Slobodan Obradović Biljana Bojičić Tijana Pandurov

SQL

Struktuirani upitni jezik

Visoka škola elektrotehnike i računarstva strukovnih studija, Beograd

Beograd, 2012

Autori:	Slobodan Obradović, Biljana Bojičić, Tijana Pandurov
Recenzenti:	Dragoslav Perić, Branislav Pavić
Tehnička obrada:	Slobodan Obradović, Biljana Bojičić, Tijana Pandurov
Izdavač:	Visoka škola elektrotehnike i računarstva strukovnih studija, Beograd, Vojvode Stepe 283
Korice:	Gabrijela Dimić, Kristijan Kuk
Tiraž:	200 primeraka
Lektor:	Milena Dorić
Štampa:	MST Gajić

ISBN 978-86-7982-144-7
CIP – Katalogizacija u publikaciji
Narodna biblioteka Srbije, Beograd

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 /
Slobodan Obradović, Biljana Bojičić, Tijana
Pandurov. - Beograd : Visoka škola
elektrotehnike i računarstva strukovnih
studija, 2012 (Beograd : MST Gajić). - 122
str. : ilustr. ; 24 cm

Tiraž 50. - Bibliografija: str. 121.

ISBN 978-86-7982-144-7
1. Бојчић, Биљана, 1973- [аутор] 2.
Пандуров, Тијана, 1976- [аутор]
a) Релационе базе података - Програмирање
b) Програмски језик "SQL"
COBISS.SR-ID 195214860

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, struktuirani 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 SQLa 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 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.

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.

Autori se zahvaljuju Bojanu Vujoviću na korisnim sugestijam i otklanjanju uočenih grešaka.

U Beogradu, decembar 2012. 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**).

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'*)

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. Oni se zasnivaju na različitim

principima: **QBE** se zasniva na relacionom računu domena, **Quel** se zasniva na relacionom računu (ne podržava relacionu algebru), a **SQL** se zasniva na relacionom računu n-torki tj. na kombinaciji relacione algebre i relacionog računa.

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.** 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 Standardisation Organisation*) 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 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 klauzula 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 (- -).

Pri imenovanju objekata treba imati u vidu da svi SUBP (Sistem za upravljanje bazama podataka, DBMS - Database Management System) 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).

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 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:

- **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:

- **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:

- **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. Pomoću ovih naredbi opisuje se šema podataka. Osnovna karakteristika tih naredbi jeste da se njihovo izvršenje ne može otkazati komandom **ROLBACK**. 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 omogućavaju definisanje različitih tipova podataka, 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 specijalnih 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.

Domen podataka

Navođenjem tipa podatka automatski se ograničava skup dozvoljenih vrednosti, odnosno određuje se domen. Domen nekog svojstva predstavlja skup mogućih (dozvoljenih) vrednosti koje to svojstvo 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.

Objekti baze podataka

Baza podataka je logički integrisani skup podataka sa centralizovanim upravljanjem. Danas su dominantne relacione baze podataka koje predstavljaju skup vremenski promenljivih tabela.

Tabela se definiše kao niz naziva kolona sa nula ili više redova (vrsta, n-torki, slogova, zapisa) kojima su zadate vrednosti kolona. U jednoj bazi ime tabele je jedinstveno (ne mogu postojati dve tabele sa istim imenom).

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.

Da bi redovi bili jedinstveni, svaka tabela mora da ima ključ. 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*). Ako ne postoji ni jedan podskup ovog skupa koja ima predhodno svojstvo, onda je to **primarni ključ** (*minimalnost*).

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). Samim tim, primarni ključ, ili bilo koji njegov deo, ne mogu imati vrednost **NULL**.

Spoljni ključ je atribut u nekoj tabeli čiji je skup vrednosti podskup skupa vrednosti primarnog ključa neke druge tabele. 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**.

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.

*Tabelu **RADNIK** kreiramo naredbom:*

```
CREATE TABLE RADNIK
        (idbr# INTEGER NOT NULL,
        ime CHAR(25) NOT NULL,
        prezime CHAR(25) NOT NULL,
        posao CHAR(10),
        kvalif CHAR(3),
        rukovodilac$ INTEGER,
        datzap DATE,
        premija FLOAT(1),
        plata FLOAT(1),
        brod$ SMALLINT);
```

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**:

```
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.

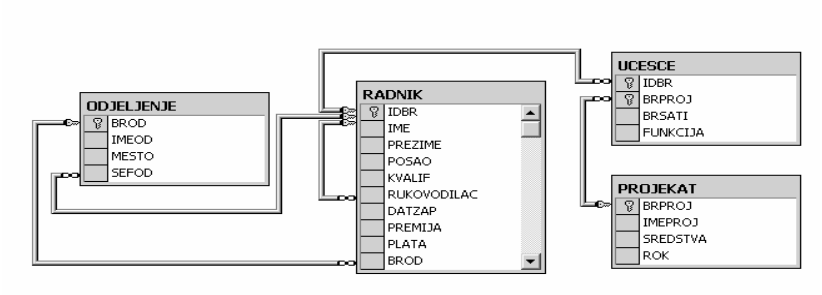
I veze tipa 1:1 mogu se ostvariti preko stranog ključa. Ostvaruju se tako što se u tabeli ODELJENJE uvodi kao atribut (spoljni ključ) primarni ključ u tabeli RADNIK. U ovom slučaju primarni ključ je preimenovan i zove se *šefod,* kako bi njegova namena bila očiglednija. Atribut *šefod* može imati samo 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). Ona određuju kako se operacije ažuriranja jedne tabele prenose na druge tabele, koje su u nekoj relaciji-vezi sa tabelom koja se ažurira.

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 **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.

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.

Primer 1. Kreirati tabelu RADNIK.[1] 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

[1] 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 **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	

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.

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.

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 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):

```
ALTER TABLE ODELJENJE
ADD ( brzap INTEGER);
```

Primer 6. Dodati atribut brod u tabelu RADNIK.

```
ALTER TABLE RADNIK
ADD brod SMALLINT;
```

Primer 7. Promeniti tip polja brsati u tabeli UCESCE na **SMALLINT**.

```
ALTER TABLE UCESCE
ALTER COLUMN 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.

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

Napomena: Funkcija koja vraća trenutni datum u MS SQL Servru je **GETDATE**().

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**. Izbacivanje definicije tabele iz baze podataka vrši se naredbom čiji je opšti oblik:

DROP TABLE ime_tabele

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. 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.

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.

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.

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, datrođ, adresa, .. >

a jedan njen deo se vidi u sledećoj tabeli.

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

INDGRAD

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

Tabela: 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 “češlja“ 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**.**

Naravno, 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.

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 klauzulama **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 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 klauzuli **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**).

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 **SELECT,** glasi:

SELECT [ALL DISTINCT] lista atributa 1
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**, **ENI**, **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 sa

- (**WHERE, HAVING**) definišu uslove koje podaci treba da zadovoljavaju pri izdvajanju, što odgovara **operatoru restrikcije** (**selekcije**). Na kraju, mogu se postaviti i zahtevi kojima se rezultati
- grupišu (**GROUP BY**), ili
- na neki način sređuju (**ORDER BY**).

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

Odredbe, klauzule **SELECT** i **FROM** su obavezne, a ako se ne postavi nikakav uslov za selekciju ili uređivanje, druge klauzule (**WHERE**, **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,** a ne mora biti isti kao redosled atributa dat u definiciji tabele),
- sprečiti pojavu višestrukih n-torki.

Značenje opcije **DISTINCT** je sledeće:

- ako ova klauzula 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 (rezultat je 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.

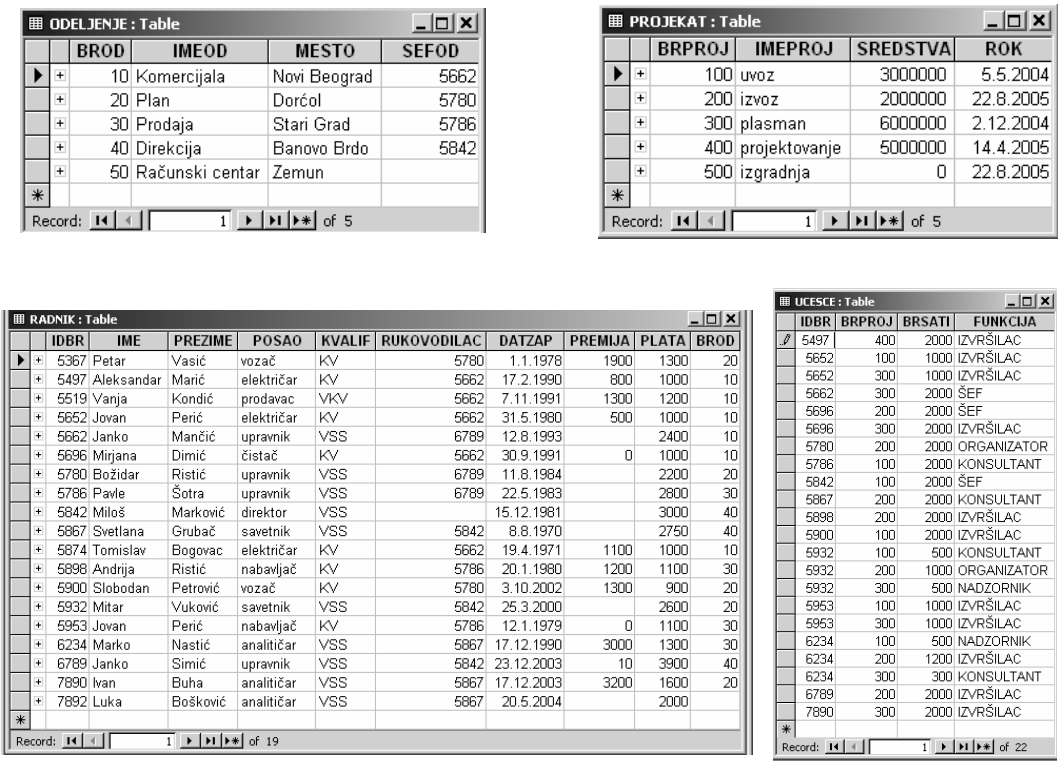
Na kraju, napomenimo da SQL ne “razume” znak # i $ u imenima atributa kao oznake za ključne atribute. To su interne oznake za prepoznavanje ključnih atributa, i ti znaci u primerima u ovoj knjizi upotrebljeni su samo zato da bi se ključni atribut razlikovao od drugih, i time primeri bili pregledniji.

Sve operacije i klauzule 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 2 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, 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.



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

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 2 - 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 klauzuli, 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 koja se žele u rezultatu:

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

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. Ovaj zadatak urađen, uzimajući u obzir i to pravilo, bio bi:

```
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 tabeli:

idbr	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

Ovaj zadatak urađen na duži način, dodajući ime tabele pre imena atributa koji treba da se prikaže, glasio bi:

```
SELECT RADNIK.idbr, RADANIK.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.

Primer 25. Prikazati broj, 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 (npr. 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 tabela:

imeod
Komercijala
Plan
Prodaja
Direkcija
Računski centar

Primer 28. Prikaži nazive svih poslova u preduzeću.

```
SELECT posao
FROM RADNIK;
```

Odgovar 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
uprvnik
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
uprvnik
vozač

Isti efekat se kod nekih SUBP postiže klauzulom jedinstven (**UNIQUE**), 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 (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 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
NULL
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 klauzulu **DISTINCT** jer u jednom odeljenju radi više radnika pa bi u odgovoru bilo ponovljeno svako odeljenje više puta. Broj odeljenja prikazuje 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.
- u primeru b) ne moramo koristiti klauzulu **DISTINCT** jer je šifra odeljenja primarni ključ relacije i samim tim je jedinstven.
- u primeru a) moramo iza klauzule **AS** (kao) koristiti znake navoda ili uglaste zagrade [ ] (u Accessu) jer novo ime je sastavljeno od više reči, a u primeru b) ne moramo.

Napomena: U nekim sistemima za upravljanje bazama podataka (SUBP) nije potrebno navoditi klauzulu **AS,** već se samo stavi razmak, tj. prazno mesto (space), to je ekvivalentno klauzuli **AS** (ovo nije slučaj u Accessu). 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 - klauzula 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 klauzulu **WHERE** (*gde je*), i ona nam 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 sledeća je:

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

a argumenti mogu biti:

- jedan atribut,
- skup atributa (u zagradama odvojeni zapetom) 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 klauzule:

- **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 (ili ne nalazi - **NOT**) između vrednosti **argument2** i **argument3** uključujući i granice toga intervala.

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 (**TRUE**) ako relacioni izraz daje barem jednu n-torku kao rezultat, u suprotnom je neistinit (**FALSE**).

Pored ove, pomenute tri opcije šire verzije SQL-a imaju još neke mogućnosti, kao na primer:

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

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

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 klauzulu **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 5 775.

```
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 kao sam nije primarni ključ.

Primer 39. Prikazati funkciju zaposlenog čiji je identifikacioni broj 5 652 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 3000000, 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 2 000.

Primer 45. Prikazati imena i plate zaposlenih čija je plata 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 i to kao: "!= " ili "<>".

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, MySQLu, jer se znak nejednakosti može pisati kao "<>". 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, MySQLu i Oracleu.

```
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 prisutno u MS Accessu, MySQLu, MS SQL Serveru.

```
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

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. Tako iako je bila namera da se prikažu 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';
```

Primer 49. Prikazati ime, posao i kvalifikaciju zaposlenih koji su zaposleni 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# ;
```

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';
```

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

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

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

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

Primer 52. Prikaži ime, posao i platu zaposlenih u odeljenju 30, čija je plata veća od 2 000.

```
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**, a 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.

```
ELECT 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. Prikazati 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 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, a svaki od njih je ispravan.

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

Bez korišćenja zagrada zadatak bi se mogao uraditi na sledeći način, što nije tačno!

```
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 na upit, s obzirom na to da uslovi nisu interpretirani na odgovarajući način.

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 2 600 do 3 000.

```
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 1 000 i 2 000 (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 1 000 i 2 000, ne može se koristi operator **BETWEEN**.

Neki SUBP, na primer Oracle, imaju na raspolaganju i klauzule **ANY** (**SOME**), **ALL** koje se mogu koristiti u **WHERE** klauzuli. Kod drugih SUBP, na primer MS Access, MS SQL Server, ove klauzule se mogu koristiti samo u ugnježdenim upitima. U MySQLu **ANY**( ), **SOME**( ), i **EVERY**( ) grupne su funkcije. Upotreba ovih klauzula 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 {1 000, 1 500, 2 000}.

```
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 {1 000, 1 500, 2 000}.

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

Uređivanje izveštaja po vrednosti izabranih atributa - klauzula 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. Klauzula **ORDER BY** određuje eksplicitno redosled n-torke u rezultatu upita po nekom kriterijumu (po abecedi, po veličini itd.) u rastućem ili opadajućem poretku. Klauzula **ORDER BY** je uvek poslednja klauzula u **SELECT** bloku, jer najpre se selektuju n-torke, a na kraju se uređuju. I ova opcija može imati više atributa po kojima se vršiređanje.

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 klauzulu **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.

Ovaj primer smo mogli uraditi i ovako:

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

Razlika u odnosu da 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.

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 tada bi bio:

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

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

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;
```

ime	kvalif	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

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	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

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 vrednost **NULL** grupišu zajedno.

Rad sa tekstualnim podacima - operator LIKE

Za rad sa tekstualnim podacima vrlo često se koristi klauzula LIKE (liči na, kao, poput). Upotreba ove klauzule 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 Oraclu, MySQL i MS SQL Serveru proizvoljan niz znakova zamenjuje se džoker znakom **%** (a 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 6 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';

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

brproj	imeproj
100	uvoz
200	izvoz

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%';

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

ime
Petar
Svetlana
Tomislav
Mitar

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.

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

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

ime
Miloš

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, kao što se i vidi, 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 rastećum redosledu.

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

prezime	kvalif
Perić	KV
Petrović	KV
Perić	KV

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

prezime	kvalif
Perić	KV
Perić	KV
Petrović	KV

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.

1. 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*>).
2. 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).
3. Kada nije nastupio momenat delovanja nekog atributa (plata ili premija za mesec mart biće poznati su tek tokom aprila).

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 (**PRIMATY 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 istinosna 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**.

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;
```

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)

b) nemaju premiju.

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

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** služi da se u upitu sistematski 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. Npr. neutralna vrednost za sabiranje je broj 0. U

raznim SUBP u tu svrhu koriste se različite funkcije, u MS Accessu to je funkcija **NZ** (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 izgledao bi:

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

Zadatak urađen u MS Accessu izgledao bi:

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

Zadatak urađen u MS SQL Serveru izgledao bi:

SELECT idbr, ime, brod, plata+ **ISNULL**(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[, broj]). 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).

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 klauzuli **ORDER BY** mora se koristiti ceo izraz po kome se vrši uređivanje (kao u prethodnom slučaju) ili redni broj izraza u naredbi **SELECT** (kao u narednom slučaju), a ne može se koristiti natpis 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;
```

Klauzula GROUP BY

Klauzula **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.

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 kluzula GROUP BY izrazi u klauzuli SELECT mogu da budu samo oblika:

- konstante,
- grupne funkcije,
- bilo koji od izraza iz klauzule GRUOP 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, a prava uloga klauzule **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 (po odeljenjima).

Napomena: Atributi po kojima se vrši grupisanje moraju biti navedeni i u **SELECT** naredbi, a u nekim RDBMS, kao recimo u Accessu, MS SQL Serveru 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.

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**, mogu se 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 Accesu.

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. Sve ove funkcije navode se u klauzuli **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 mesta **spos,**
- **INSTR**(string, sstring[, spos]) – traži podstrins **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 (konketenacija) koristi se operator ||, pa bi isti zadatak urađen Oracleu 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 projektovanja 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 vrednsoti, 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	uprvnik	3
Ivan	analitičar	1
Luka	analitičar	1

Funkcije za rad sa datumima

Datumi su izuzetno važni za poslovne apliakcije, 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 mjesecu 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 kar. po modelu u datum.

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

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

ime	godina
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: Radni staž je podatak koji je suštinski važan za poslovanje firme jer se na bazi njega, na primer, izračunava plata zaposlenih. Ali taj podatak je stalno vremenski 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). Grupa može da se formira pomoću klauzule **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 klauzule **GROUP BY**. Grupe se mogu 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.

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 Oracleu i SQL Serveru

Funkcija	*Oracle*	*SQL Server*
Average	**AVG**	**AVG**
Count	**COUNT**	**COUNT**
Maximum	**MAX**	**MAX**
Minimum	**MIN**	**MIN**
Standard deviation	**STDDEV**	**N/A**
Summation	**SUM**	**SUM**
Variance	**VARIANCE**	**N/A**

Agregatne funkcije u MY SQL-u

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

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

COUNT(*izraz*)-vraća broj ne-null vrednosti u vrstama koje vraća **SELECT** lista.

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

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

GROUP_CONCAT(*izraz*)-vraća string konketeniranih 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 Oracleom.

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 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** klauzuli, niti je izvršeno grupisanje ponekom 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 Accesu.

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 najveca,
            ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK;
```

najmanja	najveća	srednja	broj zaposlenih
900	3900	1797,37	19

Upotreba WHERE klauzule u grupnim funkcijama

Klauzula **WHERE** se upotrebljava u grupnim funkcijama kada proračune vršimo samo nad n-torkama koji zadovoljavaju dati uslov. Redosled operacija u ovakvim upitima je sledeći: najpre se izdvoje n-torke koji zadovoljavaju dati uslov, 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 upotrebom klauzule 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 klauzuli **SELECT**, bez obzira na to da li želimo da budu prikazani u rezultatu ili ne (vidljivi ili nevidljivi).

Grupne funkcije izvršavaju proračun nad setom vrednosti i vraćaju jednu vrednost. Sa izuzetkom **COUNT**(*) agregatne funkcije, sve druge 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** klauzulom. 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** klauzuli.

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 klauzuli, 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** klauzule. 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 klauzuli WHERE menjali uslov, odnosno postojalo bi više identičnih upita sa različitim uslovima. To se može izbeći upravo korišćenjem GROUP BY klauzule umesto klauzule 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** klauzula 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 klauzule 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	uprvnik	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	uprvnik	1

Grupne funkcije i upotreba klauzule HAVING (uslovi za grupe)

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

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

idbr	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 projekata.

```
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 /).

Klauzule **GROUP BY** i **WHERE** mogu se koristiti zajedno, pri tome ***GROUP BY mora uvek biti iza WHERE klauzule***, jer najpre treba izvršiti selekciju (smanjiti broj n-torki), zatim se one grupišu sa GROUP BY, a onda se dodatno izabiraju grupe klauzulom **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.

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, jer godina ima 12 meseci.

Upotreba NVL funkcije

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 **NVL (atrib, vrednost)**. Tako se vrednost **0** u koloni **premija** dodeljuje zaposlenima koji nemaju premiju funkcijom **NVL** (*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**. Da bi se upiti spajali ovim operatorima, liste atrubuta ili izraza u klauzulama 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.

Primer 117. Prikazati ime, broj odeljenja i platu za zaposlene koji rade u odeljenju 20 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;
```

Korištenjem opratora **UNION ALL** dobijaju se duplirani zapisi za one slučajeve gde su ispunjena oba uslova, tako 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 dobli zapis o radniku Slobodanu dva puta jer on zadovoljava oba postavljena uslova, da je iz odeljenja 20 i da mu je plata manja od 1 200.

Napomena: Neki SUBP ne podržavaju svaku od ovih funkcija. Tako na primer SQL Server, MS Access imaju samo **UNION**, a **INTERSECT** i **MINUS** ne podržavaju. U Oracleu možemo napraviti primere za ove operacije. 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, subquery) je ugrađen u neki drugi upit. Rezultat podupita se koristi kao konkretna vrednost (ako se radi o upitu koji vreća jedan slog, single row subquery). Ako podupit vraća skup vrednosti, mora se koristiti klauzula IN. Podupiti se mogu navesti i u klauzuli **WHERE** i u klauzuli **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 klauzule **FROM**. Tada tabela koja predstavlja rezultat podupita određuje virtuelnu tabelu iz koje se izdvajaju sn-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** klauzulu drugog upita. Prvi, unutrašnji upit, trebao bi 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 klauzele 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	uprvnik	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 priakzani 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 Accessu).

```
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 Accessu:

```
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 odeljneju 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 ugnježdenim upitima mogu se koristiti funkcije i grupisanje po raznim kriterijumima.

Primer 135. Prikazati imena odeljenja u kojima su ukupna primanja svih radnika u odeljenju veća od 10 000.

```
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 primanje 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)
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	uprvnik	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.

```
SELECT ime, brod, posao, plata
FROM RADNIK
WHERE  brod  IN  ( SELECT brod, MAX(plata)
                   FROM RADNIK
                   GROUP BY brod );
```

ime	brod	posao	plata
Janko	10	upravnik	2400
Pavle	30	upravnik	2800
Mitar	20	savetnik	2600
Janko	40	upravnik	3900
Luka	NULL	analitičar	2000

Primer urađen u SUBP Oracle:

```
SELECT ime, brod, posao, plata
FROM RADNIK
WHERE  (brod, plata)   IN  ( SELECT brod, MAX(plata)
                             FROM RADNIK
                             GROUP BY brod );
```

Primer 143. Prikaži imena radnika koji su se poslednji zaposlili u svakom odeljenju.

```
SELECT ime, datzap AS [Datum zaposlenja], brod
FROM RADNIK
WHERE  datzap IN (SELECT MAX(datzap)
                  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

Primer urađen u SUBP Oracle:

```
SELECT ime, datzap, brod
FROM RADNIK
WHERE  (datzap, brod) IN
                  (SELECT MAX(datzap), brod
                  FROM RADNIK
                  GROUP BY brod);
```

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** klauzulu 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 tabela

Spajanje tabela se koristi kada se informacije koje želimo da prikažemo (lista atributa u **SELECT** klauzuli) 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. Kako JOIN spaja informacije iz više tabela u jednu, rezultat zavisi od toga kako zahtevamo da se spajaju podaci, pa, shodno tome, 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 (10X10). 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 i ime table 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 postoji poklapanje vrednosti za polje (ili polja) preko kojeg 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, najpre, šta će se desiti ako ne povežemo tabele:

```
SELECT RADNIK.ime, RADNIK.posao, ODELJENJE.brod, ODELJENJE.imeod,
       ODELJENJE.mesto
FROM ODELJENJE, RADNIK;
```

Napomena: U slučaju 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	uprvnik	40	Direkcija	Banovo Brdo
Ivan	analitičar	20	Plan	Dorćol

Napomena: Spajanjem tabela RADNIK i ODELJENJE preko atributa brod u ovom upiti dobili smo traženi rezultat. Kao rezultat dobijeno je 18 zapisa, iako imamo 19 zaposlenih u preduzeću. Podaci o radniku šifre 7 892 (Luka Bošković) nisu prikazani jer on još nije raspoređen u neko od odeljenja.

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.

Alternativa za INNER JOIN (spajanje pomoću WHERE klauzule)

Pored upotrebe **INNER JOIN**-a tabele se mogu spojiti pomoću klauzule **WHERE**.

Primer 146. Uradimo sada Primer 144 upotrebivši klauzulu **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=ODALJENJE.brod;
```

Primer 147. Uradimo sada Primer 145 upotrebivši klauzulu **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 klauzuli **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 1000 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 klauzule 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 postpak 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 klauzule **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, aribut **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 atrbuta, to jest: RADNIK.idbr ili UCESCE.idbr.

Primer 151. Prikazati ime radnika i mesto u kome rade za sve radnike čija je plata između 1 500 i 2 500 (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 skraćenica (alias) umesto imena tabela

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 (alijasa) 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** klauzuli navede se puno ime tabele i dodeli se skraćenica za tabelu koja se koristi u tom upitu.

Netačni su sledeći primeri:

I)
```
SELECT ime, posao, brod, imeod, mesto
FROM ODELJENJE AS O , RADNIK AS R
WHERE R.brod=O.brod;
```

U ovom slučaju nisu korišćeni alijasi ispred svih imena atributa u **SELECT** klauzuli.

II)
```
SELECT R.ime, R.posao, O.brod, O.imeod, O.mesto
FROM ODELJENJE AS O , RADNIK AS R
WHERE brod=O.brod;
```

U ovom slučaju nisu korišćeni alijasi ispred imena atributa u **WHERE** klauzuli.

```
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 alijasi ispred nekih imena atributa u **SELECT** klauzuli. 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 R.ime, R.posao, P.brproj, P.imeproj, P.sredstva, P.rok
FROM RADNIK R INNER JOIN UCESCE  U ON R.idbr=U.idbr
              INNER JOIN PROJEKAT P ON P.brproj=U.brproj;
```

Napomena: U ovom primeru nije korišćeno AS, već je samo stavljen znak razmaka (space) iza imena tabele, a zatim navedena skraćenica za tabelu.

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** klauzule. 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**), ili levo spajanje (**LEFT 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 tzv. **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 158. 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 Accessu:

```
SELECT *
FROM ODELJENJE RIGHT JOIN RADNIK ON ODELJENJE.brod = RADNIK.brod
WHERE RADNIK.premija>2000;
```

Primer 159. 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 160. 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;
```

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

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 alijasa:

```
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;
```

Primer 161. 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	uprvnik
Računski centar	NULL	NULL

Primer 162. 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 163. 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 164. 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 165. 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 166. 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 167. 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 168. 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 169. 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 170. 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 171. 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 172. 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 173. Prikazati imena radnika koji imaju istu kvalifikaciju kao Mitar.

a) pomoću unutrašnjag 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 174. 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 175. 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 176. 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. Dakl,e **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.

Primer 177. 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 178. Izlistaj šifru i ime odeljenja, kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odeljenju.

Primer urađen u MS Accessu:

```
SELECT ODELJENJE.brod, ODELJENJE.imeod AS [ime odeljenja],
       Max(RADNIK.plata) AS [najveća plata], First(RADNIK.idbr) AS [šifra radnika],
       First(RADNIK.ime) AS [ime radnika], First(RADNIK.posao) AS posao
FROM ODELJENJE INNER JOIN RADNIK ON ODELJENJE.brod = RADNIK.brod
GROUP BY ODELJENJE.brod, ODELJENJE.imeod;
```

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.

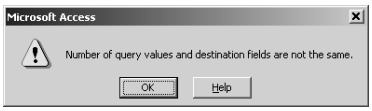
Postoje tri tipa ovih naredbi:

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 179. Dodati podatke o novom projektu broj 600, imena Obrazovanje za koji se izdvajaju sredstva od 2 000 000.[1]

INSERT INTO PROJEKAT
VALUES (*600, 'Obrazovanje', 2000000*);

Odgovor sistema je:

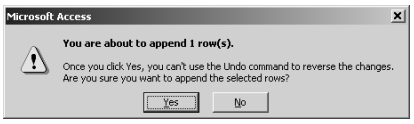


Odgovor je očekivan zbog toga što nismo naveli vrednosti za sve atribute i nedostaje podataka za atribut rok. Dodati da je rok za ovaj projekat 22.10.2005.

[1] 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.

INSERT INTO PROJEKAT
VALUES (*600, 'Obrazovanje', 2000000, '22.10.2005'*);

Odgovor sistema je:



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, analogno 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 180. 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 odradio 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. To je problem 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, to nije 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 181. 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 182. 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 Inkrement”, pa mu je potrebno dodeliti neku vrednost, koja već ne postoji u tabeli. Naravno, da vrednost za atribut MESTO nije neophodno uneti.

Primer 183. 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 3 000, platu 2 000 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 184. 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.

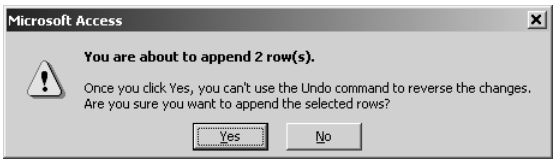
INSERT INTO RADNIK (ime, idbr, kvalif, datzap)
VALUES *('Mirko', 7891, 'VSS', '29-jun-02');*

Napomena: Ovaj upit ne radi, javlja grešku, uzrok može biti taj što se vrednost atributa, koja je primarni ključ, ponavlja, zatim ponavlja se neka vrednost atributa koja treba da ima jedinstvenu vrednost, ili ne unosi se neka vrednost atributa koja ne sme imati vrednost NULL (na primer prezime).

Primer 185. 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 186. 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 187. Kreirati tabelu ZAPOSLENI sa svim istim atributima kao i tabela 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.

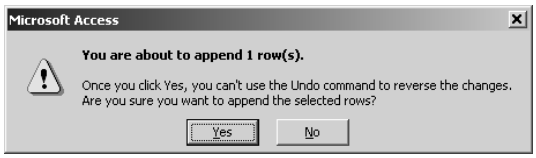
Primer 188. Kreirati tabelu KVALIF_VKV sa atributima IDBR, IME, BROD i u nju smestiti sve zaposlene koji imaju kvalifikaciju VKV.

```
SELECT idbr, ime, brod INTO KVALIF_VKV
FROM RADNIK
WHERE kvalif='VKV';
```

Primer 189. Dodati zaposlenog čije je IDBR 7890 na projekat Uvoz.

```
INSERT INTO UCESCE
SELECT RADNIK.idbr, PROJEKAT.brproj
FROM RADNIK, PROJEKAT
WHERE RADNIK.idbr=7890 AND PROJEKAT.imeproj='Uvoz';
```

Odgovor sistema je:



Primer 190. 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 191. 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 odeljenja 30.

Primer 192. 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 193. Izbrisati sve podatke iz tabele KVALIF_VKV.

I) DELETE * FROM KVALIF_VKV;

II) DELETE KVALIF_VKV.* **FROM** KVALIF_VKV;

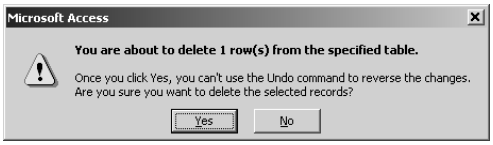
III) DELETE FROM KVALIF_VKV;

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 194. U relaciji RADNIK obriši sve podatke o radniku **idbr** = *5 953* koji je otišao u penziju.

DELETE FROM RADNIK
WHERE idbr = *5953***;**

Odgovor sistema je:



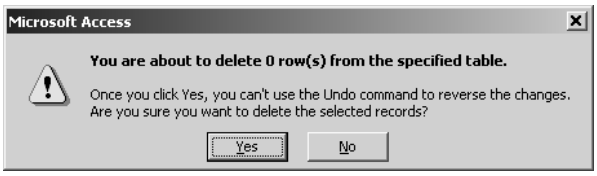
Primer 195. Izbrisati podatke o radnicima koji su se zaposlili pre 17-feb-90.

```
DELETE FROM RADNIK
WHERE RADNIK.datzap < '17-feb-90 ';
```

Primer 196. 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:

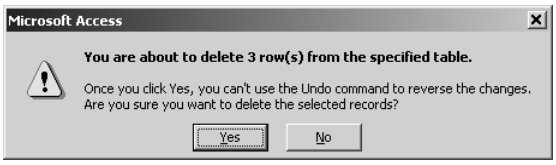


U ovom primeru neće biti izbrisan ni jedan zapis jer ne postoji odeljenje imena *priprema*.

Primer 197. Zaposlene čiji 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'));
```

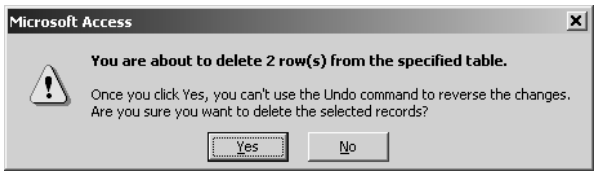
Odgovor sistema je:



Primer 198. 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 199. Obrisati sa svih projekata radnike čija je premija 0.

```
DELETE *
FROM UCESCE
WHERE idbr IN (SELECT idbr
               FROM RADNIK
               WHERE premija=0);
```

Menjanje postojećih podataka

Modifikacija postojećih podataka (ažuriranje u užem smislu) izvodi se komandom **UPDATE - SET**. Opšti oblik naredbe glasi:

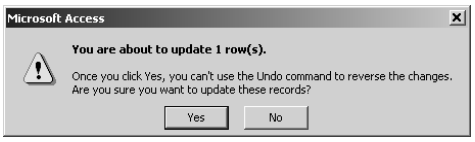
```
UPDATE ime relacije
SET atribut1=vrednost1[, atribut2=vrednost2, ...]
WHERE  [ lista uslova ]
```

tako da se ovom naredbom može menjati i vrednost samo jednog podatka unutar jedne n-torke.

Primer 200. Zaposlenom čija je šifra **idbr** = ***5932*** dodeliti rukovodioca čija je šifra *5842.*

```
UPDATE RADNIK SET rukovodilac= 5842
WHERE  idbr = 5932;
```

Odgovor sistema je:

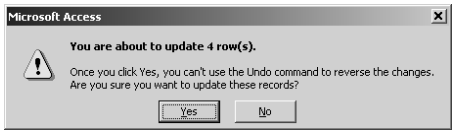


Ovom naredbom može se menjati i vrednost jednog atributa unutar više n-torki.

Primer 201. Prebaci sve zaposlene iz odeljenja 30 u odeljenje 20, a naredba glasi:

```
UPDATE RADNIK SET brod= 20
WHERE  brod = 30;
```

Odgovor sistema je:



Primer 202. 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.

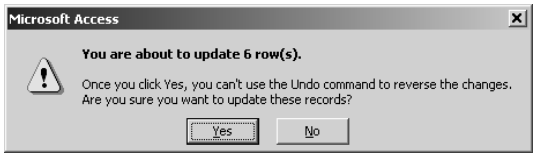
Primer 203. Odeljenje Plan premestiti na Voždovac.

```
UPDATE ODELJENJE SET MESTO='Voždovac'
WHERE IMEOD='Plan';
```

Primer 204. 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:



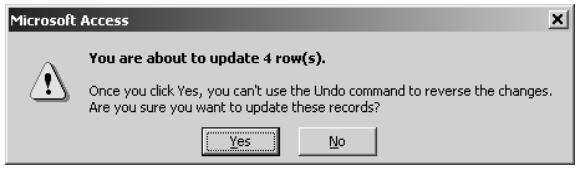
Primer 205. Zaposlenima u odeljenju smeštenom na Dorćolu dodeliti premiju u iznosu 40% od plate, ne uzimajući u obzir direktore i upravnike.

```
UPDATE RADNIK
SET premija=plata*0.4
WHERE brod IN (SELECT brod
                FROM ODELJENJE
                WHERE mesto='Dorćol')
                AND posao NOT IN ('direktor', 'upravnik');
```

Primer 206. 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 tu svrhu ima takozvane akcione upite pomoću kojih se mogu ne samo ažurirati podaci već i kreirati pogledi i tabele.

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).

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 u bazi, 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 tim 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.

Kako ažuriranje (dakle izmena podataka) ovakve relacije najčešće nije moguće, posebno ako je pogled kreiran nad više tabela (što je i logično), 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**, a navedeni atributi njeni atributi. Naravno, 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 o onima iz njegovog odeljenja. U tu svrhu, iz relacije ***RADNIK*** **<** ***idbr#, ime,*** **...** > treba izdvojiti samo one zaposlene koji su od interesa.

Primer 207. 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 šef hoće da obrađuje podatke o zaposlenima u svom odeljenju on jednostavno koristi ovaj pogled.

Primer 208. 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
                ODELJENJE.brod = RADNIK.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 209. Kreirati pogled IZVOZ koje 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**. Ova komanda uklanja pogled i iz relacione šeme i iz rečnika podataka baze podataka.

Zamena za CREATE VIEW u ACCESSu

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. Pogledajte sledeći primer:

Primer 210.

```
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 primera 208 a) koji se oslanjao na pogled kreiran u primeru 207. Pošto nemamo mogućnost da kreiramo pogled i da se na njega jednostavno pozovemo navođenjem njegovog imena u klauzuli FROM, ovde smo kôd pogleda prepisali u "glavni" upit i tako faktički 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 nekoliko slučajeva, pa ga zato treba imati u vidu.

Primer 211. Izračunati koliko u preduzeću ima različitih kvalifikacija.

```
SELECT Count(*) AS [Broj kvalifikacija]
FROM (SELECT DISTINCT kvalif
      FROM RADNIK)
```

Obzirom 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 212. Izlistaj šifru odeljenja, kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odeljenju.

```
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
```

Ono što predstavlja problem pri rešavanju ovog zadatka je 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.

Sledeći primer je skoro identičan prethodnom, ali dodavanje imena odeljenja u spisak potrebnih kolona malo komplikuje rešenje.

Primer 213. 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
```

Napomena: Ovo je validan SQL kôd i može se izvršiti na MS ACCESSu, ali program krahira pri pokušaju čuvanja ovog upita!

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 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 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 214. Dodeliti korisniku korisničkog imena Luka pravo da kreira tabele u bazi PREDUZECE.

USE PREDUZECE
GRANT CREATE TABLE TO *Luka*;

Primer 215. Dodeliti sva prava korisniku korisničkog imena Milos nad bazom PREDUZECE.

USE PREDUZECE
GRANT ALL TO *Milos*;

Primer 216. Dodeliti SELECT pravo korisniku korisničkog imena Janko nad tabelom RADNIK baze PREDUZECE.

USE PREDUZECE
GRANT SELECT ON RADNIK **TO** *Janko*;

Primer 217. 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 218. 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 219. 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 220. 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 221. Dodeliti pravo korisniku korisničkog imena boza sva prava nad kreiranim pogledom ODELJENJE20.

USE PREDUZECE
GRANT ALL ON ODELJENJE20 **TO** *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.

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. Oduzimanje prava vrši se naredbom REVOKE, čiji je opšti oblik:

REVOKE {ALL | [ALTER, DELETE, INDEX, SELECT, UPDATE(atr**)]}**
ON [kreator**.]{**tabela|pogled**}**
FROM {PUBLIC | korisnik1**[,** korisnik2 **...]}.**

Primer 222. Oduzeti korisniku korisničkog imena Luka pravo da kreira tabele u bazi PREDUZECE.

USE PREDUZECE
REVOKE CREATE TABLE TO *Luka*;

Primer 223. Oduzeti SELECT pravo korisniku korisničkog imena Janko nad tabelom RADNIK baze PREDUZECE.

USE PREDUZECE
REVOKE SELECT ON RADNIK **TO** *Janko*;

Primer 224. 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*;

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 225. 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.

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. 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).

Primer 226. Dodati podatak o zaposlenom identifikacionog broja 2 332, imena Petar, prezimena Marković u odeljenje 20. Koristiti transakcije.

```
USE preduzece
BEGIN TRAN
INSERT INTO RADNIK (idbr, ime, prezime, brod)
VALUES (2332, 'Petar', 'Marković', 20)
ROLLBACK
```

Posle izvršenja navedene transakcije, proverom podataka u tabeli RADNIK nije došlo do bilo kakve izmene, jer opcija **ROLLBACK** kod transakcija vraća stanje baze na pređašnje, tj. na stanje pre početka transakcije.

USE preduzece
BEGIN TRAN
INSERT INTO RADNIK (idbr, ime, prezime, brod)
VALUES (*2332*, *'Petar'*, *'Marković'*, *20*)
COMMIT

Posle izvršenja ove transakcije, u tabeli RADNIK postojaće još jedan zapis, o radniku koji je dodat. Za razliku od slučaja sa **ROLLBACK**, ovde na kraju transakcije imamo opciju **COMMIT** koja transakciju učini trajnom.

Primer 227. Zaposlenim identifikacionog broja 5 652 i 5 497 dati platu 1 100.

I) **SELECT** idbr, ime, prezime, plata
FROM RADNIK
WHERE idbr **IN** (*5652*, *5497*);

idbr	ime	prezime	plata
5497	Aleksandar	Marić	1000
5652	Jovan	Perić	1000

Najpre je urađena provera, da utvrdimo kolika je plata radnicima identifikacionih brojeva 5 652 i 5 497 i ustanovljenoje da imaju platu 1 000.

II) **USE** preduzece
BEGIN TRANSACTION
UPDATE RADNIK
SET plata = *1100*
WHERE idbr **IN** (*5652*, *5497*)
ROLLBACK TRANSACTION

Posle izvršene prethodne transakcije, ponovo treba uraditi proveru da se vidi da li je transakcija uradila ono što je zadato. Pokretanjem upita I) dobijamo isti rezultat kao i pre izvršenja transakcije II). To je očekivano jer na kraju transakcije II) postoji opcija **ROLLBACK,** koja onemogućava da budu prihvaćene bilo kakve promene na baze iz te transakcije.

III) **USE** preduzece
BEGIN TRANSACTION
UPDATE RADNIK
SET plata = *1100*
WHERE idbr **IN** (*5652*, *5497*)
COMMIT TRANSACTION

Posle aktivirane transakcije III) stanje u bazi se promenilo (jer je na kraju transakcije **COMMIT**), što se može proveriti pokretanjem upita I) posle koga se dobija sledeći rezultat:

idbr	ime	prezime	plata
5497	Aleksandar	Marić	1100
5652	Jovan	Perić	1100

Primer 228. Zaposlenim identifikacionog broja 7 892 dodeliti premiju 1 000, a zatim mu smanjiti platu za 10% vrednosti nove premije.

I) **SELECT** idbr, ime, prezime, plata, premija
FROM RADNIK
WHERE idbr=*7892*;

idbr	ime	prezime	plata	premija
7892	Luka	Bošković	2000	NULL

Pre pokretanja transakcija utvrđeno je da je plata pomenutog radnika 2 000, a da nema premiju. U ovom primeruje u jednom koraku potrebno je uraditi dve stavke, prvo dodeliti premiju radniku, a zatim njegovu postojeću platu smanjiti za onu vrednost koliko iznosi 10% novododeljne premije.

II) **USE** preduzece
BEGIN TRAN
UPDATE RADNIK
SET premija = *1000*
WHERE idbr=*7892*
GO
UPDATE RADNIK
SET plata = premija*0.9
WHERE idbr=*7892*
GO
ROLLBACK TRAN

Izvršavanjem transakcije II) u bazi neće doći do promena jer promene onemogućava **ROLLBACK**, tako da izvršavanjem upita I) dobija isti rezultat kao i pre pokretanja ove transakcije.

III) **USE** preduzece
BEGIN TRAN
UPDATE RADNIK
SET premija = *1000*
WHERE idbr=*7892*
GO
UPDATE RADNIK
SET plata = premija*0.9
WHERE idbr=*7892*
GO
COMMIT TRAN

Izvršavanje transakcije III) dovešće do promena u bazi, što se može proveriti pokretanjem upita I) koja daje sledeći rezultat:

idbr	ime	prezime	plata	premija
7892	Luka	Bošković	1900	1000

Svaka izmena podataka u bazi podataka evidentira se u dnevniku transakcija. Ako, recimo, nestanak struje prekine tekuću transakciju, samo deo izmena biće zapisan na disku, a neki podaci će zadržati staru vrednost. Naravno da ovo nikako ne sme da se desi, pa se zato čitava transakcija mora poništiti (na osnovu dnevnika

transakcija) prilikom prvog narednog pokretanja sistema za upravljanje bazama podataka.

Okidač (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.

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.

Primer 229. Kreirati triger imena *brisanje* koji se aktivira brisanjem zapisa iz tabele RADNIK, tako što iz tabele UCESCE briše sve zapise o učešću na projektima izbrisanih radnika iz tabele RADNIK.

```
CREATE TRIGGER brisanje ON RADNIK
FOR DELETE
AS
DELETE FROM UCESCE
WHERE idbr IN (SELECT idbr FROM DELETED);
```

Primer 230. Kreirati triger imena *procenat* koji se aktivira dodavanjem zapisa u tabelu UCESCE, tako što će svakom radniku koji se doda na neki projekat povećati platu 10% (podatak o plati je u tabeli RADNIK).

```
CREATE TRIGGER procenat ON UCESCE
FOR INSERT
AS
UPDATE RADNIK SET plata=plata*1.1
WHERE idbr= (SELECT idbr FROM INSERTED);
```

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, isl.), 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.

Primer 231. Kreirati uskadištenu proceduru *Zarada* koja prikazuje ukupne troškove za isplatu zarada radnicima po odeljenjima. Parametar broja odeljenja je promenljiv.

```
CREATE PROC Zarada
        @brod SMALLINT
        AS
                SELECT brod, SUM(plata+ISNULL(premija, 0)) AS zarada
                FROM radnik
                WHERE brod=@brod
                GROUP BY brod
GO
```

Za pokretanje ove procedure u MS SQL Serveru potrebno je odraditi sledeće:

```
EXEC Zarada 40;
```

Promenljivi parametar u ovom primeru je broj odeljenja, tako da smo ovde naveli njegovu vrednost, na primer broj odeljenja 40 i dobija se sledeći rezultat:

brod	zarada
40	9660

Navođenjem vrednosti broja nekog drugog odeljenja dobija se vrednost troškova za zarade radnika u tom odeljenju, na primer za odeljenje 20 je:

```
EXEC Zarada 20;
```

brod	zarada
20	15000

Primer 232. Napraviti proceduru koja prikazuje ime radnika i mesto odeljenja u kome radi za radnika čije ime počinje zadatim slovom. Parametar koji je promenljiv je početno slovo imena radnika.

```
CREATE PROC pocetno_slovo
        @slovo VARCHAR (1)
        AS
                SELECT  R.ime, O.mesto
                FROM RADNIK R , ODELJENJE  O
                WHERE R.brod= O.brod AND
                R.ime LIKE @slovo+'%'
GO
```

Pokretanjem ove procedure dobijaju se odgovarajući rezultati u zavisnosti od vrednosti promenljivog parametra koji se zada.

EXEC pocetno_slovo '*A*';

ime	mesto
Aleksandar	Novi Beograd
Andrija	Stari Grad

EXEC pocetno_slovo '*P*';

ime	mesto
Petar	Dorćol
Pavle	Stari Grad

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 postji). 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.

Primer 233. Kreirati uskladištenu proceduru *sp_zarada*, koja u polje **zarada** u tabeli RADNIK dodaje vrednost ukupne zarade za svakog radnika. Za to je potrebno deklarisati kursor u okviru procedure.

Napomena: Ako u tabeli RADNIK ne postoji atribut **zarada,** dodaje se korišćenjem opcije **ALTER TABLE**:

```
ALTER TABLE RADNIK
ADD zarada FLOAT(1);
```

Posle toga može se kreirati procedura:

```
CREATE PROCEDURE sp_zarada
AS
DECLARE @idbr INT
DECLARE @zarada INT

DECLARE c_pass CURSOR FOR
SELECT idbr, plata+ISNULL (premija, 0) FROM RADNIK

OPEN c_pass
WHILE (0=0)

BEGIN
FETCH NEXT FROM c_pass INTO @idbr, @zarada
UPDATE RADNIK SET zarada= @zarada
WHERE idbr=@idbr
IF (@@fetch_status <> 0) BREAK
END

CLOSE c_pass
DEALLOCATE c_pass
GO
```

Pokretanjem prethodne procedure:

```
EXEC sp_zarada
```

U kolonu zarada tabele RADNIK, biće unete vrednosti ukupne zarade, koje uračunavaju platu i premiju za svakog zaposlenog.

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.

LITERATURA

1. Vieira, R.: "*SQL Server 2000*", CET, Beograd, 2001.

2. Kreines, D.: "*Oracle SQL: The Essential Reference*", O'Reilly, 2000.

3. Alagić, S.: "*Relacione baze podataka*", Svjetlost, Sarajevo, 1984.

4. Radovan, M.: "*Projektiranje informacijskih sistema*", Informator, Zagreb, 1989.

5. Marjanović, Z.: "*ORACLE relacioni sistem za upravljanje bazom podataka*", Breza, Ljig, 1990.

6. Simić, R.: "*Organizacija podataka*", Naučna knjiga, Beograd, 1990.

7. Mišić, V.: "*Relaciona baza podataka Rdb/VMS*", Tehnička knjiga, Beograd, 1990.

8. Kovačević, A., Čukalevski, N.: "*Razvoj klijent-server sistema korišćenjem DBMS Oracle*", Viša elektrotehnička škola, Beograd, 2004.

9. Bobrowski, S.: "*Oracle 7 i obrada podataka po modelu klijent/server* ", Mikro knjiga, Beograd, 1995.

10. Vujnović, R.: "*SQL i relacijski model podataka*", Znak, Zagreb, 1995.

11. Wynkoop, S.: "*Vodič kroz SQL Server 7.0* ", CET, Beograd, 1999.

12. Obradović, S., Pandurov, T., Vučinić, B.: "*MS-Access Projektovanje baza podataka i aplikacija*", Viša elektrotehnička škola, Beograd, 2002.

13. Kaluđerčić, P., Obradović, S.. "*Projektovanje informacionih sistema - Relacione baza podataka*", Viša elektrotehnička škola, Beograd, 2003.

14. Grupa autora "*Access 2000 korak po korak* ", CET, Beograd, 1999.

15. Perić, D.: " *Aplikativni softver u poslovanju i tehnici*", Viša elektrotehnička škola, Beograd, 2004.

16. Дончев, А., Обрадович, С.: "*База от данни*", Технически университет – Габрово, Габрово, 2004.

17. Дончев, А.: " *Основи на базите от данни* ", Университетско издателство "Васил Априлов", Габрово, 1999.

18. Džaković, M.: "*Oracle 7 SQL sa osnovama relacionog modela i SQL*PLUS okruženjem*", Tehnička knjiga, Beograd, 1997.

SADRŽAJ