

# 6

## ***SQL - JEZIK RELACIONE BAZE PODATAKA***

---

Značaj standardizacije programskih jezika uočen je u ranim fazama razvoja savremene informatike, pa su se prvi standardi za programske jezike FORTRAN i COBOL pojavili već tokom šezdesetih godina. ovog veka. U oba slučaja motivacija za uvođenje standarda bilo je saznanje da se najveća vrednost u oblasti informatike nalazi u ogromnom broju programa raznovrsne namene napisanih na raznim programskim jezicima, i da tu vrednost treba zaštititi od zastarevanja usled nepredvidih izmena u programskim jezicima i hardveru. Sa prvim standardima usvojeno je načelo "vertikalne kompatibilnosti", po kome svaki novi standard treba da sadrži uz poboljšanja i sve mogućnosti prethodnog standarda. Zahvaljujući tome, bilo je obezbeđeno da se jednom razvijeni programi mogu relativno lako prenositi sa jednog računara na drugi.

Sa uvođenjem relacionih baza podataka u sve širu primenu narastala je i potreba za standardizacijom u toj oblasti. Prvi standard, koji je nastao 1986. godine bio je rezultat kompromisa koji je već godinama vladao na tržištu. Ovaj standard je neznatno dopunjen 1989. godine, a bitna poboljšanja i dopune sprovedene su standardom od 1992. godine, dopunom 1995. godine i 1999. godine i standardom od 2003. godine.

U ovom poglavlju upoznaćemo se sa standardnim jezikom relacionih baza podataka poznatim pod skraćenicom SQL. Naziv potiče od naziva na engleskom jeziku "Structured Query Language", što u prevodu glasi "Strukturirani upitni jezik".

## 6.1 Uvod u SQL - jezik

U suštini, program na bilo kom programskom jeziku čine podaci i manipulacije nad tim podacima. Primera radi, predmeti manipulacije u nekom programu na PASCAL-u su varijable različitih tipova, a rezultati tih manipulacija su isto varijable. Analogno tome, u SQL-jeziku osnovni objekti manipulacija su relacije a rezultat toga su isto relacije, čak i kada se kao rezultat manipulacije dobija skup vrednosti ili samo jedna vrednost: skup vrednosti tretiramo kao relaciju nad šemom sa jednim atributom, a jednu vrednost kao takvu relaciju sa jednom n-torkom.

Terminologija SQL-a se nešto razlikuje od one koju smo imali do sada. Umesto pojma relacije koristi se pojam tabele. Za jednu n-torku u relaciji kažemo da predstavlja jedan red tabele. Ako relaciju predstavimo sebi kao tabelu, jasno je šta se podrazumeva pod pojmom kolone: to su sve vrednosti u n-torkama relacije koje odgovaraju jednom atributu. Ova terminologija je nasleđena iz prakse koja je prethodila standardizaciji, a rezultat toga je krajnje neobična okolnost da u SQL-u, jeziku za relacione baze podataka, ne postoji ni jedna konstrukcija koja sadrži reč "RELATION".

SQL jezik podržava tri osnovne funkcije koje smo naveli u uvodnim poglavljima o bazama podataka. To su:

- definicija baze podataka: pre početka rada sa bazom podataka neophodno je definisati njenu strukturu - koje tabele postoje, koji atributi postoje u tabelama i kog su tipa, koja ograničenja postoje unutar tabela i između njih, koje pomoćne strukture (indeksi) za ubrzanje pristupa podacima postoje i za koje tabele; ova komponenta jezika odgovara DDL-jeziku baze podataka (od "Data Definition Language");
- manipulacija bazom podataka: pored upita nad bazom podataka, kojima dobijamo željene informacije, neophodno je obezbediti i ažuriranje baze podataka, odnosno unos, izmenu i brisanje podataka; ova komponenta je u stvari DML-jezik baze podataka (od "Data Manipulation Language");
- kontrola pristupu podacima: u svakoj bazi podataka neophodno je ostvariti kontrolu koji korisnici imaju pristup kojim podacima i šta mogu da rade sa tim podacima; ova komponenta predstavlja DCL-jezik baze podataka (od "Data Control Language").

Iz svega toga, možemo primetiti da naziv SQL ne odgovara u potpunosti, pošto u pitanju nije samo upitni jezik.

SQL-jezik podržava i oba režima rada sa bazom podataka koje smo naveli u poglavlju 2:

- interaktivni: korisnik zadaje jednu po jednu SQL naredbu interaktivno, preko tastature, a ishod svake se prikazuje preko monitora; pristup bazi podataka je ograničen jedino pravima korisnika;
- programski: korisnik pokreće program u kome se nalaze "ugrađene" SQL naredbe; pristup bazi podataka ograničen je pravima korisnika i sadržajem programa; pri tome, ugrađene naredbe mogu biti statičke (fiksirane u vreme prevođenja programa) ili dinamičke (konstruisane tokom izvršavanja programa).

Detalje SQL-jezika izložićemo ovde za uslove interaktivnog režima rada, a programski režim rada ćemo razmatrati u prilogu C.

## 6.2 SQL za definiciju baze podataka

Definicija neke baze podataka podrazumeva i mogućnost naknadne izmene ili uklanjanja te definicije. U standardnom SQL-jeziku se to postiže sa svega tri fraze:

<u>CREATE</u>	služi za kreiranje nekog objekta (tabele, indeksa itd.) u bazi podataka;
<u>DROP</u>	služi za uklanjanje definicije nekog objekta iz baze podataka;
<u>ALTER</u>	služi za izmenu definicije nekog objekta u bazi podataka

Pre razmatranja konkretnih naredbi napomenimo da se sama baza podataka kao celina kreira naredbom čija je sintaksna definicija

**KreiranjeBazePodataka ::= CREATE DATABASE Naziv Opcije ;**

koju zbog brojnih i složenih mogućnosti u sastavu Opcija nećemo ovde razmatrati. Isto tako, nećemo razmatrati ni naredbe izmene i uklanjanja baze podataka.

## 6.2.1 SQL tipovi podataka

Od ranije nam je poznato da je šema relacije skup atributa za koje je definisano kog su tipa podataka, pa je jasno da se u definiciji svake tabele u SQL-jeziku mora navesti kojih su tipova kolone.

Navedimo prvo najosnovnije standardne SQL tipove podataka bez navođenja alternativnih naziva. To su:

<u>INTEGER</u>	ceo broj sa ili bez predznaka čiji broj cifara zavisi od konkretne implementacije;
<u>FLOAT</u>	realan broj sa ili bez predznaka čija preciznost (broj značajnih cifara) zavisi od konkretne implementacije;
<u>DECIMAL</u> [m[,n]]	decimalni broj sa ili bez predznaka i sa ukupno m cifara od čega su n decimale; maksimalni broj cifara zavisi od konkretne implementacije;
<u>BOOLEAN</u>	logički podatak sa vrednošću TRUE ili FALSE;
<u>CHARACTER</u> [[n]]	(ili <u>CHAR</u> ): niz znakova fiksne dužine n ; maksimalni broj znakova zavisi od konkretne implementacije; ako dužina nije zadata podrazumeva se 1; konstante ovoga tipa pišu se između jednostrukih navodnika;
<u>CHARACTER VARYING</u> [n]	niz znakova promenljive dužine 0 do n, slično stringu u programskom jeziku C; maksimalni broj znakova zavisi od konkretne implementacije; konstante ovoga tipa pišu se između jednostrukih navodnika;
<u>TIMESTAMP</u>	Univerzalni podatak “godina-mesec-datum-sat-minut-sekunda-...” gde najmanja rezolucija vremena zavisi od konkretne implementacije;
<u>DATE</u>	Datumski podatak oblika yyyyymmdd (yyyy je godina, mm je mesec a dd je dan);
<u>TIME</u>	Vremenski podatak oblika hhmmss (hh je sat, mm minuti a ss sekunde).

## 6.2.2 Naredba kreiranja tabele

Prilikom kreiranja tabele, odnosno definicije njene strukture i osobina, neophodno je navesti sledeće:

- ime tabele, koje mora biti unikatno u bazi podataka;
- ime svake kolone, koja mora biti unikatno unutar tabele;
- tip svake kolone
- jedno ili više ograničenja za kolone koje ih imaju;
- jedno ili više ograničenja za celu tabelu, ako postoje.

Navedimo sada u notaciji koju smo usvojili opštu definiciju sintakse naredbe kreiranja tabele:

```
NaredbaKreiranjaTabele ::=
    CREATE TABLE Tabela ( DefinicijaKolone ...
                          [OgranicenjeTabele ...] ) ;

DefinicijaKolone ::=
    Kolona Tip|Domen OgranicenjeKolone ...
```

**Tabela** i **Kolona** formiraju se po pravilu koje važi za varijable u većini programskih jezika (prvi znak je slovo, ostali znaci su slova, cifre ili znak `_`). **Tip** je jedan od standardnih SQL tipova, a **Domen** je simbol neveden u sklopu odgovarajuće **CREATE DOMAIN** naredbe kojom je nad nekim ugrađenim tipom definisan korisnički tip (o tome će biti reči kasnije).

### 6.2.2.1 Ograničenja kolone

Uz svaku kolonu mogu se navesti ni jedno, jedno ili više ograničenja za tu kolonu. Osnovne forme za **OgranicenjeKolone** i njihovo značenje su:

<b><u>NOT NULL</u></b>	u koloni nije dozvoljena NULL vrednost;
<b><u>UNIQUE</u></b>	u koloni nije dozvoljeno ponavljanje iste vrednosti;
<b><u>PRIMARY KEY</u></b>	kolona je primarni ključ, i u njoj nije dozvoljena NULL vrednost niti ponavljanje vrednosti;
<b><u>CHECK (Predikat)</u></b>	svaka vrednost u koloni mora da zadovoljava uslov zadat logičkim izrazom koji smo označili sa <b>Predikat</b> ;
<b><u>DEFAULT = Const</u></b>	ako se prilikom unošenja jednog reda podataka u tabelu za kolonu ne zada vrednost, preuzima se podrazumevana vrednost <b>Const</b> ;

Sledeća konstrukcija služi za naznaku da je jedna kolona strani ključ i za dinamičku specifikaciju referencijalnog integriteta:

```
REFERENCES Tabela [( Kolona )]
[ ON UPDATE NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT ]
[ ON DELETE NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT ]
```

Kolona referiše ciljnu tabelu **Tabela**, i to kolonu **Kolona** ako je navedena a primarni ključ ako nije. Ako neka od klauzula operacija **ON UPDATE** ili **ON DELETE** nije zadata, za tu operaciju se podrazumeva **NO ACTION**.

Opcija **NO ACTION** i novija opcija **RESTRICT** imaju isti krajnji efekat - odbijanje izvršenja promene u ciljnoj tabeli ako se narušava referencijalni integritet, ali se razlikuju u implementaciji:

- **NO ACTION** je dijagnostički, u smislu da ne podrazumeva provere pre promene u ciljnoj tabeli, nego samo proveru tokom same operacije promene; ukoliko se detektuje narušavanje referencijalnog integriteta sistem upravljanja bazom podataka prekida operaciju i poništava efekte svih do tada izvršenih promena;
- **RESTRICT** je prediktivan, u smislu da podrazumeva provere pre bilo kakve promene u ciljnoj tabeli; to sprovodi sistem upravljanja bazom podataka, i tek kada se uveri da nigde neće biti narušen referencijalni integritet sprovodi operaciju promene.

### 6.2.2.2 Ograničenja tabele

Za celu tabelu mogu se zadati ni jedno, jedno ili više ograničenja. To je neophodno u situacijama kada ograničenja važe za skup od dve ili više kolona. Osnovne forme za `OgranicenjeTabele` i njihovo značenje su sledeće:

<code><u>UNIQUE</u> ( Kolona ,... )</code>	u koloni nije dozvoljeno ponavljanje istih kombinacija vrednosti kolona čiji su nazivi navedena;
<code><u>PRIMARY KEY</u> ( Kolona ,... )</code>	navedene kolone su primarni ključ, i u njima nije dozvoljena NULL vrednost niti je dovoljeno ponavljanje kombinacija njihovih vrednosti;
<code><u>CHECK</u> ( Predikat )</code>	u svakom redu u tabeli vrednosti navedenih kolona moraju da zadovoljavaju uslov zadat logičkim izrazom <code>Predikat</code> ;

Sledeća konstrukcija služi za naznaku da više kolona predstavlja strani ključ i za dinamičku specifikaciju referencijalnog integriteta:

```
FOREIGN KEY ( Kolona ,... )
  REFERENCES Tabela [( Kolona ,... )]
[ ON UPDATE NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT ]
[ ON DELETE NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT ]
```

Kolone naznačene u zagradama iza `KEY` referišu ciljnu tabelu `Tabela`, i to kolone `Kolona ,...` ako su navedene a primarni ključ ako nisu. Ako neka od klauzula operacija `ON UPDATE` ili `ON DELETE` nije zadata, za tu operaciju se podrazumeva `NO ACTION`.

Treba naglasiti da su sva ograničenja navedena u `CREATE TABLE` definiciji neke tabele aktivna u svakom trenutku postojanja te tabele. Svaki pokušaj da se nad nekom tabelom uradi nešto što je u suprotnosti sa bilo kojim od ograničenja biće signaliziran i odbijen od sistema za upravljanje bazom podataka. To je ogromna olakšica za projektante i programere baze podataka, koji bi u suprotnom morali da sve te provere ugrađuju u svoje aplikativne programe. Za SQL važi konstatacija da je kao jezik mešavina proceduralnog i deklarativnog, ali za `CREATE TABLE` naredbu ta konstatacija sigurno ne važi. Deklarativna moć te naredbe je ogromna, naročito u slučaju dinamičke specifikacije referencijalnog integriteta.



*Primer*

Kao ilustraciju upotrebe `CREATE TABLE` naredbe navedimo kompletnu definiciju baze podataka BIBLIOTEKA, koja je data u Prilogu A. Radi preglednosti, nazive tabela i kolona navodimo malim slovima i dati su komentari nekih ograničenja:

```
CREATE TABLE
  Oblast (
    Sifo CHAR(2) PRIMARY KEY,
    Naziv CHAR(20) NOT NULL
    UNIQUE
  );
```

**Naziv** je kandidat ključ.

```
CREATE TABLE
  Naslov (
    SifN CHAR(4) PRIMARY KEY,
    Naziv CHAR(20) NOT NULL,
    Sifo CHAR(2) NOT NULL
    REFERENCES Oblast
    ON UPDATE CASCADE
    ON DELETE NO ACTION
  );
```

Ne sme se brisati oblast za koju postoje naslovi,

```
CREATE TABLE
  Autor (
    SifA CHAR(3) PRIMARY KEY,
    Ime CHAR(15) NOT NULL
  );
```

```
CREATE TABLE
  Clan (
    SifC CHAR(3) PRIMARY KEY,
    Ime CHAR(15) NOT NULL
  );
```

```
CREATE TABLE
  Knjiga (
    SifK CHAR(3) PRIMARY KEY,
    SifN CHAR(4) NOT NULL
    REFERENCES Naslov
    ON UPDATE CASCADE
    ON DELETE NO ACTION
  );
```

Ne sme se brisati naslov za koji postoje knjige.

```

CREATE TABLE
Je_Autor (
    SifA CHAR(3) REFERENCES Autor
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    SifN CHAR(4) REFERENCES Naslov
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    Koji INTEGER NOT NULL
        CHECK ( Koji>0 )
    PRIMARY KEY (SifA,SifN)
);

```

Ne sme se brisati autor za koga postoje naslovi koje je napisao.  
 Brisanjem naslova brišu se i podaci o njegovom autorstvu.

```

CREATE TABLE
Drzi (
    SifK CHAR(3) PRIMARY KEY
        REFERENCES Knjiga
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    SifC CHAR(3) NOT NULL
        REFERENCES Clan
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    Datum DATE NOT NULL
);

```

Ne sme se brisati knjiga koja je kod člana.  
 Ne sme se brisati član kod koga je neka knjiga.

```

CREATE TABLE
Je_Rezervisana (
    SifK CHAR(3) PRIMARY KEY
        REFERENCES Knjiga
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    SifC CHAR(3) REFERENCES Clan
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    Datum DATE NOT NULL
);

```

Brisanjem knjige brišemo i podatke o tome da je ona rezervisana za nekog člana;  
 Brisanjem člana brišemo i podatke o knjigama koje su za njega rezervisane.

```

CREATE TABLE
  Pozajmica (
    SifP  INTEGER PRIMARY KEY,
    SifC  CHAR(3) REFERENCES Clan
           ON UPDATE CASCADE
           ON DELETE SET NULL,
    SifK  CHAR(3) REFERENCES Knjiga
           ON UPDATE CASCADE
           ON DELETE SET NULL,
    SifN  CHAR(3) REFERENCES Naslov
           ON UPDATE CASCADE
           ON DELETE SET NULL,
    Dana  INTEGER NOT NULL
           CHECK (Dana>0)
  );

```

Brisanjem clana, naslova ili knjige ne brišemo ostale podatke o pozajmici.

```

CREATE TABLE
  Rezervacija (
    SifN  CHAR(4) REFERENCES Naslov
           ON UPDATE CASCADE
           ON DELETE CASCADE,
    SifC  CHAR(3) REFERENCES Clan
           ON UPDATE CASCADE
           ON DELETE CASCADE,
    Datum TIMESTAMP NOT NULL,
    PRIMARY KEY (SifN,SifC)
  );

```

Brisanjem naslova brišemo i podatke o njegovim rezervacijama.

Brisanjem člana brišemo i podatke o njegovim rezervacijama.

Podatak datum je tipa **TIMESTAMP** da bi mogli da se razreše razreše prioriteta rezervacija.

Izbor dinamičkih specifikacija integriteta za slučaj uklanjanja predstavlja delikatnu odluku. Ako preterano koristimo klauzulu **NO ACTION**, nametnućemo vrlo krut režim rada sa bazom podataka koji može dovesti i do toga da ne možemo da u bazi podataka registrujemo promene koje se dešavaju u realnom sistemu koga ona predstavlja, a u ekstremnim slučajevima da čak ne možemo da uklonimo pogrešno unete podatke iz tabela. Sa druge strane, olako korišćenje klauzule **CASCADE** kod operacije **DELETE** može dovesti do neplaniranog efekta brisanja podataka koji ne treba da se brišu.

### 6.2.3 Naredba izmene definicije tabele

Naredba izmene definicije tabele je nešto složenija, pošto treba da obezbedi sledeće mogućnosti izmene tabele:

- dodavanje nove definicije kolone;
- izmena postojeće definicije kolone;
- uklanjanje postojeće definicije kolone;
- dodavanje novog ograničenja tabele;
- uklanjanje postojećeg ograničenja tabele.

Treba naglasiti da sve to mora biti sprovodivo nad tabelom koja ima neki sadržaj. U tom smislu `ALTER TABLE` i par `DROP TABLE – CREATE TABLE` nisu ekvivalentni pošto u drugom slučaju gubimo sadržaj tabele.

Sintaksna definicija naredbe izmene tabele je složena i izložena je postupno:

```

NaredbaIzmeneTabele ::=
    ALTER TABLE Tabela SpecIzmeneTabele ;

SpecIzmeneTabele ::=
    SpecIzmeneKolona | SpecIzmeneOgranicenjaTabele

SpecIzmeneKolona ::=
    SpecIzmeneJedneKolone ,...

SpecIzmeneJedneKolone ::=
    DodavanjeKolone | IzmenaKolone | UklanjanjeKolone

DodavanjeKolone ::=
    ADD [ COLUMN ] DefinicijaKolone

IzmenaKolone ::=
    DodavanjePodrazumevanja | UklanjanjePodrazumevanja

DodavanjePodrazumevanja ::=
    ALTER [ COLUMN ] Kolona SET DEFAULT = Const

UklanjanjePodrazumevanja ::=
    ALTER [ COLUMN ] Kolona DROP DEFAULT

UklanjanjeKolone ::=
    DROP [ COLUMN ] Kolona RESTRICT|CASCADE

SpecIzmeneOgranicenjaTabele :: =
    SpecIzmeneJednogOgranicenjaTabele ,...

SpecJednogOgranicenjaTabele ::=
    SpecDodavanjaOgranicenja | SpecUklanjanjaOgranicenja

SpecDodavanjaOgranicenja ::=
    ADD CONSTRAINT OgranicenjeTabele

SpecUklanjanjaOgranicenja ::=
    DROP CONSTRAINT OgranicenjeTabele RESTRICT|CASCADE

```

Ovde je neophodni nekoliko napomena i objašnjenja:

- **IzmenaKolone** je ograničena samo na mogućnost uvođenja nove ili uklanjanja podrazumevane vrednosti; u tim okolnostima, postojeća ograničenja kolone se ne mogu uklanjati a nova se mogu dodavati samo preko dodavanja novog ograničenja tabele sa naznačenom jednom kolonom;
- **UklanjanjeKolone** ne uspeva ako je navedena kolona jedina u tabeli, kao i ako je navedena klauzula **RESTRICT** a u bazi podataka postoji bar jedan pogled koji referiše kolonu koja se uklanja (o tome šta je pogled biće reči kasnije);
- **SpecUklanjanjaOgranicenja** ne uspeva ako je navedena klauzula **RESTRICT**, ograničenje definiše kandidat-ključ (preko **UNIQUE** klauzule) i u bazi podataka postoji bar jedan strani ključ koji referiše taj kandidat-ključ.

### 6.2.4 Naredba uklanjanja definicije tabele

Naredba uklanjanja definicije tabele iz baze podataka je jednostavna. Njena sintaksna definicija glasi:

```
NaredbaUklanjanjaTabele ::= DROP TABLE Tabela ;
```

Kod nekih implementacija, tabela koja se uklanja mora biti prazna. U suprotnom, sistem za upravljanje bazom podataka neće izvršiti tu naredbu.

### 6.2.5 Naredbe kreiranja i uklanjanja indeksa

Indeks je, da podsetimo, pomoćna datoteka koja treba da ubrza pristup podacima u nekoj osnovnoj datoteci. Pored toga, indeks ima još jednu namenu: zapisi u osnovnoj datoteci nalaze se u nekom fizičkom redosledu i kada pristupamo neposredno toj datoteci zapise očitavamo tim redosledom, ali ako podacima pristupamo posredstvom indeksa, očitavaćemo ih redosledom koji odgovara rastućoj ili opadajućoj vrednosti indeksnog izraza.

Naredba kreiranja indeksa u usvojenoj sintaksoj notaciji glasi:

```
NaredbaKreiranjaIndeksa ::=
    CREATE [ UNIQUE ] INDEX Indeks ON Tabela ( Kolona ,... ) ;
```

gde je značenje pojedinih delova ove definicije sledeće:

<u>UNIQUE</u>	kada se zada ova opcija, indeks mora biti unikatan, odnosno u tabeli na koju se indeks odnosi ne sme da se više puta ponovi neka vrednost <i>Kolona ,... ;</i>
<i>Indeks</i>	unikatni naziv indeksa u bazi podataka, simbol formiram po pravilu za nazive varijabli;
<i>Kolona ,... </i>	jedna ili više kolona po kojima se formira indeks.

Treba naglasiti da većina implementacija dozvoljava i indekse po izrazima odnosno funkcijama nad kolonama u sastavu indeksa, kao i rastuće i opadajuće indekse.

Nad istom tabelom po potrebi može biti definisano više indeksa. To se koristi kada nam u raznim situacijama trebaju različiti pristupi podacima i različiti redosledi podataka. Indeks može biti kreiran odmah, dok je tabela na koju se odnosi prazna, ili naknadno. Ako se kreira naknadno, indeks dobija sadržaj koji odgovara sadržaju svoje tabele. Od tog trenutka, sadržaj indeksa i tabele je sinhronizovan: svako dodavanje ili uklanjanje podataka, kao i izmena vrednosti neke od kolona koja je u sastavu indeksnog izraza, odražava se na sadržaj indeksa.

Indeks se može bilo kada i bez obzira na sadržaj svoje tabele ukloniti naredbom čija je sintaksna definicija:

```
NaredbaUklanjanjaIndeksa ::= DROP INDEX Indeks ;
```



*Primer*

U slučaju da želimo brz pristup podacima u tabeli `Je_Autor` po dva osnova, po šifri autora i po šifri naslova, moramo kreirati dva indeksa:

```
CREATE INDEX Je_Autor1 ON Je_Autor ( SifA ) ;
CREATE INDEX Je_Autor2 ON Je_Autor ( SifN ) ;
```

Za sadržaj baze podataka `BIBLIOTEKA` dat u prilogu A imali bi sledeće sadržaje indeksa `je_autor1` i `je_autor2` i ukazivanje na redove u tabeli `je_autor` :

<code>je_autor1</code> ( SIFA )		<code>je_autor</code> ( SIFA	SIFN	KOJI )		<code>je_autor2</code> ( SIFN )
-----		-----	-----	-----		-----
AP0	—————	AP0	RBP0	1	—————	PJC0
AP1	—————	JN0	RBP0	2	—————	PJC0
DM0	—————	DM0	RK00	1	—————	PP00
DM0	—————	ZP0	PP00	1	—————	PP00
IT0	—————	DM0	PP00	2	—————	PP00
JN0	—————	AP1	PJC0	1	—————	RBP0
ZP0	—————	IT0	PP00	3	—————	RBP0
ZP0	—————	ZP0	PJC0	2	—————	RK00

## 6.2.6 Naredbe kreiranja i uklanjanja pogleda

Tabele koje smo do sada razmatrali se nazivaju osnovnim tabelama, i one fizički postoje u vidu odgovarajućih datoteka. Pored takvih, u oblasti relacionih baza podataka postoje i tzv. "pogledi".

Pogled ("view" na engleskom) predstavlja izvedenu tabelu, ima redove i kolone i nastaje kao rezultat upita nad osnovnim tabelama i drugim pogledima. Redovi i kolone pogleda nisu nigde trajno zapisani. Umesto toga, svaki put kada se pristupa pogledu izvršava se upis kojim je on definisan.

Kreiranje pogleda vrši se naredbom čija je sintaksna definicija:

```
NaredbaKreiranjaPogleda ::=  
CREATE VIEW Pogled [ ( Kolona ,... ) ] AS Upit ;
```

gde je značenje pojedinih delova ove definicije sledeće:

<b>Pogled</b>	unikatni naziv pogleda u bazi podataka, simbol formiram po pravilu za nazive varijabli;
<b>Kolona ,...</b>	Ako se navedu kolone pogled se ponaša kao tabela sa brojem, redosledom i imenima kolona kako je navedeno, a u suprotnom se preuzimaju imena kolona iz osnovnih tabela i pogleda koje su navedene u naredbi upita. U oba slučaja, pogled nasleđuje tipove kolona iz osnovnih tabela i pogleda iz upita.
<b>Upit</b>	naredbu upita <code>SELECT</code> koju tek treba da obradimo a čiji rezultat izvršavanja daje "tabelu" koja predstavlja pogled.

Pod određenim okolnostima, pogled može biti "ažurabilan", odnosno može se koristiti za izmenu sadržaja osnovne tabele iz svog deficiionog upita.

Pogled se uklanja jednostavnom naredbom čija je sintaksna definicija:

```
NaredbaUklanjanjaPogleda ::=  
DROP VIEW Pogled ;
```

Uklanjanje pogleda nema nikakvog efekta na osnovne tabele iz upita.

Na poglede ćemo se osvrnuti detaljnije pošto prethodno razmotrimo SQL naredbe upita i ažuriranja.

## 6.3 SQL naredba upita **SELECT**

Naredba **SELECT** za upite predstavlja najznačajniju i najčešće korišćenu SQL naredbu za manipulaciju podacima, pa ćemo joj posvetiti najviše pažnje. Tek nakon toga obradićemo i preostale tri naredbe za manipulaciju podacima koje služe za ažuriranje baze podataka, odnosno za unos, izmenu i uklanjanje redova iz tabela.

### 6.3.1 Redni upit nad jednom tabelom

U principu, kod svakog upita zadajemo:

- koje podatke tražimo kao rezultat;
- iz kojih tabela to tražimo;
- koji uslov treba da zadovolje podaci da bi bili uključeni u rezultat;
- po kom redosledu želimo prikaz rezultata.

Pod rednim upitom nad jednom tabelom podrazumevamo naredbu upita **SELECT** nad jednom tabelom koja kao rezultat daje ni jedan red, jedan red ili niz redova podataka, od kojih svaki odgovara podacima iz jednog reda tabele koji zadovoljava eventualno zadati uslov.

Rezultat upita ne mora biti relacija u smislu unikatnosti redova koji ulaze u rezultat. To se ispoljava kada za rezultat upita biramo samo neke od kolona, kada može doći do pojave istovetnih redova u rezultatu. Stoga prilikom formulacije upita treba da postoji mogućnost specifikacije da li želimo eliminaciju višestrukog pojavljivanja istih redova u rezultatu ili ne.

Sa prethodnim napomenama obezbedili smo sve preduslove za opis naredbe **SELECT** za slučaj rednog upita nad jednom tabelom, čija je sintaksa:

```
RedniUpitJednaTabela ::=
    SELECT      R-Lista
    FROM       Tabela
    [ WHERE     R-Predikat ]
    [ ORDER BY { R-Izraz [ ASC | DESC ] } ,... ] ;

R-Lista ::=
    * | { [ ALL | DISTINCT ] R-Izraz [ [ AS ] Nadimak ] } ,...
```

Pojasnimo prvo kako se specificiraju podaci koje želimo da uključimo u rezultat:

- \***                    specijalni slučaj kada želimo da uključimo sve kolone tabele, i to onim redosledom kojim su navedene u naredbi kreiranja tabele;
- ALL**                tražimo da se u rezultatu prikažu svi redovi uključujući i one koji su istovetni, podrazumeva se ako se ništa ne navede;
- DISTINCT**        tražimo da se iz rezultata elimiše suvišna pojavljivanja (osim jednog) istovetnih redova;
- R-Izraz**            izraz izračunljiv nad svakim pojedinim redom tabele koji pored naziva kolona može da sadrži i operatore i konstante; najčešće je u formi navođenja jedne kolone;
- Nadimak**          izraz iza koga se nalazi **Nadimak** ponaša se kao kolona tog naziva, kako u svojstvu naziva kolone u interaktivnom prikazu rezultata tako i u situacijama kada se rezultat dalje ponaša kao tabela tokom izvršavanja složene SQL naredbe (o tome še biti reči kasnije);

Specifikacija "odakle" nalazi se neposredno iza klauzule **FROM**:

**Tabela**            naziv osnovne tabele ili pogleda nad kojim vršimo upit.

Uslov uključivanja redova tabele u formiranje rezultata navodi se iza klauzule **WHERE**:

**R-Predikat**        logički izraz koji je izračunljiv nad svakim pojedinim redom tabele; u formiranje rezultata upita ulaze samo oni redovi za koje taj izraz daje istinit rezultat. U najjednostavnijim slučajevima, **R-Predikat** je u formi relacionog izraza u kome se sa jedne strane relacionog operatora (>, <., = itd.) javlja ime kolone, a sa druge strane ime kolone ili konstanta.

Željeni redosled prikaza rezultata navodimo iza **ORDER BY** klauzule, gde navodimo jednu ili više kategorija **R-Izraz** odvojenih zarezima po kojima želimo uređenost. Najčešće su u pitanju kolone tabele. Podrazumeva se rastući redosled **ASC**, a ako uz neki **R-Izraz** želimo suprotno navodimo klauzulu **DESC** uz nju.

Pre ilustrativnih primera za proste upite nad jednom tabelom neophodno je nekoliko napomena.

Najjednostavniji mogući SQL-upit je u formi

```
SELECT * FROM Tabela ;
```

Ova naredba prikazuje sve redove tabele čije je ime navedeno iza **FROM** klauzule. U svakom redu prikazuju se vrednosti svih kolona, i to onim redosledom kojim se nalaze u zapisima u datoteci, odnosno redosledom navođenja kolona u naredbi **CREATE TABLE**. Ako bi izvršili ovakvu naredbu nad bilo kom tabelom iz naše baze podataka BIBLIOTEKA, dobili bi kao prikaz istu takvu tabelu.

Kod upita se najčešće traži prikaz samo određenih kolona, ili prikaz svih kolona po redosledu koji je drugačiji od stvarnog. To se postiže naredbom upita kod koje iza **SELECT** klauzule navodimo željene kolone. Ovakva naredba odgovara operaciji projekcije u relacionoj algebri, ali postoji jedna bitna razlika: ako to ne naglasimo, u tabeli koja nastaje kao prikaz neće biti eliminisana višestruka pojavljivanja istih vrednosti.

*Primeri*

Od sada pa na dalje, upiti navedeni levo daju za dati sadržaj baze podataka BIBLIOTEKA rezultate koji su navedeni desno od njih:

Upit za prikaz cele tabele:

```
SELECT *
FROM Oblast ;
```

```
BP  Baze podataka
RM  Racunarske mreze
PJ  Programski jezici
```

Upit za prikaz cele tabele u željenom redosledu:

```
SELECT      *
FROM        Oblast
ORDER BY Sifo ;
```

```
BP  Baze podataka
PJ  Programski jezici
RM  Racunarske mreze
```

Upit za prikaz samo jedne kolone iz tabele i bez eliminacije duplikata, pri čemu će u prikazu kolona preuzeti svoj naziv kao naslov:

```
SELECT SifN
FROM Knjiga ;
```

RBP0
RBP0
RK00
PJC0
PJC0
PJCO
PP00
PP00
PP00

Upit za prikaz samo jedne kolone iz tabele i sa eliminacijom duplikata, pri čemu će u prikazu kolona dobiti naslov 'SifraNaslova':

```
SELECT DISTINCT SifN AS SifraNaslova
FROM Knjiga ;
```

RBP0
RK00
PJC0
PP00

Upit za prikaz dve kolona sa zadavanjem uslova:

```
SELECT SifN,Naziv
FROM Naslov
WHERE Sifo ='PJ' ;
```

PP00	PASCAL programiranje
PJC0	Programski jezik C

U poslednjem primeru imamo uslov koji se svodi na to da vrednost kolone mora biti jednaka datoj konstanti. To je specijalni slučaj konstrukcije **R-Predikat** u formi relacionog izraza **R-PredikatRelacioni** čija je sintaksa

**R-PredikatRelacioni ::= R-Izraz ≤ | ≤= | = | <= | >= | ≥ R-Izraz**

gde je **R-Izraz** izraz izračunjivi nad svakim pojedinim redom tabele.



### 6.3.2 Redni upit nad jednom tabelom sa svodnim rezultatom

Pod rednim upitom nad jednom tabelom sa svodnim rezultatom podrazumevamo naredbu upita **SELECT** nad jednom tabelom koja kao rezultat daje jedan red podataka koji su izvedeni iz svih redova tabele koji zadovoljavaju eventualno zadati uslov.

Specifikacija rezultata takvog upita sastoji se iz jednog ili više izraza koji su izračunjivi nad više redova tabele, pri čemu više takvih izraza odvajamo zarezima.

S obzirom na okolnost da ovakav upit daje samo jedan red rezultata, zadavanje željenog redosleda redova u rezultatu nema snisla.

Definicija sintakse naredbe **SELECT** u varijanti rednog upita nad jednom tabelom sa svodnim rezultatom glasi:

```
RedniUpitJednaTabelaSvodniRezultat ::=
    SELECT G-Lista
    FROM Tabela
    [ WHERE R-Predikat ] ;

G-Lista ::= { G-Izraz [ AS ] Nadimak } ,...
```

Konstrukciju **G-Izraz** najčešće čini jedna od posebnih SQL funkcija koje se nazivaju svodnim ili agregatnim funkcijama, ali u opštem slučaju to mogu biti izrazi sastavljeni iz više takvih funkcija, operatora i konstanti.

Navedimo svodne funkcije uz odgovarajuća objašnjenja:

<u>SUM</u> ( <u>Kolona</u> )	nalazi sumu svih ne-NULL vrednosti zadate kolone;
<u>AVG</u> ( <u>Kolona</u> )	nalazi prosečnu vrednost svih ne-NULL vrednosti zadate kolone;
<u>MIN</u> ( <u>Kolona</u> )	nalazi minimalnu vrednost svih ne-NULL vrednosti zadate kolone;
<u>MAX</u> ( <u>Kolona</u> )	nalazi maksimalnu vrednost svih ne-NULL vrednosti zadate kolone;
<u>COUNT</u> ( <u>*</u> )	daje ukupan broj redova;
<u>COUNT</u> ( [ <u>ALL</u> ] <u>Kolona</u> ,... )	daje ukupan broj ne-NULL vrednosti zadate kombinacije kolona;
<u>COUNT</u> ( <u>DISTINCT</u> <u>Kolona</u> ,... )	daje ukupan broj različitih ne-NULL vrednosti zadate kombinacije kolona;

Treba naglasiti da u sračunavanje vrednosti agregatnih funkcija ulaze samo oni redovi koji zadovoljavaju uslov u **WHERE** klauzuli.

*Primeri*

Upiti navedeni sa leve strane daju za dati sadržaj baze podataka BIBLIOTEKA rezultate navedene desno.

Upit za prikaz ukupnog broja članova (odgovara broju redova u tabeli Clan )::

```
SELECT COUNT(*)                                4
FROM Clan ;
```

Upit za prikaz broja članova koji su vršili pozajmice (odgovara broju različitih vrednosti kolone SifC u tabeli Pozajmica):

```
SELECT COUNT ( DISTINCT SifC )                 3
FROM Pozajmica ;
```

Upit za prikaz broja naslova koje je napisao autor šifre 'DM0', pri čemu će u prikazu kolona dobiti naslov 'BrojNaslova':

```
SELECT COUNT (*) AS BrojNaslova                2
FROM Je_Autor
WHERE SifC = 'DM0' ;
```

Upit za prikaz sume trajanja svih pozajmica:

```
SELECT SUM (Dana)                27
FROM Pozajmica ;
```

Upit za prikaz minimalnog i maksimalnog trajanja pozajmica:

```
SELECT MIN (Dana), MAX (Dana)    2    7
FROM Pozajmica;
```

Upit za prikaz sume i proseka trajanja pozajmica za člana šifre 'JJ1':

```
SELECT SUM (Dana), AVG (Dana)    9    5
FROM Pozajmica
WHERE SifC = 'JJ1' ;
```

U vezi poslednjeg primera, neophodna je jedna napomena za funkciju **AVG**: rezultat **AVG** funkcije preuzima tip od kolone navedene kao argument, pa je u konkretnom slučaju došlo do gubitka decimala i zaokruživanja rezultata sa 4.5 na 5.

Na kraju, skrenimo pažnju na nešto što je razumljivo samo po sebi. U specifikaciji rezultata upita koje smo do sada obradili ne smeju se mešati konstrukcije **R-Izraz** i **G-Izraz**. Primera radi, upit

```
SELECT SifC, SUM (Dana)           ?
FROM Pozajmica ;
```

nema smisla, pošto je **SifC** podatak na nivou jednog reda, a **SUM (Dana)** podatak sveden iz više redova.

Iz prethodnog primera pogrešnog upita nazire se šta se htelo: želeo se prikaz sume trajanja pozajmica po šiframa članova. Da bi se dobili rezultati takvog tipa, neophodno je koristiti svodni upit.

### 6.3.3 Svodni upit nad jednom tabelom

Vratimo se na problem prethodnog upita o sumi trajanja pozajmica po šiframa članova, ali dopunjen time da želimo da prikažemo podatke samo za one članove za koje je zadovoljen neki uslov (na primer, da je suma trajanja pozajmica veća od 10). Postupak u tom slučaju bi mogao biti sledeći (radi kompaktnosti, u ovom primeru je izostavljena kolona SIFN):

- od redova u tabeli Pozajmica formirali bi kao međurezultat novu tabelu sa dve kolone (SifC i Dana) i sa grupisanim redovima sa istim vrednostima SifC;
- na osnovu tako formirane tabele formirali bi novu tabelu iste strukture čiji bi svaki red imao jednu vrednost SifC i vrednost funkcije **SUM (Dana)** izračunate unutar grupe redova sa tom vrednošću SifC;
- na uobičajen način bi prikazali redove tako dobijene tabele koji zadovoljavaju postavljeni uslov.

Opisani postupak možemo prikazati šematski na konkretnom primeru:

SifP	SifC	SifK	Dana		SifC	Dana		SifC	Dana		Rezultat
1	JJ0	004	5	—	JJ0	5		JJ0	12		JJ0 12
2	PP0	007	2	—	JJ0	7		PP0	6		
3	JJ1	005	6	—	PP0	2		JJ1	9		
4	JJ0	008	7	—	PP0	4					
5	PP0	002	4	—	JJ1	6					
6	JJ1	009	3	—	JJ1	3					

↑  
 Uslov:  
 SUM(Dana)>10

Iz prethodnog primera je jasno da bi za formulisanje odgovarajućeg SQL morali dodatno da naglasimo sledeće:

- po kojim kolonama se vrši grupisanje i koje svodne funkcije se traže unutar grupa;
- koji uslov zadajemo za uključenje svodnih redova u rezultat.

Sada smo u mogućnosti da izložimo sintaksu **SELECT** naredbe kojom se realizuje svodni upit nad jednom tabelom:

```

SvodniUpitJednaTabela ::=
    SELECT      S-ListaKolona [ , G-Lista ]
    FROM        Tabela
    [ WHERE      R-Predikat ]
    GROUP BY    G-ListaKolona
    [ HAVING     G-Predikat ]
    [ ORDER BY { Element [ ASC|DESC ] } ,... ] ;

S-ListaKolona ::= { Kolona [ [ AS ] Nadimak ] } ,...
G-Lista ::= { G-Izraz [ [ AS ] Nadimak ] } ,...
G-ListaKolona ::= Kolona ,...

```

U vezi ove definicije neophodne su sledeće napomene i objašnjenja:

<b>S-ListaKolona</b>	kolone grupisanja koje želimo da uključimo u rezultat, podskup kolona navedenih u okviru <b>G-ListaKolona</b> ;
<b>G-Lista</b>	lista svodnih izraza <b>G-Izraz</b> koje želimo da uključimo u rezultat;
<b>R-Predikat</b>	uslov koji svaki red u tabeli <b>Tabela</b> mora da zadovoljava da bi bio uzet u postupak grupisanja;
<b>G-ListaKolona</b>	jedna ili više kolona tabele <b>Tabela</b> po kojima se vrši grupisanje;
<b>G-Predikat</b>	uslov koji svaki red formiran svođenjem mora da zadovolji da bi bio uključen u rezultat; može da sadrži konstrukcije <b>G-Izraz</b> koje nisu sadržane u <b>G-Lista</b> ;
<b>Element</b>	može biti samo kolona sadržana u <b>S-ListaKolona</b> ili izraz sadržan u <b>G-Lista</b> .

Ranije opisani postupak svodenja sa formiranjem dve međutabele nije efikasan i ne koristi se u praksi, ali je vrlo pogodan da bi razumeli funkciju svodne **SELECT** naredbe. Stoga navedimo taj postupak ponovo:

- na osnovu naziva kolona koji se javljaju u **G-ListaKolona** i koje su sadržane u **G-Predikat** i na osnovu konstrukcija **G-Izraz** koje su sadržane u **G-Lista** i **G-Predikat** formira se prva pomoćna tabela sa odgovarajućim kolonama;
- tako kreirana tabela popunjava se redovima koji se dobijaju iz tabele **Tabela** tako što se iz redova koji zadovoljavaju **R-Predikat** izostavljaju nepotrebne kolone, pri čemu su redovi sakupljeni u grupe sa jednakim vrednostima kolona navedenim u **G-ListaKolona**;
- kreira se druga pomoćna tabela iste strukture kao i prethodna;
- za svaku grupu redova u prvoj pomoćnoj tabeli sa jednakim vrednostima kolona navedenim u **G-ListaKolona** formira se jedan red druge pomoćne tabele tako što se pored vrednosti kolona navedenih u **G-ListaKolona** preuzimaju i vrednosti svih izraza iz **G-Lista** i **G-Predikat** koje su izračunate unutar te grupe;
- iz druge pomoćne tabele u rezultat koji se prikazuje uključuju se samo oni redovi koji zadovoljavaju **G-Predikat** , pri čemu se prikazuju samo kolone i izrazi navedeni u **S-ListaKolona** i **G-Lista** ;
- redovi rezultata se prikazuju prema željenom redosledu iz **ORDER BY** klauzule.



*Primeri*

Upiti sa leve strane daju za dati sadržaj baze podataka BIBLIOTEKA rezultate koji su navedeni desno:

Upit za prikaz šifara autora i broja naslova koje su napisali, pri čemu će u prikazu kao naslovi kolona javljaju zadati nadimci:

<code>SELECT</code>	<code>SifA AS Autor, COUNT(*) AS Naslova</code>	AP0	1
<code>FROM</code>	<code>Je_Autor</code>	JN0	1
<code>GROUP BY</code>	<code>SifA ;</code>	DM0	2
		ZP0	2
		AP1	1
		IT0	1

Upit za prikaz šifara članova čija je suma trajanja pozajmica veća od 10 (ovde imamo slučaj funkcije u **HAVING** klauzuli koju ne tražimo u rezultatu):

<code>SELECT</code>	<code>SifC,</code>	JJ0
<code>FROM</code>	<code>Pozajmica</code>	
<code>GROUP BY</code>	<code>SifC</code>	
<code>HAVING</code>	<code>SUM(Dana) &gt; 10 ;</code>	

Upit za prikaz šifara članova i njihovog ukupnog broja pozajmica i ukupnog trajanja pozajmica, ali samo za pozajmice duže od 2 dana:

<code>SELECT</code>	<code>SifC,COUNT(*) ,SUM(Dana)</code>	JJ0	2	12
<code>FROM</code>	<code>Pozajmica</code>	PP0	1	4
<code>WHERE</code>	<code>Dana &gt; 2</code>	JJ1	2	9
<code>GROUP BY</code>	<code>SifC ;</code>			

### 6.3.4 Upiti nad više tabela

Upiti nad više tabela podrazumevaju spajanje tabela po nekom uslovu i realizuju se tako što se u **FROM** klauzuli **SELECT** naredbe navedu umesto jednog više naziva tabela odvojenih zarezima.

U slučaju da se uslov ne navede, od redova navedenih tabela formira se Dekartov proizvod.

*Redni upit nad više tabela*

U slučaju rednog upita nad više tabela (što uključuje i poseban slučaj jedne tabele) sintaksa naredbe **SELECT** je:

```
RedniUpitViseTabela ::=
    SELECT      R-Lista
    FROM      { Tabela [ [ AS ] Nadimak ] } ,...
    [ WHERE      R-Predikat ]
    [ ORDER BY { R-Izraz [ ASC|DESC ] } ,... ] ;
```

U vezi ove definicije napomenimo:

- u **R-Lista** i **R-Predikat** mogu se javiti izrazi koji su izračunjivi nad svakim pojedinim redom koga čine sve kolone svih navedenih tabela;
- za kolone koje postoje u samo jednoj tabeli dovoljno je da navodimo samo nazive u **R-Lista**, **R-Predikat** i u **ORDER BY** klauzuli;
- za kolone koje postoje u više tabela moramo naglasiti iz koje tabele je kolona, i to u formi **Tabela.Kolona** ili **Nadimak.Kolona**;
- **Nadimak** je po pravilu kraće od **Tabela** i treba da nam olakša navođenje kolona, a obavezno je kod spajanja tabele same sa sobom;
- u **R-Predikat** se zadaje uslov spajanja, najčešće u formi uslova jednakosti vrednosti određenih kolona u tabelama;
- upit nad više tabela bez uslova spajanja daje kao rezultat Dekartov proizvod tih tabela.

*Primeri*

Upit koji daje nazive naslova i nazive njihovih oblasti (spajamo tabele Naslov i Oblast po uslovu jednakosti kolona SifO):

```
SELECT  N.Naziv,O.Naziv      PASCAL programiranje    Programski jezici
FROM    Naslov N,Oblast O   Programski jezik C      Programski jezici
WHERE   N.SifO = O.SifO     Racunarske komunikacije Racunarske mreze
ORDER BY N.Naziv;          Relacione baze podataka Baze podataka
```

Upit koji daje imena članova koji su pozajmljivali knjige (spajamo tabele Pozajmica i Clan po uslovu jednakosti kolona SifC; svako ime treba da se javi samo jednom u rezultatu, ali zbog mogućnosti da imamo članove sa istim imenom u rezultat uključujemo i šifre članova): naslovi kolona rezultata biće SifC i Ime.

```
SELECT DISTINCT C.SifC AS SifC, Ime      JJ0 J.Jankovic
FROM   Clan C,Pozajmica P               PP0 P.Petrovic
WHERE  C.SifC = P.SifC ;                 JJ1 J.Jovanovic
```

Upit koji daje šifre i nazive naslova knjiga koje članovi drže kod sebe (spajamo tabele Drzi, Knjiga i Naslov po dva uslova jednakosti kolona koja logičkim operatorom AND kombinujemo u jedinstven uslov).

```
SELECT DISTINCT N.SifN, Naziv      RBP0 Relacione baze podataka
FROM   Drzi D, Knjiga K,Naslov N    PJC0 Programski jezik C
WHERE  D.SifK = K.SifK
      AND K.SifN = N.SifN ;
```

*Redni upit sa svodnim rezultatom nad više tabela*

Za redni upit sa svodnim rezultatom nad više tabele sintaksna definicija naredbe **SELECT** je:

```
RedniUpitViseTabelaSvodniRezultat ::=
    SELECT      G-Lista
    FROM      { Tabela [ [ AS ] Nadimak ] } ,...
    [ WHERE    R-Predikat ]
```

Pri tome, uz sve ranije napomene za **G-Lista** važi još i to da elementi mogu biti iz bilo koje od tabela navedenih u klauzuli **FROM**.

*Primer*

Upit koji daje ukupno trajanje pozajmica svih naslova sa šifrom oblasti 'BP':

```
SELECT SUM(Dana) AS ZbirDana
FROM   Pozajmica P, Naslov N
WHERE  P.SifN = N.SifN
      AND Sifo = 'BP' ;
```

7

*Svodni upit nad više tabela*

Preostalo je još da razmotrimo svodni upit nad više tabela, čija sintaksna definicija glasi:

```

SvodniUpitJednaTabela ::=
    SELECT      S-ListaKolona [ , G-Lista ]
    FROM        { Tabela [ [ AS ] Nadimak ] } ,...
    [ WHERE      R-Predikat ]
    GROUP BY    G-ListaKolona
    [ HAVING     G-Predikat ]
    [ ORDER BY { Element [ ASC|DESC ] } ,... ] ;

```

U vezi ovakve forme **SELECT** naredbe treba dodatno napomenuti sledeće:

- u **S-ListaKolona** mogu da se jave kolone iz bilo koje od navedenih tabela i mogu se koristiti nadimci;
- kao argumenti funkcija u **G-Lista** mogu da se jave kolone iz bilo koje od navedenih tabela i mogu se koristiti nadimci;
- u klauzuli **WHERE** mogu da se jave kolone iz bilo koje od navedenih tabela;
- u klauzulama **GROUP BY**, **HAVING** i **ORDER BY** mogu da se jave kolone iz bilo koje od navedenih tabela, uz poštovanje ranije navedenih pravila i ograničenja koje važe za te klauzule.

*Primeri*

Upit koji daje imena članova i ukupne brojeve njihovih pozajmica (spajamo tabele Pozajmica i Clan; zbog mogućnosti da imamo članove istog imena u rezultat uvodimo i šifre članova):

```
SELECT      P.SifC, Ime, COUNT(*)          JJ0   J.Jankovic   2
FROM        Pozajmica P, Clan C           PP0   P.Petrovic   2
GROUP BY    P.SifC, Ime ;                JJ1   J.Jovanovic  2
```

Upit koji daje šifre oblasti i ukupno trajanje njihovih pozajmica, ali samo za oblasti čija šifra nije 'PJ' (spajamo tabele Pozajmica i Naslov):

```
SELECT      N.Sifo AS Sifo, SUM(Dana) AS ZbirDana      BP   13
FROM        Pozajmica P, Naslov N
WHERE       P.SifN = N.SifN
AND         N.Sifo <> 'PJ'
GROUP BY    N.Sifo ;
```

Upit koji daje nazive oblasti i broj knjiga i broj naslova iz tih oblasti, ali samo za one oblasti iz kojih ima više od jedne knjige (spajamo tabele Knjiga, Naslov i Oblast; sa COUNT(\*) određujemo broj knjiga, a broj naslova sa COUNT (DISTINCT SifN) ):

```
SELECT      O.Naziv,                          Baze podataka      2   1
COUNT(*),                                Programski jezici   6   2
COUNT(DISTINCT K.SifN)
FROM        Knjiga AS K,
Naslov AS N,
Oblast AS O
WHERE       K.SifN = N.SifN
AND         N.Sifo = O.Sifo
GROUP BY    O.Naziv
HAVING      COUNT(* )> 1 ;
```

Za sve tri navedene vrste upita nad više tabela evidentno je da spajanje redova tabela prethodi svim drugim operacijama sa redovima. Slično objašnjenju za svodni upit nad jednom tabelom, možemo radi jasnoće sebi predstaviti sledeći način izvršavanja bilo kog upita nad više tabela:

- od tabela navedenih u FROM klauzuli formira se Dekartov proizvod i tako se kao međurezultat dobija tabela koja sadrži sve kolone svih navedenih tabela;
- upit se dalje izvršava kao da je u pitanju jedna tabela.



### *Spajanje tabele same sa sobom*

Ilustrujmo sada situaciju spajanja tabele same sa sobom. Takav upit ne možemo formulisati nad našom bazom podataka BIBLIOTEKA pa ćemo se poslužiti pogodnim primerom.

Neka je data sledeća tabela kojom pored zaposlenih želimo da evidentiramo i njihovu hijerarhiju u poslu, odnosno ko je kome nadređeni:

RADNIK ( SifR, Ime, SifNad )

Atribut SifNad predstavlja šifru nadređenog. Njegova vrednost će biti neka od vrednosti SifR za sve radnike osim za direktora, kod koga će SifNad imati NULL vrednost, pošto on nema nadređenog.

Formulišimo sada upit koji daje šifre radnika, njihova imena i imena njihovih nadređenih. Rešenje je u tome da tabelu Radnik spojimo samu sa sobom po uslovu da je SifNad iz "prve" tabele jednako SifR iz "druge" tabele. Ovo je situacija kada je upotreba nadimaka tabela neizbežna:

```
SELECT
    R1.SifR AS SifR,
    R1.Ime  AS Ime,
    R2.Ime  AS ImeNad
FROM
    Radnik AS R1,
    Radnik AS R2
WHERE
    R1.SifNad = R2.SifR ;
```

### 6.3.5 Forme predikata u klauzulama WHERE i HAVING

Do sada smo u primerima imali predikate tipa relacionog izraza, a videli smo i kako se pomoću logičkog operatora **AND** iz više relacionih izraza formira složen predikat. Sada je red da razmotrimo u usvojenoj sintaksoj notaciji sve moguće forme predikata u **WHERE** i **HAVING** klauzulama.

Prvo objasnimo pojmove prostog i složenog predikata:

- prost predikat je elementarni logički izraz koji je izračunljiv nad svakim pojedinim redom neke tabele i koji se ne može rastavljati na jednostavnije logičke izraze;
- složen predikat je logički izraz koji je formiran iz jednog ili više jednostavnijih logičkih izraza primenom logičkih operatora **AND**, **OR** i **NOT**.

Za složeni predikat važi sledeća sintaksna definicija rekurzivnog karaktera:

**SlozenPredikat := [ NOT ] ProstPredikat [ AND | OR SlozenPredikat ]**

Po ovoj definiciji mogu se iz prostih formirati svi mogući složeni predikati.

Preostaje nam još da razmotrimo forme prostih predikata. SQL jezik podržava ukupno sedam vrsta prostih predikata, od kojih ćemo sada obraditi šest:

**Izraz** <= | <= | = | <> | >= | > **Izraz**

ispituje da li su vrednosti navedenih skalarnih izraza u zadatom odnosu;

**Izraz** [ NOT ] BETWEEN **Izraz** AND **Izraz**

ispituje da li je (ili nije) vrednost navedenog izraza u zadatim granicama; ovom je ekvivalentno [ NOT ] ( **Izraz** >= **Izraz** AND **Izraz** <= **Izraz** );

**Kolona** IS [ NOT ] NULL

ispituje da li je (ili nije) vrednost navedene kolone NULL;

**ZnakovniIzraz** [ NOT ] LIKE **ZnakovnaMaska**

ispituje da li navedena znakovna vrednost (tipa **CHARACTER**) zadovoljava (ili ne) zadati motiv, pri čemu se za zadavanje motiva u **ZnakovnaMaska** koriste pored običnih znakova i dva specijalna znaka: \_ ima značenje "bilo koji znak", a % "bilo koji broj znakova" što uključuje i ni jedan znak;

**Izraz** [ NOT ] IN ( **Konstanta** ,... )

ispituje da li je (ili nije) vrednost navedenog izraza jednaka nekoj od navedenih konstanti; **Izraz** i **Konstanta** moraju biti istog tipa;

**Izraz** <= | <= | = | <> | >= | > ANY ( **Konstanta** ,... )

ispituje da li je (ili nije) vrednost navedenog izraza u navedenom odnosu sa bar jednom od navedenih konstanti; **Izraz** i **Konstanta** moraju biti istog tipa;

**Izraz** <= | <= | = | <> | >= | > ALL ( **Konstanta** ,... )

ispituje da li je (ili nije) vrednost navedenog izraza u navedenom odnosu sa svim navedenim konstantama; **Izraz** i **Konstanta** moraju biti istog tipa;

Formu relacionog izraza smo već imali u primerima, a formu testiranja na NULL vrednost ne možemo da primenimo na sadržaj baze podataka BIBLIOTEKA, pa nam preostaje da ilustrujemo upotrebu preostalih formi predikata.

*Primeri*

Upit koji daje šifre članova i ukupna trajanja pozajmice trajanja od 5 do 10 dana:

```
SELECT      SifC, SUM(Dana) AS ZbirDana      PP0      6
FROM        Pozajmica                      JJ1      9
GROUP BY    SifC
HAVING      SUM(Dana) BETWEEN 5 AND 10 ;
```

Upit koji daje nazive svih naslova u kojima se nalazi reč "jezik":

```
SELECT Naziv                                Programski jezik C
FROM Naslov
WHERE Naziv LIKE %jezik% ;
```

Upit koji daje šifre knjiga koje odgovaraju naslovima šifara "RBP0" i "RK00":

```
SELECT SifK                                001
FROM Knjiga                                002
WHERE SifN IN('RBP0','RK00') ;             003
```

Prethodni upit, ali uz upotrebu ANY forme predikata:

```
SELECT SifK                                001
FROM Knjiga                                002
WHERE SifN = ANY('RBP0','RK00') ;         003
```

Upit koji daje šifre naslova za sve knjige osim za one čije su šifre '001', '002' i '003':

```
SELECT DISTINCT SifN                      PJC0
FROM Knjiga                              PP00
WHERE SifK <> ALL('001','002','003') ;
```

Forme predikata sa **IN**, **ANY** i **ALL** dolaze do izražaja kod upita sa podupitima, kada se skupovi vrednosti koje te forme koriste dobijaju izvršavanjem upita.

### 6.3.6 Upiti sa podupitima

Podupitom nazivamo **SELECT** naredbu koja se nalazi u sklopu **WHERE** ili **HAVING** klauzule i čije izvršenje prethodi vrednovanju predikata u tim klauzulama. Možemo ih klasifikovati po rezultatu kojeg daju i po načinu izvršavanja u odnosu na "spoljnu" **SELECT** naredbu.

Klasifikacija prema rezultatu daje sledeće vrste upita:

- S-Upit** "skalarni" upit, uvek daje kao rezultat jednu vrednost;
- K-Upit** "kolonski" upit, kao rezultat ili ne daje ni jednu vrednost ili daje skup vrednosti, odnosno redove sa jednom kolonom;
- R-Upit** "redni upit" opšteg tipa, kao rezultat ili ne daje ništa ili daje skup redova sa više kolona.

Klasifikaciju podupita po načinu izvršavanja sprovodimo po tome da li njegovo izvršavanje zavisi od izvršavanja spoljnog upita (upita u kome se nalazi) ili ne. Po tome razlikujemo dve vrste podupita:

- Nekorelisani podupit** podupit čije izvršavanje ni na koji način ne zavisi od izvršavanja spoljnog upita; ovakav podupit se izvršava samo jednom, i to na početku izvršavanja upita u kome se nalazi;
- Korelisani podupit** podupit čije izvršavanje zavisi od izvršavanja spoljnog upita; ovakav podupit se izvršava za svaki red tabele koju obrađuje spoljni upit.

Pošto smo klasifikovali upite po rezultatu, sada smo u mogućnosti da proširimo definicije formi prostih predikata na slučajeve sa podupitima:

$$\text{Izraz} \mid ( \text{S-Upit} ) \leq \mid \leq \mid = \mid < > \mid > = \mid > \quad \text{Izraz} \mid ( \text{S-Upit} )$$

$$\text{Izraz} \mid ( \text{S-Upit} ) \mid [ \text{NOT} ] \text{ BETWEEN Izraz} \mid ( \text{S-Upit} ) \text{ AND Izraz} \mid ( \text{S-Upit} )$$

$$\text{Izraz} \mid ( \text{S-Upit} ) \mid [ \text{NOT} ] \text{ IN ( Konstanta ,... \mid K-Upit )}$$

$$\text{Izraz} \mid ( \text{S-Upit} ) \leq \mid \leq \mid = \mid < > \mid > = \mid > \quad \text{ANY ( Konstanta ,... \mid K-Upit )}$$

$$\text{Izraz} \mid ( \text{S-Upit} ) \leq \mid \leq \mid = \mid < > \mid > = \mid > \quad \text{ALL ( Konstanta ,... \mid K-Upit )}$$

kao i da navedemo sedmu formu prostog predikata:

$[ \text{NOT} ] \text{ EXISTS ( R-Upit )}$

utvrđuje ishod podupita; ako **R-Upit** kao rezultat daje makar jedan red, **EXISTS** daje vrednost "istina", a u suprotnom daje "neistina", a u varijanti sa **NOT** suprotno.

Zbog mogućnosti da **K-Upit** unutar **IN**, **ANY** ili **ALL** ne vrati ni jedan red, usvojeno je sledeće za tu situaciju:

- **IN** i **ANY** daju vrednost 'neistina';
- **ALL** daje vrednost "istina".

Napomenimo ponovo da se podupiti mogu javljati i u **WHERE** i u **HAVING** klauzuli.

Navedimo nekoliko karakterističnih primera za upite sa podupitima nekorelisanog i korelisanog tipa i sa raznim formama predikata. Radi konciznosti nećemo svugde navoditi rezultate upita.

*Primeri*

Sastaviti upit koji daje podatke o pozajmicama nadprosečnog trajanja.

Da bi utvrdili koje pozajmice imaju nadprosečno trajanje, prvo treba da odredimo prosek trajanja svih pozajmica i da zatim trajanje svake pozajmice poredimo sa tim podatkom. Prosek trajanja svih pozajmica odredićemo nekorelisanim prostim podupitom sa svodnim rezultatom, koji se izvršava samo jednom. Ono što je interesantno u ovom slučaju jeste to da su i glavni upit i podupit nad istom tabelom.

```
SELECT *                                3      JJ1    005    6
FROM Pozajmica                        4      JJ0    008    7
WHERE Dana > ( SELECT AVG(Dana)
               FROM Pozajmica ) ;
```

Sastaviti upit koji daje imena članova čije je ukupno trajanje pozajmica veće od 10 dana.

Ovde za svakog člana iz tabele Clan treba prema njegovoj šifri da utvrdimo da li je njegovo ukupno trajanje pozajmica iznad 10, pa će podupit biti korelisan i izvršavaće se za svakog člana.

```
SELECT Ime                                J.Jankovic
FROM Clan C
WHERE 10 < ( SELECT SUM(Dana)
             FROM Pozajmica
             WHERE SifC = C.Sifc ) ;
```

Iz ovog primera vidimo da se korelacija ostvaruje time što se u **WHERE** klauzuli podupita navodi kolona iz tabele koju obrađuje spoljna **SELECT** naredba, pri čemu se poreklo te kolone naglašava nadimkom tabele (može i sa punim imenom tabele).

Sastaviti upit koji daje imena članova čiji je prosek trajanja pozajmica veći od ukupnog proseka trajanja pozajmica.

U ovom primeru prvo jednim nekorelisanim podupitom određujemo ukupni prosek trajanja pozajmica, a zatim za svakog člana korelisanim podupitom određujemo njegov prosek trajanja pozajmica.

```
SELECT Ime                                J.Jankovic
FROM Clan
WHERE ( SELECT AVG(Dana)
        FROM Pozajmica
        WHERE SifC = Clan.Sifc )
      >
      ( SELECT AVG(Dana)
        FROM Pozajmica ) ;
```

Sastaviti upit koji daje imena članova koji drže knjige.

Ovo je primer za korišćenje **IN** forme predikata. Šifre članova koji drže knjige daje nekorelisani podupit.

```
SELECT Ime                                J.Jankovic
FROM Clan                                P.Petrovic
WHERE SifC IN ( SELECT SifC
                FROM Drzi ) ;
```



Sastaviti upit koji daje podake o pozajmicama koje su trajale duže od svih pozajmica člana šifre 'PP0'.

Postoje dva rešenja. Prvo je uz korišćenje **ALL** konstrukcije

```
SELECT *
FROM Pozajmica
WHERE Dana > ALL ( SELECT Dana
                    FROM Pozajmica
                    WHERE SifC = 'PP0' ) ;
```

a drugo se zasniva na tome da ako je Dana veće od maksimuma skupa vrednosti onda je veće od svih tih vrednosti:

```
SELECT *
FROM Pozajmica
WHERE Dana > ( SELECT MAX (Dana)
                FROM Pozajmica
                WHERE SifC = 'PP0' ) ;
```

*Primer*

Sastaviti upit koji daje imena autora koji su napisali sve naslove iz oblasti čija je šifra 'PJ'.

Ovde tražimo imena onih autora čije se šifre u tabeli `Je_Autor` javljaju u kombinaciji sa šiframa svih naslova iz oblasti 'PJ'. Jedno rešenje je zasnovano na preformulaciji upita u "svi autori za koje ne postoji ni jedan naslov iz oblasti 'PJ' koji nije među naslovima koje je napisao taj autor" (slično sm postupili i u poglavlju 5). Dobijamo upit sa dva nivoa korelisanih podupita:

```
SELECT Ime
FROM Autor A
WHERE NOT EXISTS ( SELECT SifN
                    FROM Naslov
                    WHERE Sifo = 'PJ'
                    AND SifN NOT IN ( SELECT SifN
                                      FROM Je_Autor
                                      WHERE Sifa = A.Sifa ) ) ;
```

Ova `NOT EXISTS - NOT IN` konstrukcija predstavlja univerzalno rešenje za klasu problema kod kojih neka kolona treba da se javi u kombinaciji sa svim vrednostima iz datog skupa.

Drugo rešenje je zasnovano na preformulaciji upita u "svi autori za koje je broj naslova koje su napisali iz oblasti 'PJ' jednak ukupnom broju naslova iz te oblasti" i zahteva spajanje tabela u prvom podupitu:

```
SELECT Ime
FROM Autor A
WHERE ( SELECT COUNT(*)
        FROM Je_Autor AS J, Naslov AS N
        WHERE J.SifN = N.SifN
              AND Sifa = A.Sifa
              AND Sifo = 'PJ' )
      =
      ( SELECT COUNT(*)
        FROM Naslov
        WHERE Sifo = 'PJ' ) ;
```

Ovo rešenje je moguće zato što se u tabeli `Je_Autor` svaka kombinacija vrednosti `Sifa-SifN` javlja samo jedanput, inače bi prvi podupit morao da sadrži `DISTINCT` klauzulu u `COUNT` funkciji.

*Primer*

Sastaviti upit koji za svakog člana koji je pozajmljivao knjige daje šifru i ime, ali samo za one čiji je prosek trajanja pozajmica veći od opšteg proseka.

Ovo je slučaj kada se podupit nalazi u **HAVING** klauzuli:

```
SELECT    P.SifC,C.Ime
FROM      Clan C, Pozajmica P
WHERE     C.SifC=P.SifC
GROUP BY  P.SifC, Ime
HAVING    AVG(Dana) > ( SELECT AVG(Dana)
                        FROM  Pozajmica ) ;
```

### 6.3.7 Unija, razlika i presek upita

Pod unijom, razlikom i presekom upita podrazumevamo primenu odgovarajućih skupovnih operatora na skupove redova koje daju pojedini upiti. Jasno je da pri tome upiti koje na taj način kombinujemo moraju zadovoljavati uslov unijske kompatibilnosti, odnosno moraju davati redove sa istim brojem, redosledom i značenjima i tipovima vrednosti, odnosno u ublaženoj varijanti uslova sa istim brojem, redosledom i tipovima vrednosti.

Za uniju, razliku i presek upita možemo formulisati jedinstvenu definiciju

```
KombinovaniUpit ::= { Upit Operator KombinovaniUpit } | Upit
Operator ::= { UNION [ ALL ] } | EXCEPT | INTERSECT
```

sa sledećim značenjem pojedinih klauzula:

<u>UNION</u> [ <u>ALL</u> ]	unija dva upita, pri čemu se eliminišu istovetni redovi ako se ne naglasi <u>ALL</u> ;
<u>EXCEPT</u>	razlika dva upita; od redova upita ispred <u>EXCEPT</u> klauzule ostaju samo oni koji se ne nalaze u rezultatu upita iza te klauzule;
<u>INTERSECT</u>	presek dva upita; od redova oba upita ostaju samo oni koji se nalaze u rezultatima oba upita.

Za razliku upita u mnogim implementacijama koristi se i nestandardna klauzula MINUS. Inače, unija, presek i razlika upita mogu se javljati i u podupitima.

*Primeri*

Sastaviti upit koji daje šifre knjiga koje su bile u prometu (članovi ih drže kod sebe ili su ranije pozajmljivane).

```
SELECT SifK
FROM Drzi

UNION

SELECT DISTINCT SifK
FROM Pozajmica ;
```

Sastaviti upit koji daje nazive naslova sa kojima su knjige kod članova a ranije nisu pozajmljivane.

```
SELECT DISTINCT Naziv
FROM Naslov N, Knjiga K
WHERE N.SifN = K.SifN
AND SifK IN ( SELECT SifK
              FROM Drzi

              EXCEPT

              SELECT DISTINCT SifK
              FROM Pozajmica ) ;
```

Slično, samo sa `INTERSECT` umesto `EXCEPT`, bi glasio upit sa uslovom da su knjige kod članova a pozajmljivane su ranije.

```
SELECT DISTINCT Naziv
FROM Naslov N, Knjiga K
WHERE N.SifN = K.SifN
AND SifK IN ( SELECT SifK
              FROM Drzi

              INTERSECT

              SELECT DISTINCT SifK
              FROM Pozajmica ) ;
```

Pomoću skupovnih operatora može se formulisati još jedno - treće po redu - rešenje za problem pokrivanja skupa vrednosti: Sastaviti upit koji daje imena autora koji su napisali sve naslove iz oblasti čija je šifra 'PJ':

```
SELECT Ime
FROM Autor AS A
WHERE NOT EXISTS ( SELECT SifN
                    FROM Naslov
                    WHERE Sifo = 'PJ'

                    EXCEPT

                    SELECT DISTINCT SifN
                    FROM Je_Autor AS J, Naslov AS N
                    WHERE J.SifN = N.SifN
                      AND J.sifA = A.sifA
                      AND Sifo  = 'PJ' ) ;
```

### 6.3.8 Dodatne mogućnosti u naredbi SELECT

U ovom odeljku osvrnućemo se na neke dodatne mogućnosti u okviru naredbe **SELECT** koje su nastale uz kasnije verzije SQL standarda. To su:

- konstrukcija **CASE** za uslovnu selekciju vrednosti;
- konstrukcija **S-Upit** kao element **SELECT** klauzule;
- konstrukcija **R-Upit** kao element **FROM** klauzule;
- klauzula **JOIN** kao element **FROM** klauzule za sve vrste spajanja tabela.

*CASE selektor vrednosti*

CASE selektor vrednosti je izraz koji prema ispunjenosti jednog ili ni jednog od zadatih uslova daje jednu od niza navedenih vrednosti. Ovaj izraz se može pojaviti svugde gde se očekuje jedna vrednost, ali se najčešće koristi unutar SELECT klauzule.

Sintaksna definicija CASE selektora glasi:

```
SelektorCASE ::=
    CASE slucaj slucaj ... [ ELSE S-Izraz ] END
Slucaj ::= WHEN R-Predikat THEN S-Izraz
```

Kao ilustrativni primer navedimo upit koji rangira naslove prema broju pozajmica u kategorije 1 do 3:

```
SELECT
    SifN AS Naslov,
    CASE
        WHEN COUNT (*) < 2 THEN 1
        WHEN COUNT (*) BETWEEN 2 AND 4 THEN 2
        ELSE 3
    END AS Rang
FROM
    POZAJMICA
GROUP BY
    SifN ;
```

Napomenimo još i to da se CASE selektor često koristi i u SET klauzuli naredbe UPDATE.



### *S-upit kao element SELECT klauzule*

U programskim jezicima uopšte važi jedno načelo koje se zove ortogonalnost, a svodi se na to da se u konstrukcijama jezika na mestu bilo koje sintaksne kategorije može pojaviti bilo koja konstrukcija koja je generiše.

Takve situacije smo već imali u našim ranijim upitima u okviru klauzula **WHERE** i **HAVING**:

- u predikatima tipa poređenja koristili smo kao generatore skalarnih vrednosti **S-Upit** koji daje jednu vrednost - skalar;
- u predikatu pripadanja takođe se može koristiti **S-Upit**.

Po načelu ortogonalnosti, **S-Upit** kao generator skalarne vrednosti može se koristiti i u okviru elemenata navedenih iza **SELECT** klauzule. Tako, umesto upita sa spajanjem koji daje podatke o članovima i ukupne brojeve njihovih pozajmica

```
SELECT      P.SifC, Ime, COUNT(*)
FROM        Pozajmica P, Clan C
GROUP BY    P.SifC, Ime ;
```

možemo formulisati upit

```
SELECT
  SifC,
  Ime,
  ( SELECT
    COUNT(*)
  FROM
    Pozajmica
    WHERE SifC = C.SifC )
FROM
  Clan C ;
```

*R-Upit kao element FROM klauzule*

Do sada smo imali samo jednu situaciju kada se **R-Upit** koristio unutar naredbe **SELECT**. To je bilo unutar predikata **EXISTS** odnosno **NOT EXISTS**.

Po načelu ortogonalnosti, **R-Upit** se može koristiti i kao element **FROM** klauzule gde se ponaša kao tabela. Jedino što pri tome treba uraditi jeste davanje naziva takvoj "tabeli" kao i eventualno imenovanje njenih "kolona" ako nazivi nasleđeni iz te "tabele" ne odgovaraju, a to se sve postiže korišćenjem klauzule **AS**.

Kao primer navedimo upit koji za oblasti čiji su naslovi pozajmljivani daje nazive, broj naslova i broj pozajmica:

```
SELECT
  O.Naziv,
  COUNT(*),
  SUM(P.Broj)
FROM
  Oblast AS O,
  Naslov AS N,
  ( SELECT
    SifN,
    COUNT(*) AS Broj
  FROM
    Pozajmica
  GROUP BY
    SifN ) AS P
WHERE
  O.Sifo = N.Sifo
AND
  N.SifN = P.SifN
GROUP BY
  O.Naziv ;
```

### *JOIN kao element FROM klauzule*

Način kako smo do sada realizovali spajanje bio je preko uslova spajanja. Osim što to nije bilo pregledno, ovakvom formulacijom spajanja nije bilo moguće formulisati vanjska spajanja, o kojima je bilo reči u poglavlju 5.

Korišćenjem klauzule **JOIN** za formulaciju spajanja unutar **FROM** klauzule uklonjeni su svi navedeni nedostaci.

Sintaksna specifikacija za korišćenje **JOIN** kod spajanja dve tabele glasi u koracima, uz napomene gde je to neophodno:

**SpajanjeTabela ::= TabelarniIzraz IzrazJOIN TabelarniIzraz**

**TabelarniIzraz ::= Tabela | ( R-Upit ) | SpajanjeTabela [ AS Nadimak ]**  
 učesnik u spajanju može biti rezultat drugog spajanja.

**IzrazJOIN ::= DekartovProizvod | PrirodnoSpajanje | OstaloSpajanje**

**DekartovProizvod ::= Tabela CROSS JOIN Tabela**

**PrirodnoSpajanje ::= Tabela NATURAL JOIN Tabela**

Spajanje je po svim kolonama istog naziva;  
 iz rezultata se izostavlja duplo javljanje spojnih kolona.

**OstaloSpajanje ::= Tabela VrstaSpajanja JOIN Tabela OsnovSpajanja**

**VrstaSpajanja ::= VrstaUnutrasnje | VrstaVanjsko**

Unutrašnje spajanje je u stvari dosadašnje uobičajeno spajanje.

**VrstaUnutrasnje ::= \_ | INNER**

**INNER** može da se izostavi: **JOIN** i **INNER JOIN** je isto.

**VrstaVanjsko ::= LEFT | RIGHT | FULL OUTER**

**OsnovSpajanja ::= OsnovUslova | OsnovJednakostiKolona**

**OsnovUslova ::= ON R-Predikat**

**OsnovJednakostiKolona ::= USING ( { Kolona, Kolona } ,... )**

Uslov spajanja je da vrednosti polovine kolona iz levog učesnika moraju biti jednake vrednostima polovine kolona iz desnog učesnika; broj kolona je uvek paran.

*Primeri*

Prirodno spajanje koje daje šifre naslova, nazive naslova i nazive oblasti, pod pretpostavkom da se kolona naziva u tabeli OBLAST zove NazivO (u suprotnom upit ne bi vdatio ni jedan red):

```
SELECT
    N.SifN, N.Naziv, O.NazivO
FROM
    Naslov AS N NATURAL JOIN Oblast AS O ;
```

Isto to, preko unutrašnjeg spajanja po jednakosti kolona (izostavljeno je **INNER**):

```
SELECT
    N.SifN, N.Naziv, O.NazivO
FROM
    Naslov AS N JOIN Oblast AS O USING ( N.Sifo,O.Sifo );
```

Isto to, preko unutrašnjeg spajanja po uslovu (da su kolone jednake):

```
SELECT
    N.SifN, N.Naziv, O.NazivO
FROM
    Naslov AS N JOIN Oblast AS O ON ( N.Sifo = O.Sifo );
```

Sledeći upiti odgovaraju upitima sa vanjskim spajanjima koji su navedeni u okviru razmatranja dodatnih operacija relacione algebre u poglavlju 5. Za levo vanjsko spajanje imamo:

```
SELECT
    N.SifN, N.Naziv, N.Sifo, O.Sifo, O.Naziv
FROM
    Naslov AS N
    LEFT OUTER JOIN
    Oblast AS O
    USING ( N.Sifo,O.Sifo );
```

Za desno vanjsko spajanje upit glasi:

```
SELECT
    N.SifN, N.Naziv, N.Sifo, O.Sifo, O.Naziv
FROM
    Naslov AS N
    RIGHT OUTER JOIN
    Oblast AS O
    USING ( N.Sifo,O.Sifo );
```

Konačno, za popuno vanjsko spajanje imamo upit:

```
SELECT
    N.SifN, N.Naziv, N.Sifo, O.Sifo, O.Naziv
FROM
    Naslov AS N
    FULL OUTER JOIN
    Oblast AS O
    USING ( N.Sifo,O.Sifo );
```

## 6.4 SQL naredbe ažuriranja

Deo SQL jezika kojim se mogu vršiti izmene u tabelama čine tri naredbe:

**INSERT** naredba za ubacivanje novih redova u tabelu;

**UPDATE** naredba za izmene redova u tabeli;

**DELETE** naredba za uklanjanje redova iz tabele.

Pre nego što razmotrimo svaku od ovih naredbi posebno, navedimo dve bitne rezlike u odnosu na naredbu upita **SELECT**:

- naredbe ažuriranja se uvek odnose samo na jednu tabelu - ne postoji ažuriranje spoja dve ili više tabela;
- kod naredbi ažuriranja ne postoji svodenje; svaka od ovih naredbi vrši promene na nivou redova tabele na koju se odnosi.

## 6.4.1 SQL naredba ubacivanja INSERT

Kod ubacivanja novih redova u tabelu moramo u SQL naredbi navesti:

- u koju tabelu ubacujemo;
- za koje kolone dajemo vrednosti;
- vrednosti koje ubacujemo.

Sintaksna definicija naredbe ubacivanja **INSERT** u potpunosti odražava navedena tri zahteva:

**Naredba INSERT ::=**

```
INSERT INTO Tabela [ ( Kolona ,... ) ]  
{ VALUES ( Konstanta ,... ) } | R-Upit ;
```

Tabelu u koju ubacujemo navodimo sa **Tabela**, a to za koje kolone ubacujemo vrednosti možemo da naznačimo na dva načina:

- ako iza **Tabela** ne navedenmo ništa, ubacujemo vrednosti za sve kolone u tabeli i po postojećem redosledu (sve ovo definisano je **CREATE TABLE** naredbom);
- ako iza **Tabela** navedemo u zagradama jedno ili više **Kolona** odvojeno zarezima, ubacujemo vrednosti samo za te kolone, dok nenavedene kolone dobijaju NULL vrednost.

Vrednosti koje ubacujemo možemo zadati na dva načina:

- preko **VALUES** klauzule, navođenjem između zagrada jedne ili više konstanti odvojenih zarezima, pri čemu po broju, redosledu i tipu mora postojati saglasnost sa prethodno opisanom specifikacijom kolona; na ovaj način jednom **INSERT** naredbom može se ubaciti samo jedan red u tabelu;
- zadavanjem upita opšteg tipa koji kao rezultat daje ni jedan, jedan ili više redova u kojima su vrednosti po broju, tipu i redosledu saglasne sa prethodnom specifikacijom kolona; na ovaj način jednom **INSERT** naredbom može se ubaciti ni jedan, jedan ili više redova u tabelu; upit može biti bilo koji od onih koje smo prethodno obradili: nad više tabela, sa svođenjem itd.

Važno je naglasiti da se **INSERT** naredba izvršava samo ako se pri tome ne narušavaju ograničenja postavljena prilikom definisanja tabele **CREATE TABLE** naredbom. U suprotnom, sistem za upravljanje bazom podataka neće izvršiti tu naredbu u signaliziraće grešku.

*Primeri*

Naredba kojom se ubacuje podatak o novom naslovu:

```
INSERT INTO   Naslov
VALUES ( 'PJP0','Programski jezik PASCAL','PJ' ) ;
```

Neka smo kreirali tabelu NaslovPJ ( SifN, Naziv ). Sledeća naredba ubacuje u tu tabelu podatke za naslove iz oblasti 'PJ' na osnovu sadržaja tabele Naslov:

```
INSERT INTO   NaslovPJ
SELECT SifN,Naziv
FROM   Naslov
WHERE SifO = 'PJ' ;
```

Neka smo kreirali tabelu SumePozajmica ( SifC, SumaDana ) koja treba da sadrži šifre članova i ukupno trajanje pozajmica. Sledeća naredba obezbeđuje podatke za takvu tabelu (naznaka kolona nije neophodna).

```
INSERT INTO      SumaPozajmica ( SifC, SumaDana )
SELECT      SifC, SUM (Dana)
FROM        Pozajmica
GROUP BY SifC ;
```



## 6.4.2 SQL naredba izmene UPDATE

Kod izmene postojećih redova u tabeli moramo u SQL naredbi navesti:

- u kojoj tabeli vršimo izmenu;
- za koje kolone u redu menjamo vrednosti i kako;
- pod kojim uslovom menjamo neki red.

Za naredbu izmene **UPDATE** koja sadrži sve navedene elemente važi sledeća sintaksna definicija:

```
NaredbaUPDATE ::=
    UPDATE Tabela [ [ AS ] Nadimak ]
    SET { Kolona = R-Izraz | S-Upit } ...
    [ WHERE R-Predikat ] ;
```

Objašnjenje za pojedine delove ove definicije je sledeće:

- sa **Tabela** navodimo tabelu u kojoj vršimo izmenu;
- sa **Kolona** navodimo koju kolonu menjamo; **R-Izraz** sme da sadrži samo konstante i nazive kolona i mora biti izračunljiv nad svakim pojedinim redom iste tabele; **S-upit** može biti nad jednom ili više drugih tabela, prost ili svodan; ako menjamo više kolona, to navodimo odvojeno zarezima, a uobičajeno je da se svaka kolona navodi u posebnom redu;
- sa **R-Predikat** koji mora biti izračunljiv nad svakim pojedinim redom iste tabele navodimo uslov koji mora biti ispunjen da bi se u redu sprovele naznačene izmene.

Za **WHERE** klauzulu važi sve što smo naveli kod naredbe **SELECT** : dozvoljene su sve forme predikata i podupiti. Napomenimo i to da sistem za upravljanje bazom podataka ne izvršava one izmene koje narušavaju ograničenja definisana u **CREATE TABLE** naredbi.

*Primeri*

Neka smo u tabeli *Naslov* za naslov šifre 'RBP0' greškom uneli šifru oblasti 'PJ' i sada želimo to da ispravimo. Odgovarajuća **UPDATE** naredba glasi:

```
UPDATE Naslov
SET   sifO = 'BP'
WHERE sifN = 'RBP0' ;
```

Neka je jedan broj članova brisan iz evidencije, a pri tome u tabeli *Pozajmica* nismo naveli nikakvu specifikaciju referencijalnog integriteta (*SifC* nismo proglasili za strani ključ). Usled toga moramo sami da šifre tih članova u *Pozajmica* postavimo na NULL vrednost. Naredba koja to obezbeđuje glasi:

```
UPDATE Pozajmica
SET   SifC = NULL
WHERE SifC NOT IN( SELECT SifC
                   FROM  Clan ) ;
```

### 6.4.3 SQL naredba uklanjanja DELETE

Za uklanjanje redova iz tabele moramo u SQL naredbi navesti:

- iz koje tabele vršimo uklanjanje;
- pod kojim uslovom uklanjamo neki red (ako ne brišemo sadržaj cele tabele).

Za odgovarajuću SQL naredbu **DELETE** sintaksna definicija glasi:

```
NaredbaDELETE ::=
  DELETE FROM Tabela [ [ AS ] Nadimak ]
  [ WHERE R-Predikat ] ;
```

Objašnjenje za pojedine sintaksne kategorije je slično ranijim:

- sa **Tabela** navodimo tabelu u kojoj vršimo uklanjanje redova;
- sa **R-Predikat** koji mora biti izračunljiv nad svakim pojedinim redom navodimo uslov koji mora biti ispunjen da bi se red uklonio iz tabele.

I ovde se u **WHERE** klauzuli mogu javljati sve do sada obrađene forme uključujući i podupite. Takođe, sistem za upravljanje bazom podataka će odbiti uklanjanja iz tabela koja narušavaju ograničenja navedena u **CREATE TABLE** naredbama, a to su dinamičke specifikacije referencijalnog integriteta.

*Primer*

Sledeća naredba uklanja podatke o članu šifre "MM0" koji niti drži neku knjigu kod sebe niti je imao pozajmice:

```
DELETE FROM Clan
WHERE      SifC = 'MM0' ;
```

Za ukljanjanje svih neaktivnih članova iz evidencije (onih koji nisu uzeli ni jednu knjigu) možemo povremeno zadavati sledeću naredbu:

```
DELETE FROM  Clan
WHERE        SifC NOT IN ( SELECT SifC
                           FROM  Drzi

                           UNION

                           SELECT DISTINCT SifC
                           FROM  Pozajmica ) ;
```

## 6.5 Pogledi

Poglede koji predstavljaju izvedene tabele već smo pomenuli kada smo razmatrali naredbe SQL jezika za definiciju baze podataka. Jednom kreiran, pogled se ponaša kao nova tabela u bazi podataka, ali uz jednu bitnu razliku u odnosu na osnovne tabele kreirane sa **CREATE TABLE** naredbom: tabela koja bi odgovarala pogledu trajno ne postoji - ona se dobija izvršavanjem upita kojim je definisan pogled.

### 6.5.1 Osobine i prednosti pogleda

Osvrnimo se još jednom na naredbu kreiranja pogleda čija je sintaksa

```
NaredbaKreiranjaPogleda ::=
    CREATE VIEW Pogled [ ( Kolona ,... ) ] AS Upit ;
```

i uz ponavljanje nekih ranijih napomena navedimo i neke nove:

- **Pogled** ne sme da odgovara imenu tabele ili pogleda koji već postoje;
- ako se ne navede **Kolona ,...**, pogled nasleđuje nazive kolona iz upita koji ga definiše, ali postoje situacije kada to nije moguće (spajanje tabela i svodni upiti); u takvim situacijama navođenje **Kolona ,...** je neophodno;
- **Upit** može biti bilo koji upit od onih koje smo do sada obradili;
- kada je data specifikacija kolona, između nje i definicionog upita **Upit** mora da postoji saglasnost po broju, redosledu i tipu;
- jednom kreiran pogled može uvek da se koristi kao i svaka druga tabela u režimu upita, a pod određenim uslovima i u režimu ažuriranja.

Prednosti koje nam pogledi donose u radu sa relacionom bazom podataka su brojne. Navedimo najznačajnije:

- pogled predstavlja jednu vrstu "podprograma" u SQL-u; jednom kreiran, može se koristiti u podupitima u **WHERE** i **HAVING** klauzulama;
- komplikovani i često korišćeni upiti se mogu formulisati u vidu pogleda koje će korisnici jednostavno pozivati u upitima tipa **SELECT \* FROM Pogled**;
- pogled razrešava problem pojavljivanja viška podataka u svodnim upitima (kada u određenim implementacijama pravila za **SELECT** naredbu nalažu da pored traženih podataka u rezultat uključimo i nepotrebne po kojima se grupiše);
- pogledi znatno olakšavaju uspostavu kontrole pristupa bazi podataka, što će biti naknadno objašnjeno.

*Primeri*

Sledeći pogled olakšava problem uvida u čitanost naslova preko pregleda koji za svaku oblast daje šifru, naziv, i ukupno trajanje pozajmica:

```
CREATE VIEW    Citanost ( SifO, Naziv, SumaDana )
AS SELECT      O.SifO, O.Naziv, SUM(Dana)
FROM          Pozajmica P, Naslov N, Oblast O
WHERE         P.SifN = N.SifN
AND          N.SifO = O.SifO
GROUP BY      O.SifO, O.Naziv ;
```

Sa ovim treba samo zadavati jednostavni upit

```
SELECT * FROM Citanost ;.
```

Upit koji za naslove koji su imali pozajmice daje šifre, nazive i broj pozajmica može se formulisati preko pogleda

```
CREATE VIEW    BrojPozajmica ( SifN, Broj )
AS SELECT      SifN, COUNT(*)
FROM          Pozajmica
GROUP BY      SifN ;
```

i upita koji ga koristi

```
SELECT        N.SifN, N.Naziv, P.Broj
FROM          Naslov AS N
JOIN
  BrojPozajmica AS P
USING ( N.SifN, P.SifN ) ;
```

Ako bi hteli da se u rezultatu prethodnog upita pojave i naslovi bez pozajmica za koje će kao broj pozajmica biti navedena vrednost 0, to bi postigli sa upitom:

```
SELECT
  N.SifN,
  N.Naziv,
  CASE
    WHEN P.Broj IS NULL THEN 0
    ELSE P.Broj
  END
FROM
  Naslov AS N
  LEFT OUTER JOIN
  BrojPozajmica AS P
  USING ( N.SifN, P.SifN ) ;
```

Ovim je ilustrovan i veoma česta primena **CASE** selektora - zamena NULL vrednosti sa nekom konkretnom vrednošću.



## 6.5.2 Ažurabilnost pogleda

Svaki pogled je upitan, u smislu da preko njega može da se realizuju upiti, ali samo neki pogledi mogu da se koriste za izmene podataka u tabelama. Takve poglede nazivamo ažurabilnim.

Da bi pogled bio ažurabilan, odnosno upotrebljiv u `INSERT`, `UPDATE` i `DELETE` naredbama, svakom redu u njegovom rezultatu mora da odgovara samo jedan red u nekoj izvornoj tabeli, a svakoj koloni samo jedna kolona u nekoj izvornoj tabeli. U suprotnom, ne može se utvrditi koji redovi i kolone u izvornim tabelama treba da budu predmet ažuriranja.

Na osnovu prethodno izloženog opšteg kriterijuma mogu se navesti suštinska ograničenja koje pogled mora da zadovoljava da bi bio ažurabilan. Ta ograničenja se sva odnose na definicioni upit pogleda i ona su:

- **Upit** mora biti nad jednom tabelom; upit nad više tabela i kombinacija upita sa **UNION**, **EXCEPT** ili **INTERSECT** nisu dozvoljeni;
- **Upit** sme da sadrži samo imena kolona u specifikaciji rezultata; konstante, izrazi i agregatne funkcije nisu dozvoljeni;
- **Upit** ne sme da sadrži **DISTINCT** klauzulu u specifikaciji rezultata;
- **Upit** ne sme biti svodni - **GROUP BY** klauzula nije dozvoljena.

Pored suštinskih, u konkretnim implementacijama SQL jezika postoje i određena praktična ograničenja za ažurabilnost pogleda, od kojih su najčešća sledeća:

- specifikacija rezultata za **Upit** mora da sadrži kolonu koja je ograničena sa **PRIMARY KEY**;
- **Upit** ne sme da sadrži podupite.

### 6.5.3 Interni pogledi

Za poglede koje smo do sada naveli važi to da su eksterni, odnosno da pre upotrebe treba da se definišu naredbom `CREATE VIEW`. Takvi pogledi ostaju zapisani u bazi podataka i nakon upotrebe, sve dok se ne uklone naredbom `DROP VIEW`.

SQL u novijim verzijama poznaje još jednu vrstu pogleda. To su interni pogledi, definisani za jednolartnu upotrebu unutar samih `SELECT` upita koji ih koriste. To se postiže tako što se neposredno ispred `SELECT` klauzule a iza klauzule `WITH` definiše pogled.

Sintaksna definicija za upit sa takvim pogledom, pisana u više redova radi preglednosti, glasi:

```
UpitSaPogledom ::=
    WITH
        Pogled [ ( _ Kolona ... _ ) ] AS ( Upit _ )
    OsnovniUpit
```

Pri tome Pogled može biti bilo koji upit, i za njegovu sintaksnu specifikaciju važe sve ranije napomene.

Ne postoje nikakvi suštinski razlozi koji ne dozvoljavaju da se iza klauzule `WITH` definiše više internih pogleda.

*Primer*

Upit koji za sve naslove (i one bez pozajmica) daje šifre, nazive i broj pozajmica preko internog pogleda ima sledeću formulaciju:

```

WITH BrojPozajmica ( SifN, Broj )
  AS SELECT   SifN, COUNT(*)
    FROM     Pozajmica
    GROUP BY SifN ;
SELECT
  N.SifN,
  N.Naziv,
  CASE
    WHEN P.Broj IS NULL THEN 0
    ELSE P.Broj
  END
FROM
  Naslov AS N
  LEFT OUTER JOIN
  BrojPozajmica AS P
  USING ( N.SifN, P.SifN ) ;

```

## 6.6 SQL i kontrola pristupa podacima

O kontroli pristupa bazi podataka bilo je reči u uvodnim poglavljima. Suština je u tome da se ostvare sledeći vidovi kontrole:

- *ko* uopšte može da pristupa bazi podataka;
- *čemu* može da pristupi u bazi podataka
- *šta* može da radi sa onim čemu može da pristupi.

Deo SQL jezika za kontrolu pristupa bazi podataka sve to obezbeđuje putem sledećih funkcija:

- kreiranje i uklanjanje korisnika - naloga za rad za bazom podataka;
- dodela i uklanjanje opštih prava za rad sa bazom podataka;
- dodela i uklanjanje posebnih prava za rad sa bazom podataka.

Osnovu dela SQL jezika za kontrolu pristupa bazi podataka čine samo dve naredbe:

- GRANT** naredba dodele;
- REVOKE** naredba uklanjanja.

### 6.6.1 Dodela opštih prava

Kreiranje korisnika se obavlja istovremeno sa dodelom jednog od opštih prava pomoću naredbe forme:

NaredbaKreiranjaKorisnika ::=

GRANT OpstePravo TO Korisnik IDENTIFIED BY Lozinka ;

Ovde **Korisnik** predstavlja naziv (javni podatak) pod kojim se korisnik najavljuje sistemu za upravljanje bazom podataka, dok je **Lozinka** šifra (tajni podatak) kojom korisnik kompletira postupak najave. **OpstePravo** može biti jedno od sledećih:

CONNECT sva dodeljena posebna prava nad tabelama plus kreiranja pogleda nad tim tabelama;

RESOURCE prethodna prava plus pravo kreiranja osnovnih tabela;

DBA (Data Base Administrator): neograničena prava nad bazom podataka, koja po pravilu ima korisnik koji je administrator baze podataka.

Korisnik se najčešće kreira sa **CONNECT** pravom, a opšte pravo iznad toga može mu se dodeliti naredbom u formi:

NaredbaDodeleOpstegPrava ::=

GRANT OpstePravo TO Korisnik ;

### 6.6.2 Uklanjanje opštih prava

Administrator baze podataka može ukloniti neko opšte pravo nekom korisniku sledećom naredbom:

```
NaredbaUklanjanjaOpstegPrava ::=  
    REVOKE OpstePravo FROM Korisnik ;
```

Ako se uklone **RESOURCE** ili **DBA** pravo, korisniku ostaje **CONNECT** pravo.

Uklanjanjem **CONNECT** prava uklanja se i korisnik i on više ne može da se najavljuje za rad sa bazom podataka.

### 6.6.3 Dodela posebnih prava

Posebna prava koja se dodeljuju ili uklanjaju korisnicima odnose se na pojedine tabele i poglede u bazi podataka, kao i na dozvoljene radnje nad njima.

Postupna sintaksna definicija naredbe dodele posebnog prava glasi:

```

NaredbaDodelePosebnogPrava ::=
    GRANT PosebnoPravo ,... | ALL
    ON Tabela | Pogled
    TO Korisnik ,... | PUBLIC
    [ WITH GRANT OPTION ] ;

PosebnoPravo ::= PravoOcitavanja | PravoAzuriranja

PravoOcitavanja ::= PravoUpita | PravoKoriscenja | PravoReferisanja

PravoAzuriranja ::= PravoUbacivanja | PravoIzmene | PravoBrisanja

PravoUpita ::= SELECT [ ( Kolona ,... ) ]

PravoKoriscenja ::= USAGE Domen

PravoReferisanja ::= REFERENCE ( Kolona ,... )

PravoUbacivanja ::= INSERT [ ( Kolona ,... ) ]

PravoIzmene ::= UPDATE [ ( Kolona ,... ) ]

PravoBrisanja ::= DELETE

```

U vezi ove složene definicije neophodne su određene napomene:

- jedno ili više prava **PosebnoPravo** možemo dodeliti jednom ili više korisnika nad jednom tabelom ili pogledom;
- ako umesto jednog ili više korisnika navedemo **PUBLIC**, to obuhvata sve postojeće korisnike (**PUBLIC** je grupa korisnika ugrađena u SQL);
- klauzula **WITH GRANT OPTION** ima značenje da navedeni korisnici mogu dalje dodeljivati sva navedena prava;
- ako u okviru prava upita, ubacivanja i izmene navedemo jednu ili više kolona, to pravo je ograničeno na samo te kolone, a u suprotnom se odnosi na sve kolone tabele
- u okviru prava referisanja obavezno je navođenje bar jedne kolone, a pod “referisanjem” se podrazumeva pozivanje na navedene kolone u ograničenjima koje korisnik zadaje u okviru definicije neke druge tabele (klauzule **CHECK** i **REFERENCES**), kao i u ograničenjima opšte vrste (o tome će biti reči kasnije);
- prava ubacivanja, izmene i brisanja nad pogledom imaju smisla samo ako je pogled ažurabilan;
- Ako umesto posebnih prava navedemo **ALL**, to obuhvata sva prethodno objašnjena prava.



*Primeri*

Neka za bazu podataka BIBLIOTEKA imamo tri korisnika koji treba da imaju prava kako je navedeno:

- Sef: prava unosa, izmene i brisanja podataka o oblastima, naslovima, autorima, autorstvu i knjigama plus prava upita nad svim podacima;
- Radnik: prava unosa, izmene i brisanja podataka o članovima, držanju knjiga, pozajmicama i rezervacijama plus prava upita nad svim podacima;
- Clan: prava upita nad podacima o oblastima, naslovima, autorima i autorstvima

Odgovarajuće SQL naredbe kojima se ostvaruje navedeni sistem prava glase, uz objašnjenja:

Kreiranje korisnika:

```
GRANT CONNECT
  TO Sef IDENTIFIED BY LozinkaSefa ;

GRANT CONNECT
  TO Radnik IDENTIFIED BY LozinkaRadnika ;

GRANT CONNECT
  TO Clan IDENTIFIED BY LozinkaRadnika ;
```

Davanje prava upita:

```
GRANT SELECT
  ON Oblast,Naslov,Autor,Je_Autor
  TO PUBLIC ;

GRANT SELECT
  ON Knjiga,Clan,Drzi,Pozajmica,Rezervacija,Je_Rezervisana
  TO Sef,Radnik ;
```

Davanje prava ažuriranja:

```
GRANT INSERT,UPDATE,DELETE
  ON Clan,Drzi,Pozajmica,Rezervacija,Je_Rezervisana
  TO Radnik ;

GRANT INSERT,UPDATE,DELETE
  ON Oblast,Naslov,Autor,Je_Autor,Knjiga
  TO Sef ;
```

### 6.6.4 Uklanjanje posebnih prava

Sintaksna definicija naredbe za uklanjanje posebnih prava glasi:

```
NaredbaUklanjanjaPosebnogPrava ::=
    REVOKE { PosebnoPravo ,... } | ALL
    ON Tabela | Pogled
    FROM { Korisnik ,... } | PUBLIC ;
```

gde sintaksne konstrukcije **PosebnoPravo**, **Tabela**, **Pogled** i **Korisnik** imaju ranije značenje i uz sledeće napomene:

- jedno ili više prava **PosebnoPravo** možemo ukloniti jednom ili više korisnika nad jednom tabelom ili pogledom;
- ako umesto jednog ili više korisnika navedemo **PUBLIC**, to obuhvata sve postojeće korisnike;
- ako je korisnik u međuvremenu iskoristio eventualnu mogućnost da svoja prava dodeljuje drugim korisnicima, ti drugi korisnici gube tako dodeljena prava.

*Primer*

Neka za prethodno uspostavljeni sistem prava pristupa želimo radniku da uskratimo pravo brisanja članova i da ga dodelimo šefu. To se ostvaruje sledećim parom SQL naredbi:

```
REVOKE DELETE ON Clan FROM Radnik ;
```

```
GRANT DELETE ON Clan TO Sef ;
```

### 6.6.5 Formiranje grupa korisnika

Iz ranije izloženog sledi da ako ne svima nego samo jednom delu korisnika treba dodeliti neka posebna prava, tu dodelu treba spovati za svakog takvog korisnika. U uslovima stotina ili hiljada korisnika (recimo svi studenti nekog fakulteta) to može biti zahtevan posao.

Za razrešenje situacija kakva je prethodna SQL je u kasnijim verzijama predvideo koncept imenovane uloge koja se kreira naredbom čija je sintaksa

**NaredbaKreiranjaUloge ::= CREATE ROLE Uloga ;**

pri čemu **Uloga** mora biti unikatno u odnosu na druge nazive i nazive korisnika. Nakon što je tako kreirana, uloga se koristi u dva koraka:

- u prvom koraku se ulozi kao da je neki korisnik dodeljuju željena posebna prava;
- u drugom koraku se uloga kao izvedeno posebno pravo dodeljuje željenim korisnicima.

*Primer*

Dodela prava upita nad tabelom NASLOV dodeljuje se samo određenim korisnicima na sledeći način:

```
CREATE ROLE GrupaKorisnika ;  
  
GRANT SELECT ON Naslov TO GrupaKorisnika ;  
  
GRANT GrupaKorisnika TO Korisnik1,Korisnik2,Korisnik3 ;
```

### 6.6.6 Upotreba pogleda u ograničavanju prava

Kod ranijih i dela sadašnjih implementacija posebnog prava upita nad nekom tabelom ne postoji mogućnost navođenja kolona kao što je to slučaj kod posebnog prava izmene. Razlog za to je što se isti efekat može postići definisanjem pogleda nad tabelom i davanjem prava nad pogledom .

Prethodna situacija nije jedina primena pogleda u ograničavanju prava nad tabelama. U opštem slučaju, razlikujemo sledeće slučajeve ograničavanja prava nad tabelama primenom pogleda:

- ograničavanje na pojedine redove: postiže se tako što se u definicionom upitu pogleda navede odgovarajući predikat u **WHERE** klauzuli;
- ograničavanje na pojedine kolone: postiže se tako što se u definicionom upitu pogleda navedu odgovarajuće kolone u **SELECT** klauzuli;
- ograničavanje na svodne rezultate: postiže se tako što se navede definicioni upit pogleda sa svodnim rezultatom ili svodnog tipa.

Kod korišćenja pogleda u sistemu dodele posebnih prava korisnicima podrazumeva se sledeće:

- pravo se daje nad pogledom, a ne nad tabelom nad kojom je definisan;
- pravo dato nad pogledom važi tokom izvršavanja definicionog upita pogleda i nad tabelom nad kojom je pogled definisan.

*Primeri*

Pogled koji ograničava pristup tabeli `Naslov` na redove sa šifrom oblasti 'PJ':

```
CREATE VIEW NaslovPJ
AS SELECT *
   FROM Naslov
  WHERE SifO='PJ' ;
```

Pogled koji ograničava pristup tabeli `Naslov` na kolone `SifN` i `Naziv` :

```
CREATE VIEW Naslov1
AS SELECT SifN,Naziv
   FROM Naslov ;
```

Pogledi koji ograničavaju pristup tabeli na svodne podatke:

```
CREATE VIEW Naslov2(Broj)
AS SELECT COUNT(*)
   FROM Naslov ;

CREATE VIEW Naslov3(SifO,Broj)
AS SELECT    SifO,COUNT(*)
   FROM      Naslov
  GROUP BY SifO ;
```

## 6.7 SQL i održavanje integriteta podataka

Mogućnost kontrole i održavanja integriteta podataka podrazumeva da sam sistem upravljanja bazom podataka (SUBP) odbija promene u bazi podataka koje bi narušile njen integritet. Da bi SUBP mogao da “zna” šta se podrazumeva pod “integritetom”, neophodno je da se to na odgovarajući način formuliše. Konceptualno, to potpada pod DDL, odnosno jezik SUBP za definiciju podataka.

Savremeni SQL jezik uz integritet ugrađenih tipova podataka podržava specifikaciju integriteta u još 3 nivoa:

- domenskom.
- tabelarnom
- opštem,

i to je jedna od njegovih najznačajnijih deklarativnih mogućnosti.



### 6.7.1 Tipski integritet

Ovaj vid integriteta je nešto što je ugrađeno i u većinu programskih jezika. Najjednostavniji primer za to je varijabli koja je brojnog tipa može da se dodeli samo ispravna brojna vrednost. U slučaju datumskog tipa **DATE** proveru je rigoroznija: pored toga što ograničava opsege vrednosti za dane i mesece ona uključuje i proveru za prestupne godine. Slični važi i za tipove **TIME** i **TIMESTAMP**.

## 6.7.2 Domenski integritet

SQL standard od 1992. godine podržava domene kao korisničke tipove podataka, uz ograničenje da oni moraju biti prosti, odnosno definisani nad nekim baznim (ugrađenim) SQL tipom podatka. To je implementirano naredbama kreiranja i uklanjanja domena. Jednom uveden domen može se koristiti umesto **Tip** u naredbu **CREATE TABLE**.

Definicija sintakse naredbe kreiranja domena glasi

```
NaredbaKreiranjaDomena ::=
    CREATE DOMAIN Domen AS Tip
    [ DEFAULT Const ]
    [ CHECK ( Ogranicenje ) ] ,... ;
```

pri čemu za pojedine delove ove definicije važe napomene:

- Domen** naziv domena, formira po pravilu koje važi za varijable u većini programskih jezika, ne sme biti jednako nazivu ugrađenog tipa i mora biti unikatno u skupu naziva domena koji postoje u bazi podataka;
- Tip** naziv nekog ugrađenog tipa;
- Const** podrazumevana vrednost, neka konstanta tipa **Tip**;
- Ogranicenje** bilo koji SQL logički izraz - predikat u kome se javlja klauzula **VALUE** kao naznaka vrednosti kolone na koju se ograničenje odnosi.

*Primer*

Domen koji treba da posluži za definiciju kolona koje su po prirodi lično ime:

```
CREATE DOMAIN LicnoIme  
    CHECK ( VALUE IS NOT NULL ),  
    CHECK ( VALUE <> SPACE(15) );
```

`SPACE` je SQL funkcija koja daje niz razmaknica. Sa ovakom definicijom domena, u `CREATE TABLE` naredbama za tabele `Autor` i `Clan` za kolone `Ime` naveli bi umesto tipa `CHAR(15)` i ograničenja `NOT NULL` samo `LicnoIme` kao naziv domena. Pri tome, definicija kolone nasleđuje sva ograničenja domena, a u definiciji kolone mogu se navesti i dodatna ograničenja.

Prethodni primer je jednostavan. **CHECK** klauzule mogu biti i složene forme. Navedimo nekoliko mogućnosti koje ovo ilustruju;

- **CHECK ( VALUE IN ( Const ,... | K-Upit ) ) ;**
- **CHECK ( [ NOT ] EXISTS ( R-Upit ) )**, gde se u **R-Upit** u **WHERE** ili **HAVING** klauzuli javlja **VALUE** kao oznaka za vrednost podatka;
- **CHECK ( VALUE ≤ | ≤ = | < > | ≥ | ≥ S-Upit )**.

Pri tome, navedeni upiti mogu biti nad više tabela, sa svodnim rezultatom ili svodni.

Definicija sintakse naredbe uklanjanja domena glasi

```
NaredbaUklanjanjaDomena ::=
    DROP DOMAIN Domen [ RESTRICT | CASCADE ] ;
```

pri čemu su neophodne sledeće napomene:

- RESTRICT** ima za efekat odbijanje izvršenja naredbe uklanjanja domena ako u bar jednoj tabeli u bazi podataka postoji bar jedna kolona koja je definisana nad tim domenom;
- CASCADE** ima za efekat da se u definicijama svih kolona koje su se odnosile na taj domen on zameni baznim tipom nad kojim je definisan.

### 6.7.3 Tabelarni integritet

Tabelarni integritet obuvata ograničenja koja se zadaju za tabele i sa njime smo se susreli kada smo razmatrali `CREATE TABLE` naredbu. Izložimo sada na pregledan način sve vidove tabelarnog integriteta:

- Identifikacioni integritet: obezbeđuje unikatnost i ne-NULL sastav primarnog ključa, ostvaruje se preko `PRIMARY KEY` klauzule za jednu kolonu ili za celu tabelu (jedini način za složeni primarni ključ);
- Referencijalni integritet: obezbeđuje statički i dinamički referencijalni integritet između referišuće i ciljne tabele, ostvaruje se preko `FOREIGN KEY` i `REFERENCES` klauzula za jednu kolonu ili za celu tabelu (jedini način za složeni strani ključ);
- Integritet vrednosti jedne kolone ili više kolona zajedno: obezbeđuje ograničenje nad vrednošću jedne kolone, ostvaruje se preko `NOT NULL`, `UNIQUE` i `CHECK` klauzula na nivou jedne kolone (`OgranicenjeKolone` u `CREATE TABLE` naredbi) ili više kolona zajedno (`OgranicenjeTabele` u `CREATE TABLE` naredbi).

SQL standard dopušta da se u `CHECK` klauzuli umesto jednostavnih predikata koriste složeni SQL logički izrazi sa `VALUE` klauzulom, na isti način kako je to rešeno za domenski integritet. Takođe, u `CHECK` klauzuli za više kolona zajedno mogu se uspostavljati ograničenja u smislu odnosa vrednosti različitih kolona.

### 6.7.4 Opšti integritet

Uz prethodna dva vida integriteta, SQL standard od 1992. godine uveo je i podršku za najopštiji vid integriteta baze podataka u smislu poštovanja proizvoljnih pravila koja se mogu formulirati kao logički SQL izrazi. Taj vid integriteta nazvaćemo opštim, uz napomenu da je originalni naziv “Enterprise integrity”, što u slobodnom prevodu označava integritet u smislu poštovanja pravila koja “po prirodi stvari” ili “po konvenciji” važe u realnom sistemu koga svojim sadržajem predstavlja baza podataka. To je ostvareno naredbama za kreiranje i uklanjanja pravila.

Definicija sintakse naredbe kreiranja pravila glasi, uz propratne napomene:

**NaredbaKreiranjaPravila ::=**

```
CREATE ASSERTION Pravilo  
CHECK ( Ogranicenje ) ;
```

- Pravilo** naziv pravila, formira po ranije navedenom pravilu za imena, mora biti unikatno u skupu naziva pravila koja postoje u bazi podataka;
- Ogranicenje** bilo koji SQL logički izraz – predikat koji može da sadrži upite proizvoljne složenosti - sa svodnim rezultatom ili svodne, nad jednom ili više tabela, sa podupitima i sl.

Sintaksna definicija naredbe za uklanjanje pravila glasi:

**NaredbaUklanjanjaPravila ::= DROP ASSERTION Pravilo ;**

*Primer*

Pravilo koje ne dozvoljava da neki član može ikada imati rezervisana više od 3 naslova može se formulisati kao:

```
CREATE ASSERTION Max3Rezervacije
CHECK (NOT EXISTS(SELECT      SifC
                        FROM      Rezevacija
                        GROUP BY SifC
                        HAVING    COUNT(*)>3));
```

Ovde treba naglasiti da pravila definišu uslov koji mora biti zadovoljen nakon promene u bazi podataka. Ukoliko taj uslov nije zadovoljen, promena se ne prihvata.