŠILAC');
insert into ucesce values(5780, 200, 2000, N'ORGANIZATOR');
insert into ucesce values(5786, 100, 2000, N'KONSULTANT');
insert into ucesce values(5842, 100, 2000, N'ŠEF');
insert into ucesce values(5867, 200, 2000, N'KONSULTANT');
insert into ucesce values(5898, 200, 2000, N'IZVRŠILAC');
insert into ucesce values(5900, 100, 2000, N'IZVRŠILAC');
insert into ucesce values(5932, 100, 500, N'KONSULTANT');
insert into ucesce values(5932, 200, 1000, N'ORGANIZATOR');
```

```
insert into ucesce values(5932, 300, 500, N'NADZORNIK');
insert into ucesce values(5953, 100, 1000, N'IZVRŠILAC');
insert into ucesce values(5953, 300, 1000, N'IZVRŠILAC');
insert into ucesce values(6234, 100, 500, N'NADZORNIK');
insert into ucesce values(6234, 200, 1200, N'IZVRŠILAC');
insert into ucesce values(6234, 300, 300, N'KONSULTANT');
insert into ucesce values(6789, 200, 2000, N'IZVRŠILAC');
insert into ucesce values(7890, 300, 2000, N'IZVRŠILAC');

CREATE INDEX ix_prezime ON radnik(prezime);
CREATE INDEX ix_ime ON radnik(ime);
CREATE INDEX ix_brod ON radnik(brod);
CREATE INDEX ix_rukovodilac ON radnik(rukovodilac);
```

Kreiranje šeme Preduzeće u SUBP MySQL Server

Korisnik: root

```
CREATE USER student@'%' IDENTIFIED BY student;

GRANT USAGE ON * . * TO student@'%' IDENTIFIED BY student WITH
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0
MAX_USER_CONNECTIONS 0 ;

CREATE DATABASE IF NOT EXISTS preduzece ;

GRANT ALL PRIVILEGES ON preduzece . * TO student@'%';
-------------------------
```

Korisnik: student

```
CREATE TABLE odeljenje (
  brod int NOT NULL auto_increment,
  imeod varchar(50) NOT NULL,
  mesto varchar(50) ,
  sefod int ,
  PRIMARY KEY  (brod)
) ENGINE=InnoDB
DEFAULT CHARSET=utf8  COLLATE=utf8_general_ci
AUTO_INCREMENT=1;

CREATE TABLE radnik(
  idbr int NOT NULL,
  ime VARCHAR(20) NOT NULL,
  prezime VARCHAR (40) NOT NULL,
  posao VARCHAR(20),
  kvalif VARCHAR(10),
  rukovodilac int,
  datzap DATE,
  premija DECIMAL(17,2) ,
  plata DECIMAL(17,2) DEFAULT 0.00,
  brod int,
  PRIMARY KEY  (IDBR)
  ) ENGINE=InnoDB
DEFAULT CHARSET=utf8  COLLATE=utf8_general_ci;
```

```
ALTER TABLE radnik
  ADD CONSTRAINT fk_radnik_odeljenje
    FOREIGN KEY (brod) REFERENCES odeljenje (brod),
  ADD CONSTRAINT fk_radnik_rukovodilac
    FOREIGN KEY (rukovodilac) REFERENCES radnik (idbr);

CREATE TABLE projekat(
  brproj INT NOT NULL,
  imeproj VARCHAR(20) NOT NULL,
  sredstva DECIMAL(17,2) DEFAULT 0.00,
  rok DATE,
  PRIMARY KEY  (BRPROJ)
  ) ENGINE=InnoDB
DEFAULT CHARSET=utf8  COLLATE=utf8_general_ci;

CREATE TABLE ucesce(
  idbr int NOT NULL,
  brproj int NOT NULL,
  brsati int NOT NULL,
  funkcija VARCHAR(20),
  PRIMARY KEY (IDBR, BRPROJ)
) ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

ALTER TABLE ucesce
  ADD CONSTRAINT fk_ucesce_radnik
    FOREIGN KEY (idbr) REFERENCES radnik(idbr),
  ADD CONSTRAINT fk_ucesce_projekat
    FOREIGN KEY (brproj) REFERENCES projekat(brproj);

INSERT INTO odeljenje (brod, imeod, mesto)
  values (10, 'Komercijala', 'Novi Beograd');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (20, 'Plan', 'Dorćol');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (30, 'Prodaja', 'Stari Grad');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (40, 'Direkcija', 'Banovo Brdo');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (50, 'Računski centar', 'Zemun');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (60, 'Nabavka', 'Rakovica');

INSERT INTO radnik(idbr, ime, prezime, posao, kvalif, rukovodilac,
                   datzap, premija, plata, brod) values
(5367, 'Petar', 'Vasić', 'vozač', 'KV', NULL,
  '1978-01-01', 1900.00, 1300.00,   20),
(5497, 'Aleksandar', 'Marić', 'električar', 'KV', NULL,
  '1990-02-17', 800.0, 1000.0, 10),
(5519, 'Vanja', 'Kondić', 'prodavac', 'VKV', NULL,
  '1991-11-07', 1300.0, 1200.0, 10),
(5652, 'Jovan', 'Perić', 'električar', 'KV', NULL,
  '1980-05-31', 500.0, 1000.0, 10),
(5662, 'Janko', 'Mančić',     'upravnik', 'VSS', NULL,
  '1993-08-12', NULL, 2400, 10),
(5696, 'Mirjana', 'Dimić', 'čistač', 'KV', NULL,
  '1991-09-30', 0, 1000, 10),
(5780, 'Božidar', 'Ristić', 'upravnik', 'VSS', NULL,
  '1984-08-11', NULL, 2200, 20),
(5786, 'Pavle', 'Šotra', 'upravnik', 'VSS', NULL,
  '1983-05-22', NULL, 2800, 30),
```

```
(5842, 'Miloš', 'Marković', 'direktor', 'VSS', NULL,
  '1981-12-15', NULL, 3000,   40),
(5867, 'Svetlana', 'Grubač', 'savetnik', 'VSS', NULL,
  '1970-08-08', NULL, 2750, 40),
(5874, 'Tomislav', 'Bogovac', 'električar', 'KV', NULL,
  '1971-04-19', 1100, 1000, 10),
(5898, 'Andrija', 'Ristić', 'nabavljač', 'KV', NULL,
  '1980-01-20', 1200, 1100, 30),
(5900, 'Slobodan', 'Petrović', 'vozač', 'KV', NULL,
  '2002-10-03', 1300, 900, 20),
(5932, 'Mitar', 'Vuković', 'savetnik', 'VSS', NULL,
  '2000-03-25', NULL, 2600, 20),
(5953, 'Jovan', 'Perić', 'nabavljač', 'KV', NULL,
  '1979-01-12', 0, 1100, 30),
(6234, 'Marko', 'Nastić', 'analitičar', 'VSS', NULL,
  '1990-12-17', 3000, 1300, 30),
(6789, 'Janko', 'Simić', 'upravnik', 'VSS', NULL,
  '2003-12-23', 10, 3900, 40),
(7890, 'Ivan', 'Buha', 'analitičar', 'VSS', NULL,
  '2003-12-17', 3200, 1600, 20),
(7892, 'Luka', 'Bošković', 'analitičar', 'VSS', NULL,
  '2004-05-20', NULL, 2000, NULL);

update radnik set rukovodilac = 5780 where idbr = 5367;
update radnik set rukovodilac = 5662 where idbr = 5497;
update radnik set rukovodilac = 5662 where idbr = 5519;
update radnik set rukovodilac = 5662 where idbr = 5652;
update radnik set rukovodilac = 6789 where idbr = 5662;
update radnik set rukovodilac = 5662 where idbr = 5696;
update radnik set rukovodilac = 6789 where idbr = 5780;
update radnik set rukovodilac = 6789 where idbr = 5786;
update radnik set rukovodilac = 5842 where idbr = 5867;
update radnik set rukovodilac = 5662 where idbr = 5874;
update radnik set rukovodilac = 5786 where idbr = 5898;
update radnik set rukovodilac = 5780 where idbr = 5900;
update radnik set rukovodilac = 5842 where idbr = 5932;
update radnik set rukovodilac = 5786 where idbr = 5953;
update radnik set rukovodilac = 5867 where idbr = 6234;
update radnik set rukovodilac = 5842 where idbr = 6789;
update radnik set rukovodilac = 5867 where idbr = 7890;
update radnik set rukovodilac = 5867 where idbr = 7892;

update odeljenje set sefod = 5662 where brod = 10;
update odeljenje set sefod = 5780 where brod = 20;
update odeljenje set sefod = 5786 where brod = 30;
update odeljenje set sefod = 5842 where brod = 40;

insert into projekat values(100, 'uvoz', 3000000.00, '2004-05-05');
insert into projekat values(200, 'izvoz', 2000000.00, '2005-08-22');
insert into projekat values(300, 'plasman', 6000000.00, '2004-12-02');
insert into projekat values(400, 'projektovanje', 5000000.00,
  '2005-04-14');
insert into projekat values(500, 'izgradnja', 0.00, '2005-08-22');

insert into ucesce values(5497, 400, 2000, 'IZVRŠILAC');
insert into ucesce values(5652, 100, 1000, 'IZVRŠILAC');
insert into ucesce values(5652, 300, 1000, 'IZVRŠILAC');
insert into ucesce values(5662, 300, 2000, 'ŠEF');
insert into ucesce values(5696, 200, 2000, 'ŠEF');
insert into ucesce values(5696, 300, 2000, 'IZVRŠILAC');
insert into ucesce values(5780, 200, 2000, 'ORGANIZATOR');
```

```
insert into ucesce values(5786, 100, 2000, 'KONSULTANT');
insert into ucesce values(5842, 100, 2000, 'ŠEF');
insert into ucesce values(5867, 200, 2000, 'KONSULTANT');
insert into ucesce values(5898, 200, 2000, 'IZVRŠILAC');
insert into ucesce values(5900, 100, 2000, 'IZVRŠILAC');
insert into ucesce values(5932, 100, 500, 'KONSULTANT');
insert into ucesce values(5932, 200, 1000, 'ORGANIZATOR');
insert into ucesce values(5932, 300, 500, 'NADZORNIK');
insert into ucesce values(5953, 100, 1000, 'IZVRŠILAC');
insert into ucesce values(5953, 300, 1000, 'IZVRŠILAC');
insert into ucesce values(6234, 100, 500, 'NADZORNIK');
insert into ucesce values(6234, 200, 1200, 'IZVRŠILAC');
insert into ucesce values(6234, 300, 300, 'KONSULTANT');
insert into ucesce values(6789, 200, 2000, 'IZVRŠILAC');
insert into ucesce values(7890, 300, 2000, 'IZVRŠILAC');

CREATE INDEX ix_prezime ON radnik(prezime);
CREATE INDEX ix_ime ON radnik(ime);
CREATE INDEX ix_brod ON radnik(brod);
CREATE INDEX ix_rukovodilac ON radnik(rukovodilac);
```

Commit;

Kreiranje šeme Preduzeće u SUBP Oracle DB Server

Korisnik: sys

```
CREATE user preduzece IDENTIFIED BY preduzece;

GRANT
CREATE SESSION,
CREATE TABLE,
CREATE PROCEDURE,
CREATE SEQUENCE,
CREATE TRIGGER,
CREATE VIEW,
CREATE SYNONYM,
ALTER SESSION,
CREATE ANY INDEX,
CREATE PUBLIC SYNONYM,
CREATE USER,
CREATE ROLE,
CREATE ANY DIRECTORY,
QUERY REWRITE,
DROP PUBLIC SYNONYM,
UNLIMITED TABLESPACE

TO preduzece;

------------------------
```

Korisnik: preduzece

```
CREATE TABLE ODELJENJE(
  BROD NUMBER(6,0) PRIMARY KEY,
  IMEOD VARCHAR2(50) NOT NULL,
  MESTO VARCHAR2(50),
  SEFOD NUMBER(6,0)
);

CREATE TABLE RADNIK(
  IDBR NUMBER(6,0) PRIMARY KEY,
  IME VARCHAR2 (20) NOT NULL,
  PREZIME VARCHAR2 (40) NOT NULL,
  POSAO VARCHAR2(20),
  KVALIF VARCHAR2(10),
  RUKOVODILAC NUMBER (6,0),
  DATZAP DATE,
  PREMIJA NUMBER(17,2) ,
  PLATA NUMBER(17,2) DEFAULT 0.00,
  BROD NUMBER(6,0),
  CONSTRAINT fk_radnik_odeljenje
    FOREIGN KEY (BROD) REFERENCES ODELJENJE(BROD),
  CONSTRAINT fk_radnik_rukovodilac
    FOREIGN KEY (RUKOVODILAC) REFERENCES RADNIK (IDBR)
  );

CREATE TABLE PROJEKAT(
  BRPROJ NUMBER (6,0) PRIMARY KEY,
  IMEPROJ VARCHAR2(20) NOT NULL,
  SREDSTVA NUMBER(17,2) DEFAULT 0.00,
  ROK DATE
  );
```

```
CREATE TABLE UCESCE(
  IDBR NUMBER(6,0) NOT NULL,
  BRPROJ NUMBER(6,0) NOT NULL,
  BRSATI NUMBER(4,0) NOT NULL,
  FUNKCIJA VARCHAR2(20),
  CONSTRAINT ucesce_pk PRIMARY KEY (IDBR, BRPROJ),
  CONSTRAINT fk_ucesce_radnik
    FOREIGN KEY (IDBR) REFERENCES RADNIK(IDBR),
  CONSTRAINT fk_ucesce_projekat
    FOREIGN KEY (BRPROJ) REFERENCES PROJEKAT(BRPROJ)
  );

INSERT INTO odeljenje (brod, imeod, mesto)
  values (10, 'Komercijala', 'Novi Beograd');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (20, 'Plan', 'Dorćol');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (30, 'Prodaja', 'Stari Grad');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (40, 'Direkcija', 'Banovo Brdo');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (50, 'Računski centar', 'Zemun');
INSERT INTO odeljenje (brod, imeod, mesto)
  values (60, 'Nabavka', 'Rakovica');

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values (5367, 'Petar', 'Vasić', 'vozač', 'KV', NULL,
  to_date('01011978', 'MMDDYYYY'), 1900.00, 1300.00,  20);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values (5497, 'Aleksandar', 'Marić', 'električar', 'KV', NULL,
  to_date('02171990', 'MMDDYYYY'), 800.0, 1000.0, 10);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5519, 'Vanja', 'Kondić', 'prodavac', 'VKV', NULL,
  to_date('11071991', 'MMDDYYYY'), 1300.0, 1200.0, 10);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5652, 'Jovan', 'Perić', 'električar', 'KV', NULL,
  to_date('05311980', 'MMDDYYYY'), 500.0, 1000.0, 10);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5662, 'Janko', 'Mančić',     'upravnik', 'VSS', NULL,
  to_date('08121993','MMDDYYYY'), NULL, 2400, 10);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5696, 'Mirjana', 'Dimić', 'čistač', 'KV', NULL,
  to_date('09301991', 'MMDDYYYY'), 0, 1000, 10);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5780, 'Božidar', 'Ristić', 'upravnik', 'VSS', NULL,
  to_date('08111984','MMDDYYYY'), NULL, 2200, 20);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5786, 'Pavle', 'Šotra', 'upravnik', 'VSS', NULL,
```

```
  to_date('05221983', 'MMDDYYYY'), NULL, 2800, 30);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5842, 'Miloš', 'Marković', 'direktor', 'VSS', NULL,
  to_date('12151981', 'MMDDYYYY'), NULL, 3000,  40);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5867, 'Svetlana', 'Grubač', 'savetnik', 'VSS', NULL,
  to_date('08081970', 'MMDDYYYY'), NULL, 2750, 40);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5874, 'Tomislav', 'Bogovac', 'električar', 'KV', NULL,
  to_date('04191971', 'MMDDYYYY'), 1100, 1000, 10);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5898, 'Andrija', 'Ristić', 'nabavljač', 'KV', NULL,
  to_date('01201980','MMDDYYYY'), 1200, 1100, 30);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5900, 'Slobodan', 'Petrović', 'vozač', 'KV', NULL,
  to_date('10032002', 'MMDDYYYY'), 1300, 900, 20);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5932, 'Mitar', 'Vuković', 'savetnik', 'VSS', NULL,
  to_date('03252000', 'MMDDYYYY'), NULL, 2600, 20);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(5953, 'Jovan', 'Perić', 'nabavljač', 'KV', NULL,
  to_date('01121979', 'MMDDYYYY'), 0, 1100, 30);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(6234, 'Marko', 'Nastić', 'analitičar', 'VSS', NULL,
  to_date('12171990', 'MMDDYYYY'), 3000, 1300, 30);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(6789, 'Janko', 'Simić', 'upravnik', 'VSS', NULL,
  to_date('12232003', 'MMDDYYYY'), 10, 3900, 40);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(7890, 'Ivan', 'Buha', 'analitičar', 'VSS', NULL,
  to_date('12172003', 'MMDDYYYY'), 3200, 1600, 20);

INSERT INTO RADNIK(IDBR, IME, PREZIME, POSAO, KVALIF, RUKOVODILAC,
                   DATZAP, PREMIJA, PLATA, BROD)
values(7892, 'Luka', 'Bošković', 'analitičar', 'VSS', NULL,
  to_date('05202004', 'MMDDYYYY'), NULL, 2000, NULL);

update radnik set rukovodilac = 5780 where idbr = 5367;
update radnik set rukovodilac = 5662 where idbr = 5497;
update radnik set rukovodilac = 5662 where idbr = 5519;
update radnik set rukovodilac = 5662 where idbr = 5652;
update radnik set rukovodilac = 6789 where idbr = 5662;
update radnik set rukovodilac = 5662 where idbr = 5696;
update radnik set rukovodilac = 6789 where idbr = 5780;
```

```
update radnik set rukovodilac = 6789 where idbr = 5786;
update radnik set rukovodilac = 5842 where idbr = 5867;
update radnik set rukovodilac = 5662 where idbr = 5874;
update radnik set rukovodilac = 5786 where idbr = 5898;
update radnik set rukovodilac = 5780 where idbr = 5900;
update radnik set rukovodilac = 5842 where idbr = 5932;
update radnik set rukovodilac = 5786 where idbr = 5953;
update radnik set rukovodilac = 5867 where idbr = 6234;
update radnik set rukovodilac = 5842 where idbr = 6789;
update radnik set rukovodilac = 5867 where idbr = 7890;
update radnik set rukovodilac = 5867 where idbr = 7892;

update odeljenje set sefod = 5662 where brod = 10;
update odeljenje set sefod = 5780 where brod = 20;
update odeljenje set sefod = 5786 where brod = 30;
update odeljenje set sefod = 5842 where brod = 40;


insert into projekat
  values(100, 'uvoz', 3000000.00, to_date('05052004', 'MMDDYYYY'));
insert into projekat
  values(200, 'izvoz', 2000000.00, to_date('08222005', 'MMDDYYYY'));
insert into projekat
  values(300, 'plasman', 6000000.00, to_date('12022004','MMDDYYYY'));
insert into projekat
  values(400, 'projektovanje', 5000000.00,
   to_date('04142005', 'MMDDYYYY'));
insert into projekat
  values(500, 'izgradnja', 0.00, to_date('08222005', 'MMDDYYYY'));

insert into ucesce values(5497, 400, 2000, 'IZVRŠILAC');
insert into ucesce values(5652, 100, 1000, 'IZVRŠILAC');
insert into ucesce values(5652, 300, 1000, 'IZVRŠILAC');
insert into ucesce values(5662, 300, 2000, 'ŠEF');
insert into ucesce values(5696, 200, 2000, 'ŠEF');
insert into ucesce values(5696, 300, 2000, 'IZVRŠILAC');
insert into ucesce values(5780, 200, 2000, 'ORGANIZATOR');
insert into ucesce values(5786, 100, 2000, 'KONSULTANT');
insert into ucesce values(5842, 100, 2000, 'ŠEF');
insert into ucesce values(5867, 200, 2000, 'KONSULTANT');
insert into ucesce values(5898, 200, 2000, 'IZVRŠILAC');
insert into ucesce values(5900, 100, 2000, 'IZVRŠILAC');
insert into ucesce values(5932, 100, 500, 'KONSULTANT');
insert into ucesce values(5932, 200, 1000, 'ORGANIZATOR');
insert into ucesce values(5932, 300, 500, 'NADZORNIK');
insert into ucesce values(5953, 100, 1000, 'IZVRŠILAC');
insert into ucesce values(5953, 300, 1000, 'IZVRŠILAC');
insert into ucesce values(6234, 100, 500, 'NADZORNIK');
insert into ucesce values(6234, 200, 1200, 'IZVRŠILAC');
insert into ucesce values(6234, 300, 300, 'KONSULTANT');
insert into ucesce values(6789, 200, 2000, 'IZVRŠILAC');
insert into ucesce values(7890, 300, 2000, 'IZVRŠILAC');

CREATE INDEX ix_prezime ON RADNIK(prezime);
CREATE INDEX ix_ime ON RADNIK(ime);
CREATE INDEX ix_brod ON RADNIK(brod);
CREATE INDEX ix_rukovodilac ON RADNIK(rukovodilac);
CREATE INDEX ix_sefod ON ODELJENJE(sefod);

Commit;
```

LITERATURA

1. Alagić, S.: "*Relacione baze podataka*", Svjetlost, Sarajevo, 1984.
2. Radovan, M.: "*Projektiranje informacijskih sistema*", Informator, Zagreb, 1989.
3. Simić, R.: "*Organizacija podataka*", Naučna knjiga, Beograd, 1990.
4. Mišić, V.: "*Relaciona baza podataka Rdb/VMS*", Tehnička knjiga, Beograd, 1990.
5. Marjanović, Z.: "*ORACLE relacioni sistem za upravljanje bazom podataka*", Breza, Ljig, 1990.
6. Bobrowski, S.: "*Oracle 7 i obrada podataka po modelu klijent/server* ", Mikro knjiga, Beograd, 1995.
7. Vujnović, R.: "*SQL i relacijski model podataka*", Znak, Zagreb, 1995.
8. Wynkoop, S.: "*Vodič kroz SQL Server 7.0* ", CET, Beograd, 1999.
9. Grupa autora "*Access 2000 korak po korak* ", CET, Beograd, 1999.
10. Kovačević, A., Čukalevski, N.: "*Razvoj klijent-server sistema korišćenjem DBMS Oracle*", Viša elektrotehnička škola, Beograd, 2004.
11. Vieira, R.: "*SQL Server 2007*", CET, Beograd, 2008.
12. Kreines, D.: "*Oracle SQL: The Essential Reference*", O'Reilly, 2000.
13. Riordan M. R.: " *Projektovanje baza podataka*", Mikro knjiga, Beograd, 2010.
14. Дончев, А.: " *Основи на базите от данни* ", Университетско издателство "Васил Априлов", Габрово, 1999.
15. Džaković, M.: "*Oracle 7 SQL sa osnovama relacionog modela i SQL*PLUS okruženjem*", Tehnička knjiga, Beograd, 1997.
16. Дончев, А., Обрадович, С.: "*База от данни*", Технически университет – Габрово, ISBN – 954-683-223-5, Габрово, 2004.
17. A. Donchev, S. Obradović, "*Analitical determination of the horizontal partitions in a distributed database*", International scientific conference UNITECH'04, TU-Gabrovo, 18-19 November 2004, Gabrovo, proceedings, vol.I, pp. I 342-I 347, ISBN 954-683-303-7.
18. Slobodan Obradović *"Poslovno informatička obuka"*, Nacionalna služba za zapošljavanje, ISBN 86-902585-8-2, Beograd, 2004.
19. Anton Dončev, Slobodan Obradović, Svetlana Anđelić, *"METODOLOGIJA PROJEKTOVANJA BAZA PODATAKA - Algoritam za izbor primarnog ključa"*, Festival informatičkih dostignuća INFOFEST 2006, Budva
20. A. Donchev, S. Obradović, " *System Catalogue Selection in a Peer-to-Peer Database*", International scientific conference UNITECH'06, TU-Gabrovo, 24-25 November 2006, Gabrovo, proceedings, vol.I, pp. I 345-I 349, ISBN 10: 954-683-351-7, ISBN 13: 978-954-683-351-8.

21. S. Obradović, I. Vujičić, A. Žorić: *"Primena računara u poslovanju"*, S. Obradović, ISBN-86-904623-4-1, Beograd, 2006.

22. Dragan Simić, Srećko Ristić, Slobodan Obradović –*"Measurement of the Achieved Performance Levels of the Web Applications with Distributed Relational Database"*, Facta Universitatis Ser : Electonics &. Energetics Vol. 20, no 1, April 2007, pp 31-43, ISSN 0353-3670, COBISS.SR-ID 12826626

23. A. Donchev, S. Obradović, F. Abdiwi, "Cost Measurement of a Distributed Database System with Normalized Workload", International scientific conference UNITECH'07, TU-Gabrovo, 23-24 November 2007, Gabrovo, proceedings, vol.I, pp. I 478-I 482, ISSN 1313-230X.

24. Svetlana Andjelic, Slobodan Obradovic, Branislav Gacesa, "AN PERFORMANCE ANALYSIS OF THE DBMS - MySQL vs PostgreeSQL" -mr Communications-Scientific Letters of the University Zilina, Slovakia, pages 53-59, ISSN 1335-42-05 Art No 4/2008 Networks for new generations

25. Obradović, S., Pandurov, T., Vučinić, B.: "*SQL Strukturirani upitni jezik*", Visoka škola elektrotehnike i računarstva, ISBN – 978-86-85081-17-3, Beograd, 2008.

26. S. Obradović, S. Slavnić, A. Donchev, Comparison of the free DBMS alternatives for MS SQL, International scientific conference UNITECH'08, TU-Gabrovo, 21-22 November 2008, Gabrovo, proceedings, vol.1, pp.I 455-I 460, ISSN:1313-230X.

27. Kaluđerčić, P., Obradović, S.. "*Projektovanje informacionih sistema - Relacione baza podataka*", Visoka škola elektrotehnike i računarstva, Beograd, 2008.

28. S. Obradović, D. Tesić, S. Ilić, V. Marković, "Responsibility of management related to data security", International scientific conference UNITECH'09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 337-I 342, ISSN:1313-230X.

29. S. Ilić, S. Obradović, "Efficient Validation of Exceeded Limit Values in Batch Data Loading by Database Trigger", International scientific conference UNITECH'09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 393-I 398, ISSN:1313-230X.

30. Obradović, S., Pandurov, T., Vučinić, B.: "*MS-Access Projektovanje baza podataka i aplikacija*", Visoka škola elektrotehnike i računarstva, ISBN – 978-86-85081-99-6, Beograd, 2009.

31. Obradović, S., Barbarić, Ž.: " Informatika i informacione tehnologije", Visoka škola elektrotehnike i računarstva, Beograd, 2009.

32. S. Ilić, S. Obradović, "Comparison of Methods for Validation of Exceeded Limit Values in Batch Data Loading", 10th Anniversary International Scientific Conference Unitech'10, Technical University Of Gabrovo, Bulgaria, 19-20 November 2010, Pp. I-333-I-338, Issn 1313-230x

33. S. Obradović, D. Tesić, M. Mijalković, V. Marković, "The influence of the type of processing on computer crime protection", International scientific conference UNITECH'09, TU-Gabrovo, 20-21 November 2009, Gabrovo, proceedings, vol.1, pp.I 343-I 350, ISSN:1313-230X.

34. S. Urman, R.Hardmen, M. MekLaflin.: " Baza podataka Oracle 10g PL/SQL programiranje", Kompjuter biblioteka, 2005.

35. S. Dillon, C. Beck, T.Kyte; "Od početka... Programiranje u Oraclu", CET 2002

36. J. Price : "Oracle Database 10g SQL ', McGraw-Hill/Osborne, 2004
37. D. Sundjeric, T. Woodhead: "SQL Server 2000 Stored Procedure Programming", Osborne/McGraw-Hill 2001
38. R. Dewson: "Beginning SQL Server 2008 for Developers", APRESS 2008
39. J. Sack : 'SQL Server 2008 Transact-SQL Recipes", APRESS, 2008
40. S. Sehring : "MySQL Bible", Wiley Publishing, 2002
41. C.A.Bell : "Expert MySQL", APRESS, 2007
42. S. Feuerstein, G. Harrison, : "MySQL Stored Procedure Programming", O'Reilly, 2006