Sinteza fizičkog modela

Glava 6

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 vrijednosti za svaki atribut.

Jezik za opis fizičke strukture podataka definiše: način memorisanja i dodjele memorijskog prostora na disku, redosled i fizičku organizaciju podataka, dodjelu 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), izmjena sadržaja nekog polja (MODIFY), dodavanje sloga (INSERT), brisanje sloga (DELETE, REMOVE) itd.

Jezici upita omogućavaju realizaciju proizvoljnih funkcija – upita nad relacijama. Postoje dve klase ovih jezika zavisno da li se zasnivaju na relacionoj algebri ili na predikatskom (relacionom) računu. 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, presjek, Kartezijev (Dekartov) proizvod, projekcija, selekcija (restrikcija), spajanje (join), djeljenje. Postoje još neke izvedene operacije uslovljene Null vrijednostima.

Relacioni račun je neproceduralni jezik i programer umjesto 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.

Primjer relacionog računa n-torki:

```
x ∈ RADNIK
x.SIFRA, x.IME   GDE_JE   x.PLATA > 2000
AND   x.KVALIF = 'VKV'
```

Primjer relacionog računa domena:

```
x,y GDE_JE ∃ z > 2000 AND
RADNIK (SIFRA: x, IME: y, PLATA: z, KVALIF = 'VKV')
```

Zajednička karakteristika im je da su njihove konstrukcije mnogo sličnije prirodnim jezicima nego što je to slučaj sa jezicima treće generacije. Najpoznatiji primjeri ovih jezika su SQL, Quel i QBE. Oni se zasnivaju na raznim 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.

Ova 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 kao i mogućnosti za definisanje bezbjednosnih 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 primjenu, a u narednom poglavlju biće opisane mogućnosti QBE jer je njegova primena vrlo prosta i implementiran je 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, već pomenutom, radu:

"...***da koncept relacionog modela dopušta razvoj univerzalnog jezika za manipulisanje podacima baziranog na primjeni 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 nakon 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 pomenutih 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 i implementirana je danas praktično u svim programskim paketima za obradu podataka kao njihov sastavni dio, a u svojoj sintaksi ima za sve osnovne relacione operatore (spajanje, razliku, množenje, restrikciju i projekciju) ekvivalentne SQL naredbe. Tri preostala relaciona operatora (dijeljenje, presjek 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.

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, zahtijeva 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.

SQL spada u takozvane ***no-case-sensitive*** programske pakete, što znači da se naredbe, ključne riječi, imena objekata i varijabli 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 riječi, funkcije, i imena relacija,
- mala slova - imena atributa, varijabli, itd.,
- italik mala slova - vrijednosti atributa (podaci),
- u uglastim zagradama - neobavezne opcije,
- komentari - dva znaka minus (- -).

Programski paket SQL omogućava jednostavno :

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

Korisnik može u komunikaciji sa sistemom upotrjebiti dvije opcije i to:

- interaktivni način rada (korisnik direktno komunicira sa bazom, a rezultat dobija na ekranu monitora), odnosno
- ugradnjom i povezivanjem SQL-a sa nekim programskim paketom (Delphi, Visual C++, Visual Basic itd.). Mogu se koristiti i gotove aplikacije napisane sintaksom jednog od tih jezika.

U prvom slučaju korisnik mora poznavati strukturu baze podataka (nazive tabela, relacije, odnose među njima, te imena atributa u njima), kao i sintaksu upitnog jezika.

U drugom slučaju, prilikom korišćenja unapred definisanih aplikacija, dovoljno je znati samo koju informaciju želimo dobiti (za koju naravno prije toga mora biti pripremljena odgovarajuća aplikacija) pa da njenim pozivom dođemo do rezultata. Nažalost, u ovome slučaju koji je praktičan za korisnika-laika, primjena je ograničena samo na one aktivnosti i operacije koje su unaprijed pripremljene.

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

a) ***Naredbe za definisanje podataka*** (Data Definition Statements) 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** – izmjena definicije tabele, izmjena, 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 (izmjenu, dodavanje i brisanje) i izvještavanje (pribavljanje 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, izmjena vrijednosti podataka u koloni.

c) ***Naredbe za upravljanje bezbjednošću podataka*** (Data Control Functions) omogućavaju oporavak, konkurentnost, sigurnost i integritet relacione baze podataka:

- **GRANT** – dodjela 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 narednih naredbi,
- **COMMIT WORK** – prenos dejstva transakcije na bazu podataka,
- **ROLLBACK WORK** – poništavanje dejstva transakcije na bazu podataka.

Bez poznavanja navedenih komandi za definisanje podataka i rukovanje podacima ne može se ozbiljno računati na održavanje, dizajniranje i izradu elementarnih aplikacija za obradu podataka uz korišćenje moderne relacione tehnologije. Štaviše, za najveći broj standardnih zahtjeva, 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.

6.1 Naredbe za definisanje 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 vrijednosti.

Ime relacije (ili **ime_tabele**) ne smije biti nijedna od ključnih riječi programskog paketa SQL i mora počinjati slovom engleskog alfabeta. Od specijalnih znakova može sadržavati 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 rijetkost. Obično je taj broj, u dobro projektovanim sistemima, od 10 do 30. Svi sistemi za upravljanje bazama podataka postavljaju neke ograničenja u pogledu broja atributa. U Access-u taj broj ne može biti veći od 255 a u SQL Serveru od 1024.

U jednoj bazi podataka, kao što smo vidjeli, može postojati više atributa sa istim imenom, ali oni ne smiju pri tome biti u istoj relaciji (tabeli). Preporučljivo je ovu opciju (ista imena atributa u raznim tabelama) radi preglednosti i eliminacije grešaka, izbjegavati 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 vrijednosti** s druge strane, spriječavaju se moguće greške pri unošenju podatka.

Razni programski paketi omogućavaju definisanje različitih tipova podataka, ali se svi svode na nekoliko osnovnih tipova.

- **Slovni ili znakovni tip** podatka koristi se kada podatak predstavlja niz alfanumeričkih znakova. Takav podatak je na primjer 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 promjenljive dužine (**VARCHAR**).
- **Numerički (NUMERIC) tip** podatka (na primjer visina ličnog dohodka ili dužina neke ulice) može slično kao i u drugim programskim jezicima, da bude cjelobrojan (**SMALLINT,** ako je dužina 16 bita, **INTEGER**, ako je dužina 32 bit-a 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 vrijeme, za neke rokove, bez obzira da li su iskazani danima, mjesecima i godinama ili satima, minutama i sekundama.

  Datumski tip DATE je jedna cjelina, jedan podatak, iako u sebi sadrži tri numerička polja (za *dan*, *mjesec* 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 mjeseci, a mjesec 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 dvije vrijednosti.

  *Na primjer takav atribut je **prisutnost,** kod koga mogu postojati samo vrijednosti **prisutan** ili **nije_prisutan,** ili atribut **pol** kod koga mogu postojati vrijednosti **muški** i **ženski**.*

- **Memo tip** podatka je u principu slovni (alfanumerički) tip, ali sa većom dužinom (na primjer do 64 kilobajta) a koristi se za unošenje opisnih podataka (dijagnoza ili anamneza pacijenta, na primjer).

- **Bit-mapa** podatak nosi neku grafičku informaciju, dijagram ili sliku.

U tehnici definisanja podataka postoji i pomenuti slučaj **nepostojećeg podatka** kada vrijednost podatka (bilo kog tipa) nije poznata ili nije nastupio momenat njegovog prisustva u bazi, a koji u drugim programskim jezicima nije poznat, takozvana ***Null-***vrijednost. SQL dopušta svakom podatku (sem primarnog ključa) da ima i nepostojeću, Null-vrijednost.

U sledećem primjeru možemo vidjeti 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 radnika, poslu koji obavlja, njegovom rukovodiocu, datumu zaposlenja, premiji i plati.

*Tabelu **RADNIK** kreiramo naredbom:*

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

a kao rezultat dobijamo relaciju:

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

u kojoj atributi ***idbr#*** *i* ***ime*** *ne mogu imati vrijednosti* ***Null****.*

Za potpuniju informaciju o preduzeću, sem tabele RADNIK, potrebno je kreirati i tabelu ODJELJENJE 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 ODJELJENJE
        (brod# SMALLINT NOT NULL,
        ime_od CHAR(15) NOT NULL,
        mesto CHAR(20));

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

Ovako definisana tabela RADNIK sadrži u sebi i neke relacije između pojedinih radnika – *unarne veze* (neki radnici su istovremeno rukovodioci nekim drugim radnicima). Takođe je ostvarena i jedna relacija sa jednim drugim entitetom – *binarna veza* sa tabelom ODJELJENJE, jer radnici su zaposleni u nekom od odjeljenja koja se nalaze u sastavu preduzeća. Pri tome jedan radnik pripada samo jednom odjeljenju, a u jednom odjeljenju 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 ODJELJENJE (na strani **1**). Spoljni ključ prepoznajemo po znaku $ iza imena atributa.

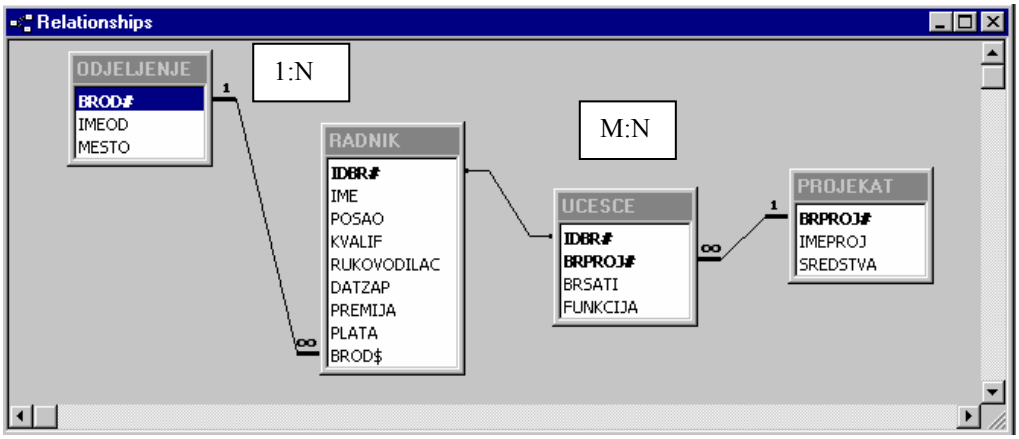
Ali, svi radnici rade na nekim konkretnim poslovima, projektima, i pri tome jedan radnik može raditi na više projekata, a istovremeno na jednom velikom projektu radi više radnika. Dakle ovo je relacija M:N (više prema više). Da bismo ostvarili ovu relaciju između dva entiteta

treba kreirati novu tabelu, nazovimo je UČEŠĆE, koja ima **složeni primarni ključ** (*idbr#,brproj#*) koji sačinjavaju primarni ključevi iz tabela RADNIK (*idbr#*) i PROJEKAT (*brproj#*).

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

Dakle, bazu podataka sačinjavaju četiri fizičke tabele (slika 6.1.):

RADNIK <idbr#,kvalif, ime, posao, rukovodilac, dat_zap, premija, plata, brod$>
ODJELJENJE <brod#,, naziv, mesto>
PROJEKAT <brproj#,, imeproj, sredstva>
UČEŠĆE <idbr#, brproj#, funkcija,>



Slika 6.1 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, spoljnji 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. Moderne verzije RDBMS-a pored naredbe CREATE TABLE imaju i mogućnost kreiranja relacije bez pisanja ove naredbe korišćenjem unaprijed pripremljenih opcija, "čarobnjaka" (WIZARD) i "alata" (TOOLS), ugrađenih u sistem za manipulisanje bazom. Korisnik u tom slučaju mora samo unijeti naziv relacije, broj atributa, naziv atributa i tip podatka koji će poprimiti taj atribut, te da li atribut može poprimiti vrijednost ***Null*** ili ne.

6.1.1 Izmjena vrijednosti ili definicije atributa i dodavanje novog atributa u postojeću tabelu - ALTER TABLE

Dodavanje novog atributa, odnosno nove kolone u tabelu:

ALTER TABLE ime_tabele
ADD (atrib tip [, atrib tip]);

Primjer 1: U tabelu ODJELJENJE dodati atribute šef_od (šef odjeljenja), i br_zap (broj zaposlenih):

ALTER TABLE ODJELJENJE
ADD (*šef_od INTEGER*, *br_zap NUMBER(2)*);

Izmjena definicije postojećeg atributa, tj. postojeće kolone u tabeli:

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

Primjer 2: U tabeli ODJELJENJE povećati dužinu atributa *br_zap* na 6 cifara.

ALTER TABLE ODJELJENJE
MODIFY (*br_zap NUMBER(6)*);

6.1.2 Izbacivanje relacije iz baze podataka - DROP TABLE

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.

4.1.3 Indeksi

Indeksi se koriste za brzi pristup po kolonama koje se indeksiraju i omogućavaju jedinstvenu vrijednost 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).

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

Primjer 3: U tabeli ODJELJENJE kreirati indeks nad artibutom *imeod*.

```
CREATE INDEX naziv_ind
ON ODJELJENJE (imeod);
```

Primjer 4: U tabeli ODJELJENJE kreirati primarni ključ-jedinstveni indeks nad artibutom *brod#*.

```
CREATE UNIQUE INDEX brod_ind
ON ODJELJENJE (brod#);
```

Indeksi se izbacuju naredbom DROP INDEX naziv_indeksa.

Primjer 5: U tabeli ODJELJENJE ukloniti indeks nad atributom *imeod*.

```
DROP INDEX naziv_ind
```

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 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 vrijednost manju od čvora roditelja a desno su čvorovi sa vrijednošću koja je veća od čvora roditelja.

Jedna od osnovnih operacija pri manipulisanju velikim brojem podataka je pomenuto **pretraživanje,** to jest dobijanje nove informacije na osnovu određenog broja prethodno poznatih vrijednosti nekih atributa. Međutim, ako je broj podataka veliki (stotine miliona, na primjer), onda pretraživanje nije ni najmanje jednostavan i kratak postupak i za najbrže savremene računare. Na primjer, 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, vrijeme za dobijanje odgovora postaje nedopustivo dugo.

Iz toga razloga se, pri pretraživanju velikih baza podataka, pribjegava njihovom **indeksiranju** po atributu (ili atributima) po kome se vrši pretraživanje. Sve tabele su po pravilu indeksirane po primarnom ključu i tada **Indeksna tabela** (uvijek 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 dijela:

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

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

GRAĐANIN < matbr#, prezime, ime, datrođ, adresa, .. >

a jedan njen dio se vidi u tabeli 6.1.

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 primjer **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 varijabli po veličini ili po datumu) i pored vrijednosti atributa **prezime** indeksirana tabela ima samo još i vrijednost 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 rječi u rječniku ili broja u telefonskom imjeniku) bez potrebe da se "češlja" cijela tabela, a onda uz pomoć vrijednosti indeksa-rednog broja zapisa, dolazimo opet u samo jednom koraku (direktnim pristupom) i do ostalih 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) podliježe nekim pravilima. Međutim, uvijek mora biti zadovoljen uslov da postoji kriterijum po kome se vrijednosti u indeksnoj tabeli mogu poređati.

Naravno nijesu svi atributi dobri kandidati za indeks. Nije noguće praviti indekse nad kolonama tipa bit-map, text ili slika, a kako je veličina indeksa ograničena, nijesu 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: spoljnji (strani) ključevi i kolone koje učestvuju u klauzulama GROUP BY i ORDER BY (koje će kasnije biti detaljno objašnjene).

GRAĐANIN

Redbr	matbr#	prez	ime	datrođ	adresa
1	1324764	Antić	Ante	10.07.54	Beograd
2	9763421	Jović	Jovan	24.12.33	Valjevo
3	4513298	Marić	Maks	13.03.76	Bor
4	3344228	Babić	Miro	02.02.77	Uzice
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 6.1 Tabela GRAĐANIN i njoj pridruženi indeks INDGRAD

Na kraju, pomenimo da indeksiranje ima i svojih nedostataka. 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 vrijeme, jer se indeksne datoteke, svaki put nakon izmjene osnovne tabele, moraju reindeksirati. 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.

Pored indeksiranja i ciljanim grupisanjem podataka na fizičkom medijumu (disku) efikasnost i brzina rada mogu se bitno povećati. Rješenje se sastoji u tome da se slogovi koji se najčešće uzastopno koriste smjeste na disk na iste, ili susjedne segmente. Tako, ako imamo slogove **S1** i **S2** smještene na isti segment diska **D1** onda će se pri dohvatu **S1** jednovremeno u bufferu 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 dijelu memorije i pristup njima je zato brz. Ponekad se svi zapisi na disku smještaju ne po redosledu unosa već uređeni po nekom redoslijedu. 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 vrijeme održavanja baze.

6.2 Naredbe za rukovanje podacima

Naredba **SELECT** služi za “dohvatanje” jednog ili više podataka iz jedne, ili iz više tabela. Rezultat ove naredbe je neka informacija a ima opet najčešće strukturu relacije, pa tako ima svoje atribute i vrijednosti atributa, ali kao fizička tabela, stalno prisutna na disku, ne postoji. 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 dio prve), prilikom pravljenja složenih upita, jer za sve vrijeme izvršavanja naredbe SELECT parcijalni rezultati postoje kao tabele u memoriji računara. Rezultat upita može biti prost neizmjenjen sadržaj jedne ili više tabela, ali isto tako može biti i neka nova vrijednost 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 primjerima koji slijede naći i takvi gdje 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 [**ALL**]

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

- definišu-selektuju atributi (**SELECT**) čije vrijednosti želimo dobiti (odgovara **operatoru projekcije)**, zatim se
- izdvajaju relacije u kojima se nalaze vrijednosti 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 zahtjevi kojima se rezultati
- grupišu (**GRUP BY**), ili
- na neki način uređuju (**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, ostale klauzule (WHERE, HAVING, ...) jednostavno se izostavljaju.

Koristeći naredbu SELECT moguće je :

- izdvojiti neke atribute,
- izmjeniti redoslijed atributa (redoslijed atributa u odgovoru isti je kao redoslijed atributa koji je naveden u naredbi SELECT a ne mora biti isti kao redoslijed atributa dat u definiciji tabele).
- spriječiti pojavu višestrukih n-torki.

Značenje opcija **ALL**, odnosno **DISTINCT** je sledeće:

- **ALL** 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, podrazumijeva se **ALL,** ali ima verzija SQL-a koje u tom slučaju podrazumijevaju i **DISTINCT** i na to treba obratiti pažnju, jer u suprotnom dobijeni rezultat možda neće biti relacija, što može izazvati greške u daljnjoj obradi. Na kraju, napomenimo da SQL ne "razumije" znak **#** i $ u imenima atributa kao oznake za prepoznavanje ključnih atributa, i da su ti znaci u primjerima u ovoj knjizi upotrebljeni samo zato da bi se ključni atribut razlikovao od ostalih, i time primjeri bili pregledniji. Sve operacije i klauzule SQL-a prikazaćemo na primjerima relizovanim na već kreiranoj bazi podataka jednog preduzeća.

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

ODJELJENJE <brod#, naziv, mesto>

PROJEKAT <brproj#, imeproj, sredstva>

UČEŠĆE <idbr#, brproj#, funkcija,>

Neka se u tabelama nalaze podaci prikazani na slici 6.2. Tabele nijesu urađene u potpunosti po pravilima normalizacije, ali to je učinjeno s namjerom 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.

6.2.1 Upiti nad jednom tabelom

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

Primjer 6: Prikaži celokupan sadržaj tabele ODJELJENJE.

SELECT * FROM ODJELJENJE;

Rezultat će biti čitava bazna tabela (slika 6.2. b)) jer znak * je sinonim za traženje svih atributa, a kod nekih RDBMS (na primjer Access), kao i Windowsu i DOS-u, zamenjuje bilo koji niz znakova (džoker znak).

RADNIK : Table

	IDBR#	IME	POSAO	KVALIF	RUKOVODI	DATZ
+	5367	Petar	vozac	KV	5780	01-j
+	5497	Aco	radnik	KV	5662	17-f
+	5519	Vaso	prodavac	VKV	5662	07-r
+	5652	Jovan	radnik	KV	5662	31-m
+	5662	Jovo	upravnik	VSS	5842	12-a
+	5696	Miro	radnik	KV	5662	30-s
+	5780	Bozo	upravnik	VSS	5842	11-a
+	5786	Pavle	upravnik	VSS	5842	22-m
+	5842	Savo	direktor	VSS		15-d
+	5867	Simo	savetnik	VSS	5842	08-a

ODJELJENJE : Table

	BROD#	IMEOD	MESTO
+	10	komercijala	Novi Beograd
+	20	plan	Dorcol
+	30	prodaja	Stari Grad
+	40	direkcija	Banovo Brdo
+	50	erc	Zemun

a)

UCESCE : Table

IDBR#	BRPROJ#	BRSATI	FUNKCIJA
5497	400	2000	IZVRŠILAC
5519	300	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
5874	300	2000	IZVRŠILAC
5898	200	2000	IZVRŠILAC
5900	100	2000	IZVRŠILAC
5932	100	500	KONSULTANT
5932	200	1000	ORGANIZATOR
5932	300	500	NADZOR
5953	100	1000	IZVRŠILAC
5953	300	1000	IZVRŠILAC
6234	100	500	NADZOR
6234	200	1200	IZVRŠILAC
6234	300	300	KONSULTANT
6789	200	2000	IZVRŠILAC
7890	300	2000	IZVRŠILAC
0	0	0	

Slika 6.2 Baza podataka preduzeća

Primjer 7: Prikaži nazive svih odjeljenja u preduzeću.

```
SELECT imeod
FROM ODJELJENJE;
```

Rezultat upita dat je na slici 6.3.

imeod
komercijala
plan
prodaja
direkcija
erc

Slika 6.3 Nazivi odjeljenja

Primjer 8: Prikaži nazive svih poslova u preduzeću.

```
SELECT posao
FROM RADNIK;
```

Rezultat upita dat je na slici 6.4 a).

Primjer 9: Prikaži nazive svih raznih poslova u preduzeću.

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

Rezultat upita dat je na slici 6.4 b).

POSAO
vozac
radnik
prodavac
radnik
upravnik
radnik
upravnik
upravnik
direktor
savetnik
radnik
nabavljac
vozac
savetnik
nabavljac
analiticar
rukovodila
analiticar

POSAO
analiticar
direktor
nabavljac
prodavac
radnik
rukovodila
savetnik
upravnik
vozac

a) b)

Slika 6.4 Prikaz poslova u preduzeću

Isti efekat se kod nekih DBMS 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, to treba praktično uvijek koristiti oblik SELECT DISTINCT**,** a ne samo SELECT, a mi ćemo u primjerima koji slijede smatrati da je to ponuđena default opcija 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 uvijek koristiti ovakav način pisanja, a obavezno ga moramo koristiti onda ako, u dvije 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 slijedeće naredbe potpuno identične sa naredbom u prethodnom primjeru:

Primjer 10: Prikaži nazive svih raznih poslova u preduzeću.

```
SELECT DISTINCT radnik.posao
FROM RADNIK ;

SELECT DISTINCT r.posao
FROM RADNIK R;
```

Atributima se unutar naredbe SELECT mogu dati i druga imena koja će biti prikazana u rezultatu upita. Ovo je naročito korisno ako se unutar ove naredbe vrše i neke matematičke operacije.

Primjer 11: Prikaži šifre svih odjeljenja u preduzeću.

a) Ova informacija može se dobiti iz tabele RADNIK:

```
SELECT DISTINCT radnik.[brod$] AS "sifre odjeljenja"
FROM RADNIK;
```

Rezultat je dat na slici 6.5 a).

sifre odjeljenja
10
20
30
40

a)

b) Isti podaci mogu se dobiti iz tabele ODJELJENJE:

```
SELECT odjeljenje.[brod#] AS  sifra
FROM ODJELJENJE;
```

Rezultat je dat na slici 6.5 b).

sifre
10
20
30
40
50

b)

Slika 6.5 Šifre odjeljenja

Treba uočiti nekoliko detalja:

- u primjeru a) moramo koristiti klauzulu **DISTINCT** jer u jednom odjeljenju radi više radnika pa bi u odgovoru bilo ponovljeno svako odjeljenje više puta.
- u primjeru b) ne moramo koristiti klauzulu **DISTINCT** jer je šifra odjeljenja primarni ključ relacije i samim tim je jedinstven.
- u primjeru a) moramo iza klauzule **AS** (kao) koristiti znake navoda ili uglaste zagrade [ ] (u Accessu) jer novo ime je sastavljeno od više riječi, a u primjeru b) ne moramo.

Napomjena: U nekim RDBMS nije potrebno navoditi klauzulu AS već se samo stavi razmak, tj. prazno mjesto (space), to je ekvivalentno klauzuli AS (ovo nije slučaj u Accessu). Tako su sledeće naredbe SELECT ekvivalentne prethodnim:

```
SELECT O.[brod#] šifra
FROM ODJELJENJE O;
```

```
SELECT DISTINCT R.[brod$] [šifre odjeljenja]
FROM RADNIK R;
```

Klauzula **WHERE**

U svim dosadašnjim primjerima u rezultatu su se pojavljivali svi redovi, jedne tabele. Moguće je primjeniti naredbu SELECT samo na neke 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 vrijednost unutar nekih granica (BETWEEN),
- Izdvajanje redova čija vrijednost pripada nekoj listi vrijednosti (IN),
- Izdvajanje redova koji ne zadovoljavaju neke uslove (NOT, IS NOT),
- Izdvajanje redova ako neka vrijednost postoji (EXISTS) itd.

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

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 (< > ili ≠).**

Ako se uporjeđuju argumenti koji se sastoje od više atributa, tada oba argumenta moraju imati jednak broj atributa, a uporjeđuje se prvi sa prvim, drugi sa drugim itd. Konačno, napomjenimo da atributi koji se uporjeđuju moraju biti istog, ili kompatibilnog tipa podataka.

Prilikom postavljanja uslova mogu se koristiti sljedeće opcije:

- **BETWEEN**

- **IN**
- **EXISTS.**

Ne ulazeći u sve mogućnosti navedenih opcija, pokažimo na primjerima kako se one koriste.

Primjer 12: Prikaži kvalifikaciju, platu i ime zaposlenih u odjeljenju 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	Andro	30
KV	1100	Pero	30

Slika 6.6 Podaci o zaposlenima u odjeljenju 30

Napomjena: Uočimo da redosled atributa u rezultatu odgovara redosledu atributa u naredbi SELECT a ne redosledu atributa u samoj fizičkoj tabeli.

Primjer 13: Prikaži ime, posao i platu zaposlenih u odjeljenju 30 čija je plata veća od 2000.

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

IME	POSAO	PLATA	BROD$
Pavle	upravnik	2800	30
		0	0

Slika 6.7 Podaci o radnicima u odjeljenju 30 sa platom većom od 2000

Primjer 14: Prikaži ime, posao upravnika i direktora.

```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE (R.posao="direktor") OR
        (R.posao="upravnik");
```

IME	POSAO	BROD$
Jovo	upravnik	10
Bozo	upravnik	20
Pavle	upravnik	30
Savo	direktor	40

Slika 6.8 Podaci o direktoru i upravnicima

Napomjena: Isti rezultat se dobija korišćenjem klauzule **IN** (u prevodu “u”) čiji opšti oblik glasi: **atribut [NOT] IN** (lista skalarnih vrijednosti).

Primjer 15: Prikaži ime i posao upravnika i direktora.

```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
        WHERE R.posao IN ("direktor", "upravnik");
```

IME	POSAO	BROD$
Jovo	upravnik	10
Bozo	upravnik	20
Pavle	upravnik	30
Savo	direktor	40

Slika 6.9 Podaci o direktoru i upravnicima

Napomjena: Poželjno je da redoslijed ispitivanja uslova regulišete upotrebom zagrada, jer u protivnom možete dobiti potpuno neočekivane i vjerovatno netačne odgovore na upit jer uslovi nisu interpretirani na odgovarajući način.

Primjer 16: Prikaži ime i posao upravnika i analitičara iz odjeljenja 10.

a) **SELECT** R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE R.posao="*upravnik*" **OR**
R.posao="*analiticar*" **AND** R.[brod$]=*10*;

IME	POSAO	BROD$
Jovo	upravnik	10
Bozo	upravnik	20
Pavle	upravnik	30
Marko	analiticar	10
		0

a) Neta

b) **SELECT** R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE (R.posao=”*upravnik*” **OR**
R.posao=”*analiticar*”) **AND** R.[brod$]=*10*;

ime	posao	brod$
Jovo	upravnik	10
Marko	analiticar	10
		0

b) Tačan odgovor

Slika 6.10 Upravnici i analitičari iz odjeljenja 10

Primjer 17: Prikaži ime i platu zaposlenih čija je plata od 2600 do 3000.

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

ime	plata
Pavle	2800
Simo	2750
Savo	3000
Mita	2600

Slika 6.11 Zaposleni sa platama od 2600 do 3000

Sintaksa opcije **BETWEEN** (u prevodu “između”) ima opšti oblik:

argument1 [NOT] BETWEEN argument2 AND argument3

a rezultat ovog testa je istinit ako se vrijednost za **argument2** nalazi (ili ne nalazi - **NOT**) između vrijednosti **argument1** i **argument3** uključujući i granice toga intervala.

Opcija **EXISTS** (u slobodnom prevodu – “postoji”) koristi se za provjeru postojanja najmanje jedne n-torke u rezultatu pretraživanja. Sintaksa ove naredbe glasi:

EXISTS (relacijski izraz).

Odgovor je istinit (**TRUE**) ako relacijski 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 primjer:

- **LIKE,**
- **MATCH,**

- **ALL- OR – ANY i**
- **UNIQUE,**

a njihovo značenje se može naći u kompletnim priručnicima (manualima) za korišćenje programskog paketa **SQL-a**.

Klauzula **ORDER BY**

Klauzula **ORDER BY** slaže n-torke po nekom redoslijedu (po abecedi, po veličini itd.) u rastućem ili opadajućem poretku. Klauzula ORDER BY je uvijek 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 vrši rađanje.

Primjer 18: Prikaži ime i kvalifikaciju zaposlenih čija imena počinju slovom **p**.

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE 'p%';
```

IME	KVALIF
Petar	KV
Pavle	VSS
Pero	KV

a) neuređen odgovor

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

IME	KVALIF
Pero	KV
Petar	KV
Pavle	VSS

b) uređen po kvalifikaciji

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

IME	KVALIF
Pavle	VSS
Pero	KV
Petar	KV

c) uređen po imenima

*Slika 6.12 Spisak zaposlenih čija imena počinju slovom **p***

Napomjena: U raznim verzijama SQL-a postoje džoker znaci, tj. znaci koji mogu zamijeniti bilo koji niz znakova **%** (a u Access-u je to znak ***) ili jedan znak **_** (u Access-u je to znak **?**).

Navedimo još nekoliko primjera kako bi 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 6 slova — **WHERE** ime **LIKE** '_ _ _ _ _ ' ;
- ime nije dužine 5 slova — **WHERE** ime **NOT LIKE** '_ _ _ _ _' ;
- u imenu je slovo **l** isped slova **N** — **WHERE** ime **LIKE** '%l %N %' ;
- broj ima **dve cifre** crticu **cifru** — **WHERE** tel **LIKE** '[0-9][0-9]-[0-9]' ;

Primjer 19: Prikaži ime, kvalifikaciju, platu i premiju uređenu:

a) po kvalifikaciji u opadajućem, po plati u rastućem a po premiji u opadajućem redoslijedu,

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

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

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

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
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	
Janko	VS	3900	10
Vaso	VKV	1200	1300
Slobo	KV	900	1300
Tomo	KV	1000	1100
Aco	KV	1000	800
Jovan	KV	1000	500
Miro	KV	1000	0
Andro	KV	1100	1200
Pero	KV	1100	0
Petar	KV	1300	1900

a)

ime	kvalif	plata	premija
Slobo	KV	900	1300
Tomo	KV	1000	1100
Aco	KV	1000	800
Jovan	KV	1000	500
Miro	KV	1000	0
Andro	KV	1100	1200
Pero	KV	1100	0
Vaso	VKV	1200	1300
Marko	VSS	1300	3000
Petar	KV	1300	1900
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	
Janko	VS	3900	10

b)

ime	kvalif	plata	premija
Ivan	VSS	1600	3200
Marko	VSS	1300	3000
Petar	KV	1300	1900
Vaso	VKV	1200	1300
Slobo	KV	900	1300
Andro	KV	1100	1200
Tomo	KV	1000	1100
Aco	KV	1000	800
Jovan	KV	1000	500
Janko	VS	3900	10
Miro	KV	1000	0
Pero	KV	1100	0
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	

c)

Slika 6.13 Uređivanje redoslijeda n-torki

Napomjena: Redoslijed sortiranja odgovara redoslijedu navođenja atributa u klauzuli **ORDER BY**. Najprije se sortira izvještaj po prvom navedenom atributu, zatim po drugom, itd. Sortiranje može biti izvršeno u rastućem redoslijedu – **ASC** (**Ascedent**) i to je pretpostavljena (**default**) vrijednost, pa se ne mora navoditi. Ako želimo da odgovor bude uređen u opadajućem redoslijedu (**Descedent**) po vrijednosti nekog atributa moramo iza imena navesti reč **DESC**. Ako postoji više n-torki sa jednakim vrijednostima atributa njihov redoslijed je onda proizvoljan. Ovakav rezultat se može izbjeći sortiranjem po atributu koji je primarni ključ, gdje su dvije iste vrijednosti atributa isključene. Ako se sortiranje vrši po nekom atributu koji ima i vrijednost ***Null*** (kao što bi mogao biti atribut *premija* u posljednjem primjeru) onda se sve vrijednost ***Null*** grupišu zajedno.

Upotreba **NULL** vrijednosti

Jedna od najvećih novina koje su donele relacione baze podataka jeste mogućnost prikazivanja nepostojećeg podatka čija vrijednost je nedefinisana. To su **Null** vrijednosti, i one ponekad moraju postojati.

1. Kada u bazi postoje atributi za koje su Null vrijednosti “normalne” jer to svojstvo nije primjenljivo na sve primjerke nekog entiteta. Takav je na primjer

atribut *premija* u tabeli RADNIK. Ako to svojstvo nije primjenljivo na većinu primjeraka 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 posjeduju (u ovom slučaju može se napraviti tabela PREMIJA<*idbr#*, *premija*>).

2. Ako vrijednost nekog atributa za neke objekte još nije poznata ili nije dozvoljena (neki radnici još uvijek nemaju telefon, rukovodioca, ili nisu raspoređeni ni u jedno odjeljenje).

3. Kada nije nastupio momenat djelovanja nekog atributa (plata ili premija za mart poznati su tek tokom aprila).

Za testiranje vrijednosti kolona koje sadrže Null vrijednosti na raspolaganju su samo dvije opcije **IS NULL** i **IS NOT NULL**, a moguće je koristiti i operatore logičkog poređenja.

Treba imati u vidu šta će biti rezultat operacije pri upotrebi operatora logičkog poređenja = i ≠ (tabele 1 i 2) i operatora **IS NULL** i **IS NOT NULL** (tabela 3) .

=	True	False	Null
True	True	False	Null
False	False	True	Null
Null	Null	Null	Null

Tabela 1.

≠	True	False	Null
True	False	True	Null
False	True	False	Null
Null	Null	Null	Null

Tabela 2.

	Is Null	Is Not Null
<value>	False	True
True	False	True
False	False	True
Null	True	False

Tabela 3.

Primjer 20: 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
Aco	KV	1000	800
Slobo	KV	900	1300
Vaso	VKV	1200	1300
Miro	KV	1000	0
Tomo	KV	1000	1100
Andro	KV	1100	1200
Pero	KV	1100	0
Jovan	KV	1000	500
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Janko	VS	3900	10

a)

b) nemaju premiju.

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

ime	kvalif	plata	premija
Jovo	VSS	2400	
Bozo	VSS	2200	
Pavle	VSS	2800	
Simo	VSS	2750	
Savo	VSS	3000	
Mita	VSS	2600	

b)

Slika 6.14 Rad sa nedefinisanim vrijednostima

Napomjena: Ovde treba uočiti razliku između objekata koji nemaju premiju (imaju Null vrijednost) i objekata koji imaju premiju a vrijednost premije može biti i 0 (radnici Pero i Miro).

Klauzula **GROUP BY**

Klauzula **GROUP BY,** koju treba da slijedi 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 na način da sve n-torke unutar grupe imaju jednake vrijednosti atributa po kojima se grupišu. Ako se navede više atributa po kojima treba vršiti grupisanje, prvo se ređaju n-torke sa jednakom vrijednošću prvog atributa, zatim se unutar tih grupa preslažu n-torke prema vrijednostima drugog atributa, itd.

Primjer 21: 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;
```

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

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

ime	kvalif	plata	premija
Slobo	KV	900	1300
Miro	KV	1000	0
Jovan	KV	1000	500
Aco	KV	1000	800
Tomo	KV	1000	1100
Pero	KV	1100	0
Andro	KV	1100	1200
Petar	KV	1300	1900
Vaso	VKV	1200	1300
Janko	VS	3900	10
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	

a) grupisanje podataka

ime	kvalif	plata	premija
Slobo	KV	900	1300
Miro	KV	1000	0
Jovan	KV	1000	500
Aco	KV	1000	800
Tomo	KV	1000	1100
Pero	KV	1100	0
Andro	KV	1100	1200
Petar	KV	1300	1900
Vaso	VKV	1200	1300
Janko	VS	3900	10
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	

b) sortiranje podataka

Slika 6.15 Sličnost dejstva klauzula GROUP BY i ORDER BY

Napomjena: Ova sličnost je samo prividna a prava uloga klauzule GROUP BY biće opisana pri upotrebi zbirnih (agregatnih) funkcija gde se određene funkcije

(na primjer srednja vrijednost) primjenjuju na grupe podataka sa nekim zajedničkim svojstvom (po odjeljenjima).

Napomjena: Atributi po kojima se vrši grupisanje moraju biti navedeni i u SELECT naredbi, a u nekim RDBMS, kao recimo u Access-u, mora se vršiti grupisanje po svim atributima navedenim u naredbi SELECT, dakle u prethodnom primjeru grupisanje se mora izvršiti i po imenu iako to nismo željeli.

4.2.2 Upiti nad jednom tabelom sa izračunavanjem novih vrijednosti

Pretraživanje neke baze podataka u svrhu dobijanja novih i relevantnih informacija je svakako najinteresantniji oblik primjene analize baze podataka. Upravo u pretraživanju je SQL pokazao svoja preimućstva nad ostalim paketima. SQL ima ugrađen veliki broj gotovih funkcija za *dobijanje zbirnih informacija* (AVG, SUM, MIN, MAX i manje poznate COUNT), za *obavljanje aritmetičkih operacija i uobličavanje rezultata* (POWER, ROUND, TRUNC, ABS, SIGN, MOD, SQRT) kao i za *rad sa tekstom*, 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 vrijednosti po nekom kriterijumu (atributu).

Primjer 22: Prikaži najmanju, najveću, srednju platu i broj zaposlenih:

a) u cjelom preduzeću,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
       AVG(plata) AS srednja, COUNT(*) AS  broj
FROM RADNIK;
```

b) u cjelom preduzeću, sa zaokrugljivanjem na dve decimale,

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

c) po odjeljenjima,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
       ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS  broj
FROM RADNIK
GROUP BY brod$;
```

d) u odjeljenju 10,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca, ROUND(AVG(plata), 2) AS srednja,
       COUNT(*) AS  broj
FROM RADNIK
WHERE brod$=10;
```

najmanja	najveca	srednja	broj
900	3900	1786,1111	18

a)

najmanja	najveca	srednja	broj
900	3900	1786,11	18

b)

najmanja	najveca	srednja	broj	brod$
1000	2400	1271,43	7	10
900	2600	1720	5	20
1100	2800	1666,67	3	30
2750	3900	3216,67	3	40

c)

najmanja	najveca	srednja	broj
1000	2400	1271,43	7

d)

Slika 6.16 Sumarne funkcije i GROUP BY klauzula

Napomjena: U primjeru a) srednja vrijednost je izračunata sa velikim brojem decimala, pa je u ostalim primerima korišćena funkcija za zaokruživanje (ROUND(art, n)) na dvije decimale.

Napomjena: U primjeru c) agregatne funkcije su primjenjene na grupe podataka, po odjeljenjima. Naime, GROUP BY omogućava dobijanje sumarne informacije za svaku različitu vrijednost kolone po kojoj se vrši grupisanje.

Napomjena: U primjeru d) prikazana je mogućnost selektivnog grupisanja na bazi WHERE klauzule. Dakle, GROUP BY klauzula zamjenjuje višestruko pisanje SELECT naredbe sa različitim uslovima.

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 primjenom klauzule **HAVING** (*koji imaju*).

SQL naredbe mogu sadržati aritmetičke izraze sastavljene od funkcija, imena kolona i konstanti povezanih aritmetičkim operatorima (+, -, * i /). Kada u izrazima treba koristiti vrijednosti kolone koja može imati Null vrijednosti onda treba koloni dodjeliti neutralnu vrijednost za željenu operaciju, na primjer pri sabiranju neutralna vrijednost je 0. Dodjela neke vrjednosti koloni koja sadrži Null vrijednosti vrši se funkcijom **NVL(atrib, vrijednost)**. Tako se vrjednost **0** u koloni **premija** dodjeljuje zaposlenima koji nemaju premiju funkcijom **NVL**(*premija*,0). U Access-u se u tu svrhu koristi funkcija NZ(*premija*).

Primjer 23: Izračunaj broj zaposlenih koji obavljaju različite poslove unutar svakog odjeljenja.

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

Primjer 24: Prikaži koje poslove obavlja više od 1 radnika unutar svakog odjeljenja.

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

brod$	posao	broj zaposleni
10	analiticar	1
10	prodavac	1
10	radnik	4
10	upravnik	1
20	analiticar	1
20	savetnik	1
20	upravnik	1
20	vozac	2
30	nabavljac	2
30	upravnik	1
40	direktor	1
40	rukovodila	1
40	savetnik	1

a) razni poslovi po odjeljenjima

brod$	posao	broj zaposleni
10	radnik	4
20	vozac	2
30	nabavljac	2

b) više od jednog radnika obavlja posao

Slika 6.17 Grupisanje po više kolona i dodatna selekcija n-torki koje zadovoljavaju uslov

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), a zatim se one grupišu sa GROUP BY, a onda se dodatno izabiraju grupe klauzulom HAVING. I ključnu riječ HAVING mora slijediti 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.

Primjer 25: Odrediti srednju godišnju platu unutar svakog odjeljenja 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
10	13000
20	19200
30	13200
40	39900

Slika 6.18 Prosječne plate

Primjer 26: Odrediti srednja godišnja primanja unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

a)
```
SELECT brod$, AVG(plata +NVL(premija,0))*12 AS  [prosek primanja], COUNT(*) AS
       [broj zaposlenih], SUM(plata + NVL(premija,0))*12 AS [ukupni prihod]
FROM RADNIK
WHERE posao NOT IN (‘direktor’, ‘upravnik’)
GROUP BY brod$;
```

b) **SELECT** brod$, **AVG**(plata +premija)*12 **AS** [prosek primanja], **COUNT**(*)AS [broj zaposlenih],
SUM(plata + (premija)*12 **AS** [ukupni prihod], **COUNT**(premija) **AS** [sa premijom]
FROM RADNIK
WHERE posao **NOT IN** (‘*direktor’*, ‘*upravnik’*)
GROUP BY brod$;

brod$	prosek primanja	broj zapos	ukupni prihod
10	26400	6	158400
20	38400	4	153600
30	20400	2	40800
40	39960	2	79920

brod$	prosek primanja	broj zapos	ukupni prihod	sa premijom
10	26400	6	158400	6
20	40800	4	122400	3
30	20400	2	40800	2
40	46920	2	46920	1

a) tačan rezultat *b) netačan rezultat*

Slika 6.19 Prosječna primanja po odjeljenjima

U primjeru 26. a) proizvod prosječnih primanja i broja zaposlenih jednak je ukupnom prihodu, dok to nije slučaj u primjeru b) kada je pri izračunavanju prosjeka uzet u obzir samo broj zaposlenih koji imaju premiju (posljednja kolona u izvještaju). Tačnije, pri izračunavanju prosječnih primanja ukupni prihod dijeljen je samo sa brojem onih koji imaju premiju, a ne sa ukupnim brojem zaposlenih u odjeljenju ne uzimajući u obzir direktora i upravnike.

U prethodnim primerima korišćeni specijalni relacioni operatori koji su opšteprihvaćeni: izračunavanje zbirnih podataka, proširivanje i preimjenovanje. Sem ovih neki isporučioci sistema za upravljanje bazama podataka nude i druge dopune. Tako na primjer, Microsoft je autor tri dopune: transform, rollup i cube.

Izračunavanje zbirnih podataka (engl. summarize operator) formira zapise koji sadrže zbirne podatke grupisane na osnovu zadatih polja. Po jedan zapis-red za svaku različitu vrednost grupe polja. Ako je navedeno više od jednog polja za grupisanje, grupe se ugnježduju. Polja navedena u listi polja u iskazu SELECT moraju biti takođe navedena i u listi polja za grupisanje (u iskazu GROUP BY) ili kao argument neke od zbirnih funkcija. Zbirne funkcije navedene u iskazu SELECT ne moraju biti u listi polja za grupisanje u iskazu GROUP BY.

U zbirnim funkcija Null vrednosti se uzimaju u obzir i čine grupu, ali ih zbirne funkcije zanemaruju. Kao posledica se javljaju netačni i neočekivani rezultati (primjer 26). Taj problem se javlja obično ako se jedan atribut u listi polja za grupisanje navede kao argument neke zbirne funkcije.

Zbirni podaci su vrlo korisni kada treba proučavati podatke na višem nivou apstrakcije od onog koji se čuva u bazi podataka.

Proširenje (engl. extend operator) je operator koji omogućava da se definišu i proračunaju nova, virtuelna polja koja se izračunavaju na osnovu vrednosti uskladištenih u bazi podataka. Ovi proračuni mogu biti proizvoljne složenosti a kombinuju se konstante i imena atrubuta pomoću aritmetičkih operacija i funkcija (primjeri 25 i 26).

Preimjenovanje (engl. rename operator) omogućava da se neki atributi, virtuelna polja dobijena primenom operatora proširenja ili tabele nazovu drugim imenom. Preimenovanje povećava razumljivost dobijenih rezultata (primjeri 22-26). Posebno je korisno kada treba ostvariti spajanje tabele sa samom sobom

(samospoj, engl. self join, primjer 34). Ovo omogućava da se svaka upotreba iste tabele definiše kao logički zasebna.

Transform je dopuna koja je postojala samo kod MS Access-u. Transform preuzima rezultate zbirne operacije (GROUP BY) i prikazuje ih rotirane za 90^0. Ova opcija je poznata kao unakrsni upit (engl. crosstab query) i postoji u najnovijoj verziji SQL Server-a. Opšti oblik bloka TRANSFORM je:

TRANSFORM zbirna funkcija

SELECT lista atributa

FROM lista tabela

WHERE lista uslova1

GROUP BY lista atributa za grupisanje

PIVOT zaglavlja atributa 1 [**IN** lista vrednosti]

Klauzula *TRANSFORM zbirna funkcija* definiše zbirne podatke od kojih će se sastojati rezultujući skup zapisa. SELECT blok mora da sadrži klauzulu GROUP BY a ne može da sadrži klauzulu HAVING. Lista atributa u odredbi SELECT i lista atributa za grupisanje u odredbi GROUP BY po pravilu su identične.

Odredba *PIVOT* zadaje polje, odnosno atribut koji će se pojavljivati u zaglavlju kolona u tabeli rezultata. Neobavezna odredba **IN** omogućava da se u rezultujućoj tabeli zadaju imena kolona i redosled pojavljivanja kakav je zadat u listi vrednosti. Bez ove odredbe u rezultujućoj tabeli kolone će biti date abecednim redosledom sleva nadesno.

Rollup operator postoji samo u SQL Server-u a omogućava da se na bazi parcijalnih zbirnih vrednosti izračunatih na osnovu GROUP BY izračunaju ukupni zbirovi. Ovaj operator se dodaje kao proširenje odredbe GROUP BY:

GROUP BY lista atributa za grupisanje WITH ROLLUP.

Cube operator je takođe proširenje odredbe GROUP BY, i omogućava da se izračunavaju zbirni podaci pri svakoj promeni vrednosti u svakoj koloni navedenoj u listi za grupisanje. Slično kao i *rollup* i *cube* nalaže da se izračunaju zbirni podaci za dodatne grupe podgrupa iz liste za grupisanje.

4.2.3 Upiti nad jednom relacijom kao argument u upitu nad drugom relacijom – ugnježdeni upiti

Zajednička karakteristika svih do sada prezentiranih primjera bila je **postavljanje upita uvijek samo nad jednom relacijom,** jer se odgovor

uvijek mogao naći unutar jedne relacije. Manipulisanje informacionim sistemom zahtijeva međutim često dobijanje odgovora za koje upit treba postaviti nad dvije ili više relacija istovremeno. U tom slučaju moraju se koristiti ugnježđeni upiti ili se mora poduzeti operacija spajanja dobijenih tabela (najčešće prirodnog spajanja), sa nekim projekcijama i/ili restrikcijama nad njima.

Pogledajmo u primjerima koji slijede tehniku postavljanja i ovakvih upita.

Pretpostavimo, na primjer, da nam je potrebna sledeća informacija: spisak imena svih zaposlenih u odjeljenju koje je locirano na Dorćolu. To se postiže ulaganjem rezultata jednog upita u WHERE klauzulu drugog upita.

Prvi, unutrašnji upit, bi trebao iz relacije ODJELJENJE dati odgovor koja je šifra odjeljenja (brod#) koje je locirano na *Dorćolu*.

Kada dobijemo da je to odjeljenje čija je šifra brod#=*20*, onda drugim, spoljnim upitom, iz relacije RADNIK tražimo spisak imena zaposlenih u odjeljenju gdje je brod$=*20*.

Primjer 27: Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

I) **SELECT** brod#
FROM ODJELJENJE
WHERE mesto=*'Dorcol'*;

BROD#
20

Odgovor na I) upit

II) **SELECT** ime, brod$
FROM RADNIK
WHERE brod$=20;

ime	brod$
Petar	20
Slobo	20
Bozo	20
Mita	20
Ivan	20

Odgovor na II) upit
Slika 6.20 Spisak zaposlenih na Dorćolu

Ulaganje odgovora jednog-unutrašnjeg upita kao vrednost u WHERE klauzulu drugog-spoljnjeg upita:

III) **SELECT** ime, brod$
FROM RADNIK
WHERE RADNIK.[BROD$] = (**SELECT** ODJELJENJE.[BROD#]
FROM ODJELJENJE
WHERE ODJELJENJE.mesto=*'Dorcol'*);

ime	brod$
Petar	20
Slobo	20
Bozo	20
Mita	20
Ivan	20

Slika 6.21 Spisak zaposlenih na Dorćolu

Primjer 28: Izlistaj ime, posao i platu zaposlenih u odjeljenju *10* koji imaju isti posao kao zaposleni u odjeljenju *plan*.

```
SELECT ime, posao, plata
FROM RADNIK
WHERE  brod$ = 10  AND  Posao  IN  (SELECT posao
                                    FROM RADNIK
                                    WHERE brod$ IN (SELECT brod#
                                                    FROM ODJELJENJE
                                                    WHERE imeod='plan'));
```

Slika 6.22 Zaposleni u odjeljenju plan koji rade iste poslove koji postoje u komercijali

Primjer 29: Ko su najbolje plaćeni radnici u svakom odjeljenju.

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

Napomjena: Spoljašnji i unutrašnji upit mogu biti povezani po vrijednostima više atributa. U tom slučaju ako se uporređuju argumenti koji se sastoje od više atributa, oba argumenta moraju imati jednak broj atributa, a uporređuje se prvi sa prvim, drugi sa drugim itd. Konačno, napomjenimo da atributi koji se upoređuju moraju biti istog, ili kompatibilnog tipa podataka.

Kao što smo to već napomjenuli, *imena atributa mogu se pisati navođenjem i imena relacije kojoj taj atribut pripada, a ovo je neophodno uraditi ako se odlučimo da u raznim relacijama jedne baze podataka koristimo ista imena atributa*.

Složenija pretraživanja, kao što smo to vidjeli u prethodnim primjerima, 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 vrijednosti smještene u memoriju računara. Naredba SELECT daje, naime, kao rezultat virtuelnu relaciju (koja ne egzistira i fizički, na disku, ali je dostupna za vrijeme 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 izvještaju pojavljuju

samo podaci iz jedne relacije, kao što je u svim prethodnim primjerima bio slučaj. Dakle, povezivanje tabela dinamičkom zamjenom rezultata jednog, unutrašnjeg upita u WHERE klauzulu drugog, spoljnjeg upita, može se primijeniti 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.

4.2.4 Spajanje relacija

Spajanje relacija (engl. Join) je operacija kombinovanja podataka iz više relacija koje su nastale u procesu projektovanja baze kao rezultat razdvajanja podataka iz jedne relacije u više relacija sa ciljem eliminisanja redundanse i anomalija pri upisu, brisanju i ažuriranju. Spojevi između relacija razvrstavaju se prema načinu poređenja kolona između kojih je uspostavljena veza i načinu na koji se tumače rezultati poređenja. Formalno gledano, sve vrste spajanja mogu se ostvariti pomoću odgovarajućeg uslova u odredbi WHERE. Vrlo često je to i bolje rešenje jer tada mašina baze podataka lakše optimizuje izvršenje iskaza. Uopšteno gledano spojevi se dijele u dvije grupe:

- Spojevi koji izdvajaju samo zapise za koje je uslov spajanja ispunjen (tačan, True). To su unutrašnji spojevi (engl. Inner Joins).
- Spojevi koji izdvajaju sve zapise kao unutrašnji spojevi plus preostale iz jednog, drugog ili oba skupa. To su takozvani spoljni spojevi (eng. Outer Joins).

Unutrašnji spojevi mogu biti ostvareni po jednakosti (engl. equi join) ili nejednakosti bilo koje vrste (engl. theta join).

Spajanje po jednakosti (**EQUIJOIN**)

Najčešće se koristi spajanje po jednakosti ili jednakovredni spojevi. Pri tome izdvajaju se samo zapisi koji u zadatim poljima imaju jednake vrijednosti. Posebna vrsta spajanja po jednakosti je *prirodno spajanje* koje zadovoljava sljedeće uslove.

- Operator porjeđenja mora biti jednakost.
- U spajanju moraju učestvovati sva polja koja su zajednička za obe relacije.
- U skupu rezultata pojavljuje se samo jedan skup zajedničkih polja.

Neki SUBP, kao na primjer Access kod tabela dobijenih prirodnim spajanjem nude i određene pogodnosti. Access pri spajanju tabela kod kojih postoji veza *jedan prema više* i zajednička polja u rezultatu potiču iz tabele na strani više, automatski se vrši takozvano *preslikavanje redova* (engl. Row Fix-Up) ili automatsku zamenu (engl. AutoLookup). Kada korisnik obrasca unese neku vrijednost za kontrolu vezanu za jedno polje koje se pojavljuje u spoju, Access automatski popuni odgovarajućim vrijednostima kontrole vezane za polja na strani *više*.

Spajanje po nejednakosti, teta spoj (**THETAJOIN**)

Teta spojevi su svi spojevi koji se zasnivaju na operatoru poređenja koji nije jednakost, odnosno na operatorima. <, >, <>, >=, <=. U praksi se rijetko koriste, a najčešće pri pronalaženju zapisa koji sadrže vrijednosti koje veće ili manje od nekih prosječnih ili zbirnih vrijednosti.

Tabela se može spajati i sa samom sobom (**SELF-JOIN**), kada unutar tabele postoji relacija između pojedinih n-torki. Potreba za ovom vrstom spajanja javlja u slučajevima kada postoje unutrašnje veze između pojedinačnih zapisa u jednoj tabeli, unarne veze. To je slučaj sa tabelom RADNIK, jer su neki zaposleni rukovodioci nekim drugim radnicima.

Spoljni spojevi (**OUTER JOIN**)

Po svojoj prirodi mogu biti lijevi (engl. left outer join), desni (engl. right outer join) i potpuni (engl. full outer join). Izdvajaju i kombinuju sve zapise koji zadovoljavaju uslov spajanja plus preostale iz jednog, drugog ili oba skupa. Vrijednosti koje nedostaju (bez parnjaka) zamjenjuju Null vrijednostima.

Lijevi spoljni spoj učitava sve zapise iz skupa na strani *jedan* u vezi tipa *jedan prema više*, dok desni spoljni spoj učitava sve zapise na strani *više*.

Kod nekih SUBP, na primjer Access i SQL Server vrstu spajanja zasniva na redoslijedu kojim su imena relacija navedena u odredbi FROM u iskazu SELECT. Prva tabela je lijeva a druga desna. Samim tim će sledeća dva iskaza dati isti rezultat: sve zapise iz relacije A i samo one zapise iz skupa B koji ispunjavaju uslov (iskaz je tačan):

SELECT * FROM A LEFT OUTER JOIN B ON uslov

SELECT * FROM B RIGHT OUTER JOIN A ON uslov.

Potpuni spoljni spoj, prikazuje sve zapise iz obe relacije, pri čemu one koji zadovoljavaju uslov kombinuje. Ako neki SUBP ne podržava potpuni spoljni spoj, on se može simulirati pomoću unije lijevog i desnog spoljnog spoja.

Primjer 30: Izlistaj ime, posao, ime odjeljenja i mjesto gdje rade svi zaposleni.

```
SELECT ime,  posao, RADNIK.[brod$], ODJELJENJE.[brod#], imeod, mesto
FROM ODJELJENJE,  RADNIK
WHERE ODJELJENJE.[brod#] = RADNIK.[brod$];
```

U Access-u:
```
SELECT RADNIK.IME, RADNIK.POSAO, RADNIK.[BROD$], ODJELJENJE.[BROD#],
       ODJELJENJE.IMEOD, ODJELJENJE.MESTO
FROM ODJELJENJE INNER JOIN RADNIK ON ODJELJENJE.[BROD#] = RADNIK.[BROD$];
```

IME	POSAO	BROD$	BROD#	IMEOD	MESTO
Aco	radnik	10	10	komercijala	Novi Beograd
Vaso	prodavac	10	10	komercijala	Novi Beograd
Jovo	upravnik	10	10	komercijala	Novi Beograd
Miro	radnik	10	10	komercijala	Novi Beograd
Tomo	radnik	10	10	komercijala	Novi Beograd
Jovan	radnik	10	10	komercijala	Novi Beograd
Marko	analiticar	10	10	komercijala	Novi Beograd
Petar	vozac	20	20	plan	Dorcol
Slobo	vozac	20	20	plan	Dorcol
Bozo	upravnik	20	20	plan	Dorcol
Mita	savetnik	20	20	plan	Dorcol
Ivan	analiticar	20	20	plan	Dorcol
Pavle	upravnik	30	30	prodaja	Stari Grad
Andro	nabavljac	30	30	prodaja	Stari Grad
Pero	nabavljac	30	30	prodaja	Stari Grad
Simo	savetnik	40	40	direkcija	Banovo Brdo
Savo	direktor	40	40	direkcija	Banovo Brdo
Janko	rukovodila	40	40	direkcija	Banovo Brdo

Slika 6.23 Spajanjem tabela dobija se semantički bogatija relacija

Primjer 31: Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

```
SELECT ime, brod$
FROM RADNIK, ODJELJENJE
WHERE RADNIK.[brod$]= ODJELJENJE.[brod#]
        AND mesto='Dorcol' ;
```

IME	BROD$
Petar	20
Slobo	20
Bozo	20
Mita	20
Ivan	20

Slika 6.24 Spajanjem tabela dobija se isti rezultat

Napomjena: Rezultat je naravno isti kao u primjeru 27, ali je postupak dobijanja različit, i u ovom slučaju bi vrijeme dobijanja odgovora bilo značajno duže.

U svim prethodnim primjerima u klauzuli WHERE korišćeno je spajanje po jednakosti (ODJELJENJE.[brod#] = RADNIK.[brod$]), ali moguće je spajanje po bilo kom operatoru poređenja. Ključnu riječ WHERE, u naredbi SELECT moraju da slijede 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 Dekartov proizvod (**CARTESIAN JOIN**) relacija navedenih u listi relacija. To je poznati Dekartov proizvod dva skupa. Tako, na primjer naredba:

SELECT * FROM A, B;

daje Dekartov proizvod relacija **A** i **B**. Ako relacija A ima 1000 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 vrlo veliki broj zapisa. Sada slijedi 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 preko operacije prirodnog spajanja to se ne preporučuje zbog drastičnog povećanja vremena obrade upita.

Neka je u bazu dodat još jedan zaposleni koji još nije raspoređen ni u jedno odjeljenje:

IDBR#	IME	POSAO	KVALIF	RUKOVODILAC	DATZAP	PREMIJA	PLATA	BROD$
7891	Mirko	analiticar	VSS		29-jun-02	3000	2000	

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. Ako hoćemo da u odgovor uključimo i podatke iz druge tabele, koji nemaju parnjaka u prvoj tabeli, to se zove desno spajanje (**RIGHT JOIN**), a obrnuto je levo spajanje (**LEFT JOIN**).

Primjer 32: Izlistaj sve podatke o odjeljenjima i radnicima za radnike čija je premija veća od 2000 .

```
SELECT *
FROM ODJELJENJE, RADNIK
WHERE ODJELJENJE.[brod#] = RADNIK.[brod$](+) AND premija >2000;
```

U Access-u:

```
SELECT *
FROM ODJELJENJE RIGHT JOIN RADNIK ON ODJELJENJE.[BROD#] = RADNIK.[BROD$]
WHERE (((RADNIK.PREMIJA)>2000));
```

BROD#	IMEOD	MESTO	IDBR#	IME	POSAO	KVALIF	RUKOV	DATZAP	PREMIJA	PLATA	BROD$
10	komercijala	Novi Beograd	6234	Marko	analiticar	VSS	5786	17-dec-90	3000	1300	10
40	direkcija	Banovo Brdo	7890	Ivan	analiticar	VSS	5786	17-dec-90	3200	1600	40
			7891	Mirko	analiticar	VSS		29-jun-02	3000	2000	

Slika 6.25 U odgovoru su i podaci o neraspoređenom radniku

Primjer 33: Izlistaj sve podatke o odjeljenjima i radnicima za odjeljenja čija imena počinju slovima **d** i **e**.

```
SELECT *
FROM ODJELJENJE, RADNIK
WHERE (ODJELJENJE.[brod#](+) = RADNIK.[brod$]) AND
      (imeod BETWEEN 'd%' AND 'f%');
```

U Access-u:

```
SELECT *
FROM ODJELJENJE LEFT JOIN RADNIK ON ODJELJENJE.[BROD#] = RADNIK.[BROD$]
WHERE ODJELJENJE.[IMEOD] BETWEEN 'd*' AND 'f*';
```

BROD#	IMEOD	MESTO	IDBR#	IME	POSAO	KVALIF	RUKOV	DATZAP	PREMIJA	PLATA	BROD$
40	direkcija	Banovo Brdo	5867	Simo	savetnik	VSS	5842	08-avg-70		2750	40
40	direkcija	Banovo Brdo	5842	Savo	direktor	VSS		15-dec-81		3000	40
40	direkcija	Banovo Brdo	7890	Ivan	analiticar	VSS	5786	17-dec-90	3200	1600	40
40	direkcija	Banovo Brdo	6789	Janko	rukovodila	VS		23-dec-99	10	3900	40
50	erc	Zemun									

Slika 6.26 U odgovoru su i podaci o odjeljenju bez radnika

Primjer 34: Prikaži imena, posao i broj odjeljenja radnika kojima je rukovodilac *Savo*.

```
SELECT R.ime AS [ime radnika], R.posao, ŠEF.ime AS [ime šefa], R.brod
FROM RADNIK R,  RADNIK ŠEF
WHERE ŠEF.idbr = R.rukovodilac AND ŠEF.ime="Savo";
```

self join : Select Query

ime radnika	POSAO	ime šefa	BROD
Jovo	upravnik	Savo	10
Bozo	upravnik	Savo	20
Pavle	upravnik	Savo	30
Simo	savetnik	Savo	40
Mita	savetnik	Savo	20

Slika 6.27 U odgovoru su i podaci o odjeljenju bez radnika

Primjer 35: Izlistaj šifre radnika koji rade na dva i više projekta.

```
SELECT Učešće.Idbr#, COUNT(Idbr) AS [Broj projekata]
FROM Učešće
GROUP BY Učešće.Idbr#
HAVING (((Count(*))>=2))
ORDER BY Učešće.Idbr#;
```

Idbr	Broj projekata
5652	2
5932	3
5953	2
6234	3

Slika 6.28 Šifre radnika koji rade na više projekata

Primjer 36: Izlistaj broj sati, šifru, ime i platu radnika koji rade na dva i više projekta.

```
SELECT UČEŠĆE.idbr#, sum(UČEŠĆE.brsati) AS [broj sati], RADNIK.idbr#,
       RADNIK.ime, RADNIK.plata, Count(*) AS [broj projekata]
FROM RADNIK, UČEŠĆE
WHERE RADNIK.IDBR# = UČEŠĆE.IDBR#
GROUP BY radnik.idbr#, RADNIK.IME
HAVING (((Count(*))>1));
```

UCESCE.IDBR	broj sati	RADNIK.IDBR	IME	plata	broj projekata
5652	2000	5652	Jovan	1000	2
5932	2000	5932	Mita	2600	3
5953	2000	5953	Pero	1100	2
6234	2000	6234	Marko	1300	3

Slika 6.29 Imena i plate radnika koji rade na više projekata

Primjer 37: Izlistaj šifru radnika i broj sati na projektu za radnike koji rade na projektu *uvoz*.

```
SELECT DISTINCT UČEŠĆE.idbr#, UČEŠĆE.brsati, UČEŠĆE.brproj#, PROJEKAT.brproj#,
                PROJEKAT.imeproj
FROM PROJEKAT, UČEŠĆE
WHERE PROJEKAT.brproj# = UČEŠĆE.brproj# AND
                (UČEŠĆE.brproj# = (SELECT PROJEKAT.brproj#
                                  FROM PROJEKAT
                                  WHERE PROJEKAT.imeproj = 'uvoz'));
```


IDBR	BRSATI	UCESCE.BRPR	PROJEKAT.BRPI	IMEPROJ
5786	2000	100	100	uvoz
5842	2000	100	100	uvoz
5900	2000	100	100	uvoz
5953	1000	100	100	uvoz
5652	1000	100	100	uvoz
5932	500	100	100	uvoz
6234	500	100	100	uvoz

Slika 6.30 Izvještaj o radnicima i broju sati na projektu 'uvoz'

Primjer 38: Izlistaj šifru odjeljenja kao i šifru, ime, platu i posao najbolje plaćenog radnika u svakom odjeljenju.

SQL - **SELECT** RADNIK.brod# **AS** BROD, **first**(RADNIK.idbr#) **as** [šifra], **first**(RADNIK.ime) **As** [ime], **Max**(RADNIK.plata) **AS** [najveća plata], **first**(RADNIK.posao) **as** [posao]
FROM RADNIK
GROUP BY RADNIK.brod;

BROD	šifra	ime	najveća plata	posao
10	5497	Aco	2400	radnik
20	5367	Petar	2600	vozac
30	5786	Pavle	2800	upravnik
40	5867	Simo	3900	savetnik

Slika 6.31 Šifra, ime, plata i posao najbolje plaćenog radnika u svakom odjeljenju

Napomjena: U prethodnom primjeru predikat prvi *First* obezbjeđuje da se u odgovoru pojavi samo prva n-torka ako ih ima više sa istom (najvećom) platom. U narednom primjeru nije poštovana konvencija da se ključne riječi SQL jezika pišu velikim slovima. To naravno nije dovelo do pojave grešaka jer SQL nije osjetljiv na velika i mala slova. Dakle **as**, **As** i **AS** su ista riječ, 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.

Primjer 39: Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.

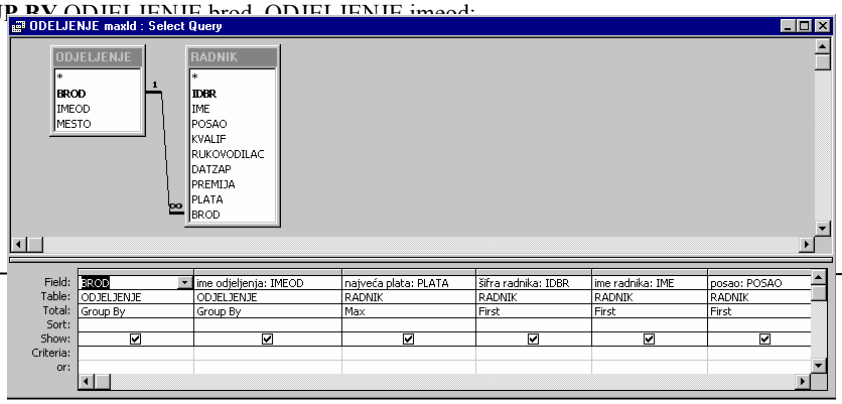
SQL

SELECT ODJELJENJE.brod#, imeod **AS** [ime odjeljenja], **Max**(plata) **AS** [najveća plata], **First**(RADNIK.idbr) **AS** [šifra radnika], **First**(ime) **AS** [ime radnika], **First**(posao)
FROM ODJELJENJE, RADNIK
WHERE ODJELJENJE.brod# = RADNIK.brod$
GROUP BY ODJELJENJE.brod#, imeod;

Access

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

QBE –



gornje okno

donje okno

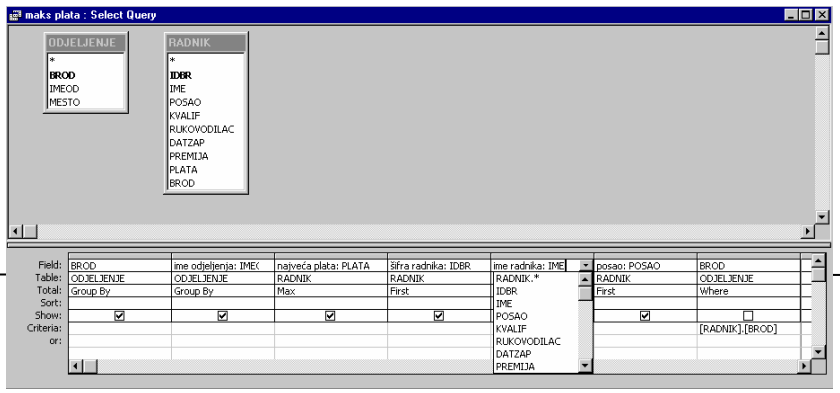
Dvodimenzionalni upit

BROD	ime odjeljenja	najveća plata	šifra radnika	ime radnika	posao
10	komercijala	2400	5497	Aco	radnik
20	plan	2600	5367	Petar	vozac
30	prodaja	2800	5786	Pavle	upravnik
40	direkcija	3900	5867	Simo	savetnik

Slika 6.32 Rezultat sva tri upita je isti

Napomjena: Na slici 6.32. a) prikazan je jedan alat u Access-u koji se zasniva na izradi ***upita po primjeru*** (engl. **Q**uery **B**y **E**xample, **QBE**). Za razliku od programskih i drugih upitnih jezika ovaj jezik ima dvodimenzionalnu sintaksu (tabela sa kolonama i vrstama). Druga važna osobina jeste da je sam upit izražen ''primjerom'' (**''by example''**). Umjesto da je data procedura za dobijanje željenog odgovora, korisnik daje primjer kako hoće da bude izračunat željeni odgovor, korisnik daje primjer tabele koju želi da dobije. Upit se u QBE-u izražava pomoću kostura tabele koja daje relacionu shemu. Tačnije, korisnik izabira ovaj kostur iz postojećih tabela. Dakle, najprije izabere tabele (gornje okno), a onda iz pojedinih tabela atribute (donje okno). Kada izabere atribute onda za svaki od njih po potrebi definiše skup željenih vrijednosti – domena, tj. uslova (polje **Criteria**). Zbog toga se ovaj metod smatra relacionim računom domena. Unutar jednog domena može se tražiti zadovoljenje jednog od više uslova (**OR**), ili istovremeno zadovoljenje uslova u raznim domenima (**AND**). Između tabela mogu se praviti relacije – prirodno spajanje (kao na slici 6.32.), ili se te relacije prave preko uslova za neki atribut (kao na slici 6.33. preko atributa brod# u tabeli ODJELJENJE i atributa brod$ u tabeli RADNIK). Ovi jezici nemaju mogućnost kreiranja ugnježdenih upita već je samo moguće izvršiti spajanje tabela.

Primjer 40: Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje



plaćenog radnika u svakom odjeljenju.

Slika 6.33 Izbor atributa i postavljanje ograničenja za skup dozvoljenih vrijednosti

Napomjena: Rezultat ovog *upita po primjeru* je isti kao na slici 6.32 b).

Napomjena: Standardne, ugrađene funkcije izabiraju se u polju **Total**, u polju **Sort** se zadaju uslovi za uređivanje (sortiranje) odgovora. Tabele se definišu u polju **Table**, a polja se izabiraju iz padajućih listi u polju **Field**. Neka polja su neophodna radi zadavanja uslova za grupisanje ili spajanje tabela ali nisu neophodna u izvještaju pa se mogu učiniti nevidljivim. To se radi preko polja **Show**, a u primjeru na slici 6.33. poslednje polje preko kojeg je ostvareno spajanje tabela nije vidljivo u izvještaju.

4.2.5 Ažuriranje baze podataka

Dodavanje, izmjena (ažuriranje u užem smislu) i brisanje podataka vrši se naredbama **INSERT**, **UPDATE** i **DELETE**. Kod primjene 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 a izvodi se naredbom **INSERT**. Opšti oblik glasi:

INSERT INTO ime relacije **[**lista atributa**] VALUES [**lista *vrijednosti* **]**

uz napomenu da se **ovom naredbom odjednom u relaciju mogu unijeti vrijednosti atributa samo jedne n-torke**. Podrazumijeva se dodavanje na kraju tabele, jer redoslijed navođenja n-torki u relaciji nije bitan.

Postoje tri tipa ovih naredbi:

1. za ubacivanje vrijednosti SVIH atributa jedne n-torke,
2. za ubacivanje vrijednosti NEKIH atributa jedne n-torke,
3. za ubacivanje podataka iz jedne tabele u drugu.

1. *Za ubacivanje vrijednosti SVIH atributa jedne n-torke, nije potrebno specificirati nazive atributa.*

Primjer 41: U relaciju **RADNIK** dodaj podatke o novom zaposlenom koji ima šifru **7891,** ime mu je "**Mirko"**, radi na poslovima **analitičara**, ima visoku stručnu spremu (**VSS**), još nema šefa(Null), počeo je da radi 29-jun-02, ima premiju 3000, platu 2000 i još nije raspoređen ni u jedno odjeljenje (Null).

```
INSERT INTO RADNIK VALUES
( 7891, Mirko, analiticar, VSS, Null, 29-jun-02, 3000, 2000, Null);
```

Napomjena: Ukoliko podatak za neki od atributa nije poznat, kao podatak se unosi vrijednost ***Null***. Redosled podataka u listi VALUES mora biti isti kao u definiciji tabele.

2. *Za ubacivanje vrijednosti NEKIH atributa jedne n-torke, potrebno je specificirati nazive atributa i njihove vrijednosti i to u identičnom poretku.*

Primjer 42: U relaciju RADNIK dodaj podatke o novom zaposlenom, ime mu je "**Mirko"**, koji ima šifru **7891,** ima 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);
```

3. *Za ubacivanje vrijednosti 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 vrijednosti (pomoću naredbe SELECT) i to u identičnom poretku.*

Primjer 43: 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 ('analiticar', 'vozac');
```

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 nepromijenjeni.

Brisanje podataka u relaciji može se izvesti pojedinačno, ili grupno. Komandom **DELETE** uvijek se briše cijela n-torka, a ne samo pojedina vrijednost nekog atributa. Sintaksa naredbe u opštem slučaju glasi:

```
DELETE FROM ime relacije [ WHERE lista uslova ]
```

Primjer 44: U relaciji RADNIK obriši sve podatke o radniku **idbr# =** *5953* koji je otišao u penziju.

```
DELETE FROM RADNIK WHERE idbr# = 5953;
```

Ako nije naveden uslov (WHERE) brišu se svi podaci. Iz relacije RADNIK mogu se obrisati svi oni koji su se zaposlili prije *17-feb-90*.

Primjer 45: Izbrisati podatke o radnicima koji su se zaposlili posle 17-feb-90.

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

Iz relacije RADNIK mogu se obrisati svi oni koji su radili u odjeljenju *priprema*.

Primjer 46: Izbrisati podatke o svim radnicima koji rade u odjeljenju *priprema*.

```
DELETE FROM RADNIK
WHERE (RADNIK.brod$) = (SELECT brod#
                        FROM ODJELJENJE
                        WHERE imeod= 'priprema');
```

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

```
UPDATE ime relacije
SET atribut1=vrijednost1, atribut2=vrijednost2,
WHERE  [ lista uslova ]
```

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

Primjer 47: Šef zaposlenog čija je šifra **idbr# = *5932*** je rukovodilac čija šifra je *5842,* a naredba glasi:

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

Ali ovom se naredbom može mjenjati i vrijednost jednog podatka unutar više n-torki.

Primjer 48: Prebaci sve zaposlene iz odjeljenja 30 u odjeljenje 20, a naredba glasi:

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

Napomjena: Pri upotrebi naredbi za ažuriranje treba biti vrlo oprezan jer one mjenjaju stanje baze, tj. vrijednosti podataka. Zbog toga se preporučuje da se najprje pomoću obične **SELECT** naredbe izdvoje n-torke na koje bi se primjenile naredbe za ažuriranje, pa kada nakon analize rezultata nepobitno utvrdimo da se radi o željenoj grupi n-torki tek onda kreirati odgovarajuću naredbu **UPDATE** ili **DELETE**. Access u tu svrhu ima takozvane akcione upite pomoću kojih se mogu ne samo ažurirati podaci već i kreirati pogledi i tabele.

4.2.6 Operatori za rad sa skupovima

Operatori unije (engl. relational union), presjeka (engl. relational intersection), razlike (engr. relational difference) i Dekartov proizvod se zasnivaju na teoriji skupova, ali su donekle izmijenjeni jer se primjenjuju nad relacijama. Da bi se primjenile prve tri operacije relacije moraju biti union-kompatibilne, odnosno moraju imati isti broj atributa i odgovarajući atributi moraju imati iste korespondentne domene. Čak je dovoljno da su atributi istog tipa.

Unija je u suštini relaciona verzija sabiranja. Rezultat je relacija koja sadrži sve zapise jedne na koju su dodati zapisi druge relacije, ali su duplikati eleminisani. Unija omogućava dodavanje novih zapisa u relaciju.

Presjek je operacija koja vraća zapise koji su zajednički u obije relacije i kojom se u stvari pronalaze duplikati. Presjek se realizuje pomoću spoljnjeg spajanja. Neka postoji tabela RADNIK20 koja sadrži zapise o zaposlenima u odjeljenju 20, sa izmjenjenim podacima o radniku čiji *idbr* je 7890 (slika 6.34).

radnik20

IDBR	IME	POSAO	KVALIF	RUKOVODILAC	DATZAP	PREMIJA	PLATA	BROD
5367	Petar	vozac	KV	5780	1.1.1978	1900	1300	20
5780	Bozo	upravnik	VSS	5842	11.8.1984		2200	20
5900	Slobo	vozac	KV	5780	3.10.1978	1300	900	20
5932	Mita	savetnik	VSS	5842	25.3.1965		2600	20
7890	Ivanka	analiticar	VS	5786	17.12.1990	3200	1600	

Slika 6.34 Sa pogledom ODJELJENJE20

Presjek relacija RADNIK i RADNIK20 se realizuje kao lijevo spajanje iz kojeg su isključeni oni zapisi kod kojih je vrijednost *radnik20.idbr* jednaka Null.

Primjer 49: Kreirati presjek relacija RADNIK i RADNIK20.

```
SELECT R.IDBR#, R.IME, R.BROD$, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON RADNIK.IDBR# = radnik20.idbr
WHERE radnik20.idbr IS NOT NULL;
```

IDBR#	IME	BROD$	IDBR	IME	BROD
5367	Petar	20	5367	Petar	20
5900	Slobo	20	5900	Slobo	20
5780	Bozo	20	5780	Bozo	20
5932	Mita	20	5932	Mita	20
7890	Ivanka	20	7890	Ivanka	

Slika 6.35 Presjek dveju relacija

Odgovor na ovaj upit prikazan na slici 6.35 sadrži sve zapise iz relacije RADNIK20, bez obzira što vrijednosti neprimarnih atributa nijesu iste u obje relacije.

Razlika relacija sadrži one zapise iz prve relacije koji ne postoje u drugoj relaciji. Razlika relacija RADNIK i RADNIK20 se realizuje kao lijevo spajanje iz kojeg su isključeni oni zapisi kod kojih je vrijednost *radnik20.idbr* nije jednaka Null. Operator spoljnjeg spajanja učitava sve zapise iz obe tabele i postavlja Null vrijednosti u polja koja nemaju odgovarajuće parnjake u drugoj (ovog puta lijevoj) relaciji (slika 6.36). Operator IS NULL u odredbi WHERE izdvaja iz ovog skupa samo one zapise u kojima su Null vrijednosti (odnosno one bez parnjaka, slika 6.37).

Primjer 50: Kreirati razliku relacija RADNIK i RADNIK20.

```
SELECT R.IDBR#, R.IME, R.BROD$, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON RADNIK.IDBR# = radnik20.idbr
WHERE radnik20.idbr IS NULL;
```

IDBR#	IME	BROD$	IDBR	IME	BROD
5367	Petar	20	5367	Petar	20
5497	Aco	10			
5900	Slobo	20	5900	Slobo	20
5519	Vaso	10			
5662	Jovo	10			
5696	Miro	10			
5780	Bozo	20	5780	Bozo	20
5786	Pavle	30			
5867	Simo	40			
5842	Savo	40			
5874	Tomo	10			
5898	Andro	30			
5932	Mita	20	5932	Mita	20
5953	Pero	30			
5652	Jovan	10			
6234	Marko	10			
7890	Ivan	20	7890	Ivanka	
6789	Janko	40			

Slika 6.36 Lijevo spajanje relacija

IDBR#	IME	BROD$	IDBR	IME	BROD
5497	Aco	10			
5519	Vaso	10			
5662	Jovo	10			
5696	Miro	10			
5786	Pavle	30			
5867	Simo	40			
5842	Savo	40			
5874	Tomo	10			
5898	Andro	30			
5953	Pero	30			
5652	Jovan	10			
6234	Marko	10			
6789	Janko	40			

Slika 6.37 Presjek dveju relacija

Napomjena: Slika 6.35 i slika 6.37 pokazuju da su dva zapisa ista ako su im jednaki primarni ključevi, a vrijednosti ostalih atributa mogu biti različite. Imena odgovarajućih atributa ne moraju biti ista, ali su njihovi tipovi isti ili kompatibilni.

6.3 Kreiranje i korišćenje pogleda (VIEW)

Koncept baza podataka daje mogućnost da više raznih 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, prikaz 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 uvijek ako nam zatreba, 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.

Pogled je prema tome virtuelna imenovana relacija koja se takođe kreira naredbom SELECT**,** ali se u bazi podataka memoriše na specifičan način. Definicija pogleda se čuva 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. Kako ažuriranje (dakle izmjena podataka) ovakve relacije nije moguće (što je i logično), to se rad sa VIEW relacijama koristi često i kao način zaštite integriteta podataka. Treba odmah napomenuti da mnogi RDBMS nemaju mogućnost rada sa pogledima (na primjer Access). Naredba za kreiranje pogleda - VIEW relacije glasi:

CREATE VIEW ime_pogleda **[** atribut1, atribut2, ….. **]** **AS**
SELECT ……..

gdje je **ime_pogleda** naziv novoformirane virtuelne relacije **VIEW**, a navedeni atributi njeni atributi. Naravno, nije potrebno ni naglašavati da u sistemu ne smije da postoji još neka bazna (ili virtuelna) relacija sa istim imenom. Na primjer, za rukovodioca odjeljenja 20 nisu značajni podaci o svim radnicima već samo o onima iz njegovog odjeljenja. U tu svrhu iz relacije ***RADNIK*** < ***idbr#, ime, ….*** > treba izdvojiti samo one zaposlene koji su od interesa:

Primjer 51: Kreirati pogled ODJELJENJE20 koje sadrži podatke o odjeljenju i ime, posao, kvalifikaciju i platu zaposlenih u odjeljenju 20:

CREATE VIEW ODJELJENJE20 **AS**
SELECT O.imeod, O.imeod, R. ime, R.posao, R.kavalif, R.plata
FROM RADNIK R, ODJELJENJE O
WHERE O.brod#=R.brod$ **AND** O.brod#=*20*;

Napomjena: U ovome primjeru nisu uvedeni novi atributi, iako ih **VIEW** relacija može imati. Imena atributa u relaciji **VIEW,** i osnovnim relacijama iz kojih je izvedena, su u ovome primjeru ostala ista.

Svaki put kada korisnik hoće da obrađuje podatke o zaposlenima u svom odjeljenju on jednostavno koristi ovaj **VIEW**.

Primjer 52: Prikaži ime, posao, kvalifikaciju i mjesto gde rade zaposleni sa visokom stručnom spremom u odjeljenju 20:

a)
SELECT O20.mesto, O20.ime, O20.posao, O20.kvalif
FROM ODJELJENJE20 O20
WHERE O20.kvalif="*VSS*"**;**

b)
SELECT mesto, ime, posao, kvalif
FROM ODJELJENJE O **INNER JOIN** RADNIK R **ON** O.brod = R.brod
WHERE (((RADNIK.kvalif)="*vss*") **AND** ((RADNIK.brod)=*20*))**;**

MESTO	IME	POSAO	KVALIF
Dorcol	Bozo	upravnik	VSS
Dorcol	Ivan	analiticar	VSS
Dorcol	Mita	savetnik	VSS

Slika 6.38 Sa pogledom ODJELJENJE20 radimo kao sa fizičkom tabelom

Napomjena: Isti rezultat (slika 6.38) dobija se i upitom a) i upitom b) u ovom primjeru.

Napomjena: Ažuriranje baze podataka preko pogleda ima brojna ograničenja posebno ako je pogled definisan nad više tabela. Najkraće rečeno, da bi 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.

Napomjena: Pogledi objezbjeđuju:

- *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*, mijenjaju se definicije pogleda a ne aplikacioni programi koji koriste podatke iz baze podataka preko pogleda.

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.

6.4 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 zlonamjernog korišćenja, izmjene 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 rješavaju 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 provjera i koje akcije treba preduzeti kada definisani uslovi nisu ispunjeni. O ovome se vodi računa pri definisanju 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 mjenjaju od slučaja do slučaja, ona moraju biti podržana i u samim aplikacijama.

Pravila integriteta se djele na dvije 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.

Termin sigurnost podataka odnosi se na mehanizme zaštite baze podataka od neovlašćenog korišćenja. Sigurnost podataka ima mnogo aspekata (kriptografija, zaštita pri prenosu, fizičko obezbjeđenje, itd.) od kojih će biti opisana samo zaštita od neovlašćenog korišćenja koju pružaju softveri za upravljanje bazama podataka.

Najrasprostranjeniji princip sigurnosti podataka je dodjela, tj., ograničavanje prava pristupa i korišćenja. Obično se svakom korisniku dodjeljuju 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 dodjeli prava korišćenja pomoću naredbe GRANT. Opšti oblik naredbe GRANT je:

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

Drugim korisnicima vlasnik može dodjeliti sva prava (**ALL**) ili samo ona iz liste. Ako dozvoljavamo korisniku samo da gleda podatke treba mu dodjeliti pravo **SELECT**, a ako može i da ih briše onda i pravo **DELETE**. Promjena 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 **...]}.**

Koncept baze podataka daje prave efekte 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 pristupi istom ili istim podacima sa ciljem čitanja ali i izmjene podataka. U tom slučaju može doći do pojave pogrešnih rezultata i ažurnost i integritet podataka mogu biti ugroženi. Da bi se spriječile 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 smije 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 izmjene 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.

Sve promjene nad bazom podataka izazvane SQL naredbama najčešće se odražavaju samo na stanje podataka u operativnoj memoriji korisnikovog računara ili servera. Ako želimo da se izmjene odraze na stvarne podatke na disku (server) potrebno je potvrditi ove promjene, 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 (izmjena velikog broja podataka).

Svaka izmjena podataka u bazi podataka evidentira se u dnevniku transakcija. Ako recimo nestanak struje prekine tekuću transakciju, samo dio izmjena biće zapisan na disku, a neki podaci će zadržati staru vrjednost. Naravno da ovo nikako ne smje da se desi, pa se zato čitava transakcija mora poništiti (na osnovu dnevnika transakcija) prilikom prvog narednog pokretanja sistema za upravljanje bazama podataka.

Oporavak baze podataka (**RECOVERY**) predstavlja proces vraćanja baze podataka u stanje za koje se zna da je korektno, 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 primjeraka-kopija jednog te istog podatka na nekom od memorijskih uređaja (disku ili traci). Proces oporavka može se opisati na sledeći način:

- periodično se cjela baza podataka **kopira** (*dump, backup*) na neku arhivsku memoriju,
- za svaku promjenu u bazi u takozvani ***log file*** (*žurnal*) zapisuje 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 promjene, a same transakcije se ponove.

Na kraju možemo reći da SQL ima znatno veće mogućnosti od onih koje pruža relaciona algebra i ima snagu koju obezbeđuje relacioni račun. SQL podržava pet osnovnih operatora relacione algebre: uniju, razliku, Dekartov odnosno Kartezijev proizvod (koji je predstavljen klauzulom FROM), projekcija se izvršava u SELECT klauzuli a uslovi (predikatski račun) za selekciju, tj. izdvajanje n-torki sadržani su u WHERE klauzuli.

SQL dopušta da se međurezultati smeštaju u privremene relacije, i da se zadaju, odnosno kodiraju proizvoljni izrazi relacione algebre.

SQL pruža znatno veće mogućnosti od relacione algebre, a neke od njih su: agregatne funkcije, sortiranje, itd.

Mnoge SQL implementacije omogućavaju da SQL upiti budu deo programa napisanih u jezicima opšte namjene 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.

6.5 Aplikativni programi

Aplikativni programi služe isključivo i prvenstveno za “približavanje” informacionog sistema korisniku. Svaki potencijalni korisnik (blagajnik ili magacioner u nekom preduzeću, na primjer) ne mora, konačno i ne može,

poznavati do u detalje cijeli informacioni sistem, jer koristi samo jedan njegov dio i to na standardan, uvijek isti način. Magacioner u skladištu jedne velike firme (podrazumijeva se da ima još magacionera i još skladišta) evidentira samo podatke o pristigloj i isporučenoj robi iz svoga magacina. Njega prema tome ne interesuje ni koncepcija baze podataka, ni ostale mogućnosti informacionog sistema, za njega je važno da jednostavno, brzo i pouzdano može da unese podatke za koje je odgovoran i dobije samo one informacije koje on treba.

Srž svakog aplikativnog programa je prema tome neki, ka manipulisanju podacima okrenut programski paket, oko koga se onda pravi programsko okruženje koje omogućava korisniku da radi samo određene operacije bez poznavanja **SQL**-a ili nekog drugog "višeg" jezika, i bez poznavanja detaljne organizacije informacionog sistema.

Softver koji omogućava izradu takvih programa (koji sa stanovišta projektanta informacionog sistema predstavlja samo radno okruženje kojim on informacioni sistem "približava" neukom korisniku ne povećavajući njegove mogućnosti u bilo kom smislu) nalazi se danas na tržištu u više varijanti, u zavisnosti od toga za koje računarske sisteme je predviđen.

Kako su mogućnosti PC-a postale izuzetno velike (prije samo petnaestak godina i najveći računski centri bili su opremljeni lošije od nekog danas bolje opremljenog PC-a posljednje generacije), to je i većina tih programa prilagođena za korišćenje na PC-u.

Jedan od prvih programskih paketa, prilagođen i skromnim konfiguracijama personalnih računara koji rade pod operacionim sistemom DOS, bio je dBase (verzija III, III+, IV), nakon njega stekao je popularnost Clipper i FoxPro (opet u nekoliko razvojnih varijanti), koji je predviđen i za WINDOWS okruženje, čime su mogućnosti prezentacije rezultata (posebno grafičkih) umnogome poboljšane, slijede, takozvane vizuelne varijante (Visual BASIC, Visual FoxPro, Access) kod kojih su mogućnosti prezentacije rezultata dovedene skoro do savršensva.

Kod većih računarskih sistema u početku je dominirao programski jezik COBOL (COBOL je bio namijenjen za hijerarhiske sisteme), koji je razvojem relacionih baza **podataka** i naročito uvođenjem distribuiranih baza podataka, danas skoro potpuno istisnut iz primjene novim softverskim paketima od kojih su najpoznatiji: ORACLE, SQL Server, INFORMIX, SYBASE, INGRES itd.

S obzirom na to da su personalni računari danas po svojim mogućnostima potpuno uporedivi sa velikim računskim centrima od prije 10 do 15 godina, to se pomenuti sistemi za upravljanje relacionim bazama podataka mogu danas implementirati i na PC.

Isto tako, prvobitne verzije paketa, rađene isključivo za PC (dBase III na primjer), u svojim novijim izdanjima (dBase for Windows na primjer) imaju pored sopstvenih naredbi za manipulisanje podacima ugrađen i SQL kao standard u ovoj oblasti.

Konačno, i programski jezici opšte namjene, kao što su na primjer PASCAL ili C++, mogu da prihvataju i tzv. **DBF** fajlove, to jest tabelarno sređene podatke (dakle relacije), pa uz implementaciju nekog upitnog jezika (SQL-a, na primjer) manipulacija podacima i pisanje aplikativnih programa postaju mogući i na taj način.

Uobičajeni postupak razvoja i pisanja aplikacije sastoji se od:

- pisanja programa u višem programskom jeziku (koji ima "ugrađene" naredbe upitnog jezika),
- "prevođenja i povezivanja" (kompilacije i linkovanja) napisanih programa,
- testiranja programa i ispravljanja grešaka, te
- pravljenja verzije koju će korisnik moći da koristi.

Prema tome, pisanje aplikacija, ima sledeće dobre strane:

- korisnik ne mora da poznaje upitni jezik,
- korisnik ne mora da poznaje konfuguraciju baze podataka,
- mogućnost slučajnih grešaka je svedena na minimum,
- bezbjednost podataka je povećana i
- olakšan je prenos programa (jer korisnik ne mora da vodi računa o kompatibilnosti)

ali, nažalost i jednu vrlo lošu:

- korisnik, naime, može da uradi samo ono što je unapred predviđeno, tj. što mu aplikacija dozvoli**.**

6.6 Primjer razvoja aplikacije

Pokažimo na primjeru razvoja jednostavne aplikacije šta se dobija, a šta gubi. Pri tome se nećemo ograničiti ni na jedan određeni programski jezik nego pokušati objasniti sintezu aplikacije na nivou logike programiranja. Pretpostavimo da je potrebno za službenika na šalteru sekretarijata za saobraćaj napraviti aplikaciju koja će mu omogućiti da može:

- uneti novog vozača u registar vozača,

- izbrisati nekog vozača iz registra vozača,
- uneti novo vozilo u registar vozila,
- izbrisati neko vozilo iz registra vozila,
- naći ime i prezime vlasnika poznatog vozila.

Pretpostavimo dalje:

- da nam na raspolaganju stoji neki viši programski jezik,
- da su nam poznata osnovna pravila programiranja,
- da možemo koristiti programski paket za analizu i sintezu informacionih sistema,
- da su formirane dvije tabele (relacije) VOZAC i VOZILO (vidi primjer 8.3) i da
- da je omogućen pristup tim tabelama.

Kostur programa i dijalog sa korisnikom piše se uvijek u odabranom višem programskom jeziku, a onda kada i gdje je to potrebno ugrađuju se elementi upitnog jezika.

U uvodnom dijelu programa treba formirati MENU-listu mogućnosti koja će se ponuditi korisniku, i koja treba da se pojavi na ekranu u obliku neke "forme". Ta lista sadrži 5 procedura, a mogla bi da izgleda ovako:

```
Odaberite opciju ukucavanjem odgovarajućeg broja:
1. uvođenje novog vozača u registar vozača
2. brisanje vozača iz registra
3. uvođenje novog vozila u registar vozila
4. brisanje nekog vozila iz registra
5. nalaženje imena i prezimena vlasnika vozila na
   osnovu registarskih tablica njegovog vozila
```

Postupak u programiranju ove aplikacije može da ima ovakav tok:

PROGRAM

- Izabranu opciju (broj iz ponuđene liste) memorisati u jednu programsku varijablu, na primjer **xxx.**
  Provjeriti da li je ukucan broj veći od nule ili manji od 6.

- *Ako jeste:*
  - vratiti program na početnu listu sa napomenom o grešci.

- *Ako nije:*
  - pozvati proceduru broj **xxx,**
  - vratiti program na početni MENU.

PROCEDURE

PROCEDURA 1 za poznatu vrijednost **xxx=1** omogućava unošenje novog vozača u registar vozača preko ekrana monitora.

- matični broj vozača, koji treba da bude uveden u evidenciju vozača, upisati na ekran (u za to pripremljen formular), i smjestiti ga u memoriju računara u, na primjer, varijablu yyy,
- provjeriti da li u tabeli VOZAČ već postoji osoba sa tim matičnim brojem V.matbr=′yyy′. Provjera se može izvesti SQL - upitom:

```
SELECT V.matbr
FROM VOZAČ V
WHERE V.matbr=′yyy′;
```

- *Ako u odgovoru na upit postoji jedna n-torka:*
  - dati odgovarajuću informaciju na ekran (na primjer: “Vozač pod tim imenom i prezimenom već se nalazi u evidenciji”),
  - zatvoriti tabelu VOZAČ,
  - vratiti na početni izbor.
- *Ako u odgovoru na upit ne postoji nijedna n-torka onda:*
  - formirati masku na ekranu za unošenje podataka o vozacu i unijeti podatke o njemu. Oblik te maske kao i način unošenja podataka zavisi od programskog paketa koji se koristi. **SQL** naredba bi mogla da glasi:

```
INSERT INTO VOZAČ (sv#,prezime,ime,……..)
VALUES (28079351173513, Petrovic, Petar, ……..);
```

  - zatvoriti tabelu VOZAČ,
  - vratiti se na početni MENU,
  - kraj procedure.

PROCEDURA 2 (**xxx=2**) omogućava brisanje vozača iz evidencije.

- Matični broj vozača, koji treba da bude brisan iz evidencije vozača, upisati na ekran (u za to pripremljen formular), i smjestiti ga u memoriju računara u, na primjer, varijablu yyy**.**
- provjeriti (na isti način kao i u **PROCEDURI 1**) da li u tabeli VOZAČ već postoji takva osoba.

- *Ako ne postoji n-torka sa unesenim matbr=´yyy´:*
  - vratiti program na početni izbor.

- *Ako postoji:*
  - prikazati podatke o njoj na ekranu,
  - postaviti pitanje ”Da li želite brisanje vozača iz evidencije”?

- *Ako je odgovor potvrdan:*
  - Izvršiti brisanje (na primjer):

```
DELETE FROM VOZAC  V
WHERE  V.matbr# = ´yyy´;
```

  - zatvoriti tabelu VOZAČ.
- *Ako je odgovor na pitanje bio negativan:*
  - zatvoriti tabelu VOZAČ,
  - vratiti na početni izbor,
  - kraj procedure.

Procedure 3 i **4** su logički analogne sa **Procedurama 1** i **2** samo se umjesto tabele VOZAČ aktivira tabela VOZILA pa nema potrebe da ih navodimo.

PROCEDURA 5 (**xxx=5**), omogućava nalaženje vlasnika vozila na osnovu registarskog broja njegovog vozila:

- registarski broj vozila, čijeg vlasnika hoćemo da pronađemo, upisati na ekran (u za to pripremljen formular), i smjestiti ga u memoriju računara u, na primjer, varijablu yyy
- pronaći vozilo na osnovu registarskog broja regbr#=yyy i izdvojiti matični broj vlasnika.

SQL – program bi mogao da glasi:

```
SELECT  V.matbr
FROM VOZILA   V
WHERE regbr# = ´VA-132-345´;
```

- *Ako postoji tražena n-torka u tabeli VOZILA:*
  - memorisati vrijednost V.matbr (zzz=V.matbr)
    otvoriti tabelu VOZAČ
  - pronaći vozača sa matičnim brojem V.matbr#=zzz.

Upit bi mogao da ima oblik:

```
SELECT *
FROM VOZAČ  V
WHERE  V.matbr# =´zzz´;
```

- *Ako postoji tražena n-torka u tabeli VOZAČ prikazati sve podatke na ekranu i zatim:*
  - zatvoriti tabelu VOZILA,
  - zatvoriti tabelu VOZAČ,
  - vratiti program na glavni izbor.

- *Ako ne postoji nijedna n-torka sa tim matičnim brojem dati odgovarajuću informaciju na ekran i zatim:*
  - zatvoriti tabelu VOZILA,
  - zatvoriti tabelu VOZAČ,
  - vratiti program na glavni izbor.

- Ako ne postoji nijedna n-torka u tabeli VOZILA sa traženim registarskim brojem vozila:
  - dati odgovarajuću informaciju na ekran,
  - zatvoriti tabelu VOZILA,
  - vratiti program na glavni izbor,
  - kraj procedure.

Službenik sekretarijata za saobraćaj u nekoj opštini, za koga je ovaj program pripremljen, ne zna, i ne treba da zna, ništa o strukturi baze podataka, a i pored toga može efikasno da je koristi. Nažalost, samo u jednom njenom aspektu, onom koji je programiran aplikacionim programom.

Drugim rječima, korisniku su na rasploganju samo pet operacija i ništa van toga. Ako je, recimo, neki automobil promjenio vlasnika (izvršena kupoprodaja), službenik sekretarijata za saobraćaj nema na raspolaganju mogućnost promjene vlasnika automobila. Da bi obavio tu aktivnost, službenik mora najprije da uvede novog vlasnika u registar vozača (ako već nije registrovan), zatim da izbriše vozilo iz registra i da to isto vozilo uvede u taj isti registar kao novo vozilo i pridruži ga prethodno uvedenom vlasniku.

Neke informacije i aktivnosti u ovakvom informacionom sistemu nijesu moguće ni na koji način. Tako, recimo, nije moguće saznati koliko u Beogradu ima automobila marke Mercedes novijih od dvije godine, ili ko je vlasnik nekog ukradenog automobila sa kojeg je skinuta tablica. Treba reći da u bazi podataka postoje svi potrebni podaci za dobijanje ovakvih informacija (na primjer, u bazi postoji podatak o broju motora vozila), ali to prilikom projektovanja aplikacije nije uzeto u obzir, i ne može se ostvariti bez izmjena aplikacije, tj. bez izrade nove aplikacije.