



XL

Vladimir Blagojević

RELACIONE  
BAZE PODATAKA I

ICNT Beograd 2006

ISBN 86-86531-07-5

## *PREDGOVOR PRVOM IZDANJU*

Poštovani čitaoci,

Pred Vama je prvo izdanje u formi "elektronske knjige" naslova "RELACIONE BAZE PODATAKA I" koji je proistekao na osnovu prethodna tri izdanja štampane knjige "RELACIONE BAZE PODATAKA" uz znatno dodavanje novog sadržaja.

Ovo izdanje je vezano za jubilej Matematičke gimnazije u Beogradu - 40 godina postojanja - i tim povodom

- autor dr Vladimir Blagojević,
- izdavač "ICNT" iz Beograda, i
- štamparija "MST Gajić" iz Beograda

zajedno ustupaju bez naknade Matematičkoj gimnaziji u Beogradu 180 primeraka u formi PDF teksta sa navigacijom na CD medijumu.

Ono "I" u naslovu nije slučajno: U drugom polугоđu ove školske godine autor planira da izvede dodatnu nastavu u Matematičkoj gimnaziji sa sadržajem koji obuhvata napredne teme iz oblasti baza podataka:

- Upravljanje transakcijama;
- Oporavak od kvara;
- Objektno-relacione baze podataka i SQL jezik;
- Programiranje na nivou baze podataka;
- Skladišta podataka, otkrivanje znanja i analitička obrada podataka.

Te i još neke teme biće uključene u "elektronsku knjigu" pod naslovom "RELACIONE BAZE PODATAKA II" čije je izdavanje predviđeno tokom naredne školske godine.

Beograd, 19. septembra 2006. godine

Autor

# *RELACIONE BAZE PODATAKA 1*

- 1 POJAM BAZE PODATAKA
- 2 SISTEM UPRAVLJANJA BAZOM PODATAKA
- 3 MODELI I STRUKTURE PODATAKA
- 4 RELACIONI MODEL PODATAKA
- 5 FORMALNI UPITNI JEZICI
- 6 SQL - JEZIK RELACIONE BAZE PODATAKA
- 7 ZAVISNOSTI I NORMALNE FORME
- 8 PROJEKTOVANJE RELACIONE BAZE PODATAKA
  
- A RELACIONA BAZA PODATAKA 'BIBLIOTEKA'
- B SINTAKSNA I ALGORITAMSKA NOTACIJA
- C UGRAĐENI PROGRAMSKI SQL

LITERATURA

# 1

## ***POJAM BAZE PODATAKA***

---

Baze podataka, a posebno relacione baze podataka o kojima je reč u ovoj knjizi, predstavljaju najviši domet u oblasti informatike, odnosno automatskog čuvanja, pretraživanja i obrade podataka, što je danas isključivo zasnovano na računarskoj tehnologiji. Pre nego što pojasnimo taj pojam, osvrnućemo se ukratko na istorijat automatske obrade podataka.



## 1.1 Kratak istorijat automatske obrade podataka

Od svojih početaka, koji se mogu datirati u drugoj polovini 19. veka, automatska obrada podataka se razvijala u dva osnovna pravca:

- numerika, odnosno izračunavanja po složenim obrascima i postupcima;
- informatika, odnosno čuvanje, pretraživanje i obrada podataka.

Prvi sistem za automatsku numeričku obradu podataka osmislio je engleski naučnik Čarls Bebidž (Charles Babbage) još 1837 godine. Bio je to jedan izuzetno složen mehanički uređaj sa osnovnim komponentama savremenog računarskog sistema - procesor, radna memorija, ulazna jedinica (bušene kartice) i izlazna jedinica (štampana traka). Ovaj uređaj nazvan "Analitička mašina" bio je toliko ispred svog vremena da je njegova znatno pojednostavljena verzija napravljena tek 1888. godine, kada je izvršena i prva programirana numerička obrada podataka u istoriji: sa tačnošću od 29 cifara izračunato je prvih 40 umnožaka broja  $\pi$ . Dalji razvoj tehnologije doveo je do elektromehaničkih a od 1946. godine nastupa era elektronskih računara za numeričku obradu podataka (striktno govoreći, elektronski su bili samo centralna jedinica za obradu podataka i radna memorija, a ostalo je bilo elektromehaničko).

Automatsko čuvanje, pretraživanje i obrada podataka datiraju od 1884. godine, kada je američki pronalazač Herman Holerit (Herman Hollerith) razvio i izradio sistem za automatsku obradu podataka o popisu stanovništva u SAD. Podaci su se nalazili na bušenim karticama koje su se ručno jedna po jedna ubacivale u uređaj za očitavanje. Obrada podataka se sastojala od prebrojavanja, a programiranje je bilo svedeno na izbor vrste prebrojavanja i izvodilo se prespajanjem određenih električnih kontakata. Rezultati su se očitavali preko brojčanika. Dalji razvoj ovakvih uređaja doveo je prvo do uređaja za automatsko čitanje bušenih kartica, zatim do uređaja za izdavanje rezultata u vidu bušenih kartica, kao i uređaja za automatsko sortiranje i izdvajanje bušenih kartica na osnovu izabranog sadržaja. Kao ulazne i izlazne jedinice pojavile su se tokom prve polovine 20. veka elektromehaničke tastature i pisaće mašine, a ubzo nakon pojave prvog elektronskog računara počela je primena takvih računara i u oblasti čuvanja, pretraživanja i obrade podataka.

Sa pojavom prvog elektronskog računara postepeno se gubi razlika između računara za navedene dve oblasti primene. To je na izvestan način i prirodno: ako informatiku posmatramo kao trodelni kompleks čuvanja, pretraživanja i obrade podataka, numeriku možemo posmatrati kao poseban slučaj informatike sa jednostavnim čuvanjem i pretraživanjem i složenom obradom podataka.

Od tada, elektronski računar postaje uređaj univerzalnog karaktera, pri čemu se željena primena postiže odgovarajućim programom. Vremenom se jasno izdvajaju dve osnovne komponente onog što nazivamo računarskim sistemom:

- hardver, odnosno uređaji u materijalnom smislu;
- softver, odnosno programi koje uređaji izvršavaju tokom rada.

U periodu od 1946. godine, koju uzimamo za početak moderne računarske tehnologije, svaka od navedenih komponenti doživela je burni razvoj. Revolucionarni napredak omogućen je uvođenjem modernih tehnologija kao što su poluprovodnička integrisana kola i magnetni i optički zapis podataka, čije potencijale najbolje ilustruju sledeći aktuelni podaci:

- procesor sa 30.000.000 tranzistora na kristalu površine  $1\text{cm}^2$  i debljine 0.1 mm;
- radna memorija kapaciteta 1.000.000.000 B (znakova) na kristalu iste veličine;
- masovna memorija kapaciteta 500.000.000.000 B veličine džepne knjige.

Uz sve to, došlo je do spektakularnog pada cena kakav nije zabeležen ni u jednoj drugoj oblasti. Primera radi, ekvivalent sadašnjeg PC računara bi, pod pretpostavkom da je uopšte mogao da se napravi, 1960. godine bio bar 10.000.000 puta skuplji.

Detalji računarske tehnologije i njenog razvoja su daleko iznad potreba koje nameće ova knjiga, pa stoga navodimo samo minimum tehnoloških prekretnica bez kojih baze podataka ne bi bile ostvarive:

#### Hardver:

- složene i brze centralne jedinice za obradu podataka - procesori ;
- radne memorije kapaciteta od  $10^9$  B i više;
- masovne memorije - diskovi sa direktnim pristupom željenim podacima;
- kapacitet masovne memorije od  $10^{11}$  B i više.

#### Softver:

- standardni viši programski jezici i programski prevodioci: omogućeno je efikasno programiranje na način blizak prirodnom jeziku umesto na mašinskom jeziku, kao i automatsko prevođenje na mašinski jezik;
- programski poveziivači: omogućeno je formiranje složenih programskih celina iz više odvojeno napisanih, prevedenih i proverenih delova;
- operativni sistemi: omogućeno je efikasno programiranje ulaza i izlaza podataka kao i čuvanja podataka, i postignuta je nezavisnost programa od detalja konstrukcije pojedinih ulaznih i izlaznih uređaja.

## 1.2 Pojam baze podataka

"Baza podataka" je jedan od onih opštih pojmova u informatici koji se ili ne definiše ili definiše raznoliko, od slučaja do slučaja. Primera radi, navedimo nekoliko tipičnih definicija:

*"Baza podataka je model određenog segmenta stvarnog sveta na najnižem nivou apstrakcije".*

*"Baza podataka je skup međusobno povezanih podataka koji se čuvaju zajedno i među kojima ima samo onoliko ponavljanja koliko je neophodno".*

*"Bazu podataka čine povezani podaci i skup programa za pristup tim podacima".*

*"Baza podataka je skup povezanih podataka i svega onog što je neophodno za njihovo održavanje i korišćenje".*

*"Baza podataka je najsavršeniji vid informacionog sistema".*

Svaka od navedenih definicija naglašava neke od osobina baze podataka. S obzirom da je baza podataka usko povezana sa informacionim sistemom, odnosno da čini jedan njegov sastavni deo, neophodno je prvo da pojasnimo taj pojam, koji opet uključuje još opštiji pojam sistema.

### 1.2.1 Pojam sistema

U stvarnosti smo okruženi raznim "stvarima" (osobama, stvarima u materijalnom smislu, konceptualnim celinama itd.) sa raznim svojstvima i između kojih mogu postojati odnosi koji se mogu menjati sa vremenom. Uobičajeni termin za "stvar" u prethodnom smislu je "objekat", koji može podrazumevati "stvar" kako u materijalnom smislu (osoba, automobil, knjiga itd.) tako i konceptualno, bez jasne materijalne egzistencije (oblast izdavaštva, naslov dela, predmet u školi, itd.). U tom smislu, sistem je skup objekata i odnosa koji postoje između njih.

U izvesnom smislu, čitav svet predstavlja jedan sistem sa ogromnim brojem objekata i odnosa, ali nas takvo poimanje stvarnosti ne vodi nigde i praktično je beskorisno. Umesto toga, prema konkretnim potrebama ograničavamo se samo na određeni skup objekata i odnosa između njih. Dalje, sve objekte u sistemu ne tretiramo kao jedan skup objekata, nego ih grupišemo prema sličnim osobinama u skupove objekata. Na sličan način, sve odnose između svih pojedinačnih objekata grupišemo u skupove odnosa prema kriterijumu istovetnosti skupova iz kojih su objekti koji su u odnosu i istovetnosti prirode odnosa. Ilustrujmo to na slučaju jedne biblioteke.



*Primer:*

Biblioteku kao jedan mali deo stvarnosti možemo smatrati za sistem koga čine određeni objekti i odnosi. Umesto da nabrajamo sve pojedinačne objekte, kao skupove objekata možemo uočiti

$$\{ \{ \text{naslov} \}, \{ \text{knjiga} \}, \{ \text{član} \}, \{ \text{autor} \}, \{ \text{polica} \}, \{ \text{sto} \}, \{ \text{stolica} \} \}$$

ali ako odbacimo nebitne objekte i ograničimo se samo na one koji su bitni za rad biblioteke, dobijamo sledeće skupove objekata u sistemu

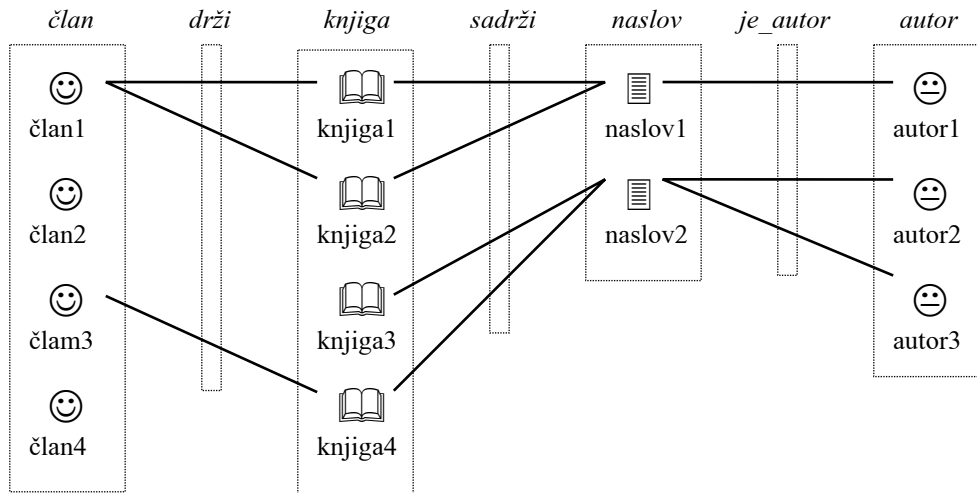
$$\{ \{ \text{naslov} \}, \{ \text{knjiga} \}, \{ \text{član} \}, \{ \text{autor} \} \}$$

Slično tome, ako odbacimo sve nebitne odnose između objekata i ograničimo se samo na one koji su bitni za biblioteku, dobijamo skupove odnosa

$$\{ \{ \text{drži}[\text{knjiga}-\text{član}] \}, \{ \text{sadrži}[\text{knjiga}-\text{naslov}] \}, \{ \text{je\_autor}[\text{autor}-\text{naslov}] \} \}$$

gde smo uz nazive odnosa u uglastim zagradama radi jasnoće naveli i skupove iz kojih su objekti koji su u odnosu.

Za naš sistem u nekom trenutku odgovarao bi grafički prikaz:



Slika 1-1: Sistem BIBLIOTEKA

Na ovom prikazu skupovi su označeni isprekidanim pravougaonicima. Ono što treba imati na umu jeste to da ovaj prikaz odgovara stanju našeg sistema u jednom trenutku. Stanje sistema može vremenom da se menja, što se svodi na promenu broja elemenata pojedinih skupova objekata ili/i skupova odnosa. Navedimo nekoliko primera:

- pojava novog člana: skupu *član* će dodaje novi element član5;
- vraćanje knjige od strane člana: raskida se veza između odgovarajućih elemenata skupova *član* i *knjiga*, odnosno nestaje jedan element iz skupa *drži* ; u situaciji kada svi članovi vrate knjige a ne uzmu ni jednu novu, skup *drži* će biti prazan;
- gubitak knjige koja je bila kod člana: raskidaju se veze između člana i knjige i knjige i naslova, odnosno iz skupova *drži* i *sadrži* nestaje po jedan element, i nestaje jedan element iz skupa *knjiga*.

Iz prethodnog primera zaključujemo da smo u naš sistem uključili samo za nas bitne objekte i odnose i odbacili sve što je nebitno. Na osnovu toga, možemo formulisati opštu definiciju sistema:

*Definicija:*

*Sistem čine izabrani skupovi objekata i izabrani skupovi odnosa između tih objekata.*

## 1.2.2 Pojam informacionog sistema

Za potrebe rada biblioteke koju smo naveli kao primer sistema, svaki bibliotekar bi uspostavio evidenciju o autorima, naslovima, knjigama, članovima, kao i o tome koje knjige su na pozajmici i kod kojih članova. Pre pojave računara, takva evidencija se vodila na karticama, koje su radi lakšeg manipulisanja bile složene uspravno u drvene kutije i to sortirano po nekom obeležju. Svi postupci korišćenja i održavanja te evidencije bili su ručni. Navedimo dva primera:

- korišćenje: radi uvida u to kod koga je određena knjiga, bibliotekar mora da pronađe karticu te knjige i da sa nje pročita ime člana kod koga je i od kada ta knjiga; posle toga, kartica se vraća na mesto odakle je uzeta;
- održavanje: da bi evidentirao jednu pozajmicu, bibliotekar prvo nalazi karticu knjige i upisuje koji član ju je pozajmio i kada, a zatim nalazi karticu člana i tu upisuje koju knjigu je pozajmio i kada; posle toga, svaka kartica se vraća na mesto odakle je uzeta.

Na osnovu takve evidencije bibliotekar je u stanju, pod uslovom da je uredno održava, da u svakom trenutku zna stanje biblioteke i da na osnovu toga obezbedi njen normalan rad. Odsustvo takve evidencije bi dovelo do nesagledivih teškoća u radu. Navedimo nekoliko primera za to:

- da bi saznao da li je neka knjiga u biblioteci, bibliotekar bi morao da pretraži sve police u biblioteci;
- da bi saznao kod koga se nalazi knjiga koja je pozajmljena, bibliotekar bi morao da redom pita sve članove o tome;
- da bi saznao ko su uopšte članovi biblioteke, bibliotekar bi redom morao da pita o tome sve osobe koje bi to mogle da budu (sve učenike jedne škole, sve stanovnike jednog grada, itd.).

Primena računara u vođenju navedene evidencije unosi automatizam u korišćenju i održavanju, ali u suštini ne menja ništa. I evidencija na karticama i evidencija na računaru su primer onoga što nazivamo informacionim sistemom:

### *Definicija*

*Informacioni sistem je skup podataka o nekom sistemu i skup postupaka za njihovo održavanje i korišćenje.*

Skup podataka u informacionom sistemu biblioteke čine drvene kutije sa karticama kod ručnog vođenja evidencije, odnosno datoteke sa slogovima kod primene računara za evidenciju. Za ovaj drugi slučaj, imali bi sledeći skup podataka koji bi odgovarao stanju sistema prikazanom na slici 1-1:

<i>član</i>	<i>drži</i>	<i>knjiga</i>	<i>sadrži</i>	<i>naslov</i>	<i>je_autor</i>	<i>autor</i>
član1	član1-knjiga1	knjiga1	knjiga1-naslov1	naslov1	naslov1-autor1	autor1
član2	član1-knjiga2	knjiga2	knjiga2-naslov1	naslov2	naslov2-autor2	autor2
član3	član3-knjiga4	knjiga3	knjiga3-naslov2		naslov2-autor3	autor3
član4		knjiga4	knjiga4-naslov2			

Slika 1-2: Podaci o sistemu BIBLIOTEKA.

U vezi podataka, kako smo ih organizovali na slici 1-2 (moguće su i druge organizacije), možemo da uočimo:

- svakom skupu objekata u sistemu odgovara jedna datoteka u informacionom sistemu;
- svakom skupu odnosa u sistemu odgovara jedna datoteka u informacionom sistemu;
- svakom elementu bilo kog skupa u sistemu odgovara jedan slog datoteke u informacionom sistemu.

U slučaju ručno vođene evidencije o biblioteci, skup postupaka za održavanje i korišćenje podataka čine pisana ili nepisana pravila koja se sprovode manuelno, dok se postupci za korišćenje podataka svode na ručno nalaženje odgovarajućih kartica.

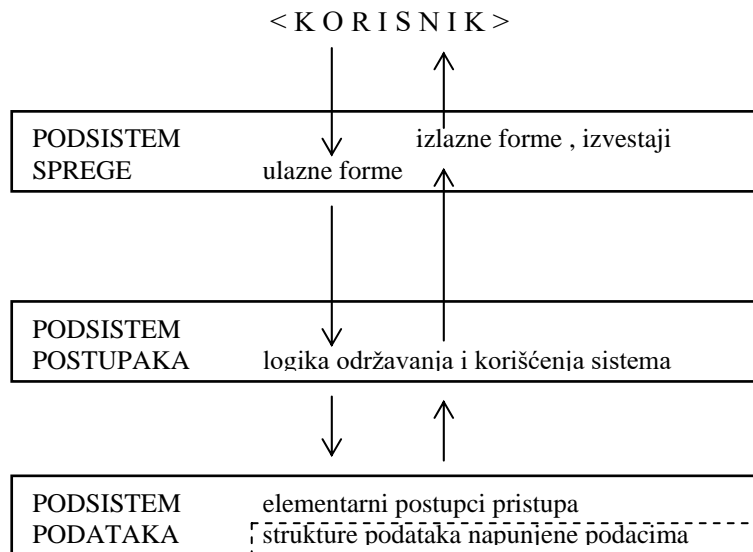
Kod evidencije uz primenu računara skup postupaka za održavanje i korišćenje podataka realizovan je preko programa kojima se automatski, brzo i nepogrešivo obavljaju sve manipulacije sa podacima.

### 1.2.3 Struktura informacionog sistema

Za pojašnjenje pojma baze podataka neophodno je još da se osvrnemo i na strukturu informacionog sistema. odnosno na njegove sastavne delove. To su:

- podsystem sprege: deo zadužen za komunikaciju sa korisnikom u oba smera (unos podataka i izdavanje podataka);
- podsystem postupaka: deo zadužen za postupke održavanja i korišćenja koji su specifični za dati sistem i koji se svode na korišćenje elementarnih postupaka pristupa podacima (održavanje i očitavanje) ;
- podsystem podataka: deo zadužen za čuvanje podataka za dat sistem, zajedno sa elementarnim postupcima pristupa podacima.

Ovakva struktura sa naznačenim tokovima podataka prikazana je na slici 1-3:



Slika 1-3: Struktura informacionog sistema.

Logika održavanja i korišćenja sistema je specifična za svaki konkretni sistem i oslanja se na skladno korišćenje elementarnih postupaka pristupa podacima.

Deo operativnog sistema koji implementira podsystem podataka na elementarnom nivou naziva se fajl-sistem (file-system).

### 1.2.4 Pojam baze podataka

Šta je to baza podataka i u čemu se ona razlikuje od običnog podsistema podataka (fajl sistema)? Pokušaćemo to da razjasnimo na jednom krajnje jednostavnom primeru.

Posmatrajmo slučaj jedne škole u kojoj su se u različitim vremenima javljale i razrešavale potrebe za računarskim informacionim sistemom. Neka je prvo nastala i razrešena potreba administracije škole za vođenjem matičnih podataka o učenicima i podataka o njihovim ocenama. Nešto kasnije, formirana je evidencija za potrebe biblioteke, koja pored matičnih podataka o učenicima koji su članovi biblioteke obuhvata i podatke o tome koje su knjige trenutno kod njih. Na kraju, uspostavljena je i evidencija za potrebe raznih sekcija, koja pored matičnih podataka o učenicima obuhvata i podatke o oblastima kojima se oni bave u sekcijama.



U prethodno navedenim okolnostima, imali bi informacioni sistem koji bi se sastojao iz odvojenih delova, prema potrebama tri korisnika podataka (administracije, biblioteke i sekcija), kako je prikazano na slici 1-4a, pri čemu smo matične podatke o učenicima (šifra, ime i prezime, adresa, telefon) označili sa UČENIK, a podatke za ostale potrebe administracije, biblioteke i sekcija sa OCENE, KNJIGE i OBLASTI, respektivno.



Slika 1-4a: Informacioni sistem škole - varijanta a.

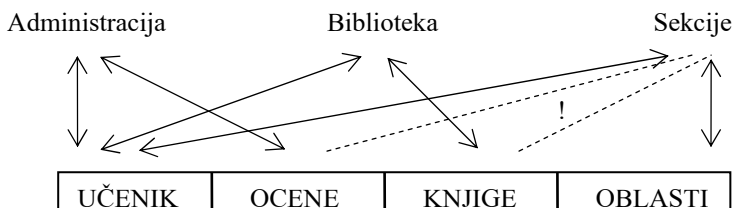
Ovakva situacija potpuno odvojenih delova informacionog sistema, odnosno odvojenih podataka i odvojenih postupaka za održavanje i korišćenje (programa) ima niz nedostataka:

- u slučaju različitog načina zapisivanja podataka (usled korišćenja različitih programskih jezika za manipulaciju podacima), kombinovanje podataka iz odvojenih delova sistema (na primer, učenici, uspeh i oblasti) je teško rešiv ili nerešiv problem;
- čak i u slučaju istog načina zapisivanja podataka, kombinovanje podataka iz odvojenih delova sistema može biti teško rešiv problem ako se u odvojenim delovima sistema koriste različite šifre za iste učenike;
- matični podaci o učenicima (ime i prezime, adresa, telefon itd.) evidentirani su višestruko za one učenike koji su članovi biblioteke ili/i neke sekcije; u slučaju izmene u matičnim podacima, ona se mora sprovesti u oba ili sva tri dela informaciona sistema.

Organizacija informacionog sistema u vidu potpuno odvojenih delova ima i jednu prednost:

- postoje jasno razgraničena prava pristupa podacima: svi korisnici mogu da pristupaju matičnim podacima, ali samo administracija ima pristup podacima o uspehu, biblioteka podacima o pozajmicama knjiga, a sekcije podacima o oblastima interesovanja.

Osnovni uzrok nepogodnosti prethodno navedene organizacije je upravo u podeljenosti informacionog sistema na odvojene delove, odnosno u fragmentaciji podataka. Takvi informacioni sistemi su jedno vreme bili nužnost zbog stanja računarske tehnologije, ali kada su se pojavili savremeniji računari i disk jedinice većeg kapaciteta stekli su se uslovi za objedinjenu organizaciju informacionog sistema. U našem primeru škole, imali bi situaciju prikazanu na slici 1-4b:



Slika 1-4b: Informacioni sistem škole - varijanta b.

Ovakva objedinjena (integrisana) organizacija informacionog sistema otklanja sve nepogodnosti organizacije u vidu odvojenih delova. Konkretno:

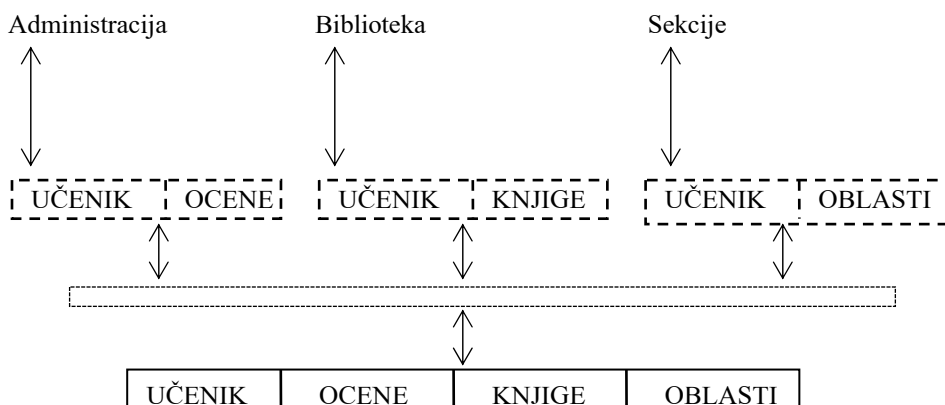
- matični podaci o učenicima se evidentiraju samo jednom, pa ne postoji potreba za višestrukim unosom niti izmenom.

Međutim, objedinjena organizacija informacionog sistema prikazana na slici 1-4b unela je jednu nepogodnost koju nismo imali ranije:

- pošto su podaci objedinjeni, svi korisnici imaju pristup svim delovima podataka, pa i onima koji se njih ne tiču; na slici 1-4b isprekidano su naznačeni samo potencijalno najopasniji nepoželjni pristupi za korisnika "Sekcije", a sličnu situaciju imamo i sa korisnikom "Biblioteka".

Gubitak kontrole pristupa podacima koji je nastao objedinjavanjem podataka je neprihvatljiv, pa zaključujemo da je u organizaciji podataka pored integrisanosti obezbediti još nešto kako bi otklonili taj nedostatak.

Integrisanost podataka u našem informacionom sistemu je vrlo poželjna osobina, ali je dovela do gubitka jedne druge važne osobine, a to je kontrola pristupa pojedinih korisnika podacima. Razlog tome je što smo u obe do sada prikazane varijante informacionog sistema koristili tradicionalnu organizaciju po kojoj programi direktno pristupaju podacima posredstvom operativnog sistema, odnosno njegovog dela koji se zove fajl-sistem. Ono što bi bilo idealno sa stanovišta naših zahteva prikazano je na slici 1-4c:



Slika 1-4c: Informacioni sistem škole - varijanta c.

Organizacija informacionog sistema prikazana na slici 1-4c zadržava sve prednosti obe prethodne organizacije a ne sadrži ni jednu njihovu manu. Konkretno:

- **integrisanost**: podaci su memorisani objedinjeno, bez redudanse (višestrukog pojavljivanja jednih te istih podataka);
- **organizovanost prema potrebama korisnika**: svaki korisnik "vidi" samo one podatke koji su mu neophodni.

U vezi podataka prikazanih isprekidano na slici 1-4c neophodne su određene napomene: Ti podaci fizički ne postoje, nigde nisu zapisani; oni su izvedeni, u smislu da se kada god to zatreba izvode iz integriranih podataka koji fizički postoje u vidu odgovarajućih datoteka na medijumima masovne memorije.

Informacioni sistem prikazan na slici 1-4c predstavlja informacioni sistem zasnovan na onom što se naziva "baza podataka". Možemo formulirati i jasnu definiciju pojma baze podataka kao dela informacionog sistema. U definiciji je podvučeno ono što je specifično za bazu podataka.

### *Definicija*

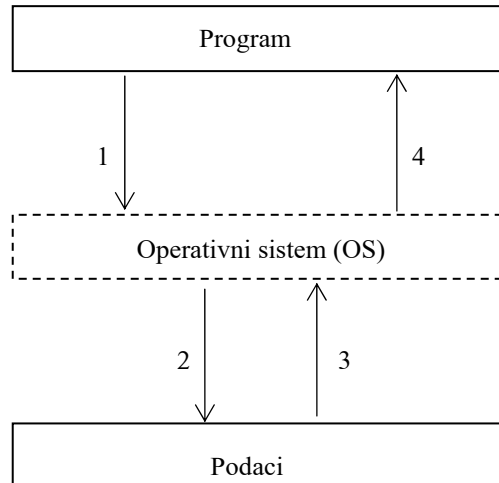
*Baza podataka je integrirani skup podataka o nekom sistemu organizovan prema potrebama korisnika i elementarni skup postupaka za njihovo održavanje i korišćenje.*

Ono što je bitno da uočimo sa slike 1-4c je sledeće: korisnici odnosno njihovi programi ne pristupaju direktno podacima preko operativnog sistema, već posredno, preko komponente koja je naznačena isprekidano i za koju je uobičajeni naziv "sistem upravljanja bazom podataka". Ta komponenta je pored ostalog zadužena i za sva "preslikavanja" između podataka u oba smera. Efekat je taj da svaki korisnik dobija privid organizacije podataka prema svojim potrebama.

Postoji još jedna prednost informacionog sistema zasnovanog na bazi podataka u odnosu na tradicionalni informacioni sistem. To je maksimalna nezavisnost programa od podataka:

- kod tradicionalnog informacionog sistema programi direktno pristupaju podacima posredstvom operativnog sistema i postoji visok stepen njihove zavisnosti od podataka; ako u nekoj datoteci dodamo neki novi podatak u strukturi sloga, potrebno je izvršiti izmenu opisa datoteke u svim programima koji joj pristupaju, bez obzira na to što oni ne koriste taj novi podatak;
- kod korišćenja baze podataka programi pristupaju podacima preko sistema upravljanja bazom podataka koji usled svog mehanizma preslikavanja "sakriva" detalje organizacije datoteka, tako da postoji minimalni stepen zavisnosti programa od podataka; prethodno navedeno dodavanje novog podatka ne iziskuje nikakve izmene u programima koji taj podatak ne koriste.

Na slici 1-5a prikazan je tradicionalni pristup podacima, direktno preko operativnog sistema, sa naznačenom sekvencom događaja:

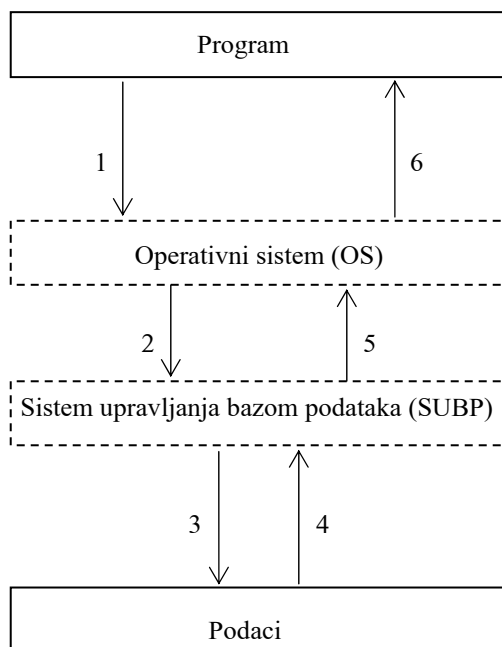


Slika 1-5a: Pristup podacima - tradicionalno.

Za primer čitanja jednog sloga podataka za naš informacioni sistem škole navedimo šta se sve dešava kod tradicionalnog pristupa, kod koga definicija sloga u programu odgovara stvarnoj strukturi sloga u datoteci:

- 1 program izdaje zahtev OS-u za učitavanje jednog sloga;
- 2 OS izdaje hardveru zahtev za učitavanje odgovarajućeg bloka podataka;
- 3 hardver dostavlja OS-u traženi blok podataka;
- 4 OS izdvaja traženi slog iz bloka podataka i dostavlja ga programu.

Na slici 1-5b prikazano je šta se dešava u varijanti sa bazom podataka:



Slika 1-5b: Pristup podacima - kod baze podataka.

Kod takvog pristupa podacima u varijanti baze podataka, slog koji je definisan u programu uopšte ne mora odgovarati strukturi sloga u nekoj datoteci, nego može biti izveden iz stvarnog sloga mehanizmom preslikavanja podataka. Za naš primer, redosled događaja je sledeći:

- 1 program izdaje zahtev SUBP-u za učitavanje jednog sloga;
- 2 SUBP preslikava taj zahtev u zahtev za učitavanje jednog stvarnog sloga OS-u;
- 3 OS izdaje hardveru zahtev za učitavanje odgovarajućeg bloka podataka;
- 4 hardver dostavlja OS-u traženi blok podataka;
- 5 OS izdvaja traženi slog iz bloka podataka i dostavlja ga SUBP-u;
- 6 SUBP iz dobijenog sloga konstruiše traženi slog i dostavlja ga programu.

Navedeni redosled događaja se odnosi na slučaj jednostavnog preslikavanja podataka, kada se traženi slog dobija na osnovu jednog "stvarnog" sloga. U praksi, preslikavanje može biti i takvo da jednom slogu u programu odgovara više "stvarnih" slogova. U tom slučaju, sekvenca 2-3-4-5 će se ponavljati potreban broj puta.



### 1.3 Zadaci baze podataka

Znatan broj zahteva koji se postavljaju pred bazu podataka treba da zadovoljava i dobro formiran informacioni sistem tradicionalnog tipa. Zbog toga te zahteve navodimo objedinjeno i sa kratkim komentarom, pri čemu su podvučeni oni zahtevi koje u potpunosti može da zadovolji samo baza podataka. Ovde navodimo samo zahteve od većeg značaja:

- organizacija prema objektima i odnosima koji postoje u sistemu na koji se baza podataka odnosi;
- integrisanost i kontrolisana redundansa: krajnji cilj integrisanosti je minimalna redundansa (višestruko ponavljanje) podataka; međutim, ponekad svesno želimo da ponavljamo određene podatke radi bržeg rada sa bazom podataka;
- organizacija prema potrebama korisnika: podrazumeva i mogućnost definisanja izvedenih slogova sa podacima;
- sigurnost: podrazumeva efikasnu kontrolu pristupa podacima, u smislu *ko* može da pristupi bazi podataka, *kojim* podacima i *šta* može da radi sa tim podacima;
- konkurentnost: podrazumeva mogućnost sinhronizovanog rada više korisnika istovremeno;
- integritet: podrazumeva automatski oporavak od nasilnih prekida u radu koji dovode do tzv. nekonzistentnih stanja usled delimično izvršenih ažuriranja (unosa, izmene ili brisanja) podataka;
- import podataka: baza podataka se često uvodi kao zamena za neki stariji informacioni sistem, i tada mora postojati mogućnost preuzimanja tih podataka;
- eksport podataka: često se javlja potreba da se postojeća baza podataka zameni nekom još savremenijom bazom podataka, i tada mora postojati mogućnost predaje podataka bazi podataka na koju se prelazi;
- performanse: baza podataka mora da obezbeđuje maksimalni učinak u smislu najveće brzine rada uz najmanje zauzeće računarskih resursa;
- ekonomičnost: odnos učinak-cena treba da je što niži (baze podataka su počele masovno da se uvode u upotrebu onda kada su postale ekonomičnije od tradicionalnih informacionih sistema);
- standardizacija: standardni (dogovoreni) način opisa organizacije baze podataka i operacija nad bazom podataka obezbeđuje maksimalnu nezavisnost i trajnost korisničkih programa za rad sa bazom podataka u odnosu na promenu baze podataka.

## 1.4 Nastanak baza podataka

Krajnje pojednostavljeni primer školskog informacionog sistema kojim smo objasnili razliku između tradicionalnog informacionog sistema i onog zasnovanog na bazi podataka odgovara onome što se dešavalo tokom razvoja informatike. Period od 1884. do 1951. godine se odlikuje datotekama (skupovima slogova) u vidu bušenih kartica, što je ograničavalo svaki program za obradu podataka za rad sa jednom ulaznom i jednom izlaznom datotekom. Od 1951. godine, kada se pojavio prvi računar sa uređajima za upisivanje i čitanje podataka sa magnetnih traka, počinje era složenijih obrada podataka sa više datoteka istovremeno. Uporedo sa tim, u velikim organizacijama koje su koristile automatsku obradu podataka sazrevao je koncept tzv. "Upravljačkog informacionog sistema" koji je podrazumevao informacioni sistem sastavljen od velikog broja datoteka i isto tako velikog broja programa. 1955. godine pojavili su se prvi magnetni diskovi, uređaji koji su omogućili direktan pristup željenim podacima, i to je samo povećalo zahteve i dovelo do novih obrada podataka. Broj odvojenih datoteka i programa u velikim organizacijama je postao toliki da je fragmentacija informacionog sistema dostigla zastrašujuće razmere. Održavanje istovetnosti višestruko evidentiranih podataka i izmene u velikom broju programa postali su najveći problem u domenu obrade podataka.

Razrešenje navedene situacije usledilo je u dva koraka. Prvo je 1965. godine u velikoj meri rešen problem fragmentacije podataka, kada je omogućeno formiranje integrisanih datoteka koje je moglo da koristi više odvojenih programa. Tvorac ovog koncepta, američki naučnik Čarls Bahman (Charles Bachman), može se smatrati tvorcem koncepta baze podataka, mada njegova tvorevina nije imala sve odlike savremene baze podataka. Trebalo je još razrešiti detalje opisa organizacije podataka i manipulacija nad podacima, što je dovršeno kroz standard usvojen 1971. i dopunjen 1978. godine. Po tom konceptu, podaci se predstavljaju preko slogova, a integracija i uspostavljanje veza među podacima se ostvaruju pokazivačkim ulančavanjem slogova u strukture tipa stabla, lista i mreža. Ove baze podataka nazvane su "formatizovanim", mada je naziv "pokazivačka" bliže suštini stvari.

Uporedo sa prethodno navedenim događajima, jedan broj naučnika radio je na ideji da se baza podataka ostvari isključivo pomoću matematičkog koncepta relacije. Ta ideja je postala realnost zahvaljujući američkom naučniku Edgaru Kodu (Edgar Codd), koji je 1971. godine formulisao teoretske osnove relacionih baza podataka, koje su pored organizacije podataka u obuhvatale i formalizam manipulacije nad podacima. Realizacija prve kompletne relacione baze podataka (Sistem R, IBM) otpočela je 1974. godine i završena je pet godina kasnije. Iz tog prototipa razvijene su i prve komercijalne relacione baze podataka, koje su se pojavile na tržištu 1981. godine. Uporedo sa time, sazrevao je koncept standardizacije organizacije i manipulacije podataka kod relacione baze podataka. Rezultat tih aktivnosti jeste standardni jezik SQL za relacione baze podataka, koji je definisan 1986. i nakon toga dopunjavan 1989., 1992., 1995. 1999. i 2003. godine.

Poslednja dopuna SQL standarda rezultat je najnovijeg trenda u oblasti baza podataka. U pitanju je koncept objektno orijentisane baze podataka koji ovde nećemo razmatrati.

## 1.5 Vrste baza podataka

Iz prethodnog osvrta na nastanak baza podataka sledi da danas postoje, izuzimajući objektno orijentisane, dve vrste baza podataka, i to:

- pokazivačke: podaci se predstavljaju slogovima, a integracija podataka i odnosi između njih predstavljaju se pokazivačkim ulančavanjem slogova u stabla, liste ili mreže;
- relacione: i podaci i njihova integracija i odnosi između njih predstavljaju se isključivo relacijama, u stvari slogovima, sa odgovarajućim sadržajem.

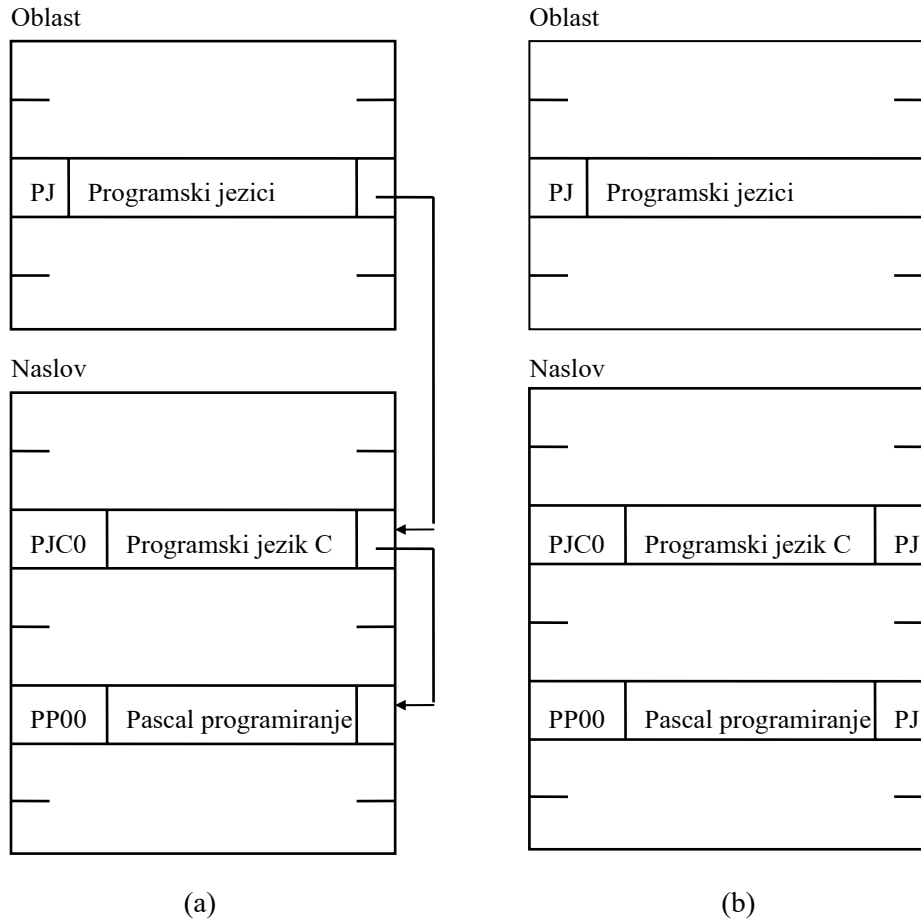
Ono što posebno treba naglasiti za pokazivačke baze podataka je sledeće:

- Njihova bitna karakteristika je u tome da su ulančane strukture podataka koje se primenjuju "vidljive" za korisnika baze podataka, odnosno da utiču na operacije manipulacije nad bazom podataka. U tom smislu, tipične su naredbe tipa: nađi početak liste, nađi sledbenika u listi, nađi nadređenog u stablu, i slično. Drugim rečima, korisnik mora u svom programu jasno da precizira kretanje kroz ulančane strukture podataka, odnosno "navigaciju" kroz bazu podataka, kako se to uobičajeno naziva.

Kod relacionih baza podataka važi sledeće:

- Pojam "navigacije" kroz bazu podataka kao takav ne postoji, pošto nema za korisnika "vidljivih" pokazivača i ulančanih struktura podataka. Pri tome, baza podataka može radi bržeg pristupa podacima sadržati mehanizam pokazivača, ali na nivou organizacije podataka koji je interni i samim tim nije "vidljiv" za korisnika.

Na slici 1-6 ilustrovana je razlika u predstavljanju veza između podataka kod pokazivačke (a) i relacione (b) baze podataka. Dat je primer veze između oblasti i naslova u bazi podataka biblioteke:



Slika 1-6: Povezivanje podataka: pokazivački (a) i relaciono (b).

Navedimo sada kako bi se kod obe vrste baze podataka ostvarila manipulacija nalaženja svih naslova koji pripadaju oblasti čiji je naziv "Programski jezici":

- pokazivačka baza podataka: prvo se nalazi slog u datoteci Oblast sa nazivom "Programski jezici"; zatim se pristupa početku liste slogova u datoteci Naslov koji pripadaju zadatoj oblasti; zatim se preko pokazivača pristupa sledećem slogu u toj listi, i tako do kraja liste;
- relaciona baza podataka: prvo se nalazi slog u datoteci Oblast sa nazivom "Programski jezici" i iz njega se očitava "PJ" kao šifra oblasti; zatim se u datoteci Naslov nalaze svi slogovi kod kojih šifra oblasti ima vrednost "PJ".

Kakav je odnos navedena dva tipa baza podataka danas i koja se više upotrebljava? S obzirom da su se pojavile ranije, pokazivačke baze podataka su prve ušle u upotrebu, ali samo u velikim državnim i privrednim organizacijama. Kada su se pojavile prve relacione baze podataka, one su se odlikovale sporošću u radu, ali je to brzo otklonjeno. Može se reći da danas dominiraju relacione baze podataka. Procenat novih instalacija pokazivačkih baza podataka je zanemarljiv u odnosu one relacionog tipa, ali zato stare instalacije kod vrlo velikih korisnika opstaju. Razlog tome je veliki broj postojećih programa i problem konverzije podataka iz jednog tipa baze podataka u drugi, što je posebno teško u uslovima neprekidnog rada kod dela velikih organizacija.



# 2

## ***SISTEM UPRAVLJANJA BAZOM PODATAKA***

---

Sistem upravljanja bazom podataka je ključni deo svake baze podataka. U ovom poglavlju ćemo razmotriti detalje u vezi tog sistema. odnosno:

- zadatke sistema upravljanja bazom podataka;
- korisnike i načine rada sa bazom podataka;
- vezu programskih jezika sa bazom podataka;
- strukturu i rad sistema upravljanja bazom podataka.

## **2.1 Zadaci sistema upravljanja bazom podataka**

Ako se podsetimo definicije baze podataka, možemo zaključiti da je sistem upravljanja bazom podataka (SUBP) "aktivni" deo baze podataka, ono što smo naznačili kao "skup elementarnih postupaka za održavanje i korišćenje".

### 2.1.1 Definicija baze podataka

SUBP se javlja kao posrednik između korisnika i operativnog sistema, i kao takav je zadužen za sledeće transformacije podataka u oba smera:

- datoteke  $\leftrightarrow$  sveukupni podaci baze podataka;
- sveukupni podaci baze podataka  $\leftrightarrow$  podaci koji odgovaraju potrebama korisnika.

Za obavljanje navedenih transformacija, SUBP mora da raspolaže sa definicijom baze podataka, odnosno sa detaljnim opisom njene strukture. U tom pogledu, sa gledišta realizacije SUBP postoje dva moguća rešenja:

- ugradnja definicije baze podataka u SUBP; ovo podrazumeva dogradnju (programiranje) SUBP za svaku konkretnu bazu podataka;
- definicija baze podataka izvan SUBP, pri čemu SUBP treba da interpretira tu definiciju; ovo podrazumeva univerzalni (interpreterski) karakter SUBP.

Kao univerzalno, usvojeno je drugo rešenje. To u vezi same definicije baze podataka podrazumeva sledeće:

- jezik visokog nivoa, na kome se može formulisati definicija bilo koje konkretne baze podataka koja je bliska prirodnom jeziku i razumljiva;
- jezik niskog nivoa, na kome se može formulisati definicija bilo koje konkretne baze podataka koja je pogodna za efikasnu interpretaciju od strane SUBP;
- poseban deo u okviru SUBP koji prevodi bilo koju definiciju visokog nivoa u odgovarajuću niskog nivoa;
- zapis definicije niskog nivoa u vidu datoteka kojima pristupa SUBP;
- deo SUBP koji interpretira zapisanu definiciju baze podataka radi njenog kreiranja.

U vezi sa prethodnim, uobičajeni su sledeći termini:

- DDL (od "Data Definition Language"): jezik visokog nivoa za opis baze podataka, kako sveukupno tako sa gledišta pojedinih korisnika;
- DDL Prevodilac: deo SUBP koji prevodi DDL definiciju na niski nivo i formira definicioni zapis baze podataka;
- Šema: definicija sveukupne baze podataka
- Podšema: definicija korisničkog viđenja baze podataka;
- Rečnik podataka: definicioni zapis baze podataka, koji sadrži šemu i sve podšeme.
- DDL interpreter: deo SUBP koji interpretira definicioni zapis baze podataka.

### 2.1.2 Manipulacija nad bazom podataka

U vezi realizacije dela SUBP za manipulaciju nad bazom podataka, odnosno za njeno održavanje i korišćenja, takođe su moguća dva pristupa:

- ugradnja konkretnih manipulacija nad konkretnom bazom podataka u SUBP; podrazumeva se dogradnja SUBP za svaku konkretnu bazu podataka;
- zadavanje konkretnih manipulacija nad konkretnom bazom podataka izvan SUBP; SUBP treba da na osnovu svake takve manipulacije i definicije baze podataka obezbedi interpretaciju, odnosno izvršavanje odgovarajuće sekvence elementarnih manipulacija nad bazom podataka; ovakav SUBP je univerzalnog karaktera.

Iz razumljivih razloga, i ovde je usvojeno drugo rešenje, čime je obezbeđena potpuna univerzalnost SUBP u smislu nezavisnosti od konkretne baze podataka i konkretnih operacija nad njom. Takav pristup podrazumeva sledeće:

- jezik visokog nivoa, na kome se na način blizak prirodnom jeziku može formulisati bilo kakva konkretna manipulacija nad bazom podataka;
- jezik niskog nivoa, na kome se može formulisati bilo kakva konkretne manipulacija nad bazom podataka, pri čemu mora da postoji mogućnost efikasne interpretacije od strane SUBP;
- poseban deo u okviru SUBP koji prevodi bilo koju formulu manipulacije iz visokog nivoa u odgovarajuću niskog nivoa;
- poseban deo u okviru SUBP koji interpretira formulu manipulacije niskog nivoa za manipulaciju nad bazom podataka.

U vezi sa navedenim, u upotrebi su sledeći termini:

- DML (od "Data Manipulation Language"): jezik visokog nivoa za opis manipulacija nad bazom podataka;
- DML Prevodilac: deo SUBP koji prevodi DML formulu na formulu manipulacije niskog nivoa;
- DML Interpreter: deo SUBP koji interpretira formulu manipulacije niskog nivoa.

### 2.1.3 Kontrola pristupa bazi podataka

Pored definicije baze podataka i formulacije manipulacija nad njom, kao značajan zadatak SUBP nameće se i kontrola pristupa bazi podataka. To podrazumeva mogućnost definicije kontrole u više nivoa, odnosno:

- *ko* može da pristupa bazi podataka, odnosno ko je sve korisnik baze podataka;
- *kojim* delovima baze podataka korisnici mogu da pristupe;
- *šta* korisnici mogu da rade sa delovima baze podataka kojima imaju pristup.

I ovde se kao prirodno nametnulo univerzalno rešenje po kome se sve te definicije nalaze izvan SUBP koji ih interpretira. To podrazumeva:

- jezik visokog nivoa, na kome se može formulisati definicija bilo kakvih prava pristupa bazi podataka koja je bliska prirodnom jeziku;
- jezik niskog nivoa, na kome se može formulisati definicija bilo kakvih prava pristupa bazi podataka koja je pogodna za efikasnu interpretaciju od strane SUBP;
- poseban deo u okviru SUBP koji prevodi bilo koju definiciju visokog nivoa u odgovarajuću niskog nivoa;
- zapis definicije niskog nivoa u jednu ili više datoteka kojim pristupa SUBP;
- korišćenje zapisane definicije prava pristupa bazi podataka u okviru ostalih funkcija SUBP.

U vezi sa kontrolom prava pristupa bazi podataka koriste se sledeći termini:

- DCL (od "Data Control Language"): jezik visokog nivoa za opis prava pristupa bazi podataka;
- DCL Prevodilac: deo SUBP koji prevodi DCL definiciju na niski nivo i formira definicioni zapis prava pristupa bazi podataka;
- Registar korisnika: definicioni zapis prava pristupa bazi podataka.

### 2.1.4 Ostali zadaci sistema upravljanja bazom podataka

U suštini, svi zadaci SUBP proizilaze iz opštih zahteva koji važe za bazu podataka i koje smo naveli u prethodnom poglavlju. Tri najznačajnija zahteva smo već detaljno obradili, a ovde sažeto navodimo još tri koje nećemo dalje razrađivati pošto to prevazilazi okvire ove knjige:

- upravljanje konkurentnim radom: SUBP u uslovima rada više korisnika mora da obezbedi pravilnu sinhronizaciju njihovog rada, kako bi se izbegle neželjene situacije kao što su korišćenje zastarelih podataka, uzajamno blokiranje i slično;
- upravljanje sistemom oporavka: SUBP u uslovima rada više korisnika treba primenom dnevnika aktivnosti (tzv. "log fajla") i drugim tehnikama da obezbedi podatke neophodne za oporavak baze podataka u slučaju nasilnog prekida rada (tzv. "pad sistema"), kao i da automatski izvrši oporavak po ponovnom pokretanju;
- statistika korišćenja baze podataka: SUBP treba da obezbedi statističke podatke o tome kojim delovima baze podataka i na kakav način se najčešće pristupa, kako bi se na osnovu toga po potrebi mogla sprovesti reorganizacija baze podataka radi što boljih performansi.

## 2.2 Korisnici i načini rada sa bazom podataka

Pod korisnicima baze podataka podrazumevamo osobe koje imaju pristup bazi podataka. Prilikom dodele prava pristupa, svakom korisniku baze podataka dodeljuju se dva podatka:

naziv: javni skraćeni naziv korisnika, podatak koji može biti poznat svima;

lozinka: tajna šifra korisnika, podatak poznat samo korisniku.

Da bi uopšte mogao da radi sa bazom podataka, korisnik mora na početku da sprovede postupak *najave* (tzv. "login"), tokom koje ukucava prvo svoj naziv a zatim i lozinku. Prilikom najave, SUBP poredi uneti naziv i lozinku sa sadržajem registra korisnika i samo u slučaju saglasnosti dozvoljava dalji rad, i to u granicama prava dodeljenih tom korisniku.

Kao prirodno se nameće pitanje: Ko unosi podatke o korisnicima i njihovim pravima pristupa? Očigledno je da mora postojati neki "super-korisnik", osoba koja jedina ima ta prava nad bazom podataka. To nas dovodi do sledeće klasifikacije korisnika baze podataka:

- Administrator: osoba sa pravom kreiranja novih korisnika i dodele prava pristupa, i najčešće bez ikakvih ograničenja u radu sa bazom podataka;
- Korisnik (obični): osoba sa pravom pristupa bazi podataka u granicama koje je odredio administrator.

Pravilo je da obični korisnici nemaju prava kreiranja novih korisnika, ali zato mogu imati prava definicije svojih delova baze podataka, kao i dodele prava pristupa drugim korisnicima nad tim delovima.

Načine rada sa bazom podataka koje samim tim treba da podržava SUBP možemo klasifikovati po više osnova. Prva klasifikacija je po sadržaju rada i svodi se na sledeća dva načina rada:

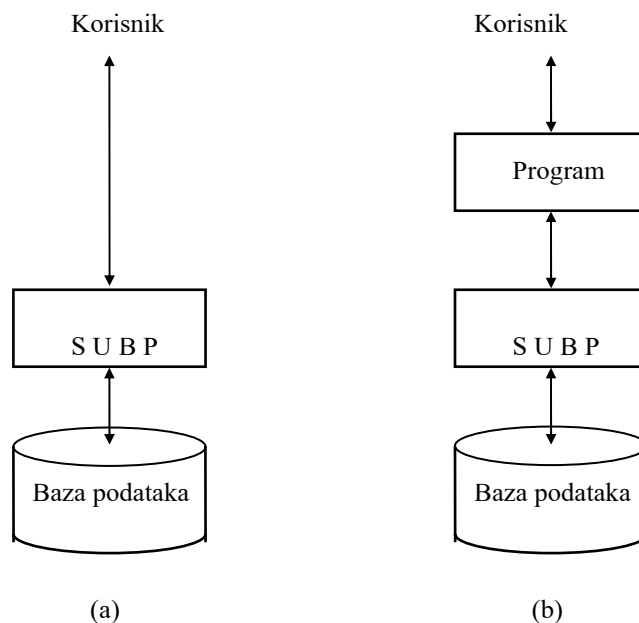
- definicioni: obuhvata unos definicija baze podataka i prava pristupa;
- manipulativni: obuhvata korišćenje baze podataka.

Administrator po pravilu radi sa bazom podataka na definicioni, a korisnik na manipulativni način. U situacijama kada definiše svoje delove baze podataka ili vrši dodelu prava pristupa nad tim delovima, i korisnik radi na definicioni način.

Druga klasifikacija polazi od forme rada i svodi se na sledeće načine rada:

- interaktivni: korisnik unosi jednu po jednu DML, DDL ili DCL naredbu, i SUBP ih jednu po jednu izvršava; u ovakvom načinu rada, korisnik komunicira direktno sa SUBP, a jedina ograničenja su ona koja proizilaze iz prava pristupa zapisanih u registru korisnika;
- programirani korisnik pokreće program preko koga vrši sav unos i pregled podataka, a program u sebi ima ugrađene DML naredbe; u ovakvom načinu rada, korisnik ne komunicira direktno sa SUBP već posredstvom programa, a prava pristupa koja ima mogu dodatno biti ograničena programom.

Na slici 2-1 šematski su prikazana oba načina rada sa bazom podataka.



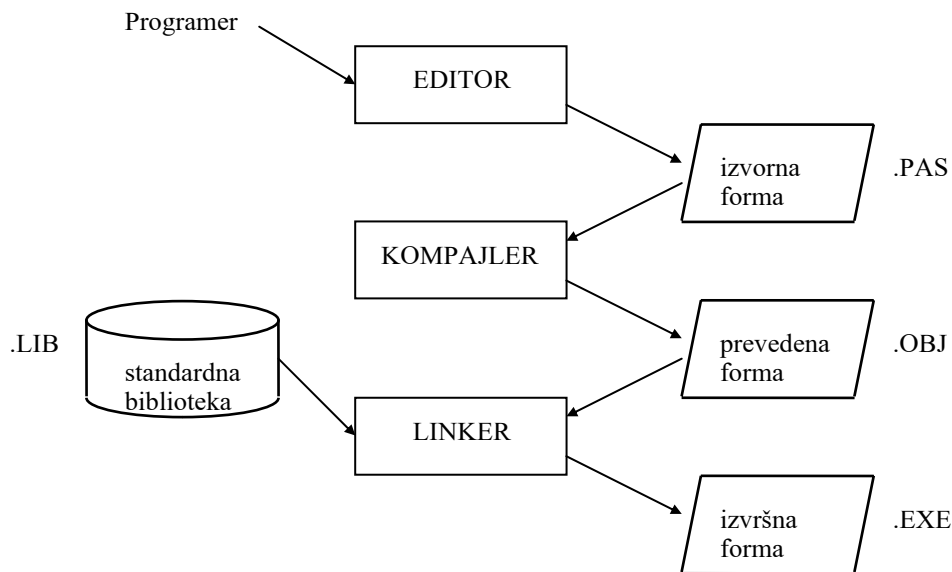
Slika 2-1: Interaktivni (a) i programski (b) način rada sa bazom podataka.

Program za rad sa bazom podataka je najčešće u izvršnoj formi, dobijen prevođenjem sa nekog programskog jezika, ali može biti i u formi DML procedure.



## 2.3 Programski jezici i baze podataka

Radi objašnjenja načina na koji je ostvarena veza između standardnih programskih jezika i baze podataka, neophodno je da se podsetimo na korake razvoja "običnog" programa (koji ne radi sa bazom podataka), od njegovog unosa u formi teksta na programskom jeziku visokog nivoa do formiranja izvršne forme koja može da se izvršava na računaru. U tu svrhu poslužiće nam šematski prikaz na slici 2-2.



Slika 2-2: Koraci formiranja "običnog" programa.

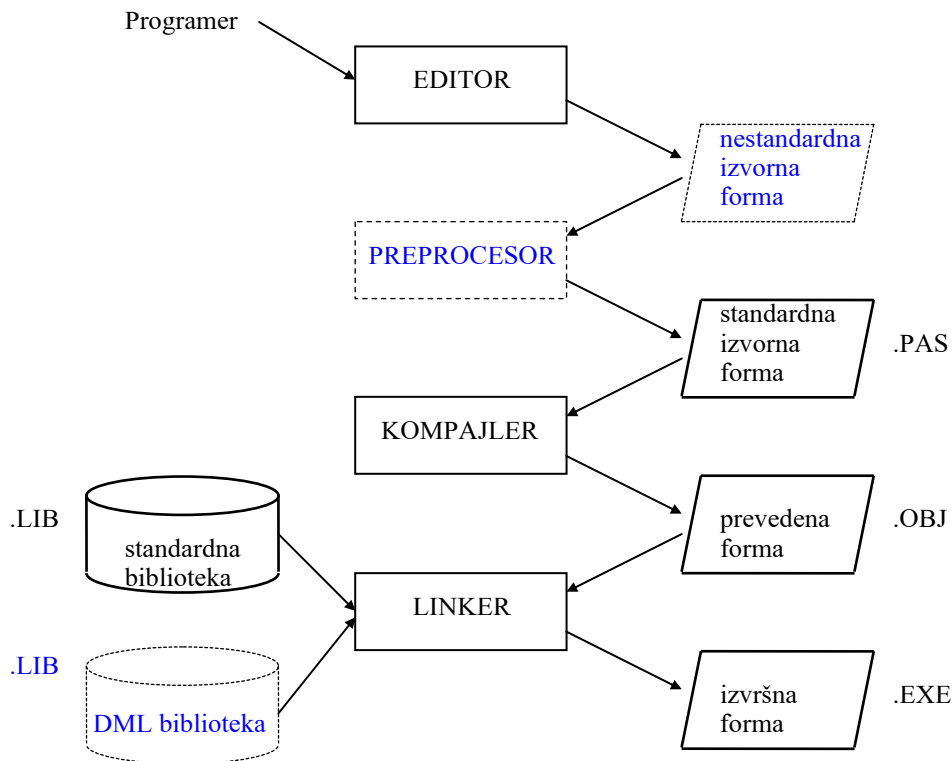
Za slučaj formiranja programa na PASCAL-u, imamo sledeće korake:

- pomoću EDITOR-a (programa za obradu teksta) formiraju se program i potrebni potprogrami u izvornoj PASCAL formi i nastaju jedna ili više datoteka sa ekstenzijom PAS;
- izvorna forma programa se prevodi u prevedenu formu pomoću KOMPAJLER-a (programa za prevođenje) i nastaju jedna ili više datoteka sa ekstenzijom OBJ;
- iz prevedene forme se uz korišćenje biblioteke standardnih procedura i funkcija pomoću LINKER-a (program za povezivanje) dolazi do programa u izvršnoj formi i nastaje datoteka sa ekstenzijom EXE.

Sa pojavom baza podataka i sve većim potrebama za programiranim radom sa njima, nastao je problem kako omogućiti da se iz standardnih programskih jezika pristupa bazi podataka. Na raspolaganju su bile samo dve mogućnosti:

- izmena standarda dopunom sintakse za potrebe rada sa bazom podataka;
- dodavanje podrške radu sa bazom podataka u okvirima postojećih standarda.

Dopuna sintakse je odbačena pošto bi to dodatno opteretilo i onako obimne standarde za programske jezike i dovelo do dodatne nerazumljivosti i pada performansi (u tom pogledu, poučan je primer programskog jezika PL/I koji je doživeo krah upravo zbog toga što je u taj jezik uneto isuviše sintaksnih mogućnosti). Usvojeno je rešenje sprežanja programskih jezika sa bazom podataka čija je suština prikazana na slici 2-3a, na kojoj su dodatni elementi u odnosu na sliku 2-2 prikazani isprekidano.



Slika 2-3a: Koraci formiranja programa za rad sa bazom podataka.

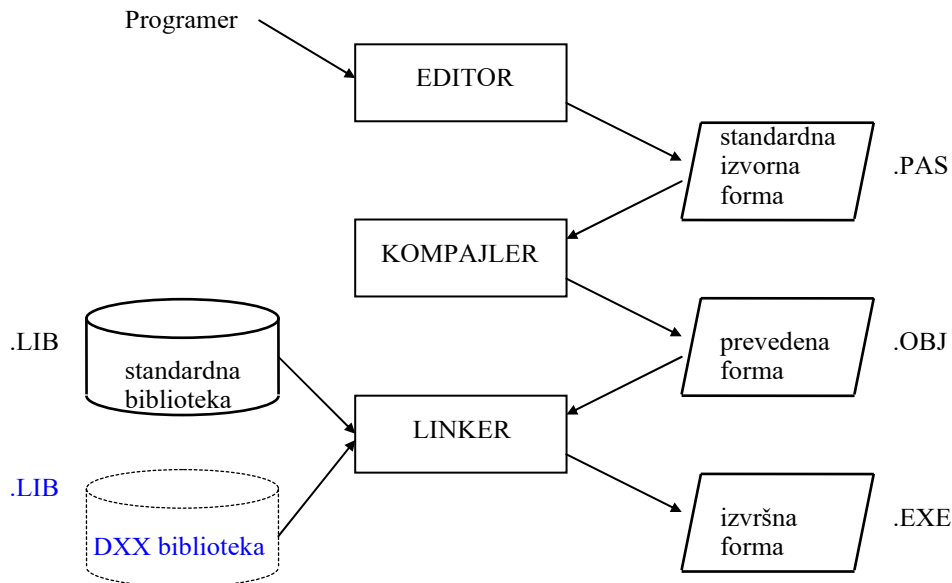
Suština rešenja prikazanog na slici 2-3a je u sledećem:

- pomoću EDITOR-a se kreira nestandardna izvorna forma koja pored standardnih instrukcija programskog jezika sadrži i DML instrukcije za rad sa bazom podataka;
- pomoću PREPROCER-a (specijalni program za obradu DML instrukcija), sve DML instrukcije se prevode u pozive posebnih DML procedura i funkcija za rad sa bazom podataka koji su po standardu za programski jezik; na taj način se dobija standardna izvorna forma programa;
- dalji postupak je isti kao i za "obične" programe, s tim što se u koraku povezivanja LINKER-om pored standardne biblioteke koristi i DML biblioteka koja sadrži sve DML procedure i funkcije za rad sa bazom podataka.

Da bi prethodno opisani postupak bio moguć, uz SUBP treba da bude isporučeno i sledeće, i to za svaki programski jezik od interesa:

- preprocesor DML instrukcija;
- biblioteka prevedenih DML procedura i funkcija.

Uz prethodno navedeni način sprežanja programskih jezika sa bazom podataka naknadno se pojavio još jedan koji je prikazan na slici 2-3b:



Slika 2-3b: Koraci formiranja programa za rad sa bazom podataka.

Suština ovog rešenja je u dopuni standarda, ali ne dopunom sintakse što je svojevremeno sasvim opravdano odbačeno, nego putem dopune predefinisanih funkcija i procedura skupom funkcija i procedura za rad sa bazom podataka. Mana ovog rešenja je to što standardizacija nije univerzalna nego je na nivou konkretnog programskog jezika (različita za svaki programski jezik), ali je prednost to što je osim manipulativnog obuhvaćen i definicioni rad sa bazom podataka (zato je oznaka dodatne biblioteke na slici 2-3b DXX umesto DML). Drugim rečima, sve što je korisniku raspoloživo u režimu interaktivnog rada sa bazom podataka može se ostvariti i u režimu programskog rada. Ovakvo rešenje je standardom od 1992 godine kojim je uvedeno dobilo naziv "Interfejs poziva" (Call-Level Interface). Danas dve najpoznatije implementacije interfejsa poziva su CLI za programski jezik C/C++ i SQLJ za programski jezik Java.

U okviru poredjenja opisana dva načina sprežanja programskih jezika sa bazom podataka treba ukazati i na jednu bitnu razliku. Kod oba rešenja postoji dodatna biblioteka prevedenih funkcija i procedura, ali su te komponente kod prvog rešenja "sakrivene" od programera (pozive generiše preprocesor), dok su kod drugog rešenja "vidljive" programeru (on ih eksplicitno poziva unutar programa).

## 2.4 Struktura i rad sistema upravljanja bazom podataka

Na osnovu svega do sada izloženog možemo ukazati na osnovne komponente SUBP i njegovu strukturu, rad i vezu sa podacima baze podataka.

Kao prvo, primetimo da se podaci baze podataka nalaze izvan SUBP koji je univerzalnog karaktera. Ti podaci se javljaju u tri vida:

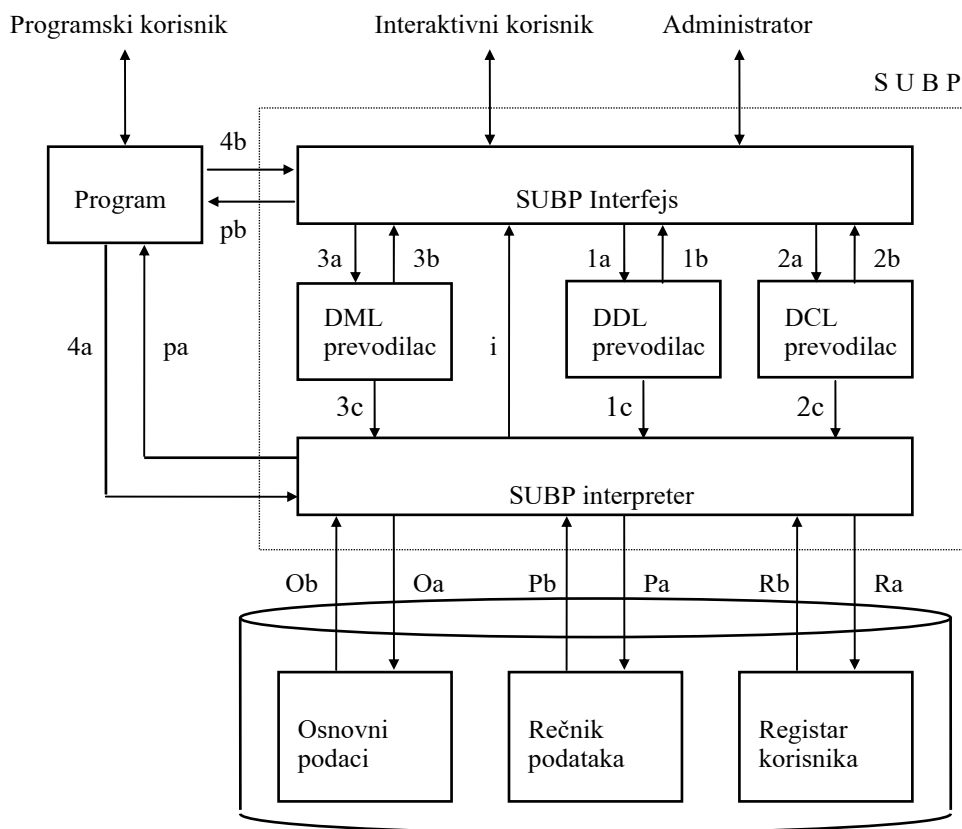
- osnovni podaci: podaci u smislu korisnog (korisničkog) sadržaja baze podataka;
- rečnik podataka: "podaci o podacima", definicioni opis baze podataka;
- registar korisnika: podaci o korisnicima i njihovim pravima pristupa.

Isto tako, i za sve korisničke programe za rad sa bazom podataka smatraćemo da se nalaze izvan SUBP i da čine posebnu komponentu informacionog sistema koju ćemo nazvati "Sistem korisničkih programa" ili skraćeno SKP. Po takvom gledištu, i SUBP i SKP čine ono što nazivamo "skup postupaka za održavanje i korišćenje".

Kao osnovne komponente SUBP uočavamo:

- SUBP interfejs (sprega sa korisnikom): deo SUBP zadužen u uslovima interaktivnog rada za svu komunikaciju sa korisnikom;
- DDL prevodilac: prevodi DDL definicije baze podataka u definiciju niskog nivoa pogodnu za interpretiranje;
- DCL prevodilac: prevodi DCL definicije korisnika i prava pristupa u definiciju niskog nivoa pogodnu za interpretiranje;
- DML prevodilac: prevodi DML opise manipulacija u formu niskog nivoa pogodnu za interpretiranje;
- SUBP interpreter: centralna komponenta SUBP, interpretira sve opise niskog nivoa i tako omogućava održavanje i korišćenje svih podataka.

Na slici 2-4 prikazana je struktura SUBP, njegov rad i veza sa podacima.



Slika 2-4: Struktura SUBP, način rada i veza sa podacima.

Integracija DML, DDL i DCL funkcija u jedinstveni SUBP interpreter spovedena je i zahvaljujući tome da se i osnovni podaci i rečnik podataka i registar podataka zapisuju na isti način u bazi podataka. Time je omogućeno da se za održavanje i korišćenje svih tih podataka koriste isti elementarni postupci.

Način rada SUBP u raznim uslovima možemo razjasniti uz pomoć slike 2-4 na kojoj su označeni bitni tokovi podataka. Posmatrajmo prvo interaktivni rad sa bazom podataka tokom koga se formira definicija baze podataka. Redosled događaja je:

- preko SUBP interfejsa unosi se DDL naredba; interfejs prosleđuje tu naredbu DDL prevodiocu (1a);
- u slučaju greške koja se može utvrditi samo na osnovu prevođenja, DDL prevodilac dostavlja interfejsu poruku o tome (1b); interfejs prikazuje tu poruku i time se završava obrada unete naredbe;
- u slučaju da nema greške navedenog tipa, DDL prevodilac formira odgovarajuću sekvencu naredbi niskog nivoa i prosleđuje je SUBP interpreteru (1c);
- SUBP interpreter izvršava prosleđenu sekvencu naredbi niskog nivoa; prvo se ispituje njena saglasnost sa rečnikom podataka (Pb); u slučaju saglasnosti, ažuriraju se rečnik podataka (Pa, upis ili izmena definicije) i osnovni podaci (Oa, kreiranje "prazne" ili izmena postojeće strukture);
- SUBP interpreter prosleđuje SUBP interfejsu poruku o ishodu prethodne aktivnosti (i) uključujući tu i situacije greške; interfejs prikazuje tu poruku.

U slučaju interaktivnog definisanja prava pristupa bazi podataka redosled događaja je nešto drugačiji, ali samo u delu koji obavlja SUBP interpreter:

- preko SUBP interfejsa unosi se DCL naredba; interfejs prosleđuje tu naredbu DCL prevodiocu (2a);
- u slučaju greške koja se može utvrditi samo na osnovu prevodenja, DCL prevodilac dostavlja interfejsu poruku o tome (2b); interfejs prikazuje tu poruku i time se završava obrada unete naredbe;
- u slučaju da nema greške navedenog tipa, DCL prevodilac formira odgovarajuću sekvencu naredbi niskog nivoa i prosleđuje je SUBP interpreteru (2c);
- SUBP interpreter izvršava prosledenu sekvencu naredbi niskog nivoa; prvo se ispituje njena saglasnost sa rečnikom podataka (Pb) i registrom korisnika (Rb); u slučaju saglasnosti, ažurira se registar korisnika (Ra, upis ili izmena definicije);
- SUBP interpreter prosleđuje SUBP interfejsu poruku o ishodu prethodne aktivnosti (i); interfejs prikazuje tu poruku.



Za interaktivni manipulativni rad sa bazom podataka, kada se menjaju ili koriste osnovni podaci, imamo sledeći redosled događaja:

- preko SUBP interfejsa unosi se DML naredba; interfejs prosleđuje tu naredbu DML prevodiocu (3a);
- u slučaju greške koja se može utvrditi samo na osnovu prevodenja, DML prevodilac dostavlja interfejsu poruku o tome (3b); interfejs prikazuje tu poruku i time se završava obrada unete naredbe;
- u slučaju da nema greške navedenog tipa, DML prevodilac formira odgovarajuću sekvencu naredbi niskog nivoa i prosleđuje je SUBP interpreteru (3c);
- SUBP interpreter izvršava prosleđenu sekvencu naredbi niskog nivoa; prvo se ispituje njena saglasnost sa rečnikom podataka (Pb) i registrom korisnika (Rb); u slučaju saglasnosti, čitaju se (Ob) ili menjaju osnovni podaci (Oa), zavisno od prirode zadate manipulacije;
- SUBP interpreter prosleđuje SUBP interfejsu poruku o ishodu i eventualne rezultate prethodne aktivnosti (i); interfejs sve to prikazuje.

U vezi interaktivnog rada napomenimo još i to da većina SUBP dozvoljava mogućnost formiranja sekvenci naredbi u vidu tekst datoteka koje se zatim kao celina prosleđuju SUBP interfejsu. Pri tome je potrebno na odgovarajući način navesti naziv takve datoteka. U takvoj varijanti rada, SUBP će sve rezultate izvođenja tih naredbi i poruke o ishodu zapisati u posebnu datoteku koju korisnik može pažljivo da pregleda. Ovakav način rada je naročito pogodan kod zadavanja obimnih definicija baze podataka.

Preostaje još da razmotrimo programski rad sa bazom podataka, koji je manipulativnog karaktera. Tu postoje dve mogućnosti. Ako je prilikom formiranja programa u program preko odgovarajućih procedura i funkcija uključen opis manipulacija na niskom nivou pogodnom za interpretiranje, jasno je da nema potrebe za SUBP interfejsom i DML prevodiocem. Redosled događaja je sledeći:

- program prosleđuje SUBP interpreteru odgovarajuću sekvencu naredbi niskog nivoa (4a);
- SUBP interpreter izvršava prosleđenu sekvencu naredbi niskog nivoa; prvo se ispituje njena saglasnost sa rečnikom podataka (Pb) i registrom korisnika (Rb); u slučaju saglasnosti, čitaju se (Ob) ili menjaju osnovni podaci (Oa), zavisno od prirode zadate manipulacije;
- SUBP interpreter prosleđuje programu poruku o ishodu prethodne aktivnosti i eventualne rezultate (pa); na programu je kako će to da koristi.

Sa druge strane, ako je u program uključen opis manipulacija na visokom nivou, SUBP interfejs i DML prevodioc su neophodni i redosled događaja je sledeći:

- program prosleđuje SUBP interfejsu odgovarajuću sekvencu DML naredbi (4b); interfejs prosleđuje tu naredbu DML prevodiocu (3a);
- u slučaju greške koja se može utvrditi samo na osnovu prevodenja, DML prevodilac dostavlja interfejsu poruku o tome (3b); interfejs prosleđuje tu poruku programu (pb) i time se završava obrada unete naredbe;
- u slučaju da nema greške navedenog tipa, DML prevodilac formira odgovarajuću sekvencu naredbi niskog nivoa i prosleđuje je SUBP interpreteru (3c);
- SUBP interpreter izvršava prosleđenu sekvencu naredbi niskog nivoa; prvo se ispituje njena saglasnost sa rečnikom podataka (Pb) i registrom korisnika (Rb); u slučaju saglasnosti, čitaju se (Ob) ili menjaju osnovni podaci (Oa), zavisno od prirode zadate manipulacije;
- SUBP interpreter prosleđuje programu poruku o ishodu prethodne aktivnosti i eventualne rezultate (pa); na programu je kako će to da koristi.

Ovaj poslednji način programskog rada je sporiji s obzirom da uključuje rad SUBP interfejsa i DML prevodioca, ali je zato fleksibilniji. Pošto se zadaju u izvornom obliku, DML naredbe nisu “fiksirane” u programu nego se mogu generisati tokom rada programa zavisno od dijaloga sa korisnikom i drugih okolnosti.

Uz navedene, postoje i dodatni delovi SUBP zaduženi za funkcionalnosti baze podataka koje ne razmatramo u ovoj knjizi. To su:

- optimizator operacija (query optimizer): deo koji na osnovu strukture operacije, strukture baze podataka i statistike korišćenja baze podataka vrši optimizaciju (ubrzanje) postupka izvršenja operacija;
- upravljач memorije (buffer manager): deo koji je zadužen za efikasno i brzo korišćenje memorijskog prostora organizovanog u tri nivoa: I primarna memorija (RAM), II sekundarna memorija (HD), III arhivska memorija;
- upravljач transakcija (transaction manager): deo koji je zadužen za upravljanje višekorisničkim konkurentni radom (radom više korisnika istovremeno);
- upravljач oporavka (recovery manager): deo koji je zadužen za oporavak baze podataka u slučaju kvara.

Uz sve navedeno što je ugrađeno u SUBP, danas postoje i brojni pomoćni programi za olakšanje poslova administratoru baze podataka, kao što su upravljanje sistemom korisnika, praćenje i podešavanje performansi, obezbeđenje sigurnosti, itd.

Šta na osnovu svega prethodnog možemo da zaključimo o sistemu upravljanja bazom podataka (SUBP)? Kao prvo SUBP je jedan izuzetno složen sistem univerzalnog karaktera, u smislu da je primenjiv za bilo koju konkretnu bazu podataka. Može se slobodno zaključiti da SUBP predstavlja vrhunski domet informatičkog inženjerstva kod koga je do savršenstva ugrađen jedan od najznačajnijih principa u savremenoj informatici. To je princip parametrizacije, odnosno izrade softvera koji generalno rešava čitavu jednu klasu problema, pri čemu se za konkretne probleme iz te klase naknadno zadaju parametri koje taj softver očitava iz odgovarajućih zapisa i interpretira ih tokom rada.

# 3

## ***MODELI I STRUKTURE PODATAKA***

---

U ovom poglavlju razmotrićemo dva pojma koji su od velikog značaja za deo informatike koji se bavi informacionim sistemima i bazama podataka. Prvi od njih, model podataka, je u samim osnovama te oblasti, dok je drugi, strukture podataka, značajan pošto se odnosi na organizaciju i način zapisivanja podataka.

### **3.1 Pojam modela podataka**

Reč "model" se uvek javlja zajedno sa nečim, u smislu "model nečega" To "nešto" nazivamo originalom. Model podataka je samo specijalni slučaj modela, pa je uputno da prvo razmotrimo opšti pojam originala i modela i postupke koji se primenjuju pri sastavljanju modela, odnosno modeliranju.

### 3.1.1 Original, model i neki opšti pojmovi

Pod modelom se podrazumeva predstava nekog originala. Original je najčešće deo našeg okruženja i kao takav realan, ali može biti i abstraktan.

Svaki original, a posebno ako je deo našeg okruženja, možemo smatrati za sistem koji čine izabrani objekti i odnosi između njih. Sistem opisujemo preko odgovarajućih skupova objekata i skupova odnosa između njih, ali pri tome uvek mora postojati mogućnost raspoznavanja individualnih objekata i odnosa u tim skupovima.

Na osnovu ovih zapažanja, u mogućnosti smo da u vezi predstavljanja nekog sistema definišemo nekoliko opštih pojmova:

- **entitet**: opšti pojam za nešto što postoji i što se može jednoznačno odrediti (prepoznati) u skupu srodnih entiteta; ovako definisan, entitet obuhvata kao specijalne slučajeve pojmove objekta i veze kao posebnog vida odnosa;
- **svojstvo**: imenovana osobina entiteta, podrazumeva vrstu (ime) svojstva i vrednost svojstva (na primer, "boja" i "plavo"); entitete opisujemo pomoću jednog ili (najčešće) više izabranih svojstava;
- **identifikator**: svojstvo ili skup svojstava koji svojom vrednošću jednoznačno određuju svaki pojedinačni objekat ili odnos u skupu (ne mogu postojati dva elementa skupa sa istim vrednostima identifikatora);
- **klasa**: imenovani skup entiteta koji se odlikuju istim vrstama svojstava (na primer, klasa "knjiga" u koju ulaze sve pojedinačne knjige koje se odlikuju svojstvima "šifra" (tzv. inventarni broj) i "naslov"; svojstva koja odlikuju neku klasu nazivamo klasifikacionim svojstvima);
- **instanca**: element u klasi, entitet, pojedinačni objekat ili odnos.

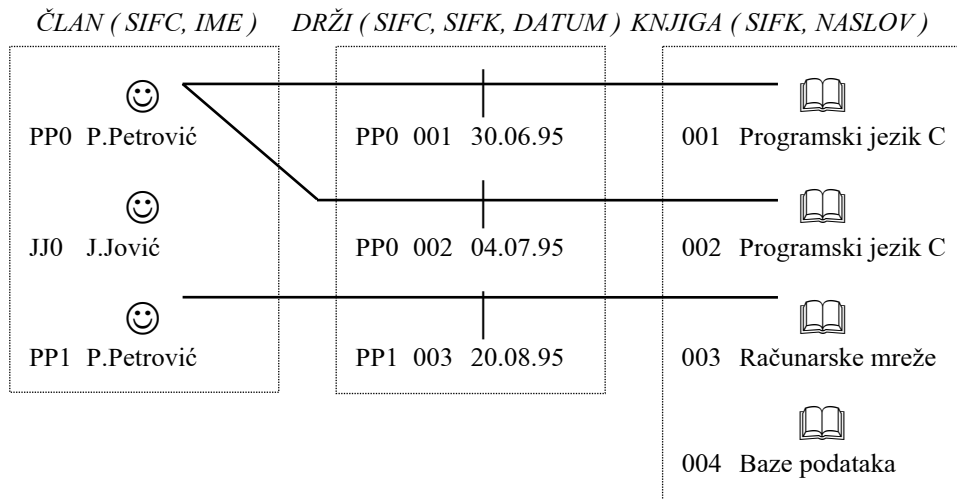
Pomoću prethodno navedenih pojmova, možemo da preciziramo šta sve karakteriše neki sistem, odnosno original:

- skup svih klasa objekata;
- skup svih klasa odnosa između objekata;
- klasifikaciona svojstva za svaku klasu objekata (najmanje jedno svojstvo, radi raspoznavanja pojedinačnih objekata);
- klasifikaciona svojstva za svaku klasu odnosa (najmanje onoliko identifikatora objekata koliko učestvuje u odnosu, radi raspoznavanja pojedinačnih odnosa);
- instance za svaku klasu objekata (pojedinačni objekti, koji se odlikuju određenim vrednostima klasifikacionih svojstava);
- instance za svaku klasu odnosa (pojedinačni odnosi, koji se odlikuju određenim vrednostima klasifikacionih svojstava).

Pri tome, posmatrani sistem je najčešće dinamički, odnosno promenljiv sa vremenom. Promene mogu biti sledeće:

- nastajanje ili nestajanje instance objekta u nekoj klasi objekata;
- nastajanje ili nestajanje instance odnosa u nekoj klasi odnosa;
- promena vrednosti jednog ili više svojstava instance objekta u nekoj klasi objekata;
- promena vrednosti jednog ili više svojstava instance odnosa u nekoj klasi odnosa.

Kao ilustracija prethodnog poslužiće nam krajnje pojednostavljeni prikaz stanja sistema BIBLIOTEKA sa jednim odnosom tipa veze, dat na slici 3-1.



Slika 3-1: Jedno stanje sistema BIBLIOTEKA.

Na ovoj ilustraciji su navedeni nazivi klasa i u zagradama klasifikaciona svojstva, a uz svaku instancu naznačene su vrednosti klasifikacionih svojstava. Napomenimo još i sledeće:

- u klasi *ČLAN* smo zbog mogućnosti da članovi imaju isto ime, što je i slučaj, uveli pored svojstva *IME* i dodatno identifikaciono svojstvo "šifra člana" *SIFC*;
- sličnu situaciju imamo i u klasi *KNJIGA* gde smo zbog ponavljanja vrednosti svojstva *NASLOV* uveli identifikator "šifra knjige" (inventarni broj) *SIFK*;
- u klasi veze *DRŽI* minimum svojstava bi bili identifikatori učesnika u vezi *SIFC* i *SIFK*, ali smo pored toga uveli i svojstvo *DATUM* koje govori o tome od kada član drži knjigu kod sebe.



Navedimo nekoliko karakterističnih primera promena u odnosu na stanje sistema BIBLIOTEKA prikazano na slici 3-1:

- nabavka nove knjige naslova "PASCAL programiranje": u klasi *KNJIGA* nastaje nova instanca sa parom vrednosti klasifikacionih svojstava 005, PASCAL programiranje;
- član šifre PP1 vraća knjigu šifre 003 koju je držao kod sebe: iz klase *DRŽI* nestaje instanca sa skupom vrednosti klasifikacionih svojstava PP1, 003, 20.08.95;
- promena prezimena člana šifre JJ0: u klasi *ČLAN* instanca sa parom vrednosti klasifikacionih obeležja JJ0, J.Jović menja drugu vrednost u novu.

Već smo naglasili da je svaki model predstava nekog originala. Pri tome, postoji jedna vrlo bitna osobina modela: sredstvo predstavljanja. Navedimo nekoliko primera za to:

- plastična maketa aviona: ovo je grub model koji predstavlja samo oblik i spoljni izgled originala, a način predstavljanja je pomoću tela odgovarajućeg oblika ali umanjenog u odnosu na original;
- planovi za izradu aviona: ovo je detaljan model originala, pošto sadrže predstavu većine njegovih svojstava, a način predstavljanja je pomoću teksta i slika;
- neka teorija u fizici: ovo je manje ili više detaljan model jednog odabranog segmenta prirode, a način predstavljanja je preko teksta i matematičkih formula;
- kartotečki informacioni sistem biblioteke: ovo je detaljan model stvarne biblioteke, a način predstavljanja je preko podataka zapisanih na karticama;
- računarski informacioni sistem biblioteke: ovo je model stvarne biblioteke sa dovoljnim nivoom detalja za efikasan rad biblioteke, a način predstavljanja je preko podataka zapisanih na magnetnim disk jedinicama.

Iz navedenih primera uočavamo da podaci mogu biti sredstvo predstavljanja, odnosno da je informacioni sistem model nekog sistema.

### 3.1.2 Informacioni sistem kao model

U opštem smislu, podatak je predstava o "nečemu" izražena nekim jezikom (dogovorenim načinom opštenja). Taj jezik ne mora nužno biti govorni jezik. Primera radi, podatak o nekoj visini predstavimo korišćenjem cifara decimalnog brojnog sistema ako ga zapišemo na papiru, ali će isti taj podatak u računaru biti predstavljen u binarnoj formi, na najjednostavnijem mogućem jeziku koji raspolaže sa samo dva simbola - 0 i 1. Na sličan način, preko grupa simbola 0 ili 1 možemo predstaviti pojedina slova i reči govornog jezika.

Informacioni sistem kao skup podataka o nekom sistemu predstavlja model tog sistema. Ilustrujmo to prikazom informacionog sistema naše biblioteke koja je kao sistem prikazana na slici 3-1.

<i>SIFC</i>	<i>IME</i>	<i>SIFC</i>	<i>SIFK</i>	<i>DATUM</i>	<i>SIFK</i>	<i>NASLOV</i>
PP0	P.Petrović	PP0	001	30.06.95	001	Programski jezik C
JJ0	J.Jović	PP0	002	04.07.95	002	Programski jezik C
PP1	P.Petrović	PP1	003	20.08.97	003	Računarske mreže
					004	Baze podataka

datoteka *ČLAN*                      datoteka *DRŽI*                      datoteka *KNJIGA*

Slika 3-2: Podaci informacionog sistema BIBLIOTEKA.

Iz ove ilustracije možemo jasno sagledati kako informacioni sistem predstavlja model nekog realnog sistema:

- svakoj klasi objekta ili veze odgovara jedna datoteka slogovnog tipa;
- svakom klasifikacionom svojstvu neke klase objekta ili veze odgovara jedan podatak u sastavu sloga odgovarajuće datoteke;
- svakoj instanci u nekoj klasi objekata ili veza odgovara jedan slog u odgovarajućoj datoteci;
- svakoj vrednosti klasifikacionog svojstva neke instance u nekoj klasi objekata ili veza odgovara jedna vrednost odgovarajućeg podatka u odgovarajućem slogu odgovarajuće datoteke.

Sadržaj svih datoteka informacionog sistema sa slike 3-2 odgovara stanju sistema BIBLIOTEKA prikazanom na slici 3-1. Kako se sa vremenom menja stanje sistema BIBLIOTEKA, tako i informacioni sistem BIBLIOTEKA mora da prati te promene da bi bio valjan model originala. Da bi to pojasnili, navedimo kako bi se ranije navedene promene za sistem BIBLIOTEKA odrazile na naš informacioni sistem:

- nabavka nove knjige naslova "PASCAL programiranje": u datoteci *KNJIGA* se dodaje novi slog sa vrednostima podataka 005, PASCAL programiranje;
- član šifre PP1 vraća knjigu šifre 003 koju je držao kod sebe: iz datoteke *DRŽI* se uklanja slog sa vrednostima podataka PP1, 003, 20.08.95;
- promena prezimena člana šifre JJ0: u datoteci *ČLAN* u slogu sa parom vrednosti podataka JJ0, J.Jović menja se vrednost drugog podatka.

Pri tome, sve izmene u podacima neophodne da bi se pratilo stanje sistema koji se modelira obezbeđuje druga komponenta koju smo naveli u definiciji informacionog sistema, a to je skup postupaka za održavanje podataka.

### 3.1.3 Baza podataka kao model

Baza podataka je specijalni, i to najsavršeniji vid komponente "Podaci" informacionog sistema, pa samim tim dosta toga što smo rekli o informacionom sistemu kao modelu nekog sistema može da se primeni i na bazu podataka. Međutim, baza podataka je u svojstvu modela tu u značajnoj prednosti. Konkretno:

- zbog integrisanosti podataka baza podataka vernije predstavlja originalni sistem; ono što je u stvarnosti jedna celina predstavljeno je kao celina i u bazi podataka;
- zbog organizacije podataka prema potrebama korisnika baza podataka je u mogućnosti na integrisane podatke kao model "preslika" na onoliko parcijalnih modela koliko to treba pojedinim korisnicima; jednom istom sistemu može odgovarati više modela sa manjim ili većim nivoom detalja.

### 3.1.4 Postupci modeliranja

U poglavlju 1, kada smo naveli primer biblioteke kao sistema, u obzir smo uzeli samo neke od postojećih skupove objekata i veza između njih. Konkretno, opredelili smo se za tretman sistema BIBLIOTEKA preko sledećih skupova klasa:

$$\text{Objekti} = \{ \{ \text{naslov} \} , \{ \text{knjiga} \} , \{ \text{član} \} , \{ \text{autor} \} \}$$

$$\text{Veze} = \{ \{ \text{drži} \} , \{ \text{sadrži} \} , \{ \text{je\_autor} \} \}$$

Međutim, kada smo u ovom poglavlju pojam originala sveli na pojam sistema, opredelili smo se za jednostavniji tretman biblioteke, preko samo dve klase objekata i jedne klase veze:

$$\text{Objekti} = \{ \{ \text{knjiga} \} , \{ \text{član} \} \}$$

$$\text{Veze} = \{ \{ \text{drži} \} \}$$

Uz to, naglasimo da smo za svaku pojedinu klasu vršili izbor klasifikacionih svojstava. Primera radi, umesto da za klasu član uzmemo detaljnu predstavu

$$\check{C}LAN ( SIFC, IME, POL, STAROST )$$

opredelili smo se za manji broj svojstava, odnosno za predstavu

$$\check{C}LAN ( SIFC, IME )$$

Postupak koji smo upravo naveli, zanemarivanje nebitnih i uzimanje u obzir samo bitnih klasa i svojstava, naziva se postupkom abstrakcije i on čini suštinu postupka razvoja svakog modela. Međutim, pre toga smo morali da uočimo klase, odnosno to da pojedinačna pojavljivanja entiteta imaju nešto zajedničko i da ih treba posmatrati kao elemente određenih klasa. Taj postupak se naziva klasifikacija. Obrnuto, ako imamo već definisane klase a treba da uočimo da su neki pojedinačni entiteti elementi tih klasa, u pitanju je postupak instancizacije.

Na osnovu prethodnog u mogućnosti smo da formulišemo osnovne postupke formiranja modela i njihov redosled:

- klasifikacija objekata: uočava se koji pojedinačni objekti čine koje klase objekata;
- klasifikacija odnosa: uočava se koji pojedinačni odnosi čine koje klase odnosa;
- abstrakcija objekata: zanemaruju se nebitne klase objekata i za dalji postupak zadržavaju se samo one od značaja;
- abstrakcija odnosa: od preostalih klasa odnosa (otpali su svi oni koji uključuju klase objekata koje smo zanemarili u prethodnom koraku) zanemaruju se nebitni i za dalji postupak zadržavaju samo oni od značaja;
- abstrakcija klasifikacionih svojstava objekata: za svaku klasu objekata zanemaruju se nebitna i uzimaju samo značajna klasifikaciona svojstva;
- abstrakcija klasifikacionih svojstava odnosa: za svaku klasu odnosa zanemaruju se nebitna i uzimaju (ako ih ima) samo značajna dodatna klasifikaciona obeležja (identifikatori za klase objekata koji su u odnosu automatski postaju klasifikaciona obeležja odnosa).

Ovi postupci su jasno zastupljeni kada se vrši modeliranje pomoću podataka, dok su u drugim slučajevima manje izraženi.

### 3.1.5 Model podataka

Informacioni sistem, odnosno podaci u sklopu njega, predstavlja model nekog sistema. Pri tome, korespondencija između takvog modela i originala je dvojaka:

- podaci svojim ustrojstvom odgovaraju ustrojstvu sistema; konkretno, datoteke i podaci unutar njihovih slogova moraju odgovarati klasama i svojstvima sistema, odnosno onom što se često naziva strukturom sistema;
- podaci svojim sadržajem odgovaraju stanju sistema; konkretno, slogovi u datotekama moraju odgovarati instancama u klasama, a vrednosti podataka u slogovima vrednostima svojstava tih instanci.

Da bi ostvarili navedenu korespondenciju, prvo moramo da kreiramo prazne datoteke sa odgovarajućim strukturama slogova, a zatim da u te datoteke unesemo podatke koji odgovaraju nekom početnom stanju našeg sistema. Od tog trenutka, zahvaljujući postupcima održavanja, takav informacioni sistem će svojim sadržajem pratiti stanje sistema koga predstavlja.

U vezi upravo opisanog početka svakog informacionog sistema, kao prirodno se nameće sledeće pitanje: *kako* znamo koje datoteke i kakve strukture treba da kreiramo? Očigledno je da moramo imati neku predstavu o ustrojstvu podataka u okviru budućeg informacionog sistema. Ta predstava o podacima, formulisana nekim pogodnim sredstvom izražavanja, jeste ono što nazivamo modelom podataka.

Model podataka je predstava ustrojstva podataka za neki informacioni sistem, odnosno bazu podataka. Ta predstava mora da sadrži odgovore na sledeća pitanja:

- koje datoteke ulaze u sastav informacionog sistema;
- koji podaci ulaze u sastav slogova svake od tih datoteka;
- kojih su tipova podataka svi ti podaci.

Navedeni odgovori nisu dovoljni sami za sebe, pošto ništa ne govore o eventualnim ograničenjima nad podacima i o načinu njihovog održavanja i korišćenja. Da bi bio sveobuhvatan, svaki model podataka mora da čine tri komponente:

- strukturalna komponenta: deo modela podataka koji smo već naveli, služi za predstavljanje ustrojstva podataka;
- integritetska komponenta: deo modela podataka koji služi za predstavljanje ograničenja nad podacima koja proizilaze iz ograničenja koja važe u originalnom sistemu (na primeru biblioteke, ograničenja su: knjiga ne može biti kod nepostojećeg člana, knjiga može biti ili kod ni jednog ili kod jednog člana, itd.);
- manipulativna komponenta: deo modela podataka koji služi za predstavljanje elementarnih postupaka održavanja i korišćenja podataka preko kojih se mogu opisati sve konkretne situacije u sistemu-okruženju.

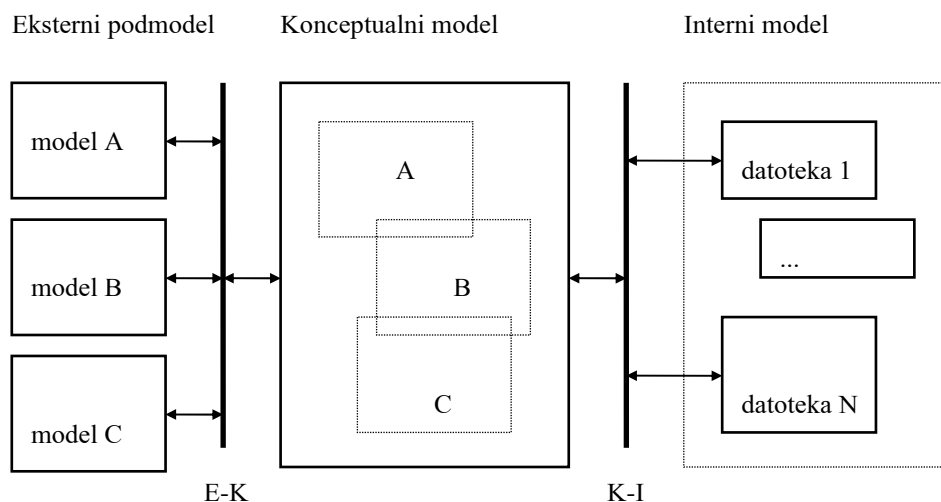


Od navedenih komponenti kao posebno značajnu treba izdvojiti strukturnu. To je sasvim razumljivo ako se ima na umu da ustrojstvo podataka neposredno utiče na ograničenja i manipulacije nad podacima. Uz to, na osnovu našeg uvodnog primera škole u poglavlju 1 možemo naslutiti složenu prirodu ustrojstva podataka, koje treba da obuhvati ustrojstvo sa gledišta krajnjih korisnika, sveukupno ustrojstvo i ustrojstvo na odgovarajućim medijumima (uglavnom magnetnim diskovima).

Pokazalo se da je razdvajanje navedenih delova strukturne komponente modela podataka osnovni način za ostvarivanje maksimalne nezavisnosti programa i podataka, pa je vrlo brzo (1975) formulisan i standard o tome. Taj standard, poznat pod oznakom ANSI/X3/SPARC, obuhvata tri strukturna podmodela, i to:

- eksterni podmodel: čini ga skup korisničkih modela, odnosno predstava ustrojstva podataka sa gledišta pojedinih korisnika; ovom odgovara pojam podšeme koji smo objasnili u poglavlju 2;
- konceptualni podmodel: čini ga predstava sveukupnog ustrojstva podataka, i kao takav sadrži sve eksterne podmodele integrisane u celinu; tome odgovara ranije uvedeni pojam šeme
- interni podmodel: čini ga detaljna predstava načina zapisa sveukupnih podataka na odgovarajućim medijumima (uglavnom magnetnim diskovima).

Kao ilustracija za prethodno, neka nam posluži slika 3-3.



Slika 3-3: Strukturni podmodeli podataka.

Ova ilustracija na posredan način uključuje i sistem upravljanja bazom podataka (SUBP). Naime:

- granicu E-K, odnosno funkciju preslikavanja u oba smera na relaciji eksterni podmodel-konceptualni model, obezbeđuje SUBP; u cilju maksimalne nezavisnosti programa i podataka, ta granica treba da je što čvršća i kao takva jedini posrednik između navedena dva podmodela;
- granicu K-I, odnosno preslikavanje u oba smera na relaciji konceptualni podmodel-interni podmodel, takođe obezbeđuje SUBP; ta granica takođe treba da je što čvršća u cilju maksimalne nezavisnosti programa i podataka.

### 3.1.6 Vrste modela podataka

Podela na vrste modela podataka je odraz podele na vrste baza podataka, pri čemu je suštinska razlika između pojedinih vrsta modela u strukturnoj komponenti. Tako, imamo sledeće vrste modela podataka (izuzimajući objektni):

- hijerahijski model podataka: jedini način predstavljanja odnosa između podataka je preko odnosa "jedan prema više", odnosno 1:N; sredstva predstavljanja su skupovi slogova, slogovi, podaci u slogovima i ukazatelji na slogove;
- mrežni model podataka: jedini način predstavljanja odnosa između podataka je preko odnosa "više prema više", odnosno M:N, što kao specijalni slučaj uključuje i odnos 1:N; sredstva predstavljanja su ista kao i kod hijerahijskog modela - skupovi slogova, slogovi, podaci u slogovima i ukazatelji na slogove;
- relacioni model podataka: jedini način predstavljanja odnosa između podataka su relacije, odnosno datoteke sa slogovima, i njihov sadržaj; sredstva predstavljanja su u suštini samo skupovi slogova, slogovi i podaci, odnosno tzv. šeme relacija.

Prva dva modela se odnose na formatizovane (pokazivačke) baze podataka koje smo pomenuli na kraju poglavlja 1.

Razlike u strukturnoj komponenti uslovile su i značajne razlike u ostalim dvema komponentama navedenih vrsta modela podataka. To je posebno izraženo kod manipulativne komponente:

- hijerahijski a posebno mrežni model podataka podrazumevaju složene operacije održavanja i korišćenja podataka, kao što su: "nađi početak liste", "nađi kraj liste", "nađi prethodnika", "nađi sledbenika", "nađi roditelja u stablu", "nađi prvog potomka u stablu", "ubaci u listu", "izbaci iz liste", itd;
- relacioni model podataka zbog svoje jednostavnosti predstavljanja podataka iziskuje samo tri elementarne operacije održavanja i pet elementarnih operacija korišćenja podataka (o tome će biti više reči u narednom poglavlju); sve moguće manipulacije nad relacionom bazom podataka mogu se izraziti preko navedenog skupa elementarnih operacija.

Sve tri navedene vrste modela spadaju u grupu tzv. semantički nižih modela podataka, pošto uključuju i internu strukturnu komponentu koja je neposredno povezana sa realizacijom baze podataka u smislu fizičkog zapisa podataka na odgovarajućim medijumima. Modeli podataka koji su semantički višeg nivoa su pre mešavina modela sistema i modela podataka nego "čisti" modeli podataka, i kao takvi služe za prvu fazu modeliranja prilikom projektovanja neke baze podataka. Pri tome je za takav model poželjna osobina jednoznačnog prevođenja u neku nižu formu, najčešće relacioni model. Jedan takav model višeg nivoa, tzv. "model objekata i odnosa", izložen je u poglavlju 8.

## **3.2 Osnovni pojmovi iz struktura podataka**

Pod strukturama podataka se podrazumeva deo informatike koji se bavi organizacijom podataka, načinom njihovog zapisivanja, kao i postupcima pretraživanja podataka. U ovom odeljku osvrnućemo se samo na neke od pojmova iz te oblasti.

### 3.2.1 Podatak, slog i datoteka

Pojmove kao što su podatak, slog i datoteka smo već koristili kada smo govorili o modelu podataka. I pored toga što su nam ti pojmovi uglavnom poznati i jasni sami po sebi, nije na odmet da ih i formalno definišemo:

- podatak: “šifrirano” svojstvo, predstava nekog svojstva izražena nekim jezikom (u informatici je to binarni jezik, tako da se podatak javlja u vidu niza bitova, od kojih svaki ima vrednost 0 ili 1);
- slog: celina sastavljena od jednog ili više podataka u određenom redosledu (celina u smislu manipulacije, odnosno čitanja i pisanja);
- datoteka: u najopštijem smislu, bilo koji podaci zapisani na nekom medijumu i koji kao celina nose zajedničko ime.

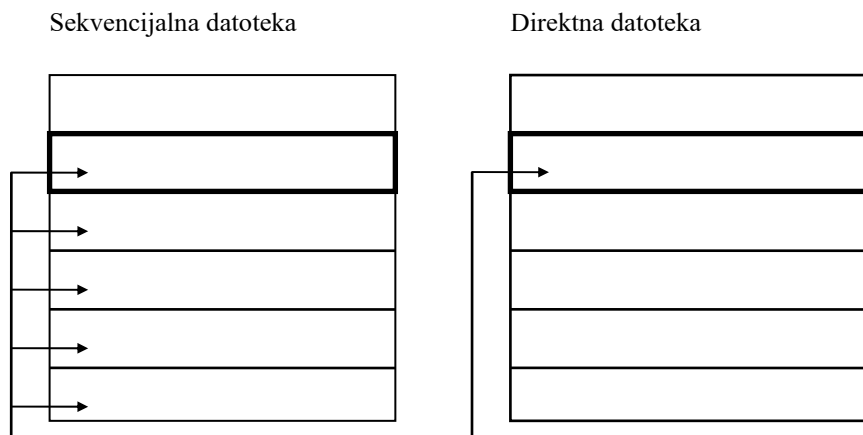
Datoteke možemo klasifikovati na više načina. Prva klasifikacija je prema organizaciji podataka u datoteci i po tom osnovu postoje dve vrste datoteka:

- neslogovne datoteke: datoteke u kojima podaci predstavljaju celinu koja se ne može rastavljati na delove (slogove); primeri za takve datoteke su zapisi crteža, slika i slično;
- slogovne datoteke: datoteke koje se sastoje iz slogova, pri čemu slogovi mogu biti iste ili različite strukture (u smislu podataka koji ulaze u njihov sastav).

Drugi osnov za klasifikaciju predstavlja način pristupa podacima. Po tome razlikujemo sledeće dve vrste datoteka:

- sekvencijalne datoteke: kod ovih datoteka je radi pristupa nekim podacima potrebno proći kroz sve podatke koji se u datoteci nalaze ispred toga;
- direktne datoteke: kod ovih datoteka se neposredno može pristupiti bilo kojim podacima bez prolaženja kroz deo podataka ispred toga.

Razjasnimo ovu podelu na primeru slogovnih datoteka prikazanih na slici 3-4, od kojih je jedna sekvencijalna a druga direktna.



Slika 3-4: Sekvencijalni i direktni pristup.

Za sekvencijalnu datoteku važe sledeća ograničenja:

- čitanje sloga: da bi se pročitao 5. slog (prikazana situacija), moraju se redom pročitati 4 sloga ispred toga;
- pisanje sloga: da bi se zapisao 5. slog, moraju se redom zapisati 4 sloga pre toga.

Sa druge strane, kod direktne datoteke nemamo navedena ograničenja. Konkretno:

- čitanje sloga: 5. slog se čita direktno; operativni sistem na osnovu poznate dužine sloga izračunava poziciju u datoteci koja odgovara tom slogu;
- pisanje sloga: slog se zapisuje direktno na poziciji koju operativni sistem izračunava na osnovu poznate dužine sloga; ako se ta pozicija nalazi iza dotadašnjeg kraja datoteke, operativni sistem kreira nedostajući prazan deo datoteke i zapisani slog postaje novi kraj datoteke.

Navedeni postupak direktnog pristupa povlači za sobom sledeće:

- poželjno je da datoteka bude sa istom dužinom slogova, pošto se vrlo teško ostvaruje direktni pristup sa slogovima promenljive dužine;
- mora biti raspoznatljiva situacija kada se iz praznog dela datoteke pokušava čitanje sloga koji pre toga nije zapisan;
- kriterijum pristupa određenom slogu mora na neki način da se pretvori u redni broj sloga u datoteci.

Poslednja klasifikacija datoteka koju pominjemo je po nameni. U tom pogledu postoje dve vrste datoteka:

- osnovne datoteke: datoteke koje sadrže osnovne podatke, odnosno slogove podataka od interesa za korisnike;
- pristupne datoteke: datoteke koje sadrže podatke koji obezbeđuju direktni pristup osnovnim podacima.

Uobičajeno je da se pristupne datoteke nazivaju indeksnim datotekama ili indeksima.

### 3.2.2 Direktan pristup podacima

Direktan pristup željenom slogu ostvaruje se zadavanjem rednog broja tog sloga u datoteci. Sa gledišta krajnjeg korisnika, ovo je zadovoljavajuće samo ako su identifikatori neki rastući celi brojevi. Takav slučaj imamo u našem primeru biblioteke, gde smo knjige redom označili šiframa (inventarnim brojevima) 001, 002 i tako redom.

U praksi postoje situacije kada je neposredno preslikavanje celobrojnih šifara na redne brojeve slogova nepodesno. Primer za to bi bila evidencija stanovnika Beograda na osnovu tzv. jedinstvenog matičnog broja. Poznato je da je matični broj sastavljen iz ukupno 13 cifara, 12 osnovnih i jedne kontrolne. Usvajanje takvih šifara za redne brojeve slogova bi značilo da čuvanje podataka od oko 2.000.000 stanovnika Grada iziskuje datoteku sa prostorom za oko 320.000.000.000 slogova. Iskorišćenje takve datoteke bilo bi oko 0.0006%.

Još češće su situacije kada šifre uopšte nisu celobrojne, nego kombinacije slova i brojeva, kada uopšte nije moguće neposredno preslikavanje šifara u redne brojeve slogova. Takav slučaj imamo u našem primeru biblioteke: šifre članova čine dva slova i redni broj.



Jedno od mogućih rešenja za navedene dve situacije jeste organizacija osnovne datoteke po metodi poznatoj pod nazivom "transformacija ključa u adresu":

- prvih  $N$  slogova datoteke čini tzv. primarno područje;
- ostatak datoteke, od sloga  $N+1$  pa dalje, čini tzv sekundarno područje;
- definiše se funkcija transformacije takva da svaku moguću vrednost šifre (ključa) po kojoj se pristupa preslikava na ceo broj u intervalu  $[1, N]$ ;

Sa takvom organizacijom, postupak je sledeći

- bez obzira na to da li je u pitanju upis ili čitanje, pomoću funkcije transformacije se na osnovu zadate šifre odredi ceo broj  $K$  u navedenom intervalu;
- postupak dodavanja odnosno upisa novog sloga je sledeći: ako je  $K$ -ti slog primarnog područja slobodan, tu se upisuje novi slog; ako se u  $K$ -tom slogu primarnog područja nalazi neki ranije upisani slog koji predstavlja početak liste slogova čija se šifra preslikava na isti ceo broj  $K$ , novi slog se dodaje u sekundarnom području kao poslednji element liste;
- postupak traženja odnosno čitanja sloga sa zadatom šifrom je sledeći: ako je  $K$ -ti slog primarnog područja slobodan, ne postoji slog sa zadatom šifrom; ako u  $K$ -tom slogu primarnog područja postoji slog čija šifra odgovara zadatoj, u pitanju je traženi slog; ako u primarnom području postoji  $K$ -ti slog ali sa neogovarajućom šifrom, pretražuje se lista slogova u sekundarnom području sve do nailaska na slog sa odgovarajućom šifrom (traženi slog postoji) ili do nailaska na kraj liste (traženi slog ne postoji).

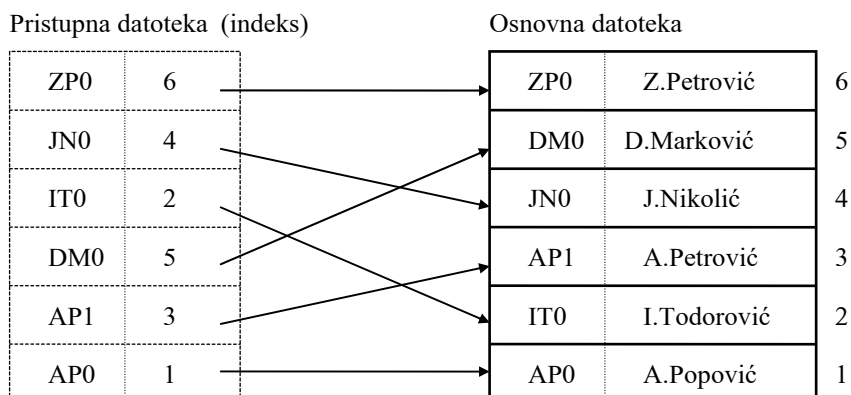
Organizacija pristupa transformacijom ključa u adresu razrešava problem pristupa podacima po bilo kakvoj šifri, ali se odlikuje i određenim manama:

- pristup se vremenom usporava kako se datoteka puni slogovima čije se šifre preslikavaju na iste cele brojeve;
- primarno područje nikad nije potpuno iskorišćeno, a naročito na početku rada, kada u datoteci ima relativno malo slogova;
- navedenom organizacijom nije moguće obezbediti ono što vrlo često treba pri korišćenju podataka, a to je uredenost u rastući ili opadajući redosled po nekom izabranom kriterijumu.

Sve navedene nedostatke otklanja organizacija direktnog pristupa preko posebne pristupne datoteke, odnosno indeksa:

- uz osnovnu direktnu datoteku koristi se pristupna direktna datoteka; svakom slogu u osnovnoj datoteci odgovara jedan slog u pristupnoj;
- svaki slog pristupne datoteke sadrži dva podatka: šifru osnovnog sloga i njegov redni broj u osnovnoj datoteci;
- radi što bržeg pretraživanja, slogovi pristupne datoteke su ulančani u stablo uređeno po rastućoj ili opadajućoj vrednosti šifre.

Kao ilustracija navedene organizacije neka nam posluži slika 3-5 koja prikazuje način zapisivanja podataka o šiframa i imenima autora naslova u biblioteci.



Slika 3-5: Direktni pristup pomoću indeksa.

Na slici je prikazan redosled slogova indeksa kakav se dobija kada se redom prolazi kroz sve slogove, a ne stvarni zapis tih slogova i njihova ulančanost u stablo. Slogovi u osnovnoj datoteci su zapisani redom, po hronologiji dodavanja. Prilikom svake izmene (dodavanje, brisanje) u osnovnoj datoteci uporedo se vrši i odgovarajuća izmena u indeksu.

Indeksi predstavljaju najbolju organizaciju direktnog pristupa, pošto obezbeđuju praktično sve što je potrebno pri manipulaciji podacima:

- brz direktni pristup traženim podacima;
- uređenost podataka po traženom kriterijumu.

Uz to, indeksi pružaju i neke dodatne pogodnosti:

- uopšte nismo ograničeni na samo jedan indeks po jednoj osnovnoj datoteci; možemo formirati onoliko indeksa koliko nam treba kriterijuma pristupa ili uređenosti podataka; tokom rada sa tako ustrojenim podacima, pre svake manipulacije moramo naglasiti koji od indeksa je tzv. aktivni indeks;
- pri formiranju indeksa nismo ograničeni na proste kriterijume koji se svode na jedan podatak u slogu osnovne datoteke; možemo koristiti i složene indeksne izraze sastavljene od više podataka, što je naročito značajno za ostvarivanje željenih redosleda osnovnih podataka.

### 3.2.3 Strukture podataka u okviru baze podataka

Na kraju, pokušajmo da na osnovu svega do sada izloženog utvrdimo koje strukture podataka su pogodne za primenu u bazi podataka. Možemo da zaključimo:

- datoteke moraju biti slogovnog tipa, s obzirom da celu klasu nekog sistema predstavljamo datotekom, a instancu slogom;
- slogovi svake datoteke moraju imati istu strukturu, s obzirom da za svaku klasu nekog sistema uvodimo klasifikaciona svojstva a instance te klase opisujemo skupom vrednosti tih svojstava;
- datoteke moraju biti direktne, sa mogućnošću što bržeg direktnog pristupa podacima i sa mogućnošću ostvarivanja željene uređenosti podataka; ovo podrazumeva i korišćenje transformacije ključa u adresu i indeksa.

*PREPORUKA ČITAOCIMA*

*Od narednog poglavlja pa na dalje koriste se sintaksna i algoritamska notacija koje su izložene u prilogu B na kraju ove knjige. Čitaocima se preporučuje da pre prelaska na naredna poglavlja prouče navedeni prilog.*

# 4

## ***RELACIONI MODEL PODATAKA***

---

Kao što je već naglašeno, relacioni model podataka predstavlja teoretske osnove za baze podataka relacionog tipa. U ovom poglavlju se u potpunosti razmatraju strukturna i integritetska komponente tog modela. Manipulativna komponenta modela se obrađuje samo u delu održavanja, a u delu korišćenja se samo pominje, s obzirom da je toj temi posvećeno poglavlje 5.

Na kraju poglavlja izloženi su koncepti univerzalnog relacionog modela.

## **4.1 Strukturna komponenta relacionog modela**

Relacioni model je jedini model podataka koji u matematičkom smislu predstavlja jedan formalan sistem. U tom smislu, on se bazira na određenom broju pojmova formalnog karaktera.

### 4.1.1 Atribut i domen

Atribut u relacionom modelu odgovara onom što smo nazvali "podatak" kada smo razmatrali modele podataka.

#### *Definicija*

*Atribut je imenovana vrsta svojstva koja se ne može rastavljati na delove bez gubitka svakog značenja.*

Navedena definicija podrazumeva elementarnost, odnosno prost atribut. Atribut sastavljen od više atributa nazivamo složenim atributom. Takav atribut možemo rastaviti na jednostavnije attribute, koji mogu biti isto složeni ili prosti. Pri tom rastavljanju ne dolazi do potpunog gubitka značenja nego samo do promene značenja. Relacioni model kako ga sada razmatramo podrazumeva da su šeme relacija svedene na formu sa prostim atributima.

#### *Primeri*

Datum je složen atribut, pošto se može rastaviti na tri prosta atributa: dan u mesecu, mesec i godinu. Tim rastavljanjem se gubi prvobitno značenje datuma kao celine, ali dobijeni prosti atributi imaju svaki svoje značenje.

Adresa je složen atribut, koji se može rastaviti na mesnu adresu i mesto, koji su takođe složeni atributi. Mesnu adresu možemo rastaviti na naziv ulice i broj, koji su prosti. Mesto čine dva prosta atributa, poštanski broj i naziv mesta.

Svako svojstvo podrazumeva pored vrste i mogućnost opisa pomoću vrednosti tog svojstva. Na primer, za vrstu svojstva "visina u cm" možemo imati vrednosti 180, 187 itd. Ili, za vrstu svojstva (odnosno atribut kako to sada nazivamo) "naziv mesta" moguće vrednosti bi bile "Beograd", "Novi Sad" i slično. Ovo nas dovodi do pojma domena, drugog osnovnog pojma relacionog modela podataka.

#### *Definicija*

*Skup svih mogućih vrednosti nekog atributa  $A_i$  naziva se domenom tog atributa i označava sa  $D_i$  ili  $dom(A_i)$ .*

Pojam domena je veoma blizak pojmu tipa podatka iz programiranja. Svakom atributu odgovara samo jedan domen, ali zato jednom domenu može odgovarati više atributa, odnosno dva ili više različitih atributa mogu imati zajednički domen.



*Primer*

Atributi "visina u cm" i "trajanje filma u min" imaju isti domen, a to je skup celih pozitivnih brojeva.

Ako posmatramo skup entiteta koji kao pripadnici iste klase imaju ista klasifikaciona svojstva, onda u jednom trenutku jednom takvom svojstvu, u stvari atributu, može odgovarati neki skup vrednosti koji je podskup domena tog atributa. Taj podskup vrednosti naziva se aktivni domen. Za razliku od domena koji obuhvata sve moguće vrednosti i konstantan je, aktivni domen se može menjati sa vremenom.

### 4.1.2 Šema relacije

U prethodnom poglavlju smo uočili da se klasa nekih entiteta (objekata ili veza) predstavlja skupom klasifikacionih svojstava. Odgovarajuće sredstvo predstavljanja u relacionom modelu podataka je ono što nazivamo šemom relacije.

#### *Definicija*

*Šema relacije  $R$  je konačan skup atributa  $\{A_i\}$  i konačan skup  $O$  ograničenja nad vrednostima tih atributa.*

U ovom poglavlju objasnićemo samo strukturni deo ove definicije ( $R = \{A_i\}$ ). Ograničenja  $O$  nad vrednostima atributa podrazumevaju da atributi ne mogu uzimati bilo koje vrednosti, nego samo određene u kombinaciji. Tu problematiku obuhvata oblast poznata pod nazivom "Zavisnosti i normalne forme", koja je izložena u okviru poglavlja 7.

Definicija šeme relacije podrazumeva da su samim tim što su zadati atributi zadati i njihovi domen. Tri bitne osobine šeme relacije, od kojih su prve dve posledica skupovne prirode prethodne definicije, su:

- atributi moraju biti unikatni, odnosno različitih naziva;
- redosled atributa nije bitan, u smislu značenja;
- šema relacije mora da sadrži bar jedan atribut.

Nebitnost redosleda je samo teoretska. Kada navodimo neku šemu relacije mi attribute moramo navesti nekim redosledom, zapisano kao

$$R ( A_1, A_2, \dots, A_N )$$

gde  $N$  predstavlja broj atributa.

U praksi, za naziv šeme relacije možemo izabrati bilo koji pogodan naziv. Isto važi i za nazive atributa, pod uslovom da su nazivi unikatni. Tako, prirodno je da šemu relacije kojom predstavljamo klasu članova biblioteke definišemo pregledno kao

$$\text{CLAN} ( \text{SIFC}, \text{IME} )$$

Iz svega do sada izloženog možemo da zaključimo sledeće:

- svojstva klase objekata ili veza nekog sistema predstavljamo šemom relacije;
- šema relacije može da se tumači i kao definicija strukture neke datoteke.

### 4.1.3 Relacija

Videli smo da je šema relacije pojam relacionog modela podataka koji služi za predstavljanje neke klase. Za klasu koja je tako definisana, instance koje ona sadrži predstavljaju se pojmom relacije, pri čemu pojam "torka" označava jednu kombinaciju vrednosti atributa:

#### *Definicija*

*Relacija  $r$  nad šemom relacije  $R$  je konačan skup torki vrednosti atributa.*

Pojasnimo ovu definiciju koja je opšteg karaktera:

- šema relacije se zadaje u formi  $R(A_1, A_2, \dots, A_N)$  sa nekim usvojenim redosledom atributa; samim tim, redosled vrednosti u svakoj torki relacije mora biti saglasan tom redosledu atributa, odnosno forme  $a_1, a_2, \dots, a_N$ ;
- za razliku od šeme relacije, koja mora sadržati bar jedan atribut, relacija može biti prazan skup, bez i jedne torke vrednosti;
- šema relacije je kao skup atributa konstantna, dok se relacija može menjati sa vremenom, kako u pogledu broja torki tako i u pogledu sadržaja torki.

Pored prethodne opisne definicije, postoji i formalna definicija relacije koja uključuje pojmove domena i Dekartovog proizvoda:

#### *Definicija*

*Neka je data šema relacije  $R(A_1, A_2, \dots, A_N)$  i neka su  $D_i = \text{dom}(A_i)$  odgovarajući domeni atributa. Relacija  $r$  nad šemom relacije  $R$  je podskup Dekartovog proizvoda domena atributa, odnosno*

$$r \subseteq D_1 \times D_2 \times \dots \times D_N$$

Kao i šema relacije, i relacija ima dve bitne osobine koje proizilaze iz skupovne prirode njene definicije:

- torke moraju biti unikatne; u relaciji nikada ne mogu da postoje dve iste torke, odnosno torke sa istim vrednostima svih orgovarajućih atributa;
- redosled torki nije bitan, u smislu ukupnog sadržaja relacije.

I ovde u izvesnom smislu važi napomena o teoretskoj prirodi nebitnosti redosleda, što proizilazi iz sledećih okolnosti:

- relaciji u praksi odgovara neka datoteka;
- svakoj torki odgovara jedan slog te datoteke,
- slogovi u datoteci zapisani su određenim redosledom (najčešće po redosledu dodavanja).

*Primer*

Za naš raniji primer sistema "Biblioteka" imali bi šemu relacije CLAN i relaciju nad njom **clan**:

CLAN (SIFC, IME )	<b>clan</b> ( SIFC    IME            )
	-----
	JJ0      J.Janković
	PP0      P.Petrović
	JJ1      J.Jovanović
	MM0      M.Marković
	-----

Ovde smo radi preglednosti uz naziv relacije naveli i nazive atributa.

#### 4.1.4 NULL - univerzalna vrednost "nepoznato"

U praksi postoje situacije kada u relacije unosimo torke za koje su vrednosti nekih od atributa nepoznate u tom trenutku. To se javlja u dva slučaja:

- ta vrednost postoji, ali nije poznata u trenutku unosa torke; na primer, u relaciji **naslov** nad šemom relacije NASLOV ( SIFN, NAZIV, SIFO ) unosimo podatke za naslov poznate šifre i naziva, ali ne znamo kojoj oblasti pripada;
- ta vrednost je nedefinisana, nema smisla; na primer, ako relacijom **knjiga** nad šemom relacije KNJIGA ( SIFK, SIFN, SIFC, DATUM ) želimo da pored svih knjiga evidentiramo i podatke o tome koji članovi drže koje knjige i od kada, podaci SIFC i DATUM su nedefinisani za knjige koje su trenutno u biblioteci.

Navedene situacije razrešene su u okviru relacionog modela uvođenjem vrednosti sa značenjem "nepoznato". Ova vrednosti je univerzalnog tipa, primenjiva za attribute bilo kakvih domena. Za tu vrednost usvojena je oznaka NULL.

Prvi od navedena dva slučaja je posledica trenutnog nepoznavanja vrednosti nekog atributa i podrazumeva mogućnost naknadnog unosa te vrednosti i kao takav je prihvatljiv. Drugi slučaj se javlja kao posledica odabrane strukture relacije, odnosno šeme relacije.

### 4.1.5 Šema relacione baze podataka

Pojmovi atributa, domena, šeme relacije i relacije čine skup osnovnih pojmova relacionog modela podataka. Prvi izvedeni pojam relacionog modela koji ćemo definisati jeste pojam šeme relacione baze podataka.

#### *Definicija*

*Šema relacione baze podataka BP je konačan skup šema relacija  $\{R_i\}$  i konačan skup U ograničenja koja važe između njih.*

Strukturni deo ove definicije čini samo skup šema relacija, što prema definiciji šeme relacije uključuje za svaku relaciju  $R_i$  i skup ograničenja  $O_i$  koja važe nad atributima *unutar* svake od tih šema. Skup ograničenja U uključuje samo ograničenja koja važe *između* pojedinih šema relacija. Takva ograničenja ulaze istovremeno u sastav integritetske komponente relacionog modela, i biće izložena u odeljku 4.2.2.

Šema relacije predstavlja definiciju relacije. Po analogiji, šema relacione baze podataka predstavlja definiciju relacione baze podataka.

*Primer*

Za naš raniji primer sistema BIBLIOTEKA imamo sledeću strukturnu komponentu šeme relacione baze podataka gde svakoj klasi odgovara jedna šema relacije:

Objekti:

CLAN ( SIFC, IME )

KNJIGA ( SIFK )

NASLOV ( SIFN, NAZIV )

AUTOR ( SIFA, IME )

Veze:

DRZI ( SIFK, SIFC, DATUM )

SADRZI ( SIFK, SIFN )

JE\_AUTOR ( SIFA, SIFN, KOJI )

Ova šema relacione baze podataka obuhvata samo predstavu trenutnog stanja sistema BIBLIOTEKA. Prilog A sadrži šemu relacione baze podataka BIBLIOTEKA sa više detalja od prethodno navedene, pošto pored trenutnog stanja obuhvata i prošlost (POZAJMICA) i budućnost (REZERVACIJA) sistema. Uz to, ne postoje sve šeme relacija veza iz prethodnog primera, pošto su neke veze predstavljene proširenim šemama relacija objekata. O tome će biti više reči u poglavlju 8.



### 4.1.6 Relaciona baza podataka

Relaciona baza podataka je drugi izvedeni pojam koji ćemo definisati u okviru relacionog modela podataka.

*Definicija*

*Relaciona baza podataka **bp** je konačan skup relacija  $\{r_i\}$  nad šemom relacione baze podataka  $BP$  odnosno  $\{R_i\}$ .*

## Primer

Nad šemom relacione baze podataka BIBLIOTEKA iz prethodnog odeljka imamo sledeću relacionu bazu podataka ***biblioteka*** koja svojim sadržajem predstavlja stanje sistema BIBLIOTEKA u nekom trenutku:

o b j e k t i

clan ( SIFC IME )			autor ( SIFA IME )		knjiga ( SIFK )		
JJ0	J.Janković		AP0	A.Popović		001	
PP0	P.Petrović		IT0	I.Todorović		002	
JJ1	J.Jovanović		AP1	A.Petrović		003	
MM0	M.Marković		JN0	J.Nikolić		004	
			DM0	D.Marković		005	
			ZP0	Z.Petrović		006	
naslov ( SIFN NAZIV )						007	
						008	
RBP0	Relacione baze podataka					009	
RK00	Računarske komunikacije					----	
PP00	PASCAL programiranje						
PJC0	Programski jezik C						
						v e z e	
drzi ( SIFK SIFC DATUM )			sadrzi( SIFK SIFN )		je_autor ( SIFA SIFN KOJI )		
001	JJ0	10.10.95	001	RBP0	AP0	RBP0	1
002	PP0	15.10.95	002	RBP0	JN0	RBP0	2
004	JJ0	18.10.95	003	RK00	DM0	RK00	1
			004	PJC0	ZP0	PP00	1
			005	PJC0	DM0	PP00	2
			006	PJC0	AP1	PJC0	1
			007	PP00	IT0	PJC0	2
			008	PP00	ZP0	PJC0	3
			009	PP00			-----

Prikazani sadržaj je saglasan delu sadržajA baze podataka BIBLIOTEKA iz priloga A, koja obuhvata još i podatke o oblastima kojima pripadaju naslovi. Možemo uočiti i zašto relacija **sadrzi** može da se izostavi ako se u relaciju **knjiga** uključe i vrednosti atributa SIFN: svakoj knjizi odgovara jedan naslov, a u relacijama **sadrzi** i **knjiga** uvek su prisutne iste vrednosti atributa SIFK.

## 4.2 Integritetska komponenta relacionog modela

Integritetska komponenta relacionog modela podataka služi za predstavljanje ograničenja koja važe nad podacima, odnosno nad vrednostima pojedinih atributa. Ta ograničenja se po prirodi mogu podeliti na tri grupe:

- ograničenja koja proizilaze iz zahteva unikatnosti torki u relacijama; takva ograničenja nazivamo identifikacionim integritetom (ili egzistencijalnim integritetom, kako se još naziva);
- ograničenja koja se eksplicitno zadaju preko skupova ograničenja  $O_i$  u okviru šema relacija  $R_i$ ; takva ograničenja su posledica ograničenja koja važe nad svojstvima u realnom sistemu;
- ograničenja koja uključuju attribute koji se mogu nalaziti u različitim relacijama i koja se zadaju preko skupa ograničenja  $U$  u okviru šeme relacione baze podataka; takva ograničenja nazivamo referencijalnim integritetom.

Druga grupa ograničenja se proučava u okviru oblasti poznate pod nazivom "Zavisnosti i normalne forme" koja je obrađena u poglavlju 7, pa ćemo u okviru integritetske komponente relacionog modela izložiti samo egzistencijalni i referencijalni integritet. S tim u vezi, neophodno je da definišemo pojmove ključeva i stranog ključa.

### 4.2.1 Super-ključ

Iz definicije relacije proizilazi da se u relaciji nikada ne mogu javiti dve identične torke. Ovo možemo formulirati i na sledeći način: svakoj vrednosti skupa svih atributa  $A_1, A_2, \dots, A_N$  šeme relacije  $R = \{A_i\}$  odgovara jedna torka u relaciji  $r$ , odnosno skup vrednosti  $a_1, a_2, \dots, a_N$ .

Ako pažljivo pogledamo šemu relacione baze podataka i samu relacionu bazu podataka za sistem BIBLIOTEKA, možemo uočiti da i pojedini podskupovi atributa u pojedinim šemama relacija imaju tu osobinu da svakoj njihovoj vrednosti (kombinaciji vrednosti njihovih atributa) odgovara jedna torka u relaciji. To su, da nabrojimo samo neke: SIFC u šemi CLAN, SIFK, SIFC u šemi DRZI (zato što jednu knjigu jedan član drži ili ne drži kod sebe), SIFK u toj istoj šemi (zato što knjigu može da drži kod sebe samo jedan član), itd. Za takve situacije kažemo da dati podskup atributa jednoznačno određuje torke u relaciji.

#### *Definicija*

*Super-ključ neke šeme relacije  $R$  je svaki podskup atributa  $X$  te šeme koji ima osobinu da jednoznačno određuje torke u relaciji.*

Činjenica da svakoj vrednosti podskupa atributa  $X$  šeme relacije  $R$  odgovara samo jedna vrednost skupa atributa  $R$  podseća na pojam funkcije iz matematike (kada imamo  $y = f(x)$  a jednoj vrednosti  $x$  odgovara samo jedna vrednost  $y$ ). Zato je uobičajeno da se kaže da  $R$  funkcijski zavisi od  $X$  ili da  $X$  funkcijski uslovljava  $R$ , a ta okolnost se zapisuje u formi

$$X \rightarrow R$$

Sa takvim označavanjem, a imajući na umu da  $R$  predstavlja skup svih atributa šeme relacije i da šema relacije može da ima više super-ključeva, skup svih super-ključeva  $S$  možemo formalno definisati kao:

$$S = \{ X_i \mid X_i \subseteq R \wedge X_i \rightarrow R \}$$

Iz definicije super-ključa proizilazi osobina da se jedna vrednost super-ključa može pojaviti u relaciji samo jednom.

Iz osobine unikatnosti torki u relaciji proizilazi da svaka šema relacije ima bar jedan super-ključ. U krajnjem slučaju, kada postoji samo jedan super-ključ, u njegov sastav ulaze svi atributi šeme relacije, odnosno  $S = R$ .

## 4.2.2 Kandidat-ključ

Posmatrajmo šemu relacije JE\_AUTOR i odgovarajuću relaciju **je\_autor**. Iz osobine unikatnosti torki u relaciji je jasno da cela šema, odnosno kombinacija atributa SIFA,SIFN,KOJI predstavlja super-ključ. Po prirodi stvari, možemo zaključiti da je i kombinacija SIFA,SIFN super-ključ, pošto se činjenica da je neki autor napisao neki naslov evidentira samo jednom torkom u relaciji. Međutim, atributi SIFA i SIFN pojedinačno, a pogotovo KOJI, nisu super-ključevi: jedan autor može napisati više naslova, naslov može imati više autora, a mnogi autori će biti prvi, drugi i tako redom za naslove koje su napisali. Kao prirodno se postavlja pitanje koji od super-ključeva su minimalni, takvi da njihovi sastavni delovi više nisu super-ključevi. To je od posebnog značaja kada se ima na umu da svaki super-ključ može da posluži kao osnov za realizaciju direktnog pristupa torki (u stvari slogu u nekoj datoteci), pri čemu je poželjna osobina minimalnosti s obzirom da će vrednosti pristupnih super-ključeva biti zapisane u pristupnoj datoteci - indeksu.

### *Definicija*

*Kandidat-ključ neke šeme relacije R je svaki podskup atributa X te šeme koji ima osobinu da jednoznačno određuje torke u relaciji, a da ni jedan njegov pravi podskup nema tu osobinu.*

Drugim rečima, kandidat-ključ neke šeme relacije R je svaki njen super-ključ koji ima osobinu minimalnosti. Takvih super-ključeva može biti više za datu šemu relacije. Oni mogu biti bez ili sa zajedničkim atributima, a jedino ne smeju da se obuhvataju, pošto tada obuhvatajući super-ključ ne bi bio minimalan. Svaka šema relacije ima bar jedan kandidat-ključ (u najgorem slučaju, to je R). Formalno, skup svih kandidat-ključeva K može se definisati kao:

$$K = \{ X_i \mid X_i \subseteq R \wedge X_i \rightarrow R \wedge \neg \exists Y ( Y \subset X_i \wedge Y \rightarrow R ) \}$$

U do sada posmatranom primeru baze podataka BIBLIOTEKA nemamo ni jednu situaciju sa više kandidat-ključeva, ali zato imamo jednu takvu situaciju u okviru kompletnog primera u prilogu A. U pitanju je šema relacije

OBLAST ( SIFO, NAZIV )

U ovoj šemi je uveden atribut šifre oblasti SIFO koji je ujedno i kandidat-ključ, ali postoji i drugi kandidat-ključ NAZIV. Naime, za razliku od šeme relacije

NASLOV ( SIFN, NAZIV, SIFO )

u kojoj je neophodno SIFN pošto može postojati više naslova istog naziva, ponavljanje naziva oblasti nije moguće, pošto se oblasti razlikuju po nazivima.

### 4.2.3 Primarni ključ

U situaciji kada neka šema relacije ima više kandidat-ključeva, za potrebe identifikacije torki relacije i realizaciju direktnog pristupa moramo da izaberemo jedan od njih kao primarni:

#### *Definicija*

*Primarni ključ neke šeme relacije  $R$  je jedan izabrani kandidat-ključ, ili jedini kandidat-ključ ako nema izbora.*

Svaka šema relacije ima jedan i samo jedan primarni ključ. Formalna definicija primarnog ključa je krajnje jednostavna:

$$P \in K$$

U situaciji kada neka šema relacije ima više kandidat-ključeva, izbor primarnog ključa vrši se po sledećim kriterijumima:

- prednost u odnosu na kandidat-ključeve koji su posledica nekog dogovorenog ograničenja uvek imaju kandidat ključevi "po prirodi stvari", oni koji su posledica nekog suštinskog ograničenja (dogovorno ograničenje može da se promeni, a suštinsko nikada);
- za kandidat-ključeve koji su jednaki po prvom kriterijumu, prednost imaju oni koji zauzimaju manje memorijskog prostora u datoteci koja odgovara relaciji.

*Primeri*

U šemi relacije OBLAST baze podataka BIBLIOTEKA iz priloga A, i SIFO i NAZIV su kandidat-ključevi "po prirodi stvari", ali je za primarni ključ izabran SIFO kao kraći.

Posmatrajmo šemu relacije DRZI u okviru iste baze podataka i zamislimo situaciju da važi ograničenje da članovi mogu da drže samo po jednu knjigu kod sebe. Tada bi imali dva kandidat-ključa, atribut SIFK i SIFC, ali bi prednost dali atributu SIFK, pošto "po prirodi stvari" knjiga van biblioteke može biti samo kod jednog člana, a dogovorno ograničenje pozajmice na jednu knjigu može biti ukinuto nekim novim pravilima rada biblioteke.

Uobičajeno je da se u šemi relacije primarni ključ naznači podvlačenjem punom linijom, a ako se žele naglasiti i drugi kandidat-ključevi oni se mogu podvući isprekidano. Po takvom označavanju, šema relacije baze podataka BIBLIOTEKA iz priloga A bi glasila:

CLAN ( <u>SIFC</u> , IME )	POZAJMICA ( <u>SIFP</u> , SIFC, SIFK, SIFN, DANA )
KNJIGA ( <u>SIFK</u> , SIFN )	REZERVACIJA ( <u>SIFN</u> , <u>SIFC</u> , DATUM )
NASLOV ( <u>SIFN</u> , NAZIV, SIFO )	JE_AUTOR ( <u>SIFA</u> , <u>SIFN</u> , KOJI )
OBLAST ( <u>SIFO</u> , <u>NAZIV</u> )	DRZI ( <u>SIFK</u> , SIFC, DATUM )
AUTOR ( <u>SIFA</u> , IME )	JE_REZERVISANA ( <u>SIFK</u> , SIFC, DATUM )

#### 4.2.4 Strani ključ

Svi ključevi o kojima je do sada bilo reči definisani su unutar jedne šeme relacije. Za objašnjenje pojma stranog ključa neophodne su nam dve šeme relacija, mada se sve može dešavati i unutar jedne jedine šeme relacije.

Posmatrajmo sledeće dve šeme relacija u okviru baze podataka BIBLIOTEKA:

NASLOV ( SIFN, NAZIV, SIFO )      OBLAST ( SIFO, NAZIV )

Ovde atribut SIFO u šemi NASLOV govori o tome kojoj oblasti naslov pripada. Primera radi, za naslov naziva "Programski jezik C" šifra oblasti je "PJ", pri čemu u relaciji **oblast** postoji torka sa tom vrednošću atributa SIFO. Šta više, takva torka *mora* da postoji. Situacija da atribut SIFO u relaciji **naslov** ima vrednost nepostojeće oblasti, odnosno vrednost koju primarni ključ SIFO šeme OBLAST nema u relaciji **oblast**, nije prihvatljiva.

Za isti par šema relacija moguća je i sledeća situacija. Nabavljena je knjiga sa novim naslovom za koju i sama ta knjiga i novi naslov treba da se uvedu u evidenciju, ali bibliotekar ne može na osnovu naziva da je razvrsta po oblasti. U takvoj situaciji, prirodno je da se novi naslov ipak evidentira u relaciji **naslov**, ali sa NULL vrednošću za trenutno nepoznatu vrednost atributa SIFO. Pri tome se ta vrednost naknadno može promeniti u pravu koja postoji u relaciji **oblast** kao vrednost primarnog ključa SIFO.

Zamislamo da smo umesto šema relacija KNJIGA i DRZI uveli jednu šemu

KNJIGA ( SIFK, SIFN, SIFC, DATUM )

kojom želimo da evidentiramo i podatke o tome koje knjige su trenutno kod kojih članova. U takvoj situaciji je sasvim prirodno da važi ograničenje da u relaciji **knjiga** atribut SIFC može imati samo jednu od sledećih vrednosti:

- ili vrednosti primarnog ključa SIFC u relaciji **clan** (knjiga je pozajmljena);
- ili vrednost NULL (knjiga je u biblioteci).

Uz to, primetimo da u izmenjenoj šemi KNJIGA postoji još jedan atribut pod sličnim ograničenjem vrednosti. To je SIFN, koje mora imati neku od vrednosti primarnog ključa SIFN u relaciji **naslov**.



U svim navedenim primerima ograničenja su važila za po jedan atribut. U opštem slučaju, ograničenja mogu da važe i nad podskupovima atributa šeme relacije.

### *Definicija*

*Strani ključ u šemi relacije  $R$  je svaki njen podskup atributa za koji važi ograničenje vrednosti u relaciji  $r$  na sledeće dve vrednosti*

- *vrednost kandidat ključa u nekoj relaciji;*
- *vrednost NULL.*

Relacija u kojoj se nalazi kandidat-ključ naziva se ciljnom relacijom. U većini realnih situacija strani ključ je prost (jedan atribut) i on referiše primarni ključ ciljne relacije, ali u opštem slučaju on može biti složen (više atributa) i može referisati umesto primarnog ključa neki kandidat-ključ.

Ilustrujmo ovo pomoću šeme relacione baze podataka BIBLIOTEKA iz priloga A, gde su strani ključevi naglašeni masnim slovima:

CLAN ( <u>SIFC</u> , IME )	POZAJMICA ( <u>SIFP</u> , <b>SIFC</b> , <b>SIFK</b> , <b>SIFN</b> , DANA )
KNJIGA ( <b>SIFK</b> , <b>SIFN</b> )	REZERVACIJA ( <b>SIFN</b> , <u>SIFC</u> , DATUM )
NASLOV ( <u>SIFN</u> , NAZIV, <b>SIFO</b> )	JE_AUTOR ( <u>SIFA</u> , <b>SIFN</b> , KOJI )
OBLAST ( <u>SIFO</u> , <u>NAZIV</u> )	DRZI ( <b>SIFK</b> , <b>SIFC</b> , DATUM )
AUTOR ( <u>SIFA</u> , IME )	JE_REZERVISANA ( <b>SIFK</b> , <b>SIFC</b> , DATUM )

Svi strani ključevi u ovom primeru su prosti (po jedan atribut), a možemo da uočimo i sledeće situacije:

- jedna šema relacije može da sadrži više stranih ključeva (šeme POZAJMICA, REZERVACIJA, DRZI, JE\_AUTOR i JE\_REZERVISANA);
- strani ključ može biti u sastavu primarnog ključa (šeme REZERVACIJA i JE\_AUTOR);
- strani ključ može biti istovremeno i primarni ključ u celini (šeme DRZI i JE\_REZERVISANA).

Svi primeri koje smo naveli do sada uključivali su po dve šeme relacija u vezi "strani ključ-primarni ključ", a isto tako je važila jednakost naziva odgovarajućih atributa. Jednakost naziva atributa je poželjna sa gledišta preglednosti šeme relacione baze podataka, ali to nije uvek moguće postići. U suštini, ono što povezuje strani ključ i primarni ključ jeste jednakost *značenja*, a jednakost naziva može ali i ne mora biti zadovoljena. Uz to, sve navedeno se može dešavati unutar jedne te iste šeme relacije.

*Primer*

Posmatrajmo šemu relacije RADNIK kojom se pored zaposlenosti evidentira i odnos nadređenosti u nekoj firmi, odnosno ko je sve kome šef:

RADNIK ( SIFR, IME, ADRESA, SIFNAD )

U ovoj šemi atribut SIFNAD predstavlja šifru neposredno nadređenog radnika (šefa). Pravila dobre organizacije rada nalažu da svaki radnik ima najviše jednog neposredno nadređenog, pri čemu je čitava organizacija u formi stabla u čijem se korenu nalazi jedini radnik koji u firmi nema neposredno nadređenog - direktor.

U ovoj šemi atribut SIFNAD je strani ključ, pošto mora imati ili vrednost postojeće šifre radnika SIFR koja je primarni ključ, ili vrednost NULL. Možemo da uočimo sledeće:

- sve se dešava unutar jedne jedine šeme relacije koja sadrži oba učesnika u vezi "strani ključ-primarni ključ";
- strani i primarni ključ (SIFNAD i SIFR) nemaju niti mogu da imaju isti naziv, ali zato imaju isto značenje identifikatora radnika.

### 4.2.5 Identifikacioni integritet

Identifikacioni integritet proizilazi iz osobine unikatnosti torki u relaciji i svodi se na formulaciju odgovarajućeg uslova.

*Uslov identifikacionog integriteta*

*Ni jedan atribut šeme relacije  $R$  koji je u sastavu primarnog ključa nikada ne sme imati NULL vrednost u relaciji  $r$ .*

Ova definicija obuhvata i specijalni slučaj primarnog ključa u celini. Ograničenje je sasvim razumljivo, pošto bi u suprotnom, kada bi dozvolili da deo primarnog ključa primi NULL vrednost, mogla nastupiti situacija da dve ili više torki u relaciji postanu identične. U oštrijoj formi, uslov identifikacionog integriteta nije ograničen samo na primarni ključ nego obuhvata i sve kandidat-ključeve.

*Primer*

Posmatrajmo šemu relacije i odgovarajuću relaciju kojima se evidentiraju podaci o autorstvu, i uočimo naznačene dve torke u relaciji:

JE_AUTOR ( <u>SIFA,SIFN</u> , KOJI )	je_autor ( SIFA SIFN KOJI )
	-----
	... AP0 RBP0 1
	AP0 PP00 1
	...
	-----

Ako bi dozvolili da deo primarnog ključa, recimo SIFN, u relaciji **je\_autor** primi vrednost NULL, sve bi bilo u redu ako bi postojala samo jedna takva vrednost, ali bi za dve takve vrednosti imali prikazanu situaciju

je_autor ( SIFA SIFN KOJI )
-----
.. AP0 NULL 1
AP0 NULL 1
...
-----

odnosno dve identične torke, što je nedozvoljeno u relacionom modelu.

Uslov identifikacionog integriteta je statičke prirode, pošto se odnosi na svako moguće stanje relacione baze podataka. Uz to, taj uslov može da unese dodatno ograničenje vrednosti stranog ključa. Naime, kada je strani ključ u sastavu primarnog ključa ili ga čini u celini, NULL vrednost nije dozvoljena.

Na osnovu osobine unikatnosti torki u relaciji i uslova identifikacionog integriteta možemo zaključiti sledeće:

- vrednosti svakog superključa u relaciji su unikatne, odnosno u relaciji se *nikada* ne mogu pojaviti dve iste vrednosti superključa;
- svakoj vrednosti superključa u relaciji odgovara samo jedna vrednost podskupa svih atributa koji nisu u njegovom sastavu.

Ovo važi i za svaki kandidat-ključ i primarni ključ, s obzirom da su oni specijalni slučajevi superključa.

## 4.2.6 Referencijalni integritet

Suština referencijalnog integriteta je u ograničenju vrednosti stranog ključa. Uslov referencijalnog integriteta koji se odnosi na svako moguće stabilno stanje relacije baze podataka je statičke prirode i u suštini odgovara definiciji stranog ključa.

### *Statički uslov referencijalnog integriteta*

*Svaki podskup atributa šeme relacije  $R$  koji predstavlja strani ključ može u relaciji  $r$  imati ili vrednosti primarnog ključa u ciljnoj relaciji ili vrednost NULL ako je ona dozvoljena.*

Sa gledišta izmena u relaciji koja sadrži strani ključ to podrazumeva da važe sledeća ograničenja:

- ne može se uneti torka sa vrednošću stranog ključa koja nije jednaka nekoj vrednosti primarnog ključa u ciljnoj relaciji ili dozvoljenoj NULL vrednosti;
- ne može se izmeniti torka tako da vrednost stranog ključa ne bude jednaka nekoj vrednosti primarnog ključa u ciljnoj relaciji ili dozvoljenoj NULL vrednosti.

U opštem slučaju, isto može da važi i za neki kandidat-ključ u ciljnoj relaciji.

U situaciji kada svi strani ključevi u šemi relacije baze podataka imaju iste nazive kao i odgovarajući primarni ključevi u ciljnim relacijama, posebna naznaka odnosno specifikacija referencijalnih integriteta ne mora biti neophodna. U opštem slučaju, takva specifikacija je poželjna a ponekad i neophodna iz sledećih razloga:

- nazivi stranih ključeva i odgovarajućih primarnih ključeva mogu biti različiti;
- mogu postojati atributi istih naziva kao i primarni ključevi a da uopšte nisu strani ključevi;
- poželjno je da se naglasi da li referencijalni integritet u pojedinim slučajevima uključuje i NULL vrednost.

Skup takvih specifikacija je ono što čini skup ograničenja  $U$  koji smo ranije naveli u sklopu definicije šeme relacije baze podataka.

Za specifikaciju referencijalnih integriteta usvojena je posebna notacija za skup vrednosti koje u relaciji  $r$  nad šemom relacije  $R$  uzima neki podskup atributa  $X$ . Navedena okolnost se zapisuje kao  $R[X]$  što se čita kao "projekcija relacije  $r$  po podskupu atributa  $X$ " i u suštini predstavlja skup torki vrednosti nad  $X$ .

Za relaciju **je\_autor** prikazanu sa leve strane desno su navedene vrednosti koje se dobijaju projekcijama po raznim podskupovima atributa:

```

je_autor( SIFA  SIFN  KOJI )  je_autor [ SIFA,SIFN ]  je_autor [ SIFN ]
-----
AP0  RBP0  1                AP0  RBP0                RBP0
JN0  RBP0  2                JN0  RBP0                RK00
DM0  RK00  1                DM0  RK00                PP00
ZP0  PP00  1                ZP0  PP00                PJC0
DM0  PP00  2                DM0  PP00
AP1  PJC0  1                AP1  PJC0
IT0  PJC0  2                IT0  PJC0
ZP0  PJC0  3                ZP0  PJC0
-----

```

Sa takvom notacijom smo u mogućnosti da sažeto specificiramo referencijalne integritete.

*Primer*

Za levo navedenu šemu relacione baze podataka BIBLIOTEKA iz priloga A imamo specifikaciju referencijalnih integriteta ispisanu sa desne strane:

CLAN ( <u>SIFC</u> , IME )	-
KNJIGA ( <u>SIFK</u> , SIFN )	KNJIGA [SIFN] $\subseteq$ NASLOV [SIFN]
NASLOV ( <u>SIFN</u> , NAZIV, SIFO )	NASLOV [SIFO] $\subseteq$ OBLAST [SIFO] $\cup$ {NULL}
OBLAST ( <u>SIFO</u> , NAZIV )	-
AUTOR ( <u>SIFA</u> , IME )	-
POZAJMICA ( <u>SIFP</u> , SIFC, SIFK, SIFN, DANA )	POZAJMICA [SIFC] $\subseteq$ CLAN [SIFC] $\cup$ {NULL} POZAJMICA [SIFK] $\subseteq$ KNJIGA [SIFK] $\cup$ {NULL} POZAJMICA [SIFN] $\subseteq$ NASLOV [SIFN]
$\cup$ {NULL}	
REZERVACIJA ( <u>SIFN</u> , <u>SIFC</u> , DATUM )	REZERVACIJA [SIFN] $\subseteq$ NASLOV [SIFN] REZERVACIJA [SIFC] $\subseteq$ CLAN [SIFC]
DRZI ( <u>SIFK</u> , SIFC, DATUM )	DRZI [SIFK] $\subseteq$ KNJIGA [SIFK] DRZI [SIFC] $\subseteq$ CLAN [SIFC]
JE_AUTOR ( <u>SIFA</u> , <u>SIFN</u> , KOJI )	JE_AUTOR [SIFA] $\subseteq$ AUTOR [SIFA] JE_AUTOR [SIFN] $\subseteq$ NASLOV [SIFN]
JE_REZERVISANA ( <u>SIFK</u> , SIFC, DATUM )	JE_REZERVISANA [SIFK] $\subseteq$ KNJIGA [SIFK] JE_REZERVISANA [SIFC] $\subseteq$ CLAN [SIFC]

Umesto NULL koristi se {NULL} pošto operacija unije zahteva skupove kao operande.

U pojedinim specifikacijama referencijalnog integriteta naglasili smo da je dozvoljena i NULL vrednost. Ako se to ne naglasi, podrazumeva se oštrije ograničenje koje ne dozvoljava NULL vrednost. Razlozi baš ovakvih dozvola NULL vrednosti biće razjašnjeni kasnije.

Za ranije navedenu šemu relacije RADNIK imali bi sledeću specifikaciju:

**RADNIK ( SIFR, IME, ADRESA, SIFNAD )**       $\text{RADNIK [SIFNAD]} \subseteq \text{RADNIK [SIFR]} \cup \{\text{NULL}\}$

Ono što je karakteristično za vezu "strani ključ-primarni ključ" jeste to da referencijalni integritet podrazumeva i ograničenja izmena u ciljnoj relaciji. Dodavanje nove torke u ciljnu relaciju nikada ne može da naruši referencijalni integritet, pošto tada nastaje nova vrednost primarnog ključa. Međutim, uklanjanje torke dovodi uvek a izmena torke ponekad do nestanka jedne vrednosti primarnog ključa iz ciljne relacije. Ako bi se ta operacija izvršavala bezuslovno, to bi narušilo referencijalni integritet u slučaju da bilo gde u bazi podataka postoje vrednosti stranih ključeva koje su jednake nestaloj vrednosti primarnog ključa. Iz tih razloga se nameće potreba za specifikacijom u smislu: kako treba postupiti u navedenim situacijama da bi referencijalni integritet bio očuvan. Takva specifikacija se naziva dinamičkom specifikacijom odnosno uslovom referencijalnog integriteta



### *Dinamička specifikacija - uslov referencijalnog integriteta*

*Neka je  $r$  ciljna relacija nad šemom  $R$  sa primarnim ključem  $X$  i neka je  $s$  referišuća relacija nad šemom  $S$  koja sadrži strani ključ  $Y$ . Tada se posebno za svaku od operacija nad ciljnom relacijom  $r$ :*

- *DELETE: uklanjanje torke, a time i jedne vrednosti primarnog ključa,*
- *UPDATE: izmena stare vrednosti primarnog ključa u novu u torci,*

*navodi jedna od sledeće četiri propratne akcije nad referišućom relacijom  $s$ :*

- *NO ACTION: operacija se ne izvršava ako u relaciji  $s$  postoji vrednost stranog ključa koja odgovara nestaloj vrednosti primarnog ključa iz ciljne relacije, a u suprotnom se izvršava;*
- *CASCADE: operacija se uvek izvršava; za slučaj uklanjanja torke iz ciljne relacije  $r$ , iz relacije  $s$  se uklanjaju sve torke u kojima strani ključ ima vrednost koja odgovara nestaloj vrednosti primarnog ključa; za slučaj izmene vrednosti primarnog ključa u  $n$ -torci ciljne relacije  $r$ , u svim torkama relacije  $s$  u kojima strani ključ ima vrednost koja odgovara nestaloj vrednosti primarnog ključa ta vrednost se menja u novu vrednost primarnog ključa;*
- *SET NULL: operacija se uvek izvršava; u relaciji  $s$  se u svim torkama u kojima strani ključ ima vrednost koja odgovara nestaloj vrednosti primarnog ključa ta vrednost menja u NULL vrednost.*
- *SET DEFAULT: operacija se uvek izvršava; u relaciji  $s$  se u svim torkama u kojima strani ključ ima vrednost koja odgovara nestaloj vrednosti ta vrednost menja u deklarisanu "podrazumevanu" vrednost navedenu prilikom kreiranja relacije.*

SET NULL akcija se ne sme navoditi za:

- strani ključ koji je ujedno i primarni ključ;
- strani ključ koji je deo složenog primarnog ključa;
- strani ključ za koji je drugim ograničenjem isključena NULL vrednost,

dok se SET DEFAULT akcija ne sme navoditi samo ako za bilo koji atribut u sastavu stranog ključa (uključujući i specijalni slučaj jednog atributa) važi:

- u sastavu je primarnog ključa;
- pri kreiranju relacije nije deklarisan podrazumevana vrednost.

Dinamička specifikacija referencijalnog integriteta se navodi za svaki strani ključ svake šeme relacije posebno, i to za svaku od dve operacije nad ciljnom tabelom koje mogu da naruše referencijalni integritet (UPDATE, DELETE). U slučaju da se ne navede ništa, podrazumeva se najoštrije ograničenje – NO ACTION.

U praksi se javlja i dodatna opcija RESTRICT koja je u osnovi istog značenja kao i NO ACTION ali sa razlikom u implementaciji, o čemu će biti reči u poglavlju 6.

U vezi sa implementacijom dinamičke specifikacije referencijalnog integriteta u sistemu upravljanja bazom podataka treba naglasiti sledeće:

- operacija nad ciljnom relacijom neke relacione baze podataka sprovodi se samo ako ni u jednoj od relacija sa stranim ključevima koje je referišu nije nastupila NO ACTION situacija odbijanja izmene; ukoliko je to trenutka odbijanja bilo nekih delimičnih izmena u bazi podataka one moraju biti poništene;
- savremeni sistem upravljanja bazom podataka treba sam da održava specificirani referencijalni integritet, bez ikakvog dodatnog programiranja od strane korisnika, kako za operacije nad ciljnom relacijom tako i za operacije nad referišućom relacijom (onom koja sadrži strani ključ).

Da bi bila kompletna, dinamička specifikacija referencijalnog integriteta za jedan strani ključ u jednoj relaciji mora da sadrži sledeće naznake:

- koja relacija je referišuća (u kojoj se nalazi strani ključ);
- koji podskup atributa referišuće relacije (najčešće je jedan atribut) čini strani ključ;
- koja relacija je ciljna (u kojoj se nalazi referisani kandidat ključ, najčešće primarni);
- koji podskup atributa ciljne relacije čini referisani kandidat-ključ;
- za koju operaciju nad ciljnom relacijom se zadaje specifikacija;
- koja je prpratna akcija nad referišućom tabelom,

što se može sve iskazati i u skraćenoj formi, po sintaksoj notaciji koja je opisana u odeljku B.1 priloga B:

```

DinamickaSpecifikacijaReferencijalnogIntegriteta ::=
  ReferisucaRelacija :
    ( Atribut ... ) ≥ CiljnaRelacija ( Atribut ... ) ,
                                UPDATE = Akcija ,
                                DELETE = Akcija ;
  { ( Atribut ... ) ≥ CiljnaRelacija ( Atribut ... ) ,
                                UPDATE = Akcija ,
                                DELETE = Akcija ; } ...

Akcija ::= NO ACTION | CASCADE | SET NULL | SET DEFAULT

```

*Primer*

Posmatrajmo našu šemu relacione baze podataka BIBLIOTEKA iz priloga A. Sledi kompletna specifikacija referencijalnog integriteta sa komentarima:

```
KNJIGA :
    (SIFN) > NASLOV (SIFN),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
```

Ne dozvoljava se brisanje naslova za kojeg postoje knjige.

```
NASLOV :
    (SIFO) > OBLAST (SIFO),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
```

Ne dozvoljava se brisanje oblasti za koju postoje naslovi.

```
POZAJMICA :
    (SIFC) > CLAN (SIFC),
                                UPDATE - CASCADE,
                                DELETE - SET NULL ;
    (SIFK) > KNJIGA (SIFK),
                                UPDATE - CASCADE,
                                DELETE - SET NULL ;
    (SIFN) > NASLOV (SIFN),
                                UPDATE - CASCADE,
                                DELETE - SET NULL ;
```

Brisanjem člana ne brišemo podatke o pozajmicama knjiga i naslova.

Brisanjem knjige ne brišemo podatke o pozajmicama članova i naslova.

Brisanjem naslova ne brišemo podatke o pozajmicama članova.

Čak i kada su sva tri podatka obrisana ostaje nekakva informacija o pozajmici.

```
REZERVACIJA :
    (SIFN) > NASLOV (SIFN),
                                UPDATE - CASCADE,
                                DELETE - CASCADE ;
    (SIFC) > CLAN (SIFC),
                                UPDATE - CASCADE,
                                DELETE - CASCADE ;
```

Brisanjem naslova brišemo i podatke o rezervacijama tog naslova jer gube smisao.

Brisanjem člana brišemo i podatke o svim njegovim rezervacijama jer više nisu potrebni.

```
DRZI :
    (SIFK) > KNJIGA (SIFK),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
    (SIFC) > CLAN (SIFC),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
```

Ne dozvoljavamo brisanje knjige koju drži neki član dok se ne reguliše naknada za knjigu.

Ne odzvoljavamo brisanje člana dok se ne regulišu knjige koje drži kod sebe.

```

JE_AUTOR :
    (SIFA) > AUTOR (SIFA),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
    (SIFN) > NASLOV (SIFN),
                                UPDATE - CASCADE,
                                DELETE - CASCADE ;

```

Ne dozvoljavamo brisanje autorstva postojećeg autora.

Brisanjem naslova brišemo i podatke o autorstvu tog naslova jer više nisu potrebni.

```

JE_REZERVISANA :
    (SIFK) > KNJIGA (SIFK),
                                UPDATE - CASCADE,
                                DELETE - CASCADE ;
    (SIFC) > CLAN (SIFC),
                                UPDATE - CASCADE,
                                DELETE - CASCADE ;

```

Brisanjem knjige brišemo i podatak o tome da je rezervisana jer gubi smisao.

Brisanjem člana brišemo i podatke o tome da ima rezervisane knjige jer više nisu potrebni.

Iz prethodne specifikacije i komentara možemo da zaključimo dve bitne stvari:

- za operaciju UPDATE prirodan izbor je akcija CASCADE; naime, operacija UPDATE nije ništa drugo nego promena vrednosti primarnog ključa ciljne relacije iz stare u novu (veoma retko u praksi ali se dešava) i prirodno je i da strani ključevi u referišućoj tabeli koji su imali tu staru vrednost dobiju novu;
- za operaciju DELETE moguća je svaka od postojeće četiri akcije; koja od njih će biti izabrana zavisi od “prirode stvari” i “sistemske logike”, odnosno logike funkcionisanja realnog sistema kojeg predstavlja baza podataka.

Inače, pravilan izbor referencijalnih integriteta predstavlja jednu od kritičnih aktivnosti pri projektovanju relacione baze podataka. Greške u tom pogledu mogu kasnije da imaju nepovoljne pa čak i teške posledice pri korišćenju baze podataka koje se mogu svrstati u dva vida:

- suviše labav referencijalni integritet – zbog preteranog korišćenja CASCADE akcija prilikom brisanja jednih podataka nastaje “bočni efekat” brisanja drugih podataka koji uopšte nisu trebali da budu obrisani;
- suviše krut referencijalni integritet – zbog preteranog korišćenja NO ACTION akcija u bazi podataka se ne mogu evidentirati pojedine promene koje se dešavaju u realnom sistemu koga ona predstavlja.

*Primer*

Posmatrajmo ranije navedenu relaciju RADNIK koja je istovremeno i referišuća i ciljna. Specifikacija referencijalnog integriteta za taj slučaj glasi:

```

RADNIK :
    (SIFNAD) > RADNIK (SIFR),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;

```

Ne dozvoljavamo brisanje radnika koji ima sebi podređene radnike.

Suviše labav referencijalni integritet, odnosno specifikacija CASCADE akcije za DELETE operaciju, imao bi teške posledice. U slučaju brisanja direktora kaskadno bi se obrisali svi njegovi podređeni, zatim njihovi podređeni i tako dalje i na kraju bi se dobila prazna relacija. Ispravan način da obrišemo radnika koji ima podređene jeste da prvo raskinemo referisanje podređenih na njega a zatim da ga obrišemo.

*Primer*

Posmatrajmo šeme relacija koje predstavljaju mrežu železničkih pruga:

STANICA ( SIFS, NAZIV )      PRUGA ( SIFSPOC, SIFSZAD, DUZINA )

Šema relacije PRUGA sadrži dva strana ključa u odnosu na istu ciljnu relaciju. To su SIFSPOC (šifra početne stanice) i SIFSZAD (šifra zadnje stanice). Prirodno je da za šemu relacije PRUGA za oba strana ključa važe iste dinamičke specifikacije referencijalnog integriteta, odnosno:

```
PRUGA :
  (SIFPOC) > STANICA (SIFS),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
  (SIFZAD) > STANICA (SIFS),
                                UPDATE - CASCADE,
                                DELETE - NO ACTION ;
```

### **4.3 Manipulativna komponenta relacionog modela**

Manipulativu komponentu relacionog modela podataka čine predstavljanje održavanja podataka i predstavljanje korišćenja podataka.



### 4.3.1 Održavanje podataka

Relacioni model podataka se odlikuje izuzetno jednostavnim predstavljanjem postupaka održavanja podataka, odnosno izmene sadržaja relacione baze podataka. Za to se koriste samo tri elementarne operacije:

- ubacivanje nove torke u relaciju;
- uklanjanje jedne torke iz relacije;
- izmena vrednosti jednog atributa jedne torke u relaciji.

Kako na postupak održavanja podataka najčešće utiče i sadržaj baze podataka, neophodna je i četvrta elementarna operacija:

- uvid u sadržaj jedne torke u relaciji.

Bilo kakav postupak izmene sadržaja relacione baze podataka može se predstaviti pomoću pogodno odabrane sekvence elementarnih operacija nad relacijama uz upotrebu dodatnih konstrukcija prisutnih u programskim jezicima (varijable, procedure, funkcije, izrazi, kontrolne strukture itd.).

Precizan opis postupaka održavanja relacione baze podataka podrazumeva da uz svaku od elementarnih operacija naznačimo i dodatne informacije o njoj:

- za sve četiri operacije: relaciju na koju se operacija odnosi;
- za ubacivanje: vrednosti koje se ubacuju kao nova torka;
- za izmenu: atribut koji se menja i novu vrednost;
- za uvid: atribut čija se vrednost očitava.

Da bi naš opis postupaka održavanja relacione baze podataka bio kompletan, uz elemente algoritamske notacije opisane u poglavlju B.2 priloga B neophodna je i dodatna notacija za elementarne operacije nad relacijama.

**Ubacivanje ::= INSERT Relacija ( Vrednost ,... )**

ubacuje novi red u relaciju, a kao **Vrednost** se može zadati konstanta ili varijabla; ne mora se nalaziti u petlji koja iterira po torkama relacije nego se može javiti bilo gde u algoritmu;

**Brisanje ::= DELETE**

brše tekuću torku iz relacije nad kojom se iterira pomoću posebne kontrolne strukture; sme se javljati samo unutar takve kontrolne strukture;

**Izmena ::= UPDATE Atribut = Vrednost**

menja vrednost jednog atributa u tekućoj n-torci u relaciji nad kojom se iterira pomoću posebne kontrolne strukture; sme se javljati samo unutar takve kontrolne strukture; kao **Vrednost** se može zadati konstanta ili varijabla;

**Uvid ::= Varijabla.Atribut**

može se vršiti isključivo unutar posebne kontrolne strukture koja iterira nad nekom relacijom; **Varijabla** je iterirajuća varijabla te kontrolne strukture a **Atribut** je neki atribut relacije nad kojom se iterira.

**DO FOR EACH ( Varijabla IN Relacija ) Postupak**

Kontrolna struktura koja iterira kroz zadatu relaciju, varijanta one iz priloga B.

Navedeni način opisa postupaka održavanja relacionog modela ilustrovan je sa dva primera koji slede. Radi effikasnosti algoritma implementirani su i rani završeci iterativnih postupaka iskokom EXIT.

### *Primer*

Za relacionu bazu podataka BIBLIOTEKA iz priloga A posmatrajmo slučaj održavanja kada neki član želi da se briše iz evidencije, pri čemu je pravilo biblioteke da član može da drži kod sebe najviše jednu knjigu.

Prema opisu načina rada naše biblioteke datom u prilogu A sledi da se taj slučaj održavanja ne sprovodi bezuslovno. Naime, ako član drži bar jednu knjigu kod sebe, zahtev za brisanjem se odbija sve dok član ne vrati knjigu ili knjige koje su kod njega ili ih ne plati ako ih je izgubio.

Prvo što treba da uradimo u analizi slučaja održavanja jeste da identifikujemo nad kojim relacijama se sprovode koje elementarne operacije. U konkretnom slučaju, prvo treba izvršiti očitavanje relacije DRZI kako bi se proverilo da li član koji želi da se briše iz evidencije drži kod sebe neku knjigu. Ako ne drži, vrši se odgovarajuće brisanje u CLAN. Ako to nije slučaj, postoji nekoliko ishoda:

- član će vratiti knjigu kasnije; do tada se ne briše iz evidencije;
- član je izgubio knjigu i želi odmah da je plati; čim je plati, vrše se odgovarajuća brisanja, prvo u relaciji DRZI a zatim u relaciji CLAN (ovaj redosled diktira dinamička specifikacija referencijalnog integriteta **NO ACTION** za atribut SifC u relaciji DRZI);
- član je izgubio knjigu i platiće je kasnije; do tada se ne briše iz evidencije.

Iz prethodnog sledi da se nad relacijom DRZI i sprovode operacije **Uvid** i **DELETE**, a nad CLAN samo **DELETE**.

Na osnovu dosadašnjeg opisa može se sastaviti i detaljna algoritamska specifikacija postupka održavanja. Radi konciznosti izostavljene su deklaracije varijabli i parametara. Radi preglednosti sve varijable počinju sa malim slovom “v” a parametri sa “p”, dok su nazivi procedura ispisani ukoso. Smatra se da je šifra člana zadata ispravno.

```

BrisanjeClana ( > pSifC, < pIshod )
: Inicijalizacija
: : pIshod = 'USPELO'
: Provera da li drzi knjige
: : vDrziKnjigu = FALSE
: : DO FOR EACH ( vDrzi IN DRZI )
: : | IF ( vDrzi.SIFC == pSifC )
: : | | vDrziKnjigu = TRUE
: Obrada ishoda provere
: : IF ( vDrziKnjigu )
: : | Drzi knjigu
: : | : Unos odgovora sta clan zeli
: : | : : OUTPUT ( "Clan ima knjigu kod sebe" )
: : | : : INPUT ( vStaZeli )
: : | : Obrada odgovora
: : | : : IF ( vStaZeli != 'PLACA_ODMAH' )
: : | : : | pIshod = 'NEUSPELO'
: Brisanje podataka (ako moze)
: : IF ( pIshod == 'USPELO' )
: : | Brisanje u DRZI (ako treba)
: : | : IF ( vDrziKnjigu )
: : | : | DO FOR EACH ( vDrzi IN DRZI )
: : | : | | IF ( vDrzi.SIFC == pSifC )
: : | : | | DELETE
: : | Brisanje u CLAN
: : | : DO FOR EACH ( vClan IN CLAN )
: : | : | IF ( vClan.SIFC == pSifC )
: : | : | DELETE

```

Simboličke konstante ishoda održavanja navedene su između navodnika. Pretpostavlja se da član plati knjigu pa se tek onda evidentira da plaća odmah.

### Primer

Za relacionu bazu podataka BIBLIOTEKA iz priloga A posmatrajmo slučaj održavanja kada neki član vraća pozajmljenu knjigu.

Prema opisu načina rada biblioteke izloženom u prilogu A taj slučaj održavanja se uvek sprovodi ali je složen. Kao prvo, treba očitati a zatim brisati podatke o tome da član drži knjigu kod sebe i evidentirati podatke o obavljenoj pozajmici. Nakon toga treba videti da li postoji rezervacija za naslov sa te knjige . Ako ona postoji, treba opslužiti prioriternu rezervaciju tako što će se evidentirati da je vraćena knjiga rezervisana za određenog člana.

Kao i u prethodnom primeru, prvo treba da uočimo sa kojim relacijama se obavljaju koje elementarne operacije. To se, uz neophodne napomene, može provesti sledećim tabelarnim pregledom gde su sa znakom “?” označene operacije koje se eventualno izvršavaju:

Relacija	Operacije	Napomene
DRZI	Uvid, DELETE	Na osnovu SifK vracene knjige se očitava SIFC i DATUM.
KNJIGA	Uvid	Na osnovu SifK vracene knjige se očitava njeno SifN.
POZAJMICA	INSERT	Evidentira se pozajmica.
REZERVACIJA	Uvid, ? DELETE	Na osnovu SifN vracene knjige proverava se da li za taj naslov postoji neopsluzena rezervacija. Ako postoji, ositava se za kog je clana i rezervacija se brise.
JE_REZERVISANA	? INSERT	Ako je opsluzena rezervacija clana, evidentira se koji je to clan i koja knjiga

Dalje se može postupno, u koracima razrade, sastaviti odgovarajuća algoritamska specifikacija. Smatra se da je šifra knjige koja se vraća zadata ispravno. Za razliku od prethodnog primera, ovde je uzeta u obzir efikasnost i kod iteriranja nad relacijama primenjen je iskok **EXIT** nakon uvida u traženu torku.

*Korak 1*

```
VracanjeKnjige ( > pSifK, < pIshod )
: Inicijalizacija
: Obrada
: Finalizacija
```

*Korak 2*

```
VracanjeKnjige ( > pSifK, < pIshod )
: Inicijalizacija
: Obrada
: : Uvid u neophodne podatke
: : Evidentiranje vracanja knjige
: : Evidentiranje obavljene pozajmice
: : Obrada eventualne rezervacije
: Finalizacija
```

*Korak 3*

```
VracanjeKnjige ( > pSifK, < pIshod )
: Inicijalizacija
: Obrada
: : Uvid u neophodne podatke
: : : Uvid koji je clan i kada uzeo knjigu
: : : Uvid kojeg je naslova knjiga
: : Evidentiranje vracanja knjige
: : Evidentiranje obavljene pozajmice
: : : Odredjivanje nove vrednosti primarnog kljuca
: : : Odredjivanje koliko je trajala pozajmica
: : : Evidentiranje podataka o pozajmici
: : Obrada eventualne rezervacije
: : : Provera da li ima rezervacije za vracenu knjige
: : : Opsluzivanje rezervacije (ako treba)
: : : : Nalazenje prioritete rezervacije
: : : : Evidentiranje da je rezervacija opsluzena
: : : : Evidentiranje da je knjiga rezervisana
: Finalizacija
```

*Korak 4 - finalni*

```

VracanjeKnjige ( > pSifK, < pIshod )
: Inicijalizacija
: : pIshod = 'NE_REZERVISANA'
: Obrada
: : Uvid u neophodne podatke
: : : Uvid koji je clan i kada uzeo knjigu
: : : : DO FOR EACH ( vDrzi IN DRZI )
: : : : | IF ( vDrzi.SIFK == pSifK )
: : : : | | vSifC = vDrzi.SIFC
: : : : | | vDatum = vDrzi.DATUM
: : : <:--|--|--EXIT
: : : Uvid kojeg je naslova knjiga
: : : : DO FOR EACH ( vKnjiga IN KNJIGA )
: : : : | IF ( vKnjiga.SIFK == pSifK )
: : : : | | vSifN = vKnjiga.SIFN
: : : <:--|--|--EXIT
: : Evidentiranje vracanja knjige
: : : DO FOR EACH ( vDrzi IN DRZI )
: : : | IF ( vDrzi.SIFK == pSifK )
: : : | | DELETE
: : <:--|--|--EXIT
: : Evidentiranje obavljene pozajmice
: : : Odredjivanje nove vrednosti primarnog kljuca
: : : : vSifP = 0
: : : : DO FOR EACH ( vPozajmica IN POZAJMICA )
: : : : | IF ( vPozajmica.SIFP > vSifP )
: : : : | | vSifP = vPozajmica.SifP
: : : : vSifP = vSifP + 1
: : : Odredjivanje koliko je trajala pozajmica
: : : : vDana = BrojDana ( vDatum )
: : : Evidentiranje podataka o pozajmici
: : : : INSERT POZAJMICA ( vSifP,vSifC,vSifK,vSifN,vDana )

```

nastavlja se

nastavak

```

: : Obrada eventualne rezervacije
: : : Provera da li ima rezervacije za naslov vracene knjige
: : : : vIma = FALSE
: : : : DO FOR EACH ( vRezervacija IN REZERVACIJA )
: : : : | IF ( vRezervacija.SIFN == vSifN )
: : : : | | vIma = TRUE
: : : <:--:--|--|--EXIT
: : : Opsluzivanje rezervacije (ako treba)
: : : : IF ( vIma )
: : : : | Nalazenje prioritete rezervacije
: : : : : vDatumVreme = 31.12.9999-24:00:00
: : : : : DO FOR EACH ( vRezervacija IN REZERVACIJA )
: : : : : : IF ( vRezervacija.SIFN == vSifN )
: : : : : : | IF ( vRezervacija.DATUM < vDatumVreme )
: : : : : : | | vDatumVreme = vRezervacija.DATUM
: : : : : : DO FOR EACH ( vRezervacija IN REZERVACIJA )
: : : : : : | IF ( vRezervacija.SIFN == vSifN )
: : : : : : | | IF ( vRezervacija.DATUM == vDatumVreme )
: : : : : : | | | vSifC = vRezervacija.SIFC
: : : : : <:--:--|--|--EXIT
: : : : Evidenciranje da je rezervacija opsluzena
: : : : : DO FOR EACH ( vRezervacija IN REZERVACIJA )
: : : : : : IF ( vRezervacija.SIFN == vSifN )
: : : : : : | IF ( vRezervacija.DATUM == vDatumVreme )
: : : : : : | | DELETE
: : : : : <:--:--|--|--EXIT
: : : : Evidenciranje da je knjiga rezervisana
: : : : : vDatum = tekuci datum
: : : : : INSERT JE_REZERVISANA ( pSifK,vSifC,vDatum )
: : : : : OUTPUT ( vSifC )
: : : : : pIshod = 'REZERVISANA'

```

U ovom i prethodnom primeru funkcije `INPUT` i `OUTPUT` obezbeđuju komunikaciju sa korisnikom.



### 4.3.2 Korišćenje podataka

Manipulativna komponenta korišćenja podataka je deo relacionog modela podataka gde njegov formalni karakter najviše dolazi do izražaja. Osnovu za taj deo relacionog modela predstavlja uža oblast Matematike poznata pod nazivom "Predikatski račun I reda". Iz te osnove izvedene su dve varijante formalnog predstavljanja korišćenja relacione baze podataka:

- relaciona algebra;
- relacioni račun.

Navedene formalne predstave su od značaja zbog toga što su poslužile kao osnova za razvoj konkretnih jezika za rad sa relacionom bazom podataka. Kao takve, one se razmatraju u narednom poglavlju.

## 4.4 Univerzalni relacioni model

Relacioni model koji smo do sada razmatrali je u strukturnom smislu tradicionalan, onakav kakvog ga je postavio Kod u svojoj publikaciji još 1971. godine. U osnovi tog modela jeste uslov prostih atributa šeme relacije, odnosno dva ograničenja:

- atributi moraju biti skalarni - kao vrednosti ne smeju imati skupove vrednosti;
- atributi moraju biti elementarni - ne smeju biti slogovi, sastavljeni od delova, odnosno ne smeju imati slogovne vrednosti.

Ovakav relacioni model dodatno je opisan kao "ravan" ("flat relational model"), po sličnosti implementacije tabela i običnih slogovnih datoteka ("flat files"), a motivacija za navedena dva ograničenja je u suštini implementacione prirode - relacije sa slogovnim i skupovnim atributima je bilo (tada) teško implementirati. Navedena ograničenja u potpunosti odražava i sintaksna definicija šeme relacije u notaciji iz priloga B koju smo do sada koristili:

**SemaRelacije ::= Naziv ( Atribut , ... )**

gde je **Atribut** prosto, odnosno skalarno i elementarno. Primera radi, u šemi relacije CLAN nije dozvoljeno korišćenje slogovnog atributa Adresa, nego se umesto toga moraju koristiti njegovi elementarni delovi, što daje šemu relacije:

CLAN ( SifC, Ime, Mesto, Ulica, Broj )

Sa druge strane, u šemi relacije CLAN nije moguće predstaviti podatak da član ima ni jedan, jedan ili više telefonskih brojeva za kontakt, pošto bi to bio skupovni atribut.

Ugradnja ograničenja implementacione prirode u neku teoriju nije dobro rešenje i to važi i za teorijske osnove relacionih baza podataka. Na tu okolnost je ukazano već krajem 70-ih godina, kada je uveden pojam "ugnježdenog relacionog modela" ("nested relational model"), za koji je ovde odabran naziv "univerzalni relacioni model". U osnovu takvog pristupa je uklanjanje ograničenja skalarnosti i elementarnosti nad atributima. Drugim rečima, atributi osim toga što mogu biti prosti mogu biti skupovni, slogovni ili kombinacija oboje koja odgovara "ugnježdenoj" šemi relacija (otud naziv "nested relational model"). Takav pristup se ne ograničava samo na jedan nivo (ugnježdene šema relacija može za atribut imati opet ugnježdenu šemu relacije), što sve odražava sledeći skup sintaksnih definicija:

```
SemaRelacije ::= Naziv ( Element ,... )
Element ::= Atribut | Slog | Skup
Atribut ::= Naziv
Slog ::= Naziv ≤ Element Element ,... ≥
Skup ::= Naziv { Element }
```

Shodno ovoj definiciji, šema relacije CLAN za slučaj da članovi imaju jednu kontakt-adresu i ni jedan, jedan ili više telefonskih brojeva bila bi

CLAN ( SifC, Ime, ADRESA < Mesto, Ulica, Broj >, TELEFON { Broj } )

ili, sa naknadnom definicijom neprostih atributa

CLAN ( SifC, Ime, ADRESA<>, TELEFON{ } )  
ADRESA = < Mesto, Ulica, Broj >    TELEFON = { Broj }

gde su naznake sloga i skupa uz nazive atributa zadržane radi preglednosti. Za slučaj da član može da ima više kontakt-adresa imali bi u razvijenoj formi

CLAN ( SifC, Ime, ADRESA {< Mesto, Ulica, Broj >}, TELEFON { Broj } )

S obzirom da se skup slogova {< ... >} može smatrati za šemu relacije (...), kao i s obzirom da šema relacije može imati i samo jedan atribut, konstrukcije {< ... >} i {...} se obe mogu zameniti konstrukcijom (...), što daje šemu relacije

CLAN ( SifC, Ime, ADRESA ( Mesto, Ulica, Broj ), TELEFON ( Broj ) )

Pri tome ADRESA mora imati bar jednu torku a TELEFON može biti i prazno.

Prethodna izmena notacije iziskuje izmenjenu sintaksnu definiciju šeme relacije:

```
SemaRelacije ::= Naziv ( Element ,... )
Element ::= Atribut | Slog | SemaRelacije
Atribut ::= Naziv
Slog ::= Naziv ≤ Element Element ,... ≥
```

Prethodno opisani univerzalni relacioni model je vremenom zaokružen, u smislu da su za njega definisane i integritetska i manipulativna komponenta ("nested relational algebra"). Takav model znatno pojednostavljuje proučavanje zavisnosti izmedju podataka u šemi relacije, čemu je posvećeno poglavlje 7. Svoju potvrdu u praksi ovaj model je dobio kroz objektno-relacione baze podataka koje dozvoljavaju, pored ostalog, slogovne attribute i relacije u relacijama.

*PREPORUKA ČITAOCIMA*

*U prilogu A na kraju ove knjige dat je primer ustrojstva i sadržaja relacione baze podataka BIBLIOTEKA. Ova baza podataka se koristi kao osnovni ilustrativni primer u svim poglavljima, pa se radi što veće preglednosti i jasnoće čitaocima preporučuje da naprave kopiju dela žriloga A-2 kako bi mogli da je koriste uz bilo koji deo ove publikacije.*

# 5

## ***FORMALNI UPITNI JEZICI***

---

U poglavlju 2 posvećenom sistemu za upravljanje bazom podataka ukazali smo i na odgovarajuće jezike u okviru podsistema definicije, kontrole i manipulacije podacima. U praksi, razvoju i implementaciji tih jezika u vidu odgovarajućeg softvera prethodi proučavanje odgovarajućih formalnih jezika u kojima nisu prisutni detalji vezani za izvršavanje na računaru. Na taj način se mogu postaviti svi važniji zahtevi i pre no što se pristupi izradi softvera za rad sa bazom podataka.

Posebno značajnu grupu formalnih jezika čine formalni upitni jezici, što je razumljivo s obzirom da su upiti najčešće izvođene operacije nad bazama podataka. Ovi jezici se mogu podeliti na dve vrste. To su:

- imperativni jezici, pomoću kojih se preko odgovarajućeg skupa operacija opisuje *kako* se dolazi do željenih podataka;
- deklarativni jezici, pomoću kojih se preko odgovarajućih kvantifikatora, logičkih simbola i predikata opisuje *šta* se želi od podataka.

U ovom poglavlju razmotrićemo obe vrste formalnih upitnih jezika.

## 5.1 Relaciona algebra

Relaciona algebra spada u kategoriju formalnih upitnih jezika imperativnog karaktera. Čini je skup operatora za rad sa relacijama, a rezultati operacija relacione algebre takođe su relacije.

### 5.1.1 Operacije relacione algebre

Prema prvobitnoj definiciji relacione algebre, ustaljeno je gledište da je čini skup od 8 operacija koje se nazivaju osnovnim, ali su samo 5 od njih elementarne, dok se preostale 3 mogu izvesti iz njih.

Pored osnovne podele na elementarne i izvedene, operacije relacione algebre mogu se prema broju operanada (relacija koje učestvuju u operaciji) klasifikovati na unarne (1 operand) i binarne (2 operanda). Uz to, postoji i podela na tradicionalne skupovne i posebne relacione operacije.

U sledećoj tabeli dati su simboli, nazivi, i klasifikacija svih 8 osnovnih operacija relacione algebre.

simbol	naziv	složenost	operanada	vrsta
$\sigma$	restrikcija	elementarna	unarna	posebna
$\pi$	projekcija	elementarna	unarna	posebna
$\cup$	unija	elementarna	binarna	skupovna
$-$	razlika	elementarna	binarna	skupovna
$\cap$	presek	izvedena	binarna	skupovna
$\times$	Dekartov proizvod	elementarna	binarna	skupovna
$\bowtie$	spajanje	izvedena	binarna	posebna
$/$	deljenje	izvedena	binarna	posebna

Za objašnjenja operacija relacione algebre neophodne su izvesne napomene o notaciji. Kao i do sada, nazivi šema relacija pisani su velikim, a nazivi relacija malim slovima. Nazivi atributa uvek se pišu velikim slovima. Sa  $X$  se najčešće označava slup atributa šeme relacije, a sa  $x$  torka odgovarajuće relacije.  $Y$ ,  $Z$  i slično najčešće označavaju podskupove od  $X$ . Radi jasnoće definicija, ponekad se sa  $r(Y)$  naglašava da je u pitanju relacija kao skup torki nad skupom atributa  $Y$ . Pod kardinalnošću relacije  $N(r)$  podrazumeva se broj torki u relaciji  $r$ .

Za uspešno rešavanje problema u oblasti relacione algebre bitno je da imamo uvid u šeme relacija koje nastaju posle svakog koraka u izračunavanju složenih izraza. Iz tog razloga, praksa je da uz naziv relacije pišemo i attribute u sastavu njene šeme.



### 5.1.2 Restrikcija

#### *Definicija*

*Restrikcija (simbol  $\sigma$ ) je elementarna, unarna i posebna operacija koja iz polazne relacije po zatom kriterijumu izdvaja podskup torki. Kriterijum je neki logički izraz koji je izračunljiv nad svakom torkom. Dobijena relacija ima istu strukturu kao i polazna.*

Ako je  $r$  relacija nad šemom  $R(X)$ , a  $P(X)$  uslov restrikcije, formalna definicija operacije restrikcije glasi:

$$\sigma_{P(X)}(r) = t(X) = \{ x \mid x \in r \wedge P(x) \}$$



### 5.1.3 Projekcija

#### *Definicija*

*Projekcija (simbol  $\pi$ ) je elementarna, unarna i posebna operacija koja iz polazne relacije po zadatom skupu atributa formira novu relaciju kao skup torki nad tim atributima. Zadati skup atributa mora biti podskup skupa atributa polazne relacije, a vrednosti atributa u torkama nastale relacije odgovaraju onima u torkama polazne relacije.*

Ako je  $r$  relacija nad šemom  $R(X)$ ,  $Y$  zadati skup atributa a  $x$  i  $y$  torke nad  $X$  i  $Y$ , formalna definicija operacije projekcije glasi:

$$\pi_Y(r) = t(Y) = \{ t \mid Y \subseteq X \wedge y \in x \}$$

pri čemu su odgovarajuće šeme relacija  $R(X)$  i  $T(Y)$ .

Primena operacije projekcije može dovesti do situacije da više torki polazne relacije daje iste vrednosti nad podskupom atributa, ali se saglasno tome da je rezultat operacije relacione algebre uvek relacija uzima samo jedna rezultattna toraka.

### Primeri

Ako iz prethodne navedene relacije **naslov** želimo da dobijemo pregled svih naziva i šifara oblasti, primenjujemo operaciju projekcije

$$\pi_{\text{NAZIV,SIFO}}(\text{naslov}) \rightarrow t(\text{NAZIV,SIFO})$$

koja kao rezultat daje relaciju **t**

<b>t ( NAZIV</b>	<b>SIFO )</b>
-----	
Relacione baze podataka	BP
Racunarske komunikacije	RM
PASCAL programiranje	PJ
Programski jezik C	PJ
-----	

Kao ilustraciju slučaja kada više torki polazne relacije daje jednu torku u nastaloj relaciji, navedimo operaciju projekcije

$$\pi_{\text{SIFO}}(\text{naslov}) \rightarrow t(\text{SIFO})$$

koja daje šifre oblasti, odnosno relaciju **t**

<b>t ( SIFO )</b>
----
BP
RM
PJ
----

u kojoj se vrednost PJ pojavljuje jednom, dok je u polaznoj relaciji **naslov** ta vrednost prisutna dva puta.

Jedna od namena operacije projekcije je i promena redosleda atributa. Kao ilustraciju toga navedimo operaciju

$$\pi_{\text{NAZIV,SIFN,SIFO}}(\text{naslov}) \rightarrow t(\text{NAZIV,SIFN,SIFO})$$

koja kao rezultat daje relaciju **t**

<b>t ( NAZIV</b>	<b>SIFN</b>	<b>SIFO )</b>
-----		
Relacione baze podataka	RBP0	BP
Racunarske komunikacije	RK00	RM
PASCAL programiranje	PP00	PJ
Programski jezik C	PJC0	PJ
-----		

Operacije relacione algebre mogu se kombinovati. Kao primer navedimo složenu operaciju

$$\pi_{\text{NAZIV}} (\sigma_{\text{SIFO}='PJ'} (\text{naslov})) \rightarrow \text{pj\_naslov} (\text{NAZIV})$$

koja kao rezultat daje nazive svih naslova za koje je šifra oblasti jednaka 'PJ', odnosno relaciju *pj\_naslov*

```

pj_naslov ( NAZIV
            -----
            PASCAL programiranje
            Programski jezik C
            -----

```

Naziv nastale relacije se bira slobodno, pod uslovom da je unikatan, odnosno da se razlikuje od naziva relacija u bazi podataka i naziva relacija nastalih ranijim izvođenjem operacija. Tu mogućnost smo iskoristili u ovom primeru i koristićemo je na dalje, posebno u složenim sekvencama operacija kada to doprinosi preglednosti.

#### *Napomene*

Na osnovu izloženog, za operaciju projekcije možemo zaključiti

- šema relacije se menja i određuje je zadati skup atributa;
- za broj torki u rezultatu važi  $N(\mathbf{t}) \leq N(\mathbf{r})$ .

### 5.1.4 Unija

#### *Definicija*

*Unija (simbol  $\cup$ ) je elementarna, binarna i skupovna operacija koja iz dve polazne relacije formira novu koja sadrži sve torke koje se nalaze u bilo kojoj ili eventualno u obe polazne relacije. Ova operacija nije moguća između bilo koje dve relacije, nego samo između onih koje zadovoljavaju uslove:*

- *šeme relacija imaju isti broj atributa;*
- *atributi šema relacija redom odgovaraju jedni drugim.*

Navedeni uslovi zajedno čine tzv. uslov unijske kompatibilnosti koji važi i za još dve skupovne operacije relacione algebre. Drugi uslov podrazumeva da atributi odgovaraju po značenju i tipu. Jednakost naziva atributa nije neophodna, a nije uvek ni moguća, što će biti ilustrovano primerima koji slede.

Ako je su **r** i **s** relacije nad šemama  $R(X)$  i  $S(X)$ , gde  $X$  označava unijski kompatibilni skup atributa u obe relacije, formalna definicija operacije unije glasi:

$$r \cup s = t(X) = \{ x \mid x \in r \vee x \in s \}$$

Saglasno osobini skupova, svaka torka koja je prisutna u obe polazne relacije javlja se samo jednom u rezultatnoj relaciji.

*Primeri*

Pozmatrajmo relacije **clan** i **autor** iz baze podataka BIBLIOTEKA. Ako želimo podatke o svim osobama u evidenciji, to postizemo direktnom primenom operacije unije

$$\text{clan} \cup \text{autor} \rightarrow \text{osoba} (\text{SIFx}, \text{IME})$$

koja kao relaciju **osoba** (nastaloj relaciji slobodno dodeljujemo novo unikatno ime) daje:

<i>osoba</i> (	SIFx	IME	)
	JJ0	J.Jankovic	
	PP0	P.Petrovic	
	JJ1	J.Jovanovic	
	MM0	M.Markovic	
	AP0	A.Popovic	
	IT0	I.Todorovic	
	AP1	A.Petrovic	
	JN0	J.Nikolic	
	DM0	D.Markovic	
	ZP0	Z.Petrovic	

Ovde su neophodne izvesne napomene u vezi uslova unijske kompatibilnosti. Značenje atributa je isto u obe relacije, orgovarajući atributi su istih tipova (string), ali su nazivi prvih atributa (SIFC i SIFA) različiti. Iz tog razloga smo bili prinuđeni da u šemi rezultatne relacije **osoba** usvojimo novi naziv atributa, ili da umesto toga usvojimo jedan od postojeća dva. Drugo, u konkretnom slučaju atributi su odgovarajući i po dužini. Kada to nije slučaj, za šemu rezultatne relacije uzima se veća dužina.

Posmatrajmo relacije **drzi** i **pozajmica** koje su deo baze podataka BIBLIOTEKA (Prilog A). Ako želimo da utvrdimo koje knjige su trenutno ili su ikada bile kod članova, pre operacije unije prvo moramo uskladiti šeme relacija. To postizemo tako što prvo izvršimo operacije

$$\pi_{\text{SIFK}}(\text{drzi}) \rightarrow t1 (\text{SIFK})$$

$$\pi_{\text{SIFK}}(\text{pozajmica}) \rightarrow t2 (\text{SIFK})$$

a zatim operaciju unije

$$t1 \cup t2 \rightarrow \text{knjiga\_u\_prometu} (\text{SIFK})$$

pri čemu se redom dobijaju relacije

<i>t1</i> ( SIFK )	<i>t2</i> ( SIFK )	<i>knjiga_u_prometu</i> ( SIFK )
----	----	----
001	004	001
002	007	002
004	005	004
----	008	007
	002	005
	009	008
	----	009
		----

u kojoj je eliminisano duplo pojavljivanje vrednosti 002 i 004. Prethodno se može napisati i u formi složenog izraza relacione algebre

$$\pi_{\text{SIFK}}(\text{drzi}) \cup \pi_{\text{SIFK}}(\text{pozajmica}) \rightarrow \text{knjiga\_u\_prometu} (\text{SIFK})$$

#### *Napomene*

Za operaciju unije možemo zaključiti:

- šeme relacija treba po potrebi prethodno uskladiti operacijom projekcije;
- za broj torki u rezultatu važi  $\max(N(\mathbf{r}), N(\mathbf{s})) \leq N(\mathbf{t}) \leq N(\mathbf{r}) + N(\mathbf{s})$ .



### 5.1.5 Razlika

#### *Definicija*

*Razlika (simbol  $-$ ) je elementarna, binarna i skupovna operacija koja iz dve polazne relacije formira novu koja sadrži sve torke prve relacije koje se ne nalaze u drugoj relaciji. Ova operacija je moguća samo između unijski kompatibilnih relacija.*

Neka su  $\mathbf{r}$  i  $\mathbf{s}$  relacije nad šemama  $R(X)$  i  $S(X)$ , gde  $X$  označava unijski kompatibilni skup atributa u obe relacije. Formalna definicija operacije razlike glasi:

$$\mathbf{r} - \mathbf{s} = \mathbf{t}(X) = \{ x \mid x \in \mathbf{r} \wedge x \notin \mathbf{s} \}$$

*Primeri*

Posmatrajmo relacije **drzi** i **pozajmica** u bazi podataka BIBLIOTEKA. Ako želimo da utvrdimo koji članovi čitaju koje knjige prvi put, moramo za parove vrednosti SIFC,SIFK formirati razliku skupova tih parova između relacija **drzi** i **pozajmica**. To postizemo sekvencom operacija

$$\pi_{\text{SIFC,SIFK}}(\text{drzi}) \rightarrow t1(\text{SIFC,SIFK})$$

$$\pi_{\text{SIFC,SIFK}}(\text{pozajmica}) \rightarrow t2(\text{SIFC,SIFK})$$

$$t1 - t2 \rightarrow \text{cita\_1}(\text{SIFC,SIFK})$$

odnosno složenim izrazom

$$\pi_{\text{SIFC,SIFK}}(\text{drzi}) - \pi_{\text{SIFC,SIFK}}(\text{pozajmica}) \rightarrow \text{cita\_1}(\text{SIFC,SIFK})$$

Pri tome, odgovarajuće relacije su:

$t1$ ( SIFC SIFK )	$t2$ ( SIFC SIFK )	$\text{cita\_1}$ ( SIFC SIFK )
-----	-----	-----
JJ0 001	JJ0 004	JJ0 001
PP0 002	PP0 007	-----
JJ0 004	JJ1 005	
-----	JJ0 008	
	PP0 002	
	JJ1 009	
	-----	

Ako za istu bazi podataka želimo uvid u šifre članova koji trenutno ne drže ni jednu knjigu kod sebe, moramo da izvršimo sekvencu operacija

$$\pi_{\text{SIFC}}(\text{clan}) \rightarrow t1(\text{SIFC})$$

$$\pi_{\text{SIFC}}(\text{drzi}) \rightarrow t2(\text{SIFC})$$

$$t1 - t2 \rightarrow ne\_drzi(\text{SIFC})$$

ili odgovarajući složen izraz

$$\pi_{\text{SIFC}}(\text{clan}) - \pi_{\text{SIFC}}(\text{drzi}) \rightarrow ne\_drzi(\text{SIFC})$$

pri čemu su odgovarajuće relacije

$t1$ ( SIFC )	$t2$ ( SIFC )	$ne\_drzi$ ( SIFC )
----	----	----
JJ0	JJ0	JJ1
JJ1	PP0	MM0
PP0	----	----
MM0		
----		

### *Napomene*

Za operaciju razlike možemo zaključiti:

- šeme relacija treba po potrebi prethodno uskladiti operacijom projekcije;
- za broj torki u rezultatu važi  $0 \leq N(\mathbf{t}) \leq N(\mathbf{r})$ .

### 5.1.6 Presek

#### *Definicija*

*Presek (simbol  $\cap$ ) je izvedena, binarna i skupovna operacija koja iz dve polazne relacije formira novu koja sadrži sve torke prve relacije koje se nalaze i u drugoj relaciji. Ova operacija je moguća samo između unijski kompatibilnih relacija.*

Ako su  $r$  i  $s$  relacije nad šemama  $R(X)$  i  $S(X)$ , gde  $X$  označava unijski kompatibilni skup atributa u obe relacije, formalna definicija operacije preseka glasi:

$$r \cap s = t(X) = \{ x \mid x \in r \wedge x \in s \}$$

Presek je kao izvedena operacija definisan preko operacije razlike

$$r \cap s = r - (r - s)$$

*Primei*

Želimo da odredimo koji članovi trenutno ponovo čitaju koju knjigu. Da bi to dobili, treba nad relacijama **drzi** i **pozajmica** izvršiti sekvencu operacija

$$\pi_{\text{SIFC}, \text{SIFK}}(\text{drzi}) \rightarrow t1(\text{SIFC}, \text{SIFK})$$

$$\pi_{\text{SIFC}, \text{SIFK}}(\text{pozajmica}) \rightarrow t2(\text{SIFC}, \text{SIFK})$$

$$t1 \cap t2 \rightarrow \text{cita\_opet}(\text{SIFC}, \text{SIFK})$$

odnosno složenim izrazom

$$\pi_{\text{SIFC}, \text{SIFK}}(\text{drzi}) \cap \pi_{\text{SIFC}, \text{SIFK}}(\text{pozajmica}) \rightarrow \text{cita\_opet}(\text{SIFC}, \text{SIFK})$$

pri čemu se redom dobijaju relacije

<i>t1</i> ( SIFC   SIFK )	<i>t2</i> ( SIFC   SIFK )	<i>cita_opet</i> ( SIFC   SIFK )
-----	-----	-----
JJ0    001	JJ0    004	PP0    002
PP0    002	PP0    007	JJ0    004
JJ0    004	JJ1    005	-----
-----	JJ0    008	
	PP0    002	
	JJ1    009	
	-----	

*Napomene*

Za operaciju preseka možemo zaključiti:

- šeme relacija treba po potrebi prethodno uskladiti operacijom projekcije;
- za broj torki u rezultatu važi  $0 \leq N(\mathbf{t}) \leq \min(N(\mathbf{r}), N(\mathbf{s}))$ .

### 5.1.7 Dekartov proizvod

#### *Definicija*

*Dekartov proizvod (simbol  $\times$ ) je elementarna, binarna i skupovna operacija koja iz dve polazne relacije formira novu, sa torkama dobijenim tako što se svaka torka iz prve relacije redom "spoji" sa svakom torkom druge relacije, pri čemu šema nastale relacije sadrži redom sve attribute polaznih relacija.*

Ako se za puni naziv atributa usvoji naziv relacije i naziv atributa (relacija.atribut), tada je Dekartov proizvod moguć uvek osim kada se primenjuje između jedne iste relacije. U tom slučaju, za jedno od pojavljivanja iste relacije usvaja se nadimak a za puni naziv atributa uzimaju se nadimak i naziv atributa (nadimak.atribut).

Ako su  $\mathbf{r}$  i  $\mathbf{s}$  relacije nad šemama  $R(X)$  i  $S(Y)$ , gde su  $X$  i  $Y$  skupovi atributa u šemama relacija, formalna definicija operacije Dekartovog proizvoda glasi:

$$\mathbf{r} \times \mathbf{s} = \mathbf{t}(XY) = \{ xy \mid x \in \mathbf{r} \wedge y \in \mathbf{s} \}$$

*Primer*

Posmatrajmo relacije **naslov** i **oblast** u bazi podataka BIBLIOTEKA. Ako izvršimo operaciju Dekartovog proizvoda

$$\mathbf{naslov} \times \mathbf{oblast} \rightarrow t(\text{SIFN, N.NAZIV, N.SIFO, O.SIFO, O.NAZIV})$$

kao rezultat dobijamo relaciju, u kojoj smo nazive polaznih relacija ispred naziva atributa sveli na početna slova

$t$	(	SIFN	N.NAZIV	N.SIFO	O.SIFO	O.NAZIV	)
*	RBP0	Relacione	baze podataka	BP	BP	Baze podataka	
	RBP0	Relacione	baze podataka	BP	RM	Racunarske mreze	
	RBP0	Relacione	baze podataka	BP	PJ	Programski jezici	
	RK00	Racunarske	komunikacije	RM	BP	Baze podataka	
*	RK00	Racunarske	komunikacije	RM	RM	Racunarske mreze	
	RK00	Racunarske	komunikacije	RM	PJ	Programski jezici	
	PP00	PASCAL	programiranje	PJ	BP	Baze podataka	
	PP00	PASCAL	programiranje	PJ	RM	Racunarske mreze	
*	PP00	PASCAL	programiranje	PJ	PJ	Programski jezici	
	PJC0	Programski	jezik C	PJ	BP	Baze podataka	
	PJC0	Programski	jezik C	PJ	RM	Racunarske mreze	
*	PJC0	Programski	jezik C	PJ	PJ	Programski jezici	

Dobijena relacija kao celina nema smisla niti joj se može pridružiti određeno značenje. Obično za takvu relaciju ne možemo ni da usvojimo naziv koji nešto znači. Međutim, nisu sve torke u ovoj relaciji bez smisla. torke označene sa \* imaju jasno značenje: u njima se uz svaki deo iz relacije **naslov** nalazi odgovarajući deo iz relacije **oblast**.

*Napomena*

Za operaciju Dekartovog proizvoda možemo zaključiti:

- šema rezultantne relacije sadrži sve attribute polaznih relacija;
- za broj torki u rezultatu važi  $N(\mathbf{t}) = N(\mathbf{r}) \cdot N(\mathbf{s})$ .

### 5.1.8 Spajanje

#### *Definicija*

*Spajanje (simbol  $\bowtie$ ) je izvedena, binarna i posebna operacija koja iz dve polazne relacije formira novu, sa torkama dobijenim u dva koraka:*

- *svaka torka iz prve relacije redom se spaja sa svim torkama iz druge relacije;*
- *iz tako dobijenih torki izdvajaju se one koje zadovoljavaju zadati uslov  $P$ .*

Neka su  $\mathbf{r}$  i  $\mathbf{s}$  relacije nad šemama  $R(X)$  i  $S(Y)$ , gde su  $X$  i  $Y$  skupovi atributa u šemama relacija. Formalna definicija operacije spajanja glasi:

$$\mathbf{r} \bowtie_{P(XY)} \mathbf{s} = \sigma_{P(XY)}(\mathbf{r} \times \mathbf{s}) = \mathbf{t}(XY) = \{ xy \mid x \in \mathbf{r} \wedge y \in \mathbf{s} \wedge P(xy) \}$$



*$\theta$ -spajanje*

Prethodna definicija operacije spajanja dozvoljava proizvoljni uslov P, pod uslovom da je izračunljiv za svaku torku relacije nastale Dekartovim proizvodom polaznih relacija. U praksi su takvi uopšteni slučajevi retki, pa se zato ograničavamo na specijalne slučajeve uslova.

Posmatrajmo relacije  $\mathbf{r}$  i  $\mathbf{s}$ , čije su šeme  $R(X)$  i  $S(Y)$ . Neka su  $X_i$  i  $Y_k$  atributi za koje redom važi  $X_i \in X$  i  $Y_k \in Y$ . Pod  $\theta$ -spajanjem

$$\mathbf{r} \mathrel{>_{X_i \theta Y_k}} \mathbf{s}$$

podrazumevamo spajanje kod koga simbol  $\theta$  označava bilo koji od operatora poređenja ( $=, \leq, <, >, \geq, \neq$ ). Ako umesto po jednog atributa iz obe relacije u uslovu spajanja učestvuju podskupovi sa istim brojem atributa, operator poređenja se primenjuju između parova atributa odgovarajući broj puta i u logičkoj konjukciji.

*Ekvi-spajanje*

I pored toga što znatno ograničava formu uslova spajanja,  $\theta$ -spajanje je i dalje suviše uopšteno da bi osim teoretskog imalo bilo kakav drugi značaj. Za razliku od toga, specijalni slučaj gde  $\theta$  predstavlja jednakost ( $=$ ) je vrlo čest u praksi. Takvo spajanje, koje se naziva ekvi-spajanje, primenjeno na relacije **naslov** i **oblast** i sa uslovom jednakosti atributa SIFO u obe relacije, dalo bi za rezultat relaciju **t** iz primera za Dekartov proizvod koja sadrži samo torke označene sa \*. Ekvi-spajanje po jednom atributu osnažava se sa

$$\mathbf{r} >_{(X_i)=(Y_k)} \mathbf{s}$$

ili još kompaktnije sa izostavljanjem znaka jednakosti, a slučaj ekvi-spajanja po više atributa označava se kao

$$\mathbf{r} >_{(X_1, X_2, \dots, X_n)=(Y_1, Y_2, \dots, Y_n)} \mathbf{s}$$

ili bez znaka jednakosti, a uslov spajanja pri tome glasi

$$X_1=Y_1 \wedge X_2=Y_2 \wedge \dots \wedge X_n=Y_n$$

odnosno upoređuju se atributi na istim pozicijama u zagradama.

### Prirodno spajanje

Videli smo iz prethodnog da ekvi-spajanje iz relacije koju daje Dekartov proizvod izdvaja samo one torke koje imaju smisla. Međutim, relacija nastala spajanjem ima jedan suvišan atribut. Naime, vrednosti atributa iz jedne i druge relacije po kojima se vrši spajanje uvek su iste, pa se dodatnom operacijom projekcije eliminiše nepotrebnii atribut. Ovo je toliko čest slučaj u praksi da je uvedena specijalna operacija prirodnog spajanja koja podrazumeva sekvencu tri elementarne operacije relacione algebre:

- Dekartov proizvod relacija;
- Restrikcija po uslovu jednakosti spojnih atributa u obe relacije;
- projekcija po razlici unije svih atributa i skupa spojnih atributa iz bilo koje od relacija.

Prirodno spajanje dve relacije po jednom atributu označava se sa

$$\mathbf{r} \bowtie_{(X_i) * (Y_k)} \mathbf{s}$$

odnosno za slučaj više spojnih atributa sa

$$\mathbf{r} \bowtie_{(X_1, X_2, \dots, X_n) * (Y_1, Y_2, \dots, Y_n)} \mathbf{s}$$

Specijalan slučaj označavanja

$$\mathbf{r} \bowtie_* \mathbf{s}$$

podrazumeva prirodno spajanje po svim atributima koji imaju jednake nazive u obe relacije.

Formalnu definiciju prirodnog spajanja možemo formulisati na sledeći način. Neka su  $\mathbf{r}$  i  $\mathbf{s}$  relacije,  $R(X)$  i  $S(Y)$  njihove šeme, a  $A \subseteq X$  i  $B \subseteq Y$  uređeni podskupovi jednakog broja atributa relacija  $\mathbf{r}$  i  $\mathbf{s}$ , respektivno. Prirodno spajanje definišemo kao

$$\mathbf{r} \bowtie_{(A) * (B)} \mathbf{s} = \pi_{XY-B}(\sigma_{(A)=(B)}(\mathbf{r} \times \mathbf{s})) = \mathbf{t}(XY-B)$$

gde jednakost uređenih podskupova  $A$  i  $B$  podrazumeva jednakost korespondentnih elemenata. Umesto  $XY-B$  sa desne strane može se navesti i  $XY-A$ .

*Primeri*

Ekvi-spajanje relacija **naslov** i **oblast** po atributima SIFO

**naslov**  $>_{(SIFO)=(SIFO)} < \mathbf{oblast} \rightarrow t(SIFN, N.NAZIV, N.SIFO, O.SIFO, O.NAZIV)$

daje kao rezultat relaciju

$t$	(	SIFN	N.NAZIV	N.SIFO	O.SIFO	O.NAZIV	)
		RBP0	Relacione baze podataka	BP	BP	Baze podataka	
		RK00	Racunarske komunikacije	RM	RM	Racunarske mreze	
		PP00	PASCAL programiranje	PJ	PJ	Programski jezici	
		PJC0	Programski jezik C	PJ	PJ	Programski jezici	

Prirodno spajanje nad istim relacijama

**naslov**  $>_{(SIFO)*(SIFO)} < \mathbf{oblast} \rightarrow t(SIFN, N.NAZIV, SIFO, O.NAZIV)$

daje rezultat bez suvišnog atributa SIFO

$t$	(	SIFN	N.NAZIV	N.SIFO	O.NAZIV	)
-----						
RBP0		Relacione	baze podataka	BP	Baze podataka	
RK00		Racunarske	komunikacije	RM	Racunarske mreze	
PP00		PASCAL	programiranje	PJ	Programski jezici	
PJC0		Programski	jezik C	PJ	Programski jezici	
-----						

Prirodno spajanje relacija **naslov** i **oblast** zadato u formi

**naslov**  $>_* < \mathbf{oblast} \rightarrow t(SIFN, NAZIV, SIFO)$

daje kao rezultat praznu relaciju, pošto se podrazumeva da se spajanje vrši po atributima jednakih naziva, a to su NAZIV i SIFO, odnosno po uslovu

$$N.NAZIV = O.NAZIV \wedge N.SIFO = O.SIFO$$

koji ne zadovoljava ni jedna torka u  $\mathbf{r} \times \mathbf{s}$ .

### *Napomene*

Za operaciju prirodnog spajanja možemo zaključiti

- rezultatna relacija sadrži sve attribute prve relacije i one attribute druge relacije koji ne pripadaju skupu atributa spajanja;
- usvojeno je da spajanje relacija bez zajedničkih (spojnih) atributa kao rezultat daje Dekartov proizvod;
- za broj torki u rezultatu važi  $N(\mathbf{t}) \leq N(\mathbf{r}) \cdot N(\mathbf{s})$ .

### 5.1.9 Deljenje

#### *Definicija*

Deljenje (simbol  $/$ ) je izvedena, binarna i posebna operacija koja predstavlja najsloženiju operaciju relacione algebre. Formalna definicija deljenja, koja postaje sasvim jasna tek posle pogodnog primera, može se iskazati na sledeći način. Neka je  $\mathbf{r}$  relacija šeme  $R(XY)$  a  $\mathbf{s}$  relacija šeme  $S(Z)$ , gde su  $X$  i  $Y$  disjunktne skupovi atributa, a  $Y$  i  $Z$  unijski kompatibilni skupovi atributa u smislu tipa i značenja. Deljenje relacije  $\mathbf{r}$  sa relacijom  $\mathbf{s}$  se definiše kao

$$\mathbf{r} / \mathbf{s} = \pi_X(\mathbf{r}) - \pi_X((\pi_X(\mathbf{r}) \times \mathbf{s}) - \mathbf{r}) = \mathbf{t}(X)$$

Rezultat operacije deljenja ima sledeće značenje: dobijaju se one vrednosti  $X$  koje se u  $\mathbf{r}$  javljaju u kombinaciji sa svim vrednostima  $Y$  koje se javljaju u  $\mathbf{s}$  kao vrednosti  $Z$ . Drugim rečima, operacija deljenja nam daje one vrednosti  $X$  u  $\mathbf{r}$  koje u kombinaciji sa vrednostima  $Y$  "pokrivaju" skup vrednosti zadat relacijom  $\mathbf{s}$ .

*Primer*

Želimo da iz baze podataka BIBLIOTEKA dobijemo uvid u šifre svih autora koji su u bilo kom svojstvu napisali sve naslove iz oblasti čija je šifra (SIFO) jednaka PJ. Drugim rečima, želimo uvid u one šifre autora (SIFA) koje se u relaciji **je\_autor** javljaju u kombinaciji sa svim šiframa naslova (SIFN) koje odgovaraju naslovima u relaciji **naslov** za koje atribut SIFO ima vrednost PJ.

U ovom primeru, kao i u većini praktičnih situacija primene operacije deljenja, obe ili jedna od relacija koje odgovaraju relacijama **r** i **s** iz definicije se ne nalaze među izvornim relacijama baze podataka, nego ih moramo izvesti.

Relaciju **autor\_naslov** koja odgovara **r** iz definicije dobijamo projekcijom

$$\pi_{SIFA, SIFN} (je\_autor) \rightarrow autor\_naslov (SIFA, SIFN)$$

a relaciju **naslov\_pj** koja odgovara relaciji **s** iz definicije dobijamo projekcijom restrikcije

$$\pi_{SIFN} (\sigma_{SIFO='PJ'} (naslov)) \rightarrow naslov\_pj (SIFN)$$

Za dalji rad trebaće nam i relacija koja odgovara relaciji  $\pi_x(r)$  sa desne strane definicije, a koju dobijamo kao

$$\pi_{SIFA} (je\_autor) \rightarrow autor\_sif (SIFA)$$

Kao rezultat prethodnog dobijaju se relacije, iznad kojih smo radi preglednosti naznačili kojoj komponenti definicije deljenja odgovaraju:

r			s			$\pi_x(r)$					
autor_naslov	(	SIFA SIFN	)	naslov_pj	(	SIFN	)	autor_sif	(	SIFA	)
		-----			----				----		
		AP0 RBP0			PP00				AP0		
		JN0 RBP0			PJC0				JN0		
		DM0 RK00			----				DM0		
		ZP0 PP00							ZP0		
		DM0 PP00							AP1		
		AP1 PJC0							IT0		
		IT0 PP00							----		
		ZP0 PJC0									
		-----									

Dalje treba da formiramo relaciju koja odgovara Dekartovom proizvodu  $\pi_x(\mathbf{r}) \times \mathbf{s}$  iz definicije, a koja odgovara situaciji kada bi svi autori napisali sve posmatrane naslove:

$$\mathbf{autor\_sif} \times \mathbf{naslov\_pj} \rightarrow \mathbf{svi\_sve}$$

Zatim oduzimamo od te relacije relaciju  $\mathbf{autor\_naslov}$ , čime dobijamo relaciju koja odgovara komponenti  $(\pi_x(\mathbf{r}) \times \mathbf{s}) - \mathbf{r}$ :

$$\mathbf{svi\_sve} - \mathbf{autor\_naslov} \rightarrow \mathbf{nije\_autor}$$

Dobija se relacija u kojoj se šifre autora SIFA javljaju u kombinaciji samo sa šiframa SIFN naslova koje oni nisu napisali. Drugim rečima, u relaciji  $\mathbf{nije\_autor}$  javljaju se šifre onih autora koji nisu napisali bar jedan od posmatranih naslova, a nedostaju šifre onih koji su napisali sve te naslove.

Na kraju, preostaje da formiramo relaciju koja sadrži ono što tražimo, šifre svih autora koji su napisali sve posmatrane naslove, i koja odgovara kompletnom izrazu iz definicije:  $\pi_x(\mathbf{r}) - \pi_x((\pi_x(\mathbf{r}) \times \mathbf{s}) - \mathbf{r})$ :

$$\mathbf{autor\_sif} - \pi_{\text{SIFA}}(\mathbf{nije\_autor}) \rightarrow \mathbf{trazene\_sif}$$

Za relacije  $\mathbf{svi\_sve}$ ,  $\mathbf{nije\_autor}$  i  $\mathbf{trazene\_sif}$  dobijamo, uz oznake kojim komponentama definicije deljenja one odgovaraju:

$\pi_x(\mathbf{r}) \times \mathbf{s}$		$(\pi_x(\mathbf{r}) \times \mathbf{s}) - \mathbf{r}$	
$\mathbf{svi\_sve}$	( SIFA SIFN )	$\mathbf{nije\_autor}$	( SIFA SIFN )
AP0	PP00	AP0	PP00
AP0	PJC0	AP0	PJC0
JN0	PP00	JN0	PP00
JN0	PJC0	JN0	JC0
DM0	PP00	DM0	PJC0
DM0	PJC0	AP1	PP00
ZP0	PP00	IT0	PJC0
ZP0	PJC0		
AP1	PP00		
AP1	PJC0		
IT0	PP00		
IT0	PJC0		

$\pi_x(\mathbf{r}) - \pi_x((\pi_x(\mathbf{r}) \times \mathbf{s}) - \mathbf{r})$	
$\mathbf{trazene\_sif}$	( SIFA )
	----
ZP0	
	----



### 5.1.10 Upiti relacione algebre

Pod upitima relacione algebre podrazumevamo izraze sa operacijama relacione algebre koje sastavljamo u cilju dobijanja željenih podataka iz relacione baze podataka.

Iz prethodnih primera, naročito poslednjeg, jasno je od kolikog je značaja da tabelama koje nastaju kao međurezultati dajemo nazive koji govore o njihovom značenju. Ako to nije moguće, poželjno je da sa strane rečima opišemo značenje međurezultata. Isto tako, važno je da notiramo i strukturu svakog međurezultata, odnosno koje attribute sadrži njegova šema relacije.

Za složenije upite vrlo je teško odmah napisati rešenje u formi jednog izraza, pa pribegavamo postupnom rešavanju problema. Postoje dva načina da to ostvarimo, i oba ćemo ilustrovati određenim brojem primera za bazu podataka BIBLIOTEKA, koje ćemo rešavati samo simbolički, bez ispisivanja sadržaja relacija.

### 5.1.11 Postupak sekvence upita

Suština ovog postupka za rešavanje upita je u tome da svaki složeni upit relacione algebre predstavimo kao sekvencu jednostavnijih izraza nad polaznim relacijama i relacijama koje nastaju kao međurezultati.

*Primer*

Treba sastaviti upit koji daje imena svih članova koji drže ili su pozajmljivali bar jednu knjigu:

Na samom početku je važno da uočimo u kojim se sve relacijama nalaze podaci potrebni za rešenje. U ovom slučaju, to su:

- **clan** , sa šiframa i imenima svih članova;
- **drzi** , sa šiframa članova koji drže knjige kod sebe;
- **pozajmica** , sa šiframa članova koji su pozajmljivali knjige;

Pre nego što se upustimo u postupno rešenje problema neophodno je da imamo makar neku opštu predstavu o tome šta sve treba da uradimo. U ovom slučaju, postupak rešavanja problema možemo rečima formulisati na sledeći način:

- projekcijom **drzi** po SIFC dobijamo relaciju sa šiframa članova koji drže knjige;
- projekcijom **pozajmica** po SIFC dobijamo relaciju sa šiframa članova koji su pozajmljivali knjige;
- unijom prethodne dve relacije dobijamo relaciju sa šiframa članova koji drže ili su pozajmljivali knjige;
- prirodnim spajanjem (po SIFC) prethodne relacije sa **clan** dobijamo relaciju koja sadrži samo podatke o članovima koji drže ili su pozajmljivali knjige;
- projekcijom prethodne relacije po IME dobijamo imena članova koji drže ili su pozajmljivali knjige.

Na osnovu toga možemo sastaviti sledeću sekvencu upita, sa šemom relacije i kratkim komentarama za svaki međurezultat:

$\pi_{\text{SIFC}}(\text{drzi}) \rightarrow \text{drz}(\text{SIFC})$  oni koji drže;

$\pi_{\text{SIFC}}(\text{pozajmica}) \rightarrow \text{poz}(\text{SIFC})$  oni koji su pozajmljivali;

$\text{drz} \cup \text{poz} \rightarrow \text{drzpoz}(\text{SIFC})$  oni koji drže ili su pozajmljivali;

$\text{clan} \succ_* \text{drzpoz} \rightarrow \text{svedrzpoz}(\text{SIFC,IME})$  sve o onima koji drže ili su pozajmljivali;

$\pi_{\text{IME}}(\text{svedrzpoz}) \rightarrow \text{rešenje}(\text{IME})$  imena onih koji drže ili su pozajmljivali

Umesto ovako detaljne sekvence, možemo da koristimo korake koji predstavljaju kombinacije pojedinih operacija.

U ovom slučaju, relativno lako možemo da sastavimo složen izraz relacione algebre kojim se ostvaruje traženi upit:

$$\pi_{\text{IME}}(\text{clan} \succ_* (\pi_{\text{SIFC}}(\text{drzi}) \cup \pi_{\text{SIFC}}(\text{pozajmica})))$$

*Primer*

Sastaviti upit koji daje imena autora koji su napisali bar jedan naslov iz oblasti čija je šifra 'PJ'.

Podaci potrebni za ovaj upit nalaze se u tabelama **autor**, **je\_autor** i **naslov**. Do rešenja se dolazi u sledećim koracima:

$\sigma_{\text{SIFO}='PJ'}(\text{naslov}) \rightarrow t1(\text{SIFN}, \text{NAZIV}, \text{SIFO})$  naslovi iz oblasti 'PJ';

$\pi_{\text{SIFN}}(t1) \rightarrow t2(\text{SIFN})$  šifre tih naslova

$\text{je\_autor} \succ_* \langle t2 \rightarrow t3(\text{SIFA}, \text{SIFN}, \text{KOJI})$  autorstva tih naslova

$\pi_{\text{SIFA}}(t3) \rightarrow t4(\text{SIFA})$  šifre tih autora

$\text{autor} \succ_* \langle t4 \rightarrow t5(\text{SIFA}, \text{IME})$  sve o tim autorima

$\pi_{\text{IME}}(t5) \rightarrow \underline{\text{rešenje}}(\text{IME})$  imena tih autora

### Primer

Sastaviti upit koji daje imena članova koji su pročitali sve naslove iz oblasti šifre 'PJ' i ni jedan iz oblasti šifre 'BP' (napomena: za članove koji drže knjige smatra se da su ih pročitali).

Podaci za realizaciju ovog upita nalaze se u više tabela: u **drzi** i **pozajmica** se nalaze šifre članova i šifre knjiga koje su pročitali, u **knjiga** se nalazi veza knjige-naslovi, a u **naslov** veza naslovi-oblasti. Do rešenja se dolazi u sledećim koracima:

$\pi_{\text{SIFC,SIFK}}(\text{drzi}) \cup \pi_{\text{SIFC,SIFK}}(\text{pozajmica}) \rightarrow t1(\text{SIFC,SIFK})$  koji članovi su pročitali koje knjige

$\pi_{\text{SIFC,SIFN}}(\text{knjiga} >_* < t1) \rightarrow t2(\text{SIFC,SIFN})$  koji članovi su pročitali koje naslove

$\pi_{\text{SIFN}}(\sigma_{\text{SIFO='PJ'}}(\text{naslov})) \rightarrow t3(\text{SIFN})$  svi naslovi oblasti 'PJ'

$t2 / t3 \rightarrow t4(\text{SIFC})$  svi članovi koji su pročitali sve naslove oblasti 'PJ'

$\pi_{\text{SIFN}}(\sigma_{\text{SIFO='BP'}}(\text{naslov})) \rightarrow t5(\text{SIFN})$  svi naslovi oblasti 'BP'

$\pi_{\text{SIFK}}(\text{knjiga} >_* < t5) \rightarrow t6(\text{SIFK})$  sve knjige iz oblasti 'BP'

$t1 >_* < t6 \rightarrow t7(\text{SIFC,SIFK})$  koji članovi su čitali koje knjige iz oblasti 'BP'

$t4 - \pi_{\text{SIFC}}(t7) \rightarrow t8(\text{SIFC})$  članovi koji su čitali sve knjige oblasti 'PJ' i ni jednu oblasti 'BP'

$\pi_{\text{IME}}(\text{clan} >_* < t8) \rightarrow \text{rešenje}(\text{IME})$  imena članova koji su čitali sve knjige iz oblasti 'PJ' i ni jednu iz oblasti 'BP'

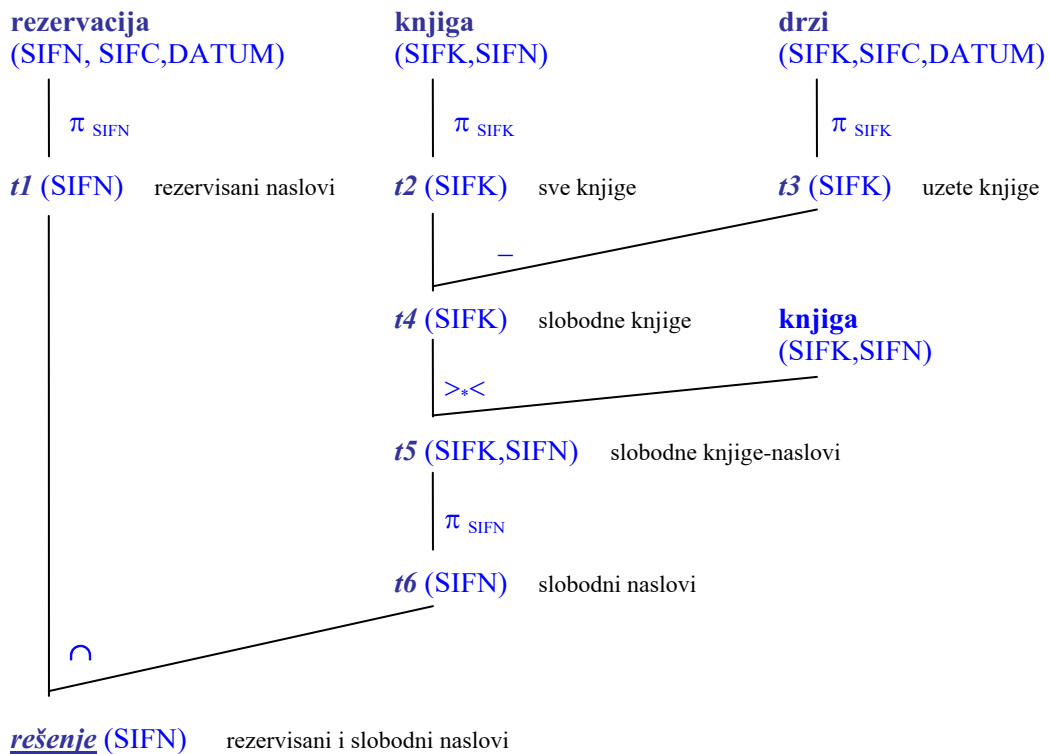
### 5.1.12 Postupak stabla upita

Ovaj postupak se zasniva na konstruisanju stabla izračunavanja vrednosti relacionog izraza, pri čemu polazeći od elementarnih izraza nad izvornim relacijama (listova stabla) formiramo sve složenije i složenije konstrukcije relacione algebre, sve dok ne dođemo do traženog rešenja (korena stabla).

### Primer

Sastaviti upit koji daje šifre naslova koje su članovi rezervisali a u biblioteci ima slobodnih knjiga sa njima.

Neophodni podaci se nalaze u sledećim tabelama: u **rezervacija** se nalaze šifre rezervisanih naslova, u **knjiga** se nalaze podaci o knjigama i šiframa naslova, a u **drzi** šifre knjiga koje su kod članova.



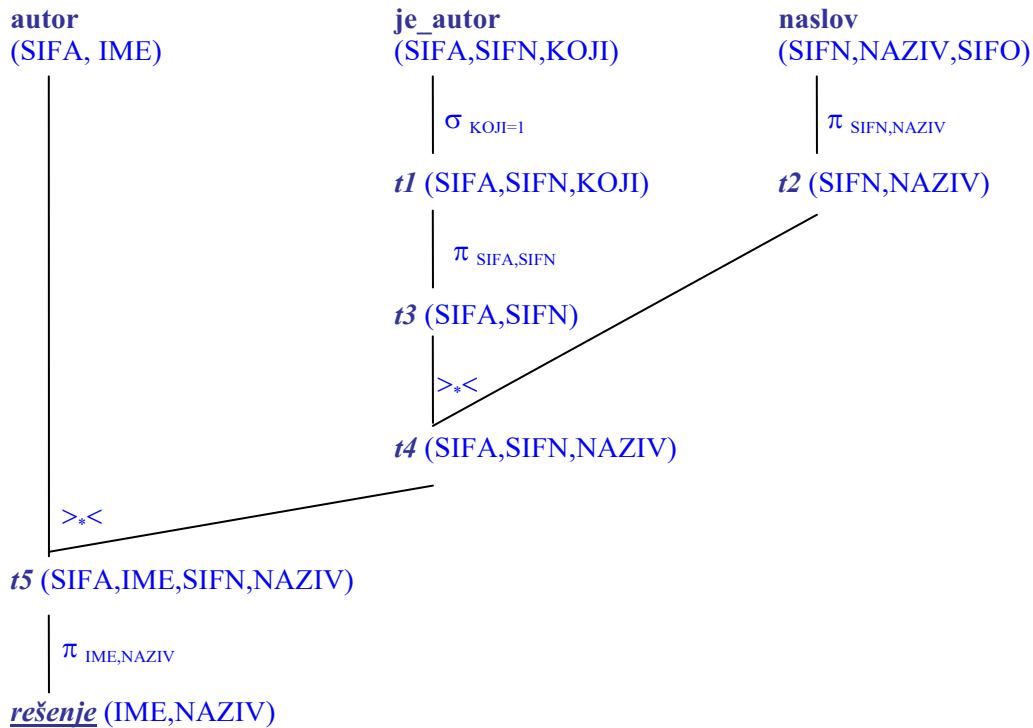
U stablu upita koje se formira na dole relacija **knjiga** se javlja dva puta. Unarne operacije navodimo uz granu koja spaja polazni i rezultatni čvor, a binarne na mestu gde se dva polazna čvora spajaju sa jednim rezultatnim. Za binarnu operaciju razlike moramo naznačiti šta se oduzima od čega, i to radimo tako što oznaku – navidomo bliže grani koja ide od relacije koja se oduzima.



*Primer*

Sastaviti upit koji daje imena autora i nazive naslova koje su napisali kao prvi autori (uzlov da atribut KOJI u **je\_autor** ima vrednost 1).

Podaci potrebni za ovaj upit nalaze se u tabelama **autor**, **je\_autor** i **naslov**.



Postupak stabla upita je ovako pregledan sve dok se stvarno radi o dijagramu i vidu stabla. Onog trenutka kada nastupi situacija višestrukog korišćenja nekog međurezultata dobija se manje pregledan dijagram u vidu mreže. Zbog toga se postupak sekvence operacija smatra pogodnijim.

### 5.1.13 Dodatne operacije spajanja

U cilju objašnjenja i ilustracije dodatnih operacija spajanja neophodan nam je sadržaj relacija **naslov** i **oblast** koji je nešto izmenjen u odnosu na onaj iz priloga A:

<b>naslov</b> ( <u>SIFN</u> NAZIV SIFO )	<b>oblast</b> ( SIFO NAZIV )
-----	-----
RBP0 Relacione baze podataka BP	BP Baze podataka
RK00 Racunarske komunikacije RM	RM Racunarske mreze
PP00 PASCAL programiranje PJ	PJ Programski jezici
PJC0 Programski jezik C PJ	<i>I Internet</i>
<i>UP01 Upravljanje projektima NULL</i>	<i>H Hardver</i>
-----	-----

Izmene (navedene iskošeno) su u sledećem: u **naslov** je dodat jedan red sa nepoznatom oblašću, a u **oblast** dva reda za koje još ne postoje naslovi.

Sa gledišta rezultata koji operacijama relacione algebre kombinuje podatke iz navedene dve relacije možemo da uočimo dva krajnja slučaja:

- Dekartov proizvod: u rezultatu su, u kombinacijama koje imaju i koje nemaju smisla, uvek sadržani svi podaci iz polaznih relacija, odnosno važe odnosi:

$$\pi_{N.SIFN, N.NAZIV, N.SIFO} ( \text{naslov} \times \text{oblast} ) = \text{naslov}$$

$$\pi_{O.SIFO, O.NAZIV} ( \text{naslov} \times \text{oblast} ) = \text{oblast}$$

- Ekvi-spajanje: u rezultatu ne moraju biti sadržani svi podaci iz polaznih relacija:

$$\pi_{N.SIFN, N.NAZIV, N.SIFO} ( \text{naslov} >_{(SIFO)=(SIFO)} < \text{oblast} ) \subseteq \text{naslov}$$

$$\pi_{O.SIFO, O.NAZIV} ( \text{naslov} >_{(SIFO)=(SIFO)} < \text{oblast} ) \subseteq \text{oblast}$$

U slučaju gore navedenih izmenjenih relacija ekvi-spajanje

$$\text{naslov} >_{(SIFO)=(SIFO)} < \text{oblast}$$

daje kao rezultat relaciju

<i>t</i> ( <u>SIFN</u> N.NAZIV N.SIFO O.SIFO O.NAZIV )
-----
RBP0 Relacione baze podataka BP BP Baze podataka
RK00 Racunarske komunikacije RM RM Racunarske mreze
PP00 PASCAL programiranje PJ PJ Programski jezici
PJC0 Programski jezik C PJ PJ Programski jezici
-----

Uočavamo da nedostaju redovi iz obe relacije koji nisu imali sa čime da se spoje - red bez šifre oblasti u **naslov** i dva reda u **oblast** kojima ne odgovara ni jedan naslov:

<b>naslov</b> ( <u>SIFN</u> NAZIV SIFO )	<b>oblast</b> ( SIFO NAZIV )
-----	-----
...	...
<i>UP01 Upravljanje projektima NULL</i>	<i>I Internet</i>
-----	<i>H Hardver</i>
	-----

U praksi situacija sa navedene dve krajnosti kombinovanja tabela nije uvek prihvatljiva. Primera radi, kompletan izveštaj o naslovima trebalo bi da sadrži podatke o svim naslovima i oblastima kojima pripadaju, s tim što bi za naslove bez oblasti tu okolnost nekako trebalo naznačiti. Isto tako, kompletan izveštaj o oblastima i naslovima trebalo bi da sadrži i oblasti iz kojih ne postoji ni jedan naslov, pri čemu bi i tu okolnost nekako trebalo naznačiti. Kao prirodan način te naznake nameće se korišćenje NULL vrednosti za nedostajuće vrednosti atributa, a u cilju obezbeđivanja opisanog načina spajanja relacija relaciona algebra je proširena sa tri dodatne operacije tzv. "vanjskog spajanja".

### Levo vanjsko ekvi-spajanje

Levo vanjsko ekvi-spajanje (simbol  $e>_{(Xp)=(Yp)}<$ ) predstavlja izvedenu, binarnu i posebnu operaciju koja iz dve polazne relacije novu, sa torkama dobijenim u dva koraka:

- prvo se vrši ekvi-spajanje polaznih relacija;
- tako dobijenom rezultatu dodaju se torke iz relacije sa leve strane operatora koje u prvom koraku nisu zadovoljavale uslov ekvi-spajanja, pri čemu se one umesto atributima relacije sa desne strane operatora dopunjuju s desna NULL vrednostima.

Za naš primer izmenjenih relacija **naslov** i oblast levo vanjsko ekvi-spajanje

**naslov**  $e>_{(SIFO)=(SIFO)}<$  **oblast**

daje kao rezultat relaciju

t	(	SIFN	N.NAZIV		N.SIFO	O.SIFO	O.NAZIV	)
		RBP0	Relacione baze podataka	BP	BP		Baze podataka	
		RK00	Racunarske komunikacije	RM	RM		Racunarske mreze	
		PP00	PASCAL programiranje	PJ	PJ		Programski jezici	
		PJC0	Programski jezik C	PJ	PJ		Programski jezici	
+		UP01	Upravljanje projektima	NULL	NULL	NULL	NULL	

U rezultatu su očuvane sve torke relacije **naslov**, odnosno važi

$$\pi_{N.SIFN, N.NAZIV, N.SIFO} ( \text{naslov } e>_{(SIFO)=(SIFO)}< \text{oblast} ) = \text{naslov}$$

Ako pod *null ( relacija )* podrazumevamo funkciju čiji je argument relacija i koja za rezultat daje relaciju iste strukture sa jednom jedinom torkom sastavljenom od NULL vrednosti, i ako su *r* i *s* relacije nad šemama *R(X)* i *S(Y)*, dolazimo do sledeće formalne definicije levog vanjskog ekvi-spajanja

$$r \ e>_{(Xp)=(Yp)}< \ s = r \ e>_{(Xp)=(Yp)}< \ s \cup ( ( r - \pi_X ( r \ e>_{(Xp)=(Yp)}< \ s ) ) \times null ( s ) )$$

gde su *Xp* i *Yp* u opštem slučaju skupovi presečnih atributa.

### Desno vanjsko ekvi-spajanje

Desno vanjsko ekvi-spajanje (simbol  $>_{(Xp)=(Yp)}<e$ ) predstavlja izvedenu, binarnu i posebnu operaciju koja iz dve polazne relacije novu, sa torkama dobijenim u dva koraka:

- prvo se vrši ekvi-spajanje polaznih relacija;
- tako dobijenom rezultatu dodaju se torke iz relacije sa desne strane operatora koje u prvom koraku nisu zadovoljavale uslov ekvi-spajanja, pri čemu se one umesto atributima relacije sa leve strane operatora dopunjuju s leva NULL vrednostima.

Za naš primer desno vanjsko ekvi-spajanje

**naslov**  $>_{(SIFO)=(SIFO)}<e$  **oblast**

daje kao rezultat relaciju

$t$	(	SIFN	N.NAZIV	N.SIFO	O.SIFO	O.NAZIV	)
		RBPO	Relacione baze podataka	BP	BP	Baze podataka	
		RK00	Racunarske komunikacije	RM	RM	Racunarske mreze	
		PP00	PASCAL programiranje	PJ	PJ	Programski jezici	
		PJC0	Programski jezik C	PJ	PJ	Programski jezici	
+		NULL	NULL	NULL	I	Internet	
+		NULL	NULL	NULL	H	Hardver	

U rezultatu su očuvane sve torke relacije **oblast**, odnosno važi

$$\pi_{O.SIFO, O.NAZIV} ( \text{naslov} >_{(SIFO)=(SIFO)}<e \text{ oblast} ) = \text{oblast}$$

Uz sve pretpostavke i napomene navedene za slučaj levog vanjskog spajanja, formalna definicija desnog vanjskog spajanja glasi:

$$r >_{(Xp)=(Yp)}<e s = r >_{(Xp)=(Yp)}<s \cup ( null(r) \times (s - \pi_Y(r >_{(Xp)=(Yp)}<s)) )$$

### Potpuno vanjsko ekvi-spajanje

Potpuno vanjsko ekvi-spajanje (simbol  $e>_{(Xp)=(Yp)}<e$ ) predstavlja izvedenu, binarnu i posebnu operaciju koja iz dve polazne relacije novu, sa torkama dobijenim u tri koraka:

- prvo se vrši ekvi-spajanje polaznih relacija;
- rezultatu prvog koraka dodaju se torke iz relacije sa leve strane operatora koje u prvom koraku nisu zadovoljavale uslov ekvi-spajanja, pri čemu se one umesto atributima relacije sa desne strane operatora dopunjuju s desna NULL vrednostima.
- rezultatu prvog i drugog koraka dodaju se torke iz relacije sa desne strane operatora koje u prvom koraku nisu zadovoljavale uslov ekvi-spajanja, pri čemu se one umesto atributima relacije sa leve strane operatora dopunjuju s leva NULL vrednostima.

Za naš primer potpuno vanjsko ekvi-spajanje

**naslov**  $e>_{(SIFO)=(SIFO)}<e$  **oblast**

daje kao rezultat relaciju

t	(	SIFN	N.NAZIV	N.SIFO	O.SIFO	O.NAZIV	)
		RBP0	Relacione baze podataka	BP	BP	Baze podataka	
		RK00	Racunarske komunikacije	RM	RM	Racunarske mreze	
		PP00	PASCAL programiranje	PJ	PJ	Programski jezici	
		PJC0	Programski jezik C	PJ	PJ	Programski jezici	
+		UP01	Upravljanje projektima	NULL	NULL	NULL	
+		NULL	NULL	NULL	I	Internet	
+		NULL	NULL	NULL	H	Hardver	

U rezultatu su očuvane sve torke relacija **naslov** i **oblast**, odnosno važi

$$\pi_{N.SIFN, N.NAZIV, N.SIFO} ( \mathbf{naslov} \mathrel{e>_{(SIFO)=(SIFO)}<e} \mathbf{oblast} ) = \mathbf{naslov}$$

$$\pi_{O.SIFO, O.NAZIV} ( \mathbf{naslov} \mathrel{e>_{(SIFO)=(SIFO)}<e} \mathbf{oblast} ) = \mathbf{oblast}$$

Uz sve prethodne pretpostavke i napomene, formalna definicija potpunog vanjskog spajanja glasi:

$$\begin{aligned} \mathbf{r} \mathrel{e>_{(Xp)=(Yp)}<e} \mathbf{s} = & \mathbf{r} \mathrel{>_{(Xp)=(Yp)}<} \mathbf{s} \cup ( ( \mathbf{r} - \pi_X ( \mathbf{r} \mathrel{>_{(Xp)=(Yp)}<} \mathbf{s} ) ) \times \mathbf{null} ( \mathbf{s} ) ) \\ & \cup ( \mathbf{null} ( \mathbf{r} ) \times ( \mathbf{s} - \pi_Y ( \mathbf{r} \mathrel{>_{(Xp)=(Yp)}<} \mathbf{s} ) ) ) \end{aligned}$$

### 5.1.14 Dodatna operacija preimenovanja

Binarna operacija relacione algebre nad relacijama koje imaju iste nazive jednog ili više atributa kao neposredni rezultat dala bi relaciju sa ponovljenim nazivima atributa, što relacioni model ne dopušta. U takvim situacijama neophodno je preimenovanje određenih atributa. Tako smo i postupili kod spajanja relacija **naslov** i **oblast**, kada smo nazivima atributa NAZIV i SIFO dodali nadimke ispred, odnosno imali smo

**naslov**  $\succ_{(SIFO)=(SIFO)} < \text{oblast} \rightarrow t(SIFN, N.NAZIV, N.SIFO, O.SIFO, O.NAZIV)$

Prethodna okolnost nameće dopunu relacione algebre elementarnom, unarnom i posebnom operacijom preimenovanja, što je i učinjeno uvođenjem opšte operacije preimenovanja  $\mathcal{R}$  naziva i/ili atributa relacije

$$\mathcal{R}_{y(B1, \dots, Bn)}(x(A1, \dots, An))$$

koja od relacije  $x$  sa nazivima atributa  $A_i$  formira relaciju  $y$  sa nazivima atributa  $B_i$  i istim sadržajem, odnosno

$$\mathcal{R}_{y(B1, \dots, Bn)}(x(A1, \dots, An)) \rightarrow y(B1, \dots, Bn)$$

U slučaju da se naziv relacije ili nazivi pojedinih atributa ne menjaju, za  $y$  i  $B_i$  se u operaciji preimenovanja navode stari nazivi. U najčešćem slučaju kada se menjaju samo nazivi atributa a ne i naziv relacije, novi naziv relacije se izostavlja:

$$\mathcal{R}_{(B1, \dots, Bn)}(x(A1, \dots, An)) \rightarrow x(B1, \dots, Bn)$$

Napomenimo na kraju da operand preimenovanja može biti osim relacije i neki izraz relacione algebre.



*Primeri*

Za ekvi-spajanje relacija **naslov** i **oblast** imali bi

**naslov**  $>_{(SIFO)=(SIFOO)} < \mathcal{R}_{(SIFOO, NAZIVO)} (\mathbf{oblast}) \rightarrow t (SIFN, NAZIV, SIFO, SIFOO, NAZIVO)$

Ako bi želeli pregled šifara i imena radnika i njihovih nadređenih iz relacije **radnik** (SIFR,IME), imali bi

**radnik**  $>_{(SIFR)=(SIFRNAD)} < \mathcal{R}_{(SIFRNAD, IMENAD)} (\mathbf{radnik}) \rightarrow t (SIFR, IME, SIFRNAD, IMENAD)$

### 5.1.15 Dodatna operacija svođenja

Kroz sve prethodne primere uočili smo da pomoću operacija relacione algebre možemo formulisati upite koji se kreću od jednostavnih do veoma složenih. Sa druge strane, postoje određeni upiti koji su jednostavni i značajni a za koje relaciona algebra sa svim svojim do sada izloženim operacijama nema rešenje. Navedimo samo neke:

- broj članova biblioteke: na osnovu sadržaja **clan** dobija se jedna torka sa jednim atributom čija vrednost odgovara broju torki u **clan**;
- maksimalno trajanje pozajmice: na osnovu sadržaja **pozajmica** dobija se jedna torka sa jednim atributom čija vrednost odgovara najvećoj vrednosti atributa DANA u **pozajmica**;
- ukupno trajanje pozajmica po naslovima; na osnovu sadržaja **pozajmica** dobija se onoliko torki koliko ima različitih naslova (vrednosti SIFN) u **pozajmica**, pri čemu svaka torka ima dva atributa - šifru naslova i zbir trajanja svih pozajmica tog naslova.

Karakteristično za sve navedene upite jeste to da se jedna torka rezultata formira na osnovu skupa torki operanda, odnosno da se primenjuje postupak svođenja. U navedenim upitima svođenja se svode na prebrojavanje i nalaženje maksimalne vrednosti, ali su isto tako moguća i druga - nalaženje minimalne vrednosti, zbira, srednje vrednosti i slično. Takve upite, inače česte u praksi, operacije relacione algebre koje smo do sada razmatrali ne omogućavaju.

U cilju prevazilaženja navedenog ograničenja izvršena je dopuna relacije algebre elementarnom, unarnom i posebnom operacijom svođenja (agregacije)

$$G_1, \dots, G_k \text{ } \mathcal{G}_{F_1(D_1), \dots, F_m(D_m)} ( \mathbf{x} (A_1, \dots, A_n) ) \rightarrow \mathbf{t} ( G_1, \dots, G_k, R_1, \dots, R_m )$$

gde su  $G_i$  atributi po kojima se vrši svođenje,  $F_j$  su funkcije svođenja,  $D_k$  su atributi koji se svode, a  $R_j$  su rezultati funkcija svođenja. Pri tome u odnosu na attribute  $A_i$  polazne relacije  $\mathbf{x}$  važi da su atributi  $G_i$  i  $D_k$  iz skupa atributa  $A_i$  relacije i da se ne preklapaju, odnosno

$$\{G_i\} \subseteq \{A_i\} \quad \{D_k\} \subseteq \{A_i\} \quad \{G_i\} \cap \{D_k\} = \emptyset$$

Uvedene su sledeće funkcije svođenja nad jednim šeme relacije atributom:

**COUNT** - broj vrednosti / broj torki;

**SUM** - zbir vrednosti (ima smisla samo za brojni atribut);

**AVG** - prosečna vrednost (ima smisla samo za brojni atribut);

**MAX** - maksimalna vrednost (ima smisla samo za uređeni atribut);

**MIN** - minimalna vrednost (ima smisla samo za uređeni atribut).

Kod sračunavanja navedenih funkcija ignorišu se NULL vrednosti atributa. Izuzetno, funkcija **COUNT** se može upotrebiti i bez atributa kao argumenta i tada vrši brojanje torki u relaciji.

Pojasnimo kako se odvija svođenje na primeru upita koji daje ukupno trajanje pozajmica po naslovima, što se može formulisati kao

$$\text{SIFN } G_{\text{SUM(DANA)}}(\text{pozajmica}) \rightarrow t(\text{SIFN}, \text{DANA})$$

Ako zamislimo da je relacija **pozajmica** uređena po atributu svođenja, odnosno SIFN, jasno se uočava kako se dobija konačni rezultat:

<b>pozajmica ( SIFP SIFC SIFK SIFN DANA )</b>					<b>t ( SIFN DANA )</b>	
1	JJ0	004	PJC0	5	SIFN $G_{\text{SUM(DANA)}}$	PJC0 11
3	JJ1	005	PJC0	6		
2	PP0	007	PP00	2		PP00 9
4	JJ0	008	PP00	7		
5	PP0	002	RBP0	4		RBP0 7
6	JJ1	009	RBP0	3		

U specijalnom slučaju, funkcija svođenja se može koristiti bez i jednog atributa svođenja, odnosno u obliku

$$G_{F1(D1), \dots, Fm(Dm)}(\mathbf{x}(A_1, \dots, A_n)) \rightarrow t(R_1, \dots, R_m)$$

kada se svođenja sprovode nad celom relacijom, odnosno nad svim torkama, pri čemu kao rezultat nastaje jedna torka. Za naš primer maksimalnog trajanja pozajmice odgovarajući upit

$$G_{\text{MAX(DANA)}}(\text{pozajmica}) \rightarrow t(\text{DANA})$$

dao bi kao rezultat jednu torku sa jednom vrednošću - 7.

Kod prethodna dva primera u relaciji rezultata **t** preuzet je iz relacije **pozajmica** naziv atributa **DANA** nad kojim je izvršeno svođenje. To je moguće kod svih funkcija svođenja osim **COUNT** kada nije ni nad jednim atributom nego broji torke u relaciji. U tom slučaju uzima se da rezultujući atributa ima naziv **COUNT** koji je rezervisan (ne sme se koristiti kao naziv izvornog atributa u relaciji). Tako, za naš primer brojanja članova imali bi:

$$G_{\text{COUNT}}(\text{clan}) \rightarrow t(\text{COUNT})$$

Na kraju, ilustrujmo sve navedene dodatne operacije svođenja i preimenovanja jednim primerom.

### Primer

Sastaviti upit koji daje imena članova koji imaju pozajmice bar dva različita naslova iz oblasti šifre 'PJ' i ni jedan iz oblasti 'BP'.

$\pi_{\text{SIFN}} (\sigma_{\text{SIFO}='PJ'} (\text{naslov})) \rightarrow t1 (\text{SIFN})$  svi naslovi oblasti 'PJ'

$\text{pozajmica} >_* < t1 \rightarrow t2 (\text{SIFP}, \text{SIFC}, \text{SIFK}, \text{SIFN}, \text{DANA})$  pozajmice naslova oblasti 'PJ'

$\pi_{\text{SIFC}, \text{SIFN}} (t2) \rightarrow t3 (\text{SIFC}, \text{SIFN})$  koji članovi su pozajmljivali koje naslove oblasti 'PJ'

$\mathcal{R}_{(\text{SIFC}, \text{BROJ})} (\text{SIFC } \mathcal{G}_{\text{COUNT}} (\text{pozajmica})) \rightarrow t4 (\text{SIFC}, \text{BROJ})$  članovi i broj naslova oblasti 'PJ' koje su pozajmljivali

$\pi_{\text{SIFC}} (\sigma_{\text{BROJ} \geq 2} (t4)) \rightarrow t5 (\text{SIFC})$  članovi koji su pozajmljivali bar dva naslova iz oblasti 'PJ'

$\pi_{\text{SIFN}} (\sigma_{\text{SIFO}='BP'} (\text{naslov})) \rightarrow t6 (\text{SIFN})$  svi naslovi oblasti 'BP'

$\text{pozajmica} >_* < t6 \rightarrow t7 (\text{SIFP}, \text{SIFC}, \text{SIFK}, \text{SIFN}, \text{DANA})$  pozajmice naslova oblasti 'PJ'

$\pi_{\text{SIFC}} (t7) \rightarrow t8 (\text{SIFC})$  članovi su pozajmljivali naslove oblasti 'BP'

$t5 - t8 \rightarrow t9 (\text{SIFC})$  članovi koji su pozajmljivali bar dva naslova oblasti 'PJ' i ni jedan naslov oblasti 'BP'

$\pi_{\text{IME}} (\text{clan} >_* < t9) \rightarrow \underline{\text{rešenje}} (\text{IME})$  imena članova koji pozajmljivali bar dva naslova oblasti 'PJ' i ni jedan naslov oblasti 'BP'

## 5.2 Relacioni račun

Relacioni račun kao deklarativni formalni upitni jezik zasnovan na predikatskom računu postoji u dve varijante:

- relacioni račun domena;
- relacioni račun torki.

Suština obe varijante je u tome što navodimo *šta* se želi kao rešenje, za razliku od relacije algebre gde moramo naznačiti *kako* se dolazi do rešenja.

### 5.2.1 Relacioni račun domena

#### *Osnove relacionog računa domena*

Osnove relacionog računa domena čine pojmovi izraza, formule i atoma. Sam naziv ovog računa potiče od okolnosti da varijable koje se javljaju u izrazima uzimaju vrednosti nad domenima atributa relacija.

Izraz koji kao rezultat upita daje skup torki ima opšti oblik:

$$\{ \langle x_1, x_2, \dots, x_m \rangle \mid P(x_1, x_2, \dots, x_m, x_{m+1}, \dots, x_n) \}$$

Varijable  $x_i$  variraju nad domenima atributa relacija. Pri tome su varijable  $x_1 \dots x_m$  slobodne, a dodatne varijable  $x_{m+1} \dots x_n$  su vezane, pošto se u formuli (predikatu)  $P$  javljaju isključivo sa kvantifikatorima  $\exists$  i  $\forall$ . Inače, mogući su i upiti u kojima dodatne vezane varijable nisu neophodne.

Za formulu relacionog računa domena važe sledeća pravila formiranja:

- svaki atom je formula;
- ako je  $P$  formula, to su i  $\neg P$  i  $(P)$ ;
- ako su  $P_1$  i  $P_2$  formule, to su i  $P_1 \wedge P_2$  i  $P_1 \vee P_2$ ;
- ako je  $P(x)$  formula a  $x$  je varijabla, onda su formule i  $\exists x(P(x))$  i  $\forall x(P(x))$ .

Neka je  $\theta$  oznaka za operator poredjenja iz skupa  $\{<, \leq, =, \neq, \geq, >\}$ . Tada su moguće forme atoma relacionog računa domena:

- $\langle x_1, x_2, \dots, x_k \rangle \in r$ , gde je  $r$  relacija sa  $k$  atributa, a varijable  $x_i$  su nad domenima tih atributa;
- $x \theta y$ , gde su  $x$  i  $y$  varijable na koje je primenjiv operator  $\theta$ ;
- $x \theta c$ , gde je  $x$  varijabla,  $c$  konstanta, a nad njima je primenjiv operator  $\theta$ .

Na kraju ovog pregleda osnova relacionog računa domena, uvedimo uobičajenu skraćenu notaciju: Umesto  $\exists x (\exists y (\dots (P(x, y, \dots)) \dots))$  koristićemo  $\exists x, y, \dots (P(x, y, \dots))$ .

*Upiti relacionog računa domena*

Kao primeri poslužiće nam u istom redosledu izlaganja prethodni upiti relacione algebre nad relacionom bazom podataka BIBLIOTEKA (Prilog A).

Pre toga, sastavimo najjednostavniji mogući upit - onaj koji daje sve podatke za naslove:

$$\{ \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \mid \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \in \mathbf{naslov} \}$$



Upit koji daje sve podatke za naslove čija je šifra oblasti 'PJ' glasi:

$$\{ \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \mid \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \in \mathbf{naslov} \wedge \text{SifO} = \text{'PJ'} \}$$

Ovaj primer ilustruje kako se u relacionom računu domena vrši restrikcija relacije. Uz to, za njega je karakteristično da struktura rezultata odgovara strukturi relacije nad kojom je upit, tako da nema potrebe za vezanim varijablama.

Sledeći primer odgovara operaciji projekcije relacije. U pitanju je upit koji daje za naslove njihove nazive i šifre oblasti, odnosno vrši projekciju relacije:

$$\{ \langle \text{Naziv}, \text{SifO} \rangle \mid \exists \text{SifN} ( \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \in \text{naslov} ) \}$$

Uočavamo da ovde struktura rezultata ne odgovara strukturi relacije nad kojom je upit, Pošto nedostaje atribut SifN, uvodimo ga kao vezanu varijablu koja strukturu rezultata “dopunjuje” do strukture relacije nad kojom je upit.

Upit koji je primer kombinacije restrikcije i projekcije relacije daje nazive naslova čija je šifra oblasti ‘PJ’:

$$\{ \langle \text{Naziv} \rangle \mid \exists \text{SifN}, \text{SifO} ( \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \in \text{naslov} \wedge \text{SifO} = \text{'PJ'} ) \}$$

Ovde su bile neophodne dve vezane “dopunjujuće” varijable radi usklađivanja struktura, kao i uslov restrukcije.

Primer koji sledi ilustruje kako se u relacionom računu domena ostvaruje unija relacija i odnosi se na upit koji daje podatke o svim osobama koje su evidentirane u bazi podataka, odnosno u relacijama **autor** i **clan**:

$$\{ \langle \text{SifX}, \text{Ime} \rangle \mid \exists \text{SifA} ( \langle \text{SifA}, \text{Ime} \rangle \in \text{autor} \wedge \text{SifA} = \text{SifX} ) \vee \\ \exists \text{SifC} ( \langle \text{SifC}, \text{Ime} \rangle \in \text{clan} \wedge \text{SifC} = \text{SifX} ) \}$$

Ovde su vezane varijable neophodne zbog različitih naziva atributa koji predstavljaju sifru osobe. U slučaju da su ti atributi istih naziva, recimo SifO, upit bi bio mnogo jednostavniji i glasio bi:

$$\{ \langle \text{SifO}, \text{Ime} \rangle \mid \langle \text{SifO}, \text{Ime} \rangle \in \text{autor} \vee \langle \text{SifO}, \text{Ime} \rangle \in \text{clan} \}$$

Sledeći primer zahteva uniju relacija čije su strukture različite, ali je zato nad podskupovima njihovih atributa unija moguća. Reč je o upitu koji daje šifre knjiga koje su u prometu, odnosno kod članova su ili su bile pozajmljivane:

$$\{ \langle \text{SifK} \rangle \mid \exists \text{SifC}, \text{Datum} ( \langle \text{SifK}, \text{SifC}, \text{Datum} \rangle \in \text{drzi} ) \vee \\ \exists \text{SifP}, \text{SifC}, \text{SifN}, \text{Dana} ( \langle \text{SifP}, \text{SifC}, \text{SifK}, \text{SifN}, \text{Dana} \rangle \in \text{pozajmica} ) \}$$

Za ovaj primer karakteristična su nezavisna “usklađivanja strukture” za dve relacije preko nezavisno odabranih vezanih varijabli.

Naredni primer ilustruje kako se u relacionom računu domena ostvaruje razlika relacija. Upit koji daje šifre članova koji ne drže ni jednu knjigu glasi:

$$\{ \langle \text{SifC} \rangle \mid \exists \text{Ime} ( \langle \text{SifC}, \text{Ime} \rangle \in \text{clan} ) \wedge \neg \exists \text{SifK}, \text{Datum} ( \langle \text{SifK}, \text{SifC}, \text{Datum} \rangle \in \text{drzi} ) \}$$

Ovaj upit se može formulisati rečima kao “Šifre članova koje postoje u **clan** i ne postoje u **drzi**”, pa je potpuno jasna upotreba kvantifikatora “ne postoji ni jedno”.

Navođenje primera koji odgovaraju skupovnim operacijama relacione algebre završićemo sa upitom koji zahteva presek relacija, a daje kao rezultat koji članovi ponovo čitaju koju knjigu, odnosno kombinacije SifC,SifK koje se nalaze i u **drzi** i u **pozajmica**:

$$\{ \langle \text{SifC}, \text{SifK} \rangle \mid \exists \text{Datum} ( \langle \text{SifK}, \text{SifC}, \text{Datum} \rangle \in \text{drzi} ) \wedge \\ \exists \text{SifP}, \text{SifN}, \text{Dana} ( \langle \text{SifP}, \text{SifC}, \text{SifK}, \text{SifN}, \text{Dana} \rangle \in \text{pozajmica} ) \}$$

Na osnovu prethodna tri primera možemo uočiti da se za uniju koristi operator “ili”, za razliku operator “i” sa negacijom  $\exists$ , a preseku operator “i”, kao i to da se za relacije nad kojima je upit uvode (ako je neophodno) nezavisne vezane varijable.

Kao primer koji ilustruje kako se u relacionom računu domena ostvaruje Dekartov proizvod neka posluži upit nad relacijama **naslovi** i **oblast**:

$$\{ \langle \text{SifN}, \text{NazivN}, \text{SifON}, \text{SifO}, \text{NazivO} \rangle \mid \langle \text{SifN}, \text{NazivN}, \text{SifON} \rangle \in \text{naslov} \wedge \langle \text{SifO}, \text{NazivO} \rangle \in \text{oblast} \}$$

Ovde smo prinudeni da koristimo nazive varijabli koji ne odgovaraju nazivima atributa, kako bi razrešili koliziju naziva Naziv i SifO u relacijama **naslov** i **oblast**. Inače, upiti koji se svode samo na Dekartov proizvod a imaju smisla su retki u praksi.

Sastavimo sada upit koji za svaki naslov daje sve podatke o njemu i njegovoj oblasti sa istom strukturom rezultata kao i u prethodnom primeru. Taj upit koji podrazumeva spajanje relacija po jednakosti veznog atributa glasi:

$$\{ \langle \text{SifN}, \text{NazivN}, \text{SifON}, \text{SifO}, \text{NazivO} \rangle \mid \langle \text{SifN}, \text{NazivN}, \text{SifON} \rangle \in \text{naslov} \wedge \langle \text{SifO}, \text{NazivO} \rangle \in \text{oblast} \wedge \text{SifON} = \text{SifO} \}$$

Razlika u odnosu na prethodni primer je jedino u dodatnom uslovu jednakosti veznih atributa relacija.

Posmatrajmo sada upit koji daje samo neke podatke o naslovima i njihovim oblastima. Neka su to šifra naslova, naziv naslova i naziv oblasti, Taj upit glasi:

$$\{ \langle \text{SifN}, \text{Naziv}, \text{NazivO} \rangle \mid \exists \text{SifON} ( \langle \text{SifN}, \text{Naziv}, \text{SifON} \rangle \in \text{naslov} \wedge \exists \text{SifO} ( \langle \text{SifO}, \text{NazivO} \rangle \in \text{oblast} \wedge \text{SifO} = \text{SifON} ) ) \}$$

Dobili smo bitno drugačiji izraz nego za situaciju kada slobodne varijable, odnosno rezultat, obuhvataju sve attribute relacija. Razlog tome je u postojanju vezanih varijabli. Naime, za svako SifON za koje je zadovoljen uslov pripadnosti u **naslov** mora da postoji bar jedno SifO za koje je zadovoljen uslov pripadnosti u **oblast** i koje je jednako tom SifON. Ovo će biti jasnije ako navedemo pravila “vidljivosti” odnosno dostupnosti varijabli u izrazu relacionog računa domena:

- slobodne varijable su dostupne globalno, unutar celog izraza;
- vezane varijable su dostupne samo unutar para zagrada koji sledi iza njihove definicije.

Uz to, treba naglasiti da se u rezultatu upita mogu pojaviti samo slobodne varijable. Inače, u ovom primeru smo drugu vezanu varijablu SifO morali da nazovemo drugačije od prve vezane varijable SifON kako bi razrešili koliziju naziva. Naime, u kontekstu u kome se definiše SifO dostupno je i SifON.

Prethodna situacija se ne javlja samo kod spajanja relacija. Da bi to pojasnili, rešimo jedan od prethodnih primera u izmenjenoj varijanti: umesto šifara članova koji ne drže ni jednu knjigu traže se njihova imena. Odgovarajući upit glasi:

$$\{ \langle \text{Ime} \rangle \mid \exists \text{SifC} ( \langle \text{SifC}, \text{Ime} \rangle \in \text{clan} \wedge \neg \exists \text{SifK}, \text{Datum} ( \langle \text{SifK}, \text{SifC}, \text{Datum} \rangle \in \text{drzi} ) ) \}$$

Slične razlike bi imali i za slučajeve unije i preseka.

Ovaj pregled upita koji je sproveden uporedno sa operacijama relacije algebre objašnjenim u ovom poglavlju završićemo primerom koji odgovara operaciji deljenja i rešava problem iz oblasti “pokrivanja” skupa vrednosti. Naime, želimo da dobijemo šifre svih autora koji su u bilo kom svojstvu napisali sve naslove iz oblasti šifre ‘PJ’. U rešavanju ovog problema koristićemo ekvivalenciju

$$\forall x(P(x)) \Leftrightarrow \neg \exists x(\neg P(x))$$

koja znatno pojednostavljuje rešenje ove klase problema. U skladu sa tim, naš problem “šifre svih autora koji su napisali sve naslove iz oblasti šifre ‘PJ’” preformulisaćemo kao “šifre svih autora za koje ne postoji ni jedan naslov iz oblasti šifre ‘PJ’ koji nisu napisali”. Odgovarajući upit naveden je postupno:

```
{ <SifA> | ∃ Ime ( <SifA,Ime> ∈ autor
      ^
      ¬ ∃ SifN,Naziv,SifO ( <SifN,Naziv,SifO> ∈ naslov ∧ SifO = 'PJ'
      ^
      ¬ ∃ Koji ( <SifA,SifN,Koji> ∈ je_autor ) ) ) }
```

Prvi deo upita odgovara u formulaciji problema tekstu “šifre svih autora”, drugi deo tekstu “za koje ne postoji ni jedan naslov iz oblasti šifre ‘PJ’”, a treći deo tekstu “koji nisu napisali”.



Ovaj kratak osvrt na relacioni račun domena završićemo sa nekoliko upita koji odgovaraju onima iz dela poglavlja posvećenog relacionoj algebri.

### *Primeri*

Upit koji daje imena svih članova koji drže ili su pozajmljivali bar jednu knjigu:

$$\{ \langle \text{Ime} \rangle \mid \exists \text{SifC} ( \langle \text{SifC}, \text{Ime} \rangle \in \text{clan} \wedge \\ ( \exists \text{SifK}, \text{Datum} ( \langle \text{SifK}, \text{SifC}, \text{Datum} \rangle \in \text{drzi} ) \vee \\ \exists \text{SifP}, \text{SifK}, \text{SifN}, \text{Dana} ( \langle \text{SifP}, \text{SifC}, \text{SifK}, \text{SifN}, \text{Dana} \rangle \in \text{pozajmica} ) ) ) \}$$

Sledeći upit daje imena autora koji su napisali bar jedan naslov iz oblasti šifre 'PJ':

$$\{ \langle \text{Ime} \rangle \mid \exists \text{SifA} ( \langle \text{SifA}, \text{Ime} \rangle \in \text{autor} \wedge \\ \exists \text{SifN}, \text{Naziv}, \text{SifO} ( \langle \text{SifN}, \text{Naziv}, \text{SifO} \rangle \in \text{naslov} \wedge \text{SifO} = \text{'PJ'} \wedge \\ \exists \text{Koji} ( \langle \text{SifA}, \text{SifN}, \text{Koji} \rangle \in \text{je\_autor} ) ) ) \}$$

Poslednji upit koji navodimo daje šifre naslova koje su članovi rezervisali a u biblioteci ima slobodnih knjiga sa njima:

$$\{ \langle \text{SifN} \rangle \mid \exists \text{SifC}, \text{Datum} ( \langle \text{SifN}, \text{SifC}, \text{Datum} \rangle \in \text{rezervacija} ) \wedge \\ \exists \text{SifK} ( \langle \text{SifK}, \text{SifN} \rangle \in \text{knjiga} \wedge \\ \neg \exists \text{SifC}, \text{Datum} ( \langle \text{SifK}, \text{SifC}, \text{Datum} \rangle \in \text{drzi} ) ) \}$$

## 5.2.2 Relacioni račun torki

### *Osnove relacionog računa torki*

Osnove relacionog računa torki čine pojmovi izraza, formule i atoma. Za razliku od relacionog računa domena, varijable koje se javljaju u izrazima relacionog računa torki uzimaju vrednosti nad celim torkama relacija.

Izraz koji kao rezultat upita daje skup torki ima opšti oblik:

$$\{ t \mid P(t, s_1, \dots, s_n) \}$$

Varijable  $t$  i  $s_i$  variraju nad torkama relacija (otud i naziv ove varijante relacionog računa). Pri tome je varijabla  $t$  slobodna, a dodatne varijable  $s_1 \dots s_n$  su vezane, pošto se u formuli (predikatu)  $P$  javljaju isključivo sa kvantifikatorima  $\exists$  i  $\forall$ . Inače, mogući su i upiti u kojima dodatne vezane varijable nisu neophodne.

Za formulu relacionog računa torki važe sledeća pravila formiranja:

- svaki atom je formula;
- ako je  $P$  formula, to su i  $\neg P$  i  $(P)$ ;
- ako su  $P_1$  i  $P_2$  formule, to su i  $P_1 \wedge P_2$  i  $P_1 \vee P_2$ ;
- ako je  $P(s)$  formula a  $s$  je varijabla, onda su formule i  $\exists s(P(s))$  i  $\forall s(P(s))$ .

Neka je  $\theta$  osnaka za operator poređenja iz skupa  $\{<, \leq, =, \neq, \geq, >\}$ . Tada su moguće forme atoma relacionog računa torki:

- $u \in r$ , gde je  $r$  relacija, a varijabla  $u$  je nad njenim torkama;
- $u[x] \theta v[y]$ , gde su  $u$  i  $v$  varijable, a  $x$  i  $y$  atributi sadržani u  $u$  i  $v$  i na koje je primenjiv operator  $\theta$ ;
- $u[x] \theta c$ , gde je  $u$  varijabla,  $x$  atribut sadržan u  $u$ , a  $c$  konstanta, pri čemu je nad  $x$  i  $c$  primenjiv operator  $\theta$ .

*Upiti relacionog računa torki*

Kao primeri poslužiće nam u istom redosledu izlaganja prethodni upiti relacione algebre nad relacionom bazom podataka BIBLIOTEKA (Prilog A).

Kao prvo, sastavimo najjednostavniji upit - onaj koji daje sve podatke za naslove:

$$\{ t \mid t \in \text{naslov} \}$$

Upit koji daje sve podatke za naslove čija je šifra oblasti 'PJ' glasi:

$$\{ t \mid t \in \text{naslov} \wedge t[\text{SifO}] = \text{'PJ'} \}$$

Ovaj primer ilustruje kako se u relacionom računu torki vrši restrikcija relacije. Uz to, za njega je karakteristično da struktura rezultata odgovara strukturi relacije nad kojom je upit (zbog  $t \in \text{naslov}$ ), tako da nema potrebe za vezanim varijablama.

Sledeći primer odgovara operaciji projekcije relacije. U pitanju je upit koji daje za naslove njihove nazive i šifre oblasti, odnosno vrši projekciju relacije:

$$\{ t \mid \exists u ( u \in \text{naslov} \wedge t[\text{Naziv}] = u[\text{Naziv}] \wedge t[\text{SifO}] = u[\text{SifO}] ) \}$$

Uočavamo da ovde struktura rezultata ne odgovara strukturi relacije nad kojom je upit (nema  $t \in \text{naslov}$ ) nego je određena pojavljivanjem  $t[\text{Naziv}]$  i  $t[\text{SifO}]$ , tako da je neophodna vezana varijabla  $u$  radi “usklađivanja struktura”.

Upit koji je primer kombinacije restrikcije i projekcije relacije daje nazive naslova čija je šifra oblasti ‘PJ’:

$$\{ t \mid \exists u ( u \in \text{naslov} \wedge t[\text{Naziv}] = u[\text{Naziv}] \wedge u[\text{SifO}] = \text{'PJ'} ) \}$$

Bitna razlika u odnosu na prethodni primer je u uslovu restrikcije vrednosti  $\text{SifO}$ .

Sledeći primer ilustruje kako se ostvaruje unija relacija i odnosi se na upit koji daje podatke o svim osobama koje su evidentirane u bazi podataka:

$$\{ t \mid \exists u ( u \in \text{autor} \wedge t[\text{SifO}] = u[\text{SifA}] \wedge t[\text{Ime}] = u[\text{Ime}] ) \vee \\ \exists u ( u \in \text{clan} \wedge t[\text{SifO}] = u[\text{SifC}] \wedge t[\text{Ime}] = u[\text{Ime}] ) \}$$

Ovde treba uočiti da smo dva puta koristili istu oznaku vezane varijable. To je bilo moguće zato što je opseg vidljivosti svake vezane varijable formula unutar zagrada desno od nje (slično vidljivosti vezanih varijabli kod relacionog računa domena).

Sledi primer koji zahteva uniju relacija čije su strukture različite, ali je zato nad podskupovima njihovih atributa unija moguća. Reč je o upitu koji daje šifre knjiga koje su u prometu, odnosno kod članova su ili su bile pozajmljivane:

$$\{ t \mid \exists u ( u \in \text{drzi} \wedge t[\text{SifK}] = u[\text{SifK}] ) \vee \exists u ( u \in \text{pozajmica} \wedge t[\text{SifK}] = u[\text{SifK}] ) \}$$

Kao i kod relacionog računa domena, ovde unija podrazumeva korišćenje konstrukcije kvantifikatora i operatora u formi “ $\exists x (P(x,...)) \vee \exists y (P(y,...))$ ”.

Kao primer koji ilustruje kako se u relacionom računu torki ostvaruje razlika relacija može da posluži upit koji daje šifre članova koji ne drže ni jednu knjigu:

$$\{ t \mid \exists u ( u \in \text{clan} \wedge t[\text{SifC}] = u[\text{SifC}] \wedge \neg \exists v ( v \in \text{drzi} \wedge u[\text{SifC}] = v[\text{SifC}] ) ) \}$$

Upotreba dve vezane varijable sa ugnježdavanjem je neophodna, što jasno proizilazi iz zahteva “da za svakog člana u **clan** ne sme da postoji ni jedna torka u **drzi** sa njegovom šifrom. Inače, za razliku je karakteristična konstrukcija izraza u formi “ $\exists x (P(x,...)) \wedge \neg \exists y (Q(x,y,...))$ ”.

Navođenje primera koji odgovaraju skupovnim operacijama relacione algebre završićemo sa upitom koji zahteva presek relacija, a daje kao rezultat koji članovi ponovo čitaju koju knjigu, odnosno kombinacije SifC,SifK koje se nalaze i u **drzi** i u **pozajmica**:

$$\{ t \mid \exists u ( u \in \text{drzi} \wedge t[\text{SifC}] = u[\text{SifC}] \wedge t[\text{SifK}] = u[\text{SifK}] \wedge \exists v ( v \in \text{pozajmica} \wedge u[\text{SifC}] = v[\text{SifC}] \wedge u[\text{SifK}] = v[\text{SifK}] ) ) \}$$

Za slučaj preseka karakteristična je konstrukcija “ $\exists x (P(x,...)) \wedge \exists y (Q(x,y,...))$ ”.



Kao primer koji ilustruje kako se u relacionom računu torki ostvaruje Dekartov proizvod neka posluži upit nad relacijama **naslov** i **oblast**:

$$\{t \mid \exists u ( u \in \text{naslov} \wedge t[\text{SifN}] = u[\text{SifN}] \wedge t[\text{NazivN}] = u[\text{Naziv}] \wedge t[\text{SifON}] = u[\text{SifO}] \wedge \\ \exists v ( v \in \text{oblast} \wedge t[\text{SifO}] = v[\text{SifO}] \wedge t[\text{NazivO}] = v[\text{Naziv}] ) ) \}$$

Ovde je karakteristična konstrukcija u formi koja je istovetna onoj za presek.

Upit koji za svaki naslov daje sve podatke o njemu i njegovoj oblasti i koji podrazumeva spajanje relacija po jednakosti veznog atributa SifO glasi:

$$\{t \mid \exists u ( u \in \text{naslov} \wedge t[\text{SifN}] = u[\text{SifN}] \wedge t[\text{NazivN}] = u[\text{Naziv}] \wedge t[\text{SifON}] = u[\text{SifO}] \wedge \exists v ( v \in \text{oblast} \wedge t[\text{SifO}] = v[\text{SifO}] \wedge t[\text{NazivO}] = v[\text{Naziv}] \wedge u[\text{SifO}] = v[\text{SifO}] ) ) \}$$

I ovde su neophodne dve vezane varijable sa ugnježdavanjem kako bi se ostavilo spajanje “svaka torka u **naslov** sa svakom torkom u **oblast**”.

Ako tražimo samo neke podatke o naslovima i njihovim oblastima, na primer šifru naslova, naziv naslova i naziv oblasti, odgovarajući upit glasi:

$$\{t \mid \exists u ( u \in \text{naslov} \wedge t[\text{SifN}] = u[\text{SifN}] \wedge t[\text{NazivN}] = u[\text{Naziv}] \wedge \exists v ( v \in \text{oblast} \wedge t[\text{NazivO}] = v[\text{Naziv}] \wedge u[\text{SifO}] = v[\text{SifO}] ) ) \}$$

Karakteristična forma je ista kao za dekartov proizvod - “ $\exists x (P(x,...)) \wedge \exists y (Q(x,y,...))$ ”.

Razmotrimo sada primer koji odgovara operaciji deljenja i rešava problem iz oblasti “pokrivanja” skupa vrednosti: želimo da dobijemo šifre svih autora koji su u bilo kom svojstvu napisali sve naslove iz oblasti šifre ‘PJ’. U rešavanju ovog problema koristićemo ranije navedenu ekvivalenciju:

$$\forall s (P(s)) \Leftrightarrow \neg \exists s (\neg P(s))$$

U skladu sa tim, naš problem preformulisano glasi “šifre svih autora za koje ne postoji ni jedan naslov iz oblasti šifre ‘PJ’ koji nisu napisali”. Odgovarajući upit relacionog računa torki naveden je postupno:

$$\{ t \mid \exists u ( u \in \text{autor} \wedge t[\text{SifA}] = u[\text{SifA}] \\ \wedge \\ \neg \exists v ( v \in \text{naslov} \wedge v[\text{SifO}] = \text{'PJ'} \\ \wedge \\ \neg \exists x ( x \in \text{je\_autor} \wedge x[\text{SifN}] = v[\text{SifN}] \wedge x[\text{SifA}] = u[\text{SifA}] ) ) ) \}$$

Prvi deo upita odgovara u formulaciji problema tekstu “šifre svih autora”, drugi deo tekstu “za koje ne postoji ni jedan naslov iz oblasti šifre ‘PJ’”, a treći deo tekstu “koji nisu napisali”.

Na kraju, navedimo primere koji odgovaraju završnim primerima relacije algebre i relacionog računa domena:

*Primeri*

Upit koji daje imena svih članova koji drže ili su pozajmljivali bar jednu knjigu:

$$\{ t \mid \exists u ( u \in \text{clan} \wedge t[\text{Ime}] = u[\text{Ime}] \wedge \\ ( \exists v ( v \in \text{drzi} \wedge u[\text{SifC}] = v[\text{SifC}] ) \vee \\ \exists v ( v \in \text{pozajmica} \wedge u[\text{SifC}] = v[\text{SifC}] ) ) ) \}$$

Upit koji daje imena autora koji su napisali bar jedan naslov iz oblasti šifre 'PJ':

$$\{ t \mid \exists u ( u \in \text{autor} \wedge t[\text{Ime}] = u[\text{Ime}] \wedge \\ \exists v ( v \in \text{naslov} \wedge v[\text{SifO}] = \text{'PJ'} \wedge \\ \exists x ( x \in \text{je\_autor} \wedge v[\text{SifN}] = x[\text{SifN}] \wedge u[\text{SifC}] = x[\text{SifC}] ) ) ) \}$$

Poslednji upit daje šifre naslova koje su članovi rezervisali a u biblioteci ima slobodnih knjiga sa njima:

$$\{ t \mid \exists u ( u \in \text{rezervacija} \wedge t[\text{SifN}] = u[\text{SifN}] \wedge \\ \exists v ( v \in \text{knjiga} \wedge u[\text{SifN}] = v[\text{SifN}] \wedge \\ \neg \exists x ( x \in \text{drzi} \wedge v[\text{SifK}] = x[\text{SifK}] ) ) ) \}$$

# 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 prioriteti 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 ( SIFA )</code>	<code>je_autor ( SIFA</code>	<code>SIFN</code>	<code>KOJI )</code>	<code>je_autor2 ( SIFN )</code>
-----	-----	-----	-----	-----
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 <b>SELECT</b> 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 prikazemo 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:

SELECT	SifA AS Autor, COUNT(*) AS Naslova	AP0	1
FROM	Je_Autor	JN0	1
GROUP BY	SifA ;	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):

SELECT	SifC,	JJ0
FROM	Pozajmica	
GROUP BY	SifC	
HAVING	SUM(Dana) > 10 ;	

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

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

### 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 sintakсна definicija rekursivnog 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:

Izraz | ( S-Upit ) <|<=|=|<>|>|=|> Izraz | ( S-Upit )  
 Izraz | ( S-Upit ) [ NOT ] BETWEEN Izraz | ( S-Upit ) AND Izraz | ( S-Upit )  
 Izraz | ( S-Upit ) [ NOT ] IN ( Konstanta ,... | K-Upit )  
 Izraz | ( S-Upit ) <|<=|=|<>|>|=|> ANY ( Konstanta ,... | K-Upit )  
 Izraz | ( S-Upit ) <|<=|=|<>|>|=|> ALL ( Konstanta ,... | K-Upit )

kao i da navedemo sedmu formu prostog predikata:

[ NOT ] EXISTS ( R-Upit )

utvrđuje ishod podupita; ako **R-Upit** kao rezultat daje makar jednan 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 tabelle **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 spovesti 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 formulisati 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.

## 7

***ZAVISNOSTI I NORMALNE FORME***

---

U svim prethodnim poglavljima smo polazili od datih šema relacija i šeme relacije baze podataka BIBLIOTEKA, ne ulazeći u to zašto je njihova struktura baš takva kakva je. Iz primera za upite, kao i sadržaja baze podataka BIBLIOTEKA, nismo mogli da uočimo nikakve nepogodnosti. U tom smislu, mogli smo da zaključimo da su te šeme kao i sama baza podataka "dobre" strukture.

U poglavlju 4 (elementi relacionog modela) uveli smo formalne definicije:

- šema relacije: skup atributa i ograničenja nad njima;
- šema relacije baze podataka: skup šema relacije i ograničenja nad njima.

Jedina ograničenja koja smo tada razmatrali bila su podrazumevana:

- unikatnost atributa u šemi relacije i unikatnost torki u relaciji (proizilazi iz skupovnog karaktera njihovih definicija);
- uslov identifikacionog integriteta za primarni ključ (ni jedan atribut u sastavu primarnog ključa ne sme imati NULL vrednost u relaciji);
- uslov referencijalnog integriteta za strani ključ (svaki strani ključ u relaciji može imati ili vrednost primarnog ključa u ciljnoj relaciji ili NULL vrednost ako je dozvoljena).

Osim poštovanja navedenih ograničenja, sadržaj svake relacije je u pogledu vrednosti ostalih atributa bio potpuno proizvoljan.

Iz prakse su poznati primeri šema relacija koje su "loše" strukture, u tom smislu da se kod relacija nad njima javlja niz nepogodnosti. Vrlo brzo je uočen i osnovni razlog za to: između atributa šeme relacije mogu postojati određene zavisnosti koje ograničavaju vrednosti tih atributa u torkama relacije. Na osnovu tih saznanja, koja su prvo bila opisnog karaktera, vremenom je formulisana posebna oblast istraživanja pod nazivom "Teorija zavisnosti". U okviru te teorije formulisane su i tzv. "normalne forme" kao kriterijumi za valjanost šeme relacije, kao i postupci "normalizacije", odnosno dekompozicije šeme relacije "loše strukture" na dve ili više šema relacija koje su sve u skladu sa željenom "normalnom formom".

## 7.1 Šeme relacija loše strukture

Loše strukture šeme relacije ilustrovaćemo kroz nekoliko pogodno odabranih primera izvedenih iz baze podataka BIBLIOTEKA. Ukazaćemo na nepogodnosti koje se pri tome javljaju

*Primer 1*

Posmatrajmo šemu relacije koja je rezultat nastojanja da se šema relacione baze podataka BIBLIOTEKA smanji tako što će se podaci o autorstvu naslova i autorima evidentirati u jednoj jedinoj relaciji šeme

**AUTOR ( SIFA, IME, SIFN, KOJI )**

u kojoj je primarni ključ SIFA,SIFN , s obzirom da je određeni autor samo jednom autor određenog naslova.

U nekom trenutku u prošlosti, kada baza podataka BIBLIOTEKA nema konačan sadržaj dat u prilogu A, mogući sadržaj relacije **autor** bio bi

autor (	SIFA	IME	SIFN	KOJI )
JN0	J.Nikolic	RBP0	2	
ZP0	Z.Petrovic	PP00	1	

Pretpostavimo sada da u bazu podataka želimo da unesemo podatke o autorstvu naslova "PJC0 Programski jezik C", koji ima dva autora, kako je dato u prilogu A. Nakon toga, imali bi sledeći sadržaj relacije **autor**:

autor (	SIFA	IME	SIFN	KOJI )
JN0	J.Nikolic	RBP0	2	
ZP0	Z.Petrovic	PP00	1	*
AP1	A.Petrovic	PJC0	1	
ZP0	Z.Petrovic	PJC0	2	*

U budućnosti, mogla bi se javiti situacija da unosimo podatke o tome da je za neki naslov autor ili koautor "ZP0 Z.Petrović", nakon čega bi u relaciji **autor** imali 3 puta uneto ime Z.Petrović. U realnim okolnostima, uz imena bi evidentirali i neki dodatni podatak o autorima, i taj podatak bi se za Z.Petrovića takođe javljao 3 puta u relaciji (torke označene sa \*).

Ako bi pokušali da izbegnemo višestruko unošenje imena autora Z.Petrovića tako što bi drugi i svaki naredni put unosili samo vrednost atributa SIFA , a za IME ostavljali NULL vrednost, stvorili bi novi problem: gubitak informacija. Naime, jedan broj upita, na primer "imena svih autora određenog naslova" ili "svi naslovi autora određenog imena" davao bi nepotpune informacije.

Iz navedenog jednostavnog primera, evidentan je osnovni nedostatak relacije **autor**, koji je posledica strukture šeme relacije **AUTOR**:

- redundansa: višestruko ponavljanje istog podatka u relaciji.

Ovaj nedostatak se ogleda kod sva tri vida ažuriranja relacije:

- višestruko unošenje: ime autora moramo uneti onoliko puta koliko je on napisao naslova
- višestruko menjanje: eventualnu promenu imena autora (ili nekog drugog podatka, recimo adrese) moramo izvršiti onoliko puta koliko je on napisao naslova;
- višestruko uklanjanje: ako želimo da potpuno uklonimo podatke o autoru, to moramo učiniti onoliko puta koliko je on napisao naslova.

Uz navedene nedostatke ažuriranja prisutna su i dva drastična nedostatka čiji je karakter egzistencijalne prirode:

- anomalija unošenja: ne možemo uneti podatke o nekom autoru ako pri tome ne unesemo i podatke o bar jednom njegovom naslovu;
- anomalija uklanjanja: uklanjanjem podatka o jedinom naslovu koji je napisao neki autor uklanjamo i podatke o tom autoru.

Svi nedostaci koje smo naveli posledica su okolnosti da pored podrazumevanih ograničenja nad šemom relacije **AUTOR** postoji i jedno dodatno:

- svakoj vrednosti atributa **SIFA** koji je deo primarnog ključa odgovara jedna vrednost atributa **IME**, odnosno kada god se u relaciji **autor** ponovi vrednost atributa **SIFA** mora se ponoviti i njoj odgovarajuća vrednost atributa **IME**.

Za razliku od posmatrane šeme relacije **AUTOR**, par šema relacija **AUTOR** i **JE\_AUTOR** iz šeme relacione baze podataka **BIBLIOTEKA** nema navedene nedostatke: ime svakog autora unosi se samo jednom i nezavisno od podataka o naslovu.



### Primer 2

Neka je u nameri da se podaci o naslovima i oblastima evidentiraju u jednoj relaciji nastala šema relacije NASLOV sledeće strukture:

**NASLOV ( SIFN, NAZIVN, SIFO, NAZIVO )**

Za stanje koje odgovara sadržaju baze podataka BIBLIOTEKA: odgovarajuća relacija **naslov** bi bila:

naslov ( SIFN NAZIVN		SIFO NAZIVO )	
-----			
RBP0	Relacione baze podataka	BP	Baze podataka
RK00	Racunarske komunikacije	RM	Racunarske mreze
PP00	PASCAL programiranje	PJ	Programski jezici *
PJC0	Programski jezik C	PJ	Programski jezici *
-----			

Ako bi uneli podatke o novom naslovu, na primer CPP0 "C++", imali bi 3 puta unete nazive oblasti PJ "Programski jezici". Pokušaj izbegavanja višestrukog unošenja podataka na način kao u prethodnom primeru doveo bi do istog neželjenog efekta - gubitka informacija.

Evidentno je da su u i relaciji **naslov** prisutni nedostaci višestrukog ažuriranja i anomalija unošenja i uklanjanja, kao i anomalije egzistencije. Uzrok tome je postojanje dodatnog ograničenja nad šemom relacije NASLOV:

- svakoj vrednosti atributa SIFO koji nije deo primarnog ključa odgovara jedna vrednost atributa NAZIVO koji takođe nije deo primarnog ključa.

Navedeni nedostaci nisu prisutni u šemi relacione baze podataka BIBLIOTEKA, kod koje umesto jedne šeme relacije NASLOV imamo dve šeme, NASLOV i OBLAST.

### Primer 3

Posmatrajmo šemu relacije koja evidentira podatke o pozajmicama i o tome koje knjige postoje i sa kojim naslovima (posebna šema relacije KNJIGA ne postoji):

#### POZAJMICA ( SIFN, SIFC, DATUM, DANA, SIFK )

Primarni ključ ove šeme je odabran uz pretpostavku da jedan član nikada istog dana ne uzima dva ili više puta knjigu istog naslova.

Za stanje koje odgovara sadržaju baze podataka BIBLIOTEKA uz dodatnu pozajmicu knjige 004 naslova PJC0 od strane člana JJ1 u trajanju od 2 dana, odgovarajuća relacija **pozajmica** bila bi

pozajmica (	SIFN	SIFC	DATUM	DANA	SIFK )
	PJC0	JJ0	01.09.95	5	004 *
	PP00	PP0	02.09.95	2	007
	PJC0	JJ1	03.09.95	6	005
	PP00	JJ0	04.09.95	7	008
	RBP0	PP0	05.09.95	4	002
	PP00	JJ1	06.09.95	3	009
	PJC0	JJ1	07.09.95	2	004 *

I u ovakvoj relaciji **pozajmica** su prisutni svi do sada navedeni nedostaci egzistencijalnog karaktera i neostaci sva tri vida ažuriranja, ali u nešto blažoj formi, i to iz sledećih razloga:

- atribut SIFN je kao primarni ključ u šemi relacije NASLOV kompaktan, pa je višestruko ponavljanje njegove vrednosti prihvatljivo;
- atribut SIFN je kao primarni ključ u šemi relacije NASLOV stabilan; njegova naknadna promena je nemoguća ili vrlo malo verovatna.

I u ovom slučaju uzrok nepogodnosti leži u postojanju jednog dodatnog ograničenja nad šemom relacije

- svakoj vrednosti atributa SIFK koji nije deo primarnog ključa odgovara jedna vrednost atributa SIFN koji je deo primarnog ključa.

Nedostaci koje smo uočili u ovom primeru nisu ispoljeni kod šeme relacione baze podataka BIBLIOTEKA, gde umesto šeme relacije POZAJMICA imamo dve šeme, POZAJMICA i KNJIGA.

### *Zaključak*

Iz prethodna 3 primera, kao i osobina primarnog ključa, možemo da zaključimo sledeće:

- ako nad nekom šemom relacije postoje situacije da jednoj vrednosti nekog atributa koji nije primarni ključ odgovara jedna vrednost nekog drugog atributa, usled višestrukog pojavljivanja ove druge vrednosti javljaće se problemi pri unošenju, menjanju i uklanjanju torki;
- okolnost da nad nekom šemom relacije jednoj vrednosti primarnog ključa odgovara jedna vrednost nekog atributa ne može da dovede do višestrukog pojavljivanja ove druge vrednosti, pošto se svaka vrednost primarnog ključa javlja samo jednom u relaciji;
- šema relacije nad kojom postoje neželjene veze između vrednosti atributa može se zameniti sa više šema relacija kod kojih ti nedostaci nisu prisutni.

Dakle, rešenje prethodno opisanih problema je u dekompoziciji šeme relacije, odnosno rastavljanju šeme relacije na više (za sada dve) šema relacija. Međutim, dekompozicija jedne šeme relacije može se sprovesti na više načina. Koje dekompozicije su loše a koje dobre i zašto?

## 7.2 Loše i dobre dekompozicije

Pre nego što razmotrimo konkretne primere, neophodno je nekoliko napomena opšte prirode:

- dekompozicija je dvojaka: dekomponuje se i šema relacije i relacija nad njom;
- dekompozicija mora biti takva da su u nastalim šemama relacije prisutni svi atributi polazne šeme, pošto bi u suprotnom došlo do gubitka dela podataka;
- dekompozicijom šeme relacije u nastalim šemama relacija se ne mogu pojaviti atributi koji se ne nalaze u polaznoj šemi;
- u najnepovoljnijem slučaju, relacija koja se dekomponuje nije prazna, pa njen sadržaj treba preneti dekompozicijom u nastale relacije.

Kao osnova za primere poslužiće nam šema relacije **POZAJMICA** i relacija **pozajmica** iz prethodnog odeljka. Pri tome, nastojaćemo da eliminišemo atribut **SIFN** iz šeme **POZAJMICA**, kako bi otklonili ranije navedene nepogodnosti.

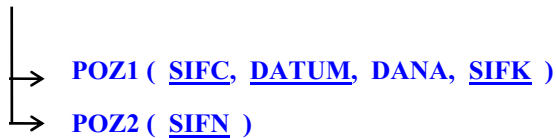
*Primer 1*

Ako za polaznu relaciju

pozajmica ( SIFN SIFC DATUM DANA SIFK )					
PJC0	JJ0	01.09.95	5	004	
PP00	PP0	02.09.95	2	007	
PJC0	JJ1	03.09.95	6	005	
PP00	JJ0	04.09.95	7	008	
RBP0	PP0	05.09.95	4	002	
PP00	JJ1	06.09.95	3	009	
PJC0	JJ1	07.09.95	2	004	

sprovedemo dekompoziciju

**POZAJMICA** ( SIFN, SIFC, DATUM, DANA, SIFK )



odgovarajuće relacije **poz1** i **poz2** dobićemo kao rezultate operacija projekcije po odgovarajućim podskupovima atributa

$\pi_{SIFC,DATUM,DANA,SIFK}(\text{pozajmica}) \rightarrow \text{poz1}(SIFC,DATUM,DANA,SIFK)$

$\pi_{SIFN}(\text{pozajmica}) \rightarrow \text{poz2}(SIFN)$

sa sledećim sadržajem

poz1 ( SIFC DATUM DANA SIFK )					poz2 ( SIFN )
JJ0	01.09.95	5	004		PJC0
PP0	02.09.95	2	007		PP00
JJ1	03.09.95	6	005		RBP0
JJ0	04.09.95	7	008		
PP0	05.09.95	4	002		
JJ1	06.09.95	3	009		
JJ1	07.09.95	2	004		

Uvidom u sadržaj nastalih relacija možemo zaključiti sledeće:

- veza između šifara naslova i ostalih podataka o pozajmicama ne postoji direktno ni u jednoj od nastalih relacija;
- ako pokušamo da rekonstruišemo sve podatke o pozajmicama prirodnim spajanjem relacija **poz1** i **poz2** dobićemo Dekartov proizvod sa 21 torkom (tabela koja sledi), pošto njihove šeme relacija nemaju ni jedan zajednički atribut; od tih torki samo 7 odgovara prvobitnom sadržaju relacije **pozajmica**, a preostalih 14 su suvišne (označeno sa ?):

<b>pozajmica</b> (	<b>SIFN</b>	<b>SIFC</b>	<b>DATUM</b>	<b>DANA</b>	<b>SIFK</b> )
-----					
	PJC0	JJ0	01.09.95	5	004
	PJC0	PP0	02.09.95	2	007 ?
	PJC0	JJ1	03.09.95	6	005
	PJC0	JJ0	04.09.95	7	008 ?
	PJC0	PP0	05.09.95	4	002 ?
	PJC0	JJ1	06.09.95	3	009 ?
	PJC0	JJ1	07.09.95	2	004
	PP00	JJ0	01.09.95	5	004 ?
	PP00	PP0	02.09.95	2	007
	PP00	JJ1	03.09.95	6	005 ?
	PP00	JJ0	04.09.95	7	008
	PP00	PP0	05.09.95	4	002 ?
	PP00	JJ1	06.09.95	3	009
	PP00	JJ1	07.09.95	2	004 ?
	RBP0	JJ0	01.09.95	5	004 ?
	RBP0	PP0	02.09.95	2	007 ?
	RBP0	JJ1	03.09.95	6	005 ?
	RBP0	JJ0	04.09.95	7	008 ?
	RBP0	PP0	05.09.95	4	002
	RBP0	JJ1	06.09.95	3	009 ?
	RBP0	JJ1	07.09.95	2	004 ?
-----					

Za navedeni primer evidentno je da je dekompozicija dovela do gubitka podataka. Konkretno, dobili smo veći broj torki od kojih samo neke odgovaraju stvarnom stanju, ali je pitanje koje? Uzrok gubitka informacija i nastajanja suvišnih n-toski pri prirodnom spajanju je evidentan:

- dekomponovane relacije nemaju ni jedan zajednički atribut (presek njihovih atributa je prazan skup). pa se svaka torka relacije **poz2** spaja sa svakom torkom relacije **poz1**, i obrnuto.

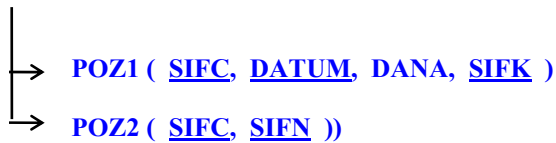
*Primer 2*

Ako za istu relaciju

pozajmica ( SIFN SIFC DATUM DANA SIFK )					
-----					
PJC0	JJ0	01.09.95	5	004	
PP00	PP0	02.09.95	2	007	
PJC0	JJ1	03.09.95	6	005	
PP00	JJ0	04.09.95	7	008	
RBP0	PP0	05.09.95	4	002	
PP00	JJ1	06.09.95	3	009	
PJC0	JJ1	07.09.95	2	004	
-----					

sprovedemo dekompoziciju

POZAJMICA ( SIFN, SIFC, DATUM, DANA, SIFK )



relacije **poz1** i **poz2** dobićemo kao rezultate odgovarajućih operacija projekcije, sa sledećim sadržajem

poz1 ( SIFC DATUM DANA SIFK )					poz2 ( SIFC SIFN )	
-----					-----	
JJ0	01.09.95	5	004		JJ0	PJC0
PP0	02.09.95	2	007		PP0	PP00
JJ1	03.09.95	6	005		JJ1	PJC0
JJ0	04.09.95	7	008		JJ0	PP00
PP0	05.09.95	4	002		PP0	RBP0
JJ1	06.09.95	3	009		JJ1	PP00
JJ1	07.09.95	2	004		-----	
-----						

Ovoga puta, nastale šeme relacija imaju kao presek zajednički atribut SIFC. Za ovaj slučaj možemo zaključiti:

- veza između šifara naslova i ostalih podataka o pozajmicama i dalje ne postoji direktno ni u jednoj od nastalih relacija;
- ako pokušamo da rekonstruišemo sve podatke o pozajmicama prirodnim spajanjem relacija **poz2** i **poz1** dobićemo relaciju sa 14 torki (tabela na sledećoj strani); od tih torki samo 7 odgovara prvobitnom sadržaju relacije **pozajmica**, a preostalih 7 su suvišne.

Gubitak podataka koji je nastupio i pored postojanja zajedničkog atributa između nastalih relacija može se objasniti na sledeći način:

- određene vrednosti zajedničkog atributa SIFC pojavljuju se više od jedanput u obe relacije, pa se neke torke relacije **poz2** spajaju sa više torki relacije **poz1**, i obrnuto.

pozajmica (	SIFN	SIFC	DATUM	DANA	SIFK )
-----					
	PJC0	JJ0	01.09.95	5	004
	PJC0	JJ0	04.09.95	7	008 ?
	PP00	PP0	02.09.95	2	007
	PP00	PP0	05.09.95	4	002 ?
	PJC0	JJ1	03.09.95	6	005
	PJC0	JJ1	06.09.95	3	009 ?
	PJC0	JJ1	07.09.95	2	004
	PP00	JJ0	01.09.95	5	004 ?
	PP00	JJ0	04.09.95	7	008
	RBP0	PP0	02.09.95	2	007 ?
	RBP0	PP0	05.09.95	4	002
	PP00	JJ1	03.09.95	6	005 ?
	PP00	JJ1	06.09.95	3	009
	PP00	JJ1	07.09.95	2	004 ?
-----					

Iz svih prethodnih primera možemo naslutiti osnovni kriterijum za očuvanje podataka pri dekompoziciji šeme relacije i relacije:

- dekompozicija je bez gubitaka podataka ako je reverzibilna, odnosno ako se prirodnim spajanjem nastalih relacija dobija polazna relacija.



*Primer 3*

U prethodnim primerima smo uvideli da je višestruko spajanje torki nastalih relacija osnovni uzročnik gubitka podataka pri dekompoziciji, pa se kao prirodno rešenje tog problema nameće dekompozicija kod koje će se u bar jednoj od nastalih relacija svaka vrednost zajedničkog atributa javljati samo jednom.

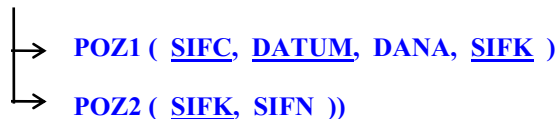
Prethodni uslov se može formulisati na sledeći način: zajednički atribut (u opštem slučaju skup zajedničkih atributa) treba da je kandidat-ključ u bar jednoj od nastalih relacija.

Sledeći takav pristup, za istu polaznu relaciju

pozajmica ( SIFN SIFC DATUM DANA SIFK )					
PJC0	JJ0	01.09.95	5	004	
PP00	PP0	02.09.95	2	007	
PJC0	JJ1	03.09.95	6	005	
PP00	JJ0	04.09.95	7	008	
RBP0	PP0	05.09.95	4	002	
PP00	JJ1	06.09.95	3	009	
PJC0	JJ1	07.09.95	2	004	

dobijamo dekompoziciju koja odgovara onoj u bazi podataka BIBLIOTEKA:

**POZAJMICA ( SIFN, SIFC, DATUM, DANA, SIFK )**



pri čemu se dobija sledeći sadržaj relacija **poz1** i **poz2**:

poz1 ( SIFC DATUM DANA SIFK )				poz2 ( SIFK SIFN )	
JJ0	01.09.95	5	004	004	PJC0
PP0	02.09.95	2	007	007	PP00
JJ1	03.09.95	6	005	005	PJC0
JJ0	04.09.95	7	008	008	PP00
PP0	05.09.95	4	002	002	RBP0
JJ1	06.09.95	3	009	009	PP00
JJ1	07.09.95	2	004		

Za ovaj slučaj važi sledeće:

- veza između šifara naslova i ostalih podataka o pozajmicama i dalje ne postoji direktno ni u jednoj od nastalih relacija;
- ako pokušamo da rekonstruišemo sve podatke o pozajmicama prirodnim spajanjem relacija **poz2** i **poz1** dobićemo 7 torki koje odgovaraju prvobitnom sadržaju relacije **pozajmica** (tabela koja sledi), pošto se svaka torka relacije **poz1** spaja sa tačno jednom torkom relacije **poz2**:

pozajmica (	SIFN	SIFC	DATUM	DANA	SIFK )
	PJC0	JJ0	01.09.95	5	004
	PP00	PP0	02.09.95	2	007
	PJC0	JJ1	03.09.95	6	005
	PP00	JJ0	04.09.95	7	008
	RBP0	PP0	05.09.95	4	002
	PP00	JJ1	06.09.95	3	009
	PJC0	JJ1	07.09.95	2	004

*Zaključak*

Neka su  $R$  i  $r$  šema relacije i relacija koje želimo da dekomponujemo na šeme relacija  $R_1$  i  $R_2$ , odnosno na relacije  $r_1$  i  $r_2$ , pri čemu postoji bar jedan zajednički atribut, odnosno važi  $R_1 \cap R_2 \neq \emptyset$ . Tada važi:

- uslov očuvanja atributa pri dekompoziciji možemo formulisati kao

$$R_1 \cup R_2 = R$$

- očuvanje podataka pri dekompoziciji (reverzibilnost) možemo u notaciji relacione algebre izraziti kao

$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) = r$$

- dekompozicija je sa očuvanjem podataka (bez gubiraka podataka, reverzibilna) ako je skup zajedničkih atributa nastalih relacija kandidat-ključ u bar jednoj od nastalih relacija, odnosno ako je zadovoljeno (značenje simbola " $\rightarrow$ " je "jednoznačno određuje"):

$$R_1 \cap R_2 \rightarrow R_1 \vee R_1 \cap R_2 \rightarrow R_2$$

### 7.3 Osnovi teorije funkcijskih zavisnosti

Do sada smo u više navrata pominjali zavisnosti između atributa šeme relacije koje se svode na to da svakoj vrednosti jednog atributa uvek odgovara samo jedna vrednost drugog atributa. U opštem slučaju, to može važiti i između vrednosti podskupova atributa šeme relacije. Za takvu situaciju se kaže da predstavlja funkcijsku zavisnost, i ona se može formalno definisati na više načina

### 7.3.1 Definicija funkcijske zavisnosti

Neka su:

- $R$  šema relacije nad nekim skup atributa;
- $r$  relacija nad šemom  $R$ ;
- $X, Y, Z$  podskupovi atributa relacije  $R$ ;
- $R[Z]$  šema relacije nad podskupom atributa  $Z$ ;
- $r[Z]$  projekcija relacije  $r$  po podskupu atributa  $Z$ ;
- $t_i$  jedna torke relacije  $r$
- $t_i[Z]$  projekcija jedne torke relacije  $r$  po podskupu atributa  $Z$ .

#### *Definicija 1*

Nad šemom relacije  $R$  postoji funkcijska zavisnost  $X \rightarrow Y$  ako u relaciji  $r$  nad tom šemom uvek važi da je  $r[XY]$  funkcija. Tada kažemo da  $Y$  funkcijski zavisi od  $X$ , odnosno da  $X$  funkcijski uslovljava  $Y$ .

#### *Definicija 2*

Nad šemom relacije  $R$  postoji funkcijska zavisnost  $X \rightarrow Y$  ako u relaciji  $r$  nad tom šemom uvek važi da se svaki element skupa  $r[X]$  preslikava na samo jedan element skupa  $r[Y]$ .

#### *Definicija 3*

Nad šemom relacije  $R$  postoji funkcijska zavisnost  $X \rightarrow Y$  ako u relaciji  $r$  nad tom šemom za bilo koje dve torke  $t_1$  i  $t_2$  za koje je  $t_1[X] = t_2[X]$  uvek važi da je i  $t_1[Y] = t_2[Y]$ . Formalno, ovo možemo iskazati kao:

$$\forall t_1 t_2 ( ( t_1 \in r \wedge t_2 \in r \wedge t_1[X] = t_2[X] ) \Rightarrow t_1[Y] = t_2[Y] )$$

### *Napomene*

U vezi pojma funkcijske zavisnost i prethodnih definicija treba naglasiti:

- Podskupovi atributa  $X$  i  $Y$  iz zavisnost mogu biti bilo koji podskupovi za koje važi  $X \subseteq R$  i  $Y \subseteq R$ , što uključuje i specijalne slučajeve njihove jednakosti sa  $R$ ;
- $X$  i  $Y$  ne moraju biti disjunktni, odnosno može važiti  $X \cap Y \neq \emptyset$ , što uključuje i specijalan slučaj  $X = Y$ ;
- Funkcijska zavisnost  $X \rightarrow Y$  je nešto što postoji "po prirodi stvari" i što proizilazi iz značenja svojstava koja odgovaraju  $X$  i  $Y$  i odnosa između njih; samo na osnovu okolnosti da u nekoj relaciji  $r$  u nekom trenutku svakoj vrednosti  $X$  odgovara samo jedna vrednost  $Y$  ne možemo zaključiti da će to važiti uvek (primer: iz sadržaja relacije **pozajmica** baze podataka BIBLIOTEKA sledi da svakoj vrednosti atributa SIFK odgovara samo jedna vrednost podskupa atributa SIFP,SIFC,SIFN,DANA, ali se to može narušiti kada se ista knjiga ponovo pozajmi);
- Na osnovu trenutnog sadržaja neke relacije  $r$  ne možemo prema prethodnom utvrditi postojanje neke funkcijske zavisnosti, ali zato možemo pouzdano utvrditi da neka pretpostavljena funkcijska zavisnost  $X \rightarrow Y$  ne važi; treba samo da formiramo projekcije  $r[X]$  i  $r[XY]$  i zatim proverimo da li one sadrže isti broj torki. Ako to nije slučaj, ne važi  $X \rightarrow Y$ , pošto bar jednoj vrednosti  $X$  odgovaraju dve ili više vrednosti  $Y$ .

### 7.3.2 Izvođenje funkcijskih zavisnosti

U praksi je vrlo brzo uočeno da na osnovu postojanja nekog skupa funkcijskih zavisnosti mogu da se izvedu dodatne funkcijske zavisnosti. Neka kao ilustracija posluže sledeća dva jednostavna primera:

*Primer*

Posmatrajmo šemu relacije POZAJMICA iz baze podataka BIBLIOTEKA. Okolnost da je atribut SIFP primarni ključ možemo izraziti kao

$$\text{SIFP} \rightarrow \text{SIFC}, \text{SIFK}, \text{SIFN}, \text{DANA}$$

Na osnovu okolnosti da se svaka vrednost atributa SIFP pojavljuje samo jednom u relaciji **pozajmica** možemo zaključiti i da svakoj vrednosti atributa SIFP odgovara samo po jedna vrednost atributa SIFC, SIFK, SIFN i DANA pojedinačno, odnosno da važi

$$\text{SIFP} \rightarrow \text{SIFC} \quad \text{SIFP} \rightarrow \text{SIFK} \quad \text{SIFP} \rightarrow \text{SIFN} \quad \text{SIFP} \rightarrow \text{DANA}$$

*Primer*

Posmatrajmo šemu relacije NASLOV iz jednog od naših ranijih primera:

$$\text{NASLOV} ( \text{SIFN}, \text{NAZIVN}, \text{SIFO}, \text{NAZIVO} )$$

Neka jednoj šifri naslova odgovara samo jedan naziv naslova, a jednoj šifri oblasti samo jedan naziv oblasti, odnosno:

$$\text{SIFN} \rightarrow \text{NAZIVN} \quad \text{SIFO} \rightarrow \text{NAZIVO}$$

Na osnovu prethodnog, nameće se da važi i zavisnost

$$\text{SIFN}, \text{SIFO} \rightarrow \text{NAZIVN}, \text{NAZIVO}$$

*Primer*

Za prethodnu šemu relacije NASLOV važi i to da jednoj šifri naslova odgovara samo jedna šifra oblasti kojoj opet odgovara samo jedan naziv oblasti, što možemo iskazati preko funkcijskih zavisnosti:

$$\text{SIFN} \rightarrow \text{SIFO} \quad \text{SIFO} \rightarrow \text{NAZIVO}$$

Na osnovu toga, jednoj šifri naslova odgovara samo jedan naziv oblasti, odnosno:

$$\text{SIFN} \rightarrow \text{NAZIVO}$$

Mogli bi tako da nabrajamo niz primera i uočavamo niz novih situacija izvođenja funkcijskih zavisnosti, ali sistematičan pristup celoj toj problematici nameće nalaženje odgovora za sledeće pitanje: Postoji li neki konačni i minimalni skup pravila izvođenja novih funkcijskih zavisnosti iz nekog zadatog skupa  $F$  funkcijskih zavisnosti, a da pri tome zadovoljava sledeća dva uslova:

- pouzdanost: ne može se izvesti ni jedna zavisnost koja ne proizilazi iz  $F$  ;
- kompletnost: mogu se izvesti sve zavisnosti koje proizilaze iz  $F$ .

Dokazano je da takav skup postoji. Čine ga tri pravila koja se po autoru nazivaju Armstrongova pravila. U njihovom navođenju polazimo od toga da su  $X, Y, Z$  i  $W$  podskupovi skupa atributa  $R$  koji predstavlja šemu relacije.

### *1. Armstrongovo pravilo: Reflektivnost*

$$Y \subseteq X \subseteq R \Rightarrow X \rightarrow Y$$

Svaki podskup atributa šeme relacije jednoznačno određuje svaki svoj sastavni deo: Pr tome postoje dva specijalna slučaja:

- za  $Y=X$  sledi  $X \rightarrow X$  , odnosno svaki podskup atributa jednoznačno određuje sam sebe;
- za  $Y=\emptyset$  sledi  $X \rightarrow \emptyset$  , odnosno svaki podskup atributa jednoznačno određuje prazan skup atributa.

### *2. Armstrongovo pravilo: Uvećanje*

$$X \rightarrow Y \wedge Z \subseteq W \Rightarrow XW \rightarrow YZ$$

Kod ovog pravila postoje tri specijalna slučaja:

- za  $Z=\emptyset$  sledi  $XW \rightarrow Y$ ;
- za  $Z=W$  sledi  $XW \rightarrow YW$ ;
- za  $Z=W=X$  sledi  $X \rightarrow YX$ .

### *3. Armstrongovo pravilo: Tranzitivnost*

$$X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$$

Specijalni slučajevi za ovo pravilo ne postoje.



Izvođenje funkcijskih zavisnosti samo na osnovu Armstrongovih pravila može biti složeno i nezgrapno, pa se za to koriste i dodatna tri pravila koja slede iz Armstrongovih pravila.

*4. pravilo: Unija*

$$X \rightarrow Y \wedge X \rightarrow Z \Rightarrow X \rightarrow YZ$$

*5. pravilo: Dekompozicija*

$$X \rightarrow Y \wedge Z \subseteq Y \Rightarrow X \rightarrow Z$$

*6. pravilo: Pseudotranzitivnost*

$$X \rightarrow Y \wedge WY \rightarrow Z \Rightarrow XW \rightarrow Z$$

Primenu pravila izvođenja funkcijskih zavisnosti ilustrovaćemo na pogodnom primeru.

*Primer*

Posmatrajmo šemu relacije iz ranijih primera

NASLOV ( SIFN, NAZIVN, SIFO, NAZIVO )

kao i skup funkcijskih zavisnosti:

$$F = \{ \text{SIFN} \rightarrow \text{NAZIVN}, \text{SIFO}, \text{NAZIVO} \quad \text{NAZIVN} \rightarrow \text{SIFO} \quad \text{SIFO} \rightarrow \text{NAZIVO} \}$$

Iz datog skupa  $F$  možemo redom izvršiti sledeća izvođenja:

po pravilu reflektivnosti:

$$\text{SIFN} \rightarrow \text{NAZIVN} \quad \text{SIFN} \rightarrow \text{SIFO}, \text{NAZIVO}$$

po pravilu uvećanja:

$$\text{SIFO} \rightarrow \text{SIFO}, \text{NAZIVO} \quad \text{SIFO}, \text{NAZIVN} \rightarrow \text{NAZIVO}, \text{NAZIVN}$$

po pravilu tranzitivnosti:

$$\text{NAZIVN} \rightarrow \text{NAZIVO}$$

### 7.3.3 Zatvarač skupa funkcijskih zavisnosti

U prethodnom primeru izveli smo svega nekoliko funkcijskih zavisnosti. U opštem slučaju, broj funkcijskih zavisnosti izveden iz nekog datog skupa funkcijskih zavisnosti  $F$  može biti vrlo veliki, pogotovo kada su u pitanju zavisnosti izvedene po pravilu reflektivnosti. Pitanje koje se pri tome nameće je sledeće:

- da li je broj izvedenih zavisnosti konačan, odnosno da li ikada nastupa situacija kada više ne možemo da izvedemo ni jednu novu funkcijsku zavisnost iz  $F$ ?

Odgovor na prethodno pitanje je potvrđan. Konačnost ukupnog broja izvedenih funkcijskih zavisnosti proizilazi iz konačnosti broja atributa koji se javljaju u  $F$  i okolnosti da su najveće moguće leve i desne strane izvedenih zavisnosti jednake skupu svih tih atributa.

U praksi, postupak izvođenja funkcijskih zavisnosti iz datog skupa zavisnosti  $F$  tekao bi na sledeći način: Prvo bi formirali novi skup  $F^+$  koji bi kao početni sadržaj imao sve zavisnosti iz  $F$ . Zatim bi izvodili jednu po jednu novu funkcijsku zavisnost i dodavali je u  $F^+$  samo ako se već ne nalazi u njemu. U jednom trenutku, nastupila bi situacija kada ne možemo da izvedemo ni jednu novu funkcijsku zavisnost koja se već ne nalazi u  $F^+$ . Takav skup  $F^+$  nazivamo zatvaračem skupa funkcijskih zavisnosti  $F$ .

#### *Definicija*

Zatvarač  $F^+$  skupa funkcijskih zavisnosti  $F$  zadatog nad skupom atributa  $R$  je minimalni skup funkcijskih zavisnosti koji zadovoljava sledeća dva uslova:

- zadati skup  $F$  je podskup  $F^+$ , odnosno  $F \subseteq F^+$ ;
- primenom Armstrongovih pravila na funkcijske zavisnosti u  $F^+$  ne može se izvesti ni jedna zavisnost koja se već ne nalazi u  $F^+$ , odnosno važi:

$$\neg \exists (X \rightarrow Y) (XY \subseteq R \wedge F^+ \Rightarrow (X \rightarrow Y) \wedge (X \rightarrow Y) \notin F^+)$$

*Primer*

Posmatrajmo šemu relacije iz prethodnog primera

NASLOV ( SIFN, NAZIVN, SIFO, NAZIVO )

i skup funkcijskih zavisnosti

$$F = \{ \text{SIFN} \rightarrow \text{NAZIVN}, \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{NAZIVO} \rightarrow \text{SIFO} \}$$

Primenom pravila izvođenja dobija se kao zatvarač  $F^+$  (navedeno od gore na dole prvo sa leve strane a zatim sa desne):

SIFN $\rightarrow \emptyset$	NAZIVO $\rightarrow \emptyset$
SIFN $\rightarrow$ SIFN	NAZIVO $\rightarrow$ SIFO
SIFN $\rightarrow$ NAZIVN	NAZIVO $\rightarrow$ NAZIVO
SIFN $\rightarrow$ SIFO	NAZIVO $\rightarrow$ SIFO, NAZIVO
SIFN $\rightarrow$ NAZIVO	SIFN, NAZIVN $\rightarrow \emptyset$
SIFN $\rightarrow$ SIFN, NAZIVN	SIFN, NAZIVN $\rightarrow$ SIFN
SIFN $\rightarrow$ SIFN, SIFO	SIFN, NAZIVN $\rightarrow$ NAZIVN
SIFN $\rightarrow$ SIFN, NAZIVO	SIFN, NAZIVN $\rightarrow$ SIFO
SIFN $\rightarrow$ NAZIVN, SIFO	SIFN, NAZIVN $\rightarrow$ NAZIVO
SIFN $\rightarrow$ NAZIVN, NAZIVO	SIFN, NAZIVN $\rightarrow$ SIFN, NAZIVN
SIFN $\rightarrow$ SIFN, NAZIVN, SIFO	SIFN, NAZIVN $\rightarrow$ SIFN, SIFO
SIFN $\rightarrow$ SIFN, NAZIVN, NAZIVO	SIFN, NAZIVN $\rightarrow$ SIFN, NAZIVO
SIFN $\rightarrow$ SIFN, SIFO, NAZIVO	SIFN, NAZIVN $\rightarrow$ NAZIVN, SIFO
SIFN $\rightarrow$ NAZIVN, SIFO, NAZIVO	SIFN, NAZIVN $\rightarrow$ NAZIVN, NAZIVO
SIFN $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO	SIFN, NAZIVN $\rightarrow$ SIFO, NAZIVO
NAZIVN $\rightarrow \emptyset$	SIFN, NAZIVN $\rightarrow$ SIFN, NAZIVN, SIFO
NAZIVN $\rightarrow$ NAZIVN	SIFN, NAZIVN $\rightarrow$ SIFN, NAZIVN, NAZIVO
SIFO $\rightarrow \emptyset$	SIFN, NAZIVN $\rightarrow$ SIFN, SIFO, NAZIVO
SIFO $\rightarrow$ SIFO	SIFN, NAZIVN $\rightarrow$ NAZIVN, SIFO, NAZIVO
SIFO $\rightarrow$ NAZIVO	SIFN, NAZIVN $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO
SIFO $\rightarrow$ SIFO, NAZIVO	...

SIFN,SIFO→∅	SIFO,NAZIVO→∅
SIFN,SIFO→SIFN	SIFO,NAZIVO→SIFO
SIFN,SIFO→NAZIVN	SIFO,NAZIVO→NAZIVO
SIFN,SIFO→SIFO	SIFN,NAZIVO→SIFO,NAZIVO
SIFN,SIFO→NAZIVO	SIFN,NAZIVN,SIFO→∅
SIFN,SIFO→SIFN,NAZIVN	SIFN,NAZIVN,SIFO→SIFN
SIFN,SIFO→SIFN,SIFO	SIFN,NAZIVN,SIFO→NAZIVN
SIFN,SIFO→SIFN,NAZIVO	SIFN,NAZIVN,SIFO→SIFO
SIFN,SIFO→NAZIVN,SIFO	SIFN,NAZIVN,SIFO→NAZIVO
SIFN,SIFO→NAZIVN,NAZIVO	SIFN,NAZIVN,SIFO→SIFN,NAZIVN
SIFN,SIFO→SIFO,NAZIVO	SIFN,NAZIVN,SIFO→SIFN,SIFO
SIFN,SIFO→SIFN,NAZIVN,SIFO	SIFN,NAZIVN,SIFO→SIFN,NAZIVO
SIFN,SIFO→SIFN,NAZIVN,NAZIVO	SIFN,NAZIVN,SIFO→NAZIVN,SIFO
SIFN,SIFO→SIFN,SIFO,NAZIVO	SIFN,NAZIVN,SIFO→NAZIVN,NAZIVO
SIFN,SIFO→SIFN,NAZIVN,SIFO,NAZIVO	SIFN,NAZIVN,SIFO→SIFO,NAZIVO
SIFN,NAZIVO→∅	SIFN,NAZIVN,SIFO→SIFN,NAZIVN,SIFO
SIFN,NAZIVO→SIFN	SIFN,NAZIVN,SIFO→SIFN,NAZIVN,NAZIVO
SIFN,NAZIVO→NAZIVN	SIFN,NAZIVN,SIFO→SIFN,SIFO,NAZIVO
SIFN,NAZIVO→SIFO	SIFN,NAZIVN,SIFO→SIFN,NAZIVN,SIFO,NAZIVO
SIFN,NAZIVO→NAZIVO	SIFN,NAZIVN,NAZIVO→∅
SIFN,NAZIVO→SIFN,NAZIVN	SIFN,NAZIVN,NAZIVO→SIFN
SIFN,NAZIVO→SIFN,SIFO	SIFN,NAZIVN,NAZIVO→NAZIVN
SIFN,NAZIVO→SIFN,NAZIVO	SIFN,NAZIVN,NAZIVO→SIFO
SIFN,NAZIVO→NAZIVN,SIFO	SIFN,NAZIVN,NAZIVO→NAZIVO
SIFN,NAZIVO→NAZIVN,NAZIVO	SIFN,NAZIVN,NAZIVO→SIFN,NAZIVN
SIFN,NAZIVO→SIFO,NAZIVO	SIFN,NAZIVN,NAZIVO→SIFN,SIFO
SIFN,NAZIVO→SIFN,NAZIVN,SIFO	SIFN,NAZIVN,NAZIVO→NAZIVN,SIFO
SIFN,NAZIVO→SIFN,NAZIVN,NAZIVO	SIFN,NAZIVN,NAZIVO→NAZIVN,NAZIVO
SIFN,NAZIVO→SIFN,SIFO,NAZIVO	SIFN,NAZIVN,NAZIVO→SIFO,NAZIVO
SIFN,NAZIVO→SIFN,NAZIVN,SIFO,NAZIVO	SIFN,NAZIVN,NAZIVO→SIFN,NAZIVN,SIFO
NAZIVN,SIFO→∅	SIFN,NAZIVN,NAZIVO→SIFN,NAZIVN,NAZIVO
NAZIVN,SIFO→NAZIVN	SIFN,NAZIVN,NAZIVO→SIFN,SIFO,NAZIVO
NAZIVN,SIFO→SIFO	SIFN,NAZIVN,NAZIVO→NAZIVN,SIFO,NAZIVO
NAZIVN,SIFO→NAZIVO	SIFN,NAZIVN,NAZIVO→SIFN,NAZIVN,SIFO,NAZIVO
NAZIVN,SIFO→NAZIVN,SIFO	SIFN,SIFO,NAZIVO→∅
NAZIVN,SIFO→NAZIVN,NAZIVO	SIFN,SIFO,NAZIVO→SIFN
NAZIVN,SIFO→SIFO,NAZIVO	SIFN,SIFO,NAZIVO→NAZIVN
NAZIVN,SIFO→NAZIVN,SIFO,NAZIVO	SIFN,SIFO,NAZIVO→SIFO
NAZIVN,NAZIVO→∅	SIFN,SIFO,NAZIVO→NAZIVO
NAZIVN,NAZIVO→NAZIVN	SIFN,SIFO,NAZIVO→SIFN,NAZIVN
NAZIVN,NAZIVO→SIFO	SIFN,SIFO,NAZIVO→SIFN,SIFO
NAZIVN,NAZIVO→NAZIVO	SIFN,SIFO,NAZIVO→SIFN,NAZIVO
NAZIVN,NAZIVO→NAZIVN,SIFO	SIFN,SIFO,NAZIVO→NAZIVN,SIFO
NAZIVN,NAZIVO→NAZIVN,NAZIVO	SIFN,SIFO,NAZIVO→NAZIVN,NAZIVO
NAZIVN,NAZIVO→SIFO,NAZIVO	SIFN,SIFO,NAZIVO→SIFO,NAZIVO
NAZIVN,NAZIVO→NAZIVN,SIFO,NAZIVO	SIFN,SIFO,NAZIVO→SIFN,NAZIVN,SIFO
	SIFN,SIFO,NAZIVO→SIFN,NAZIVN,NAZIVO
	...

...

$SIFN, SIFO, NAZIVO \rightarrow SIFN, SIFO, NAZIVO$   
 $SIFN, SIFO, NAZIVO \rightarrow NAZIVN, SIFO, NAZIVO$   
 $SIFN, SIFO, NAZIVO \rightarrow SIFN, NAZIVN, SIFO, NAZIVO$   
 $NAZIVN, SIFO, NAZIVO \rightarrow \emptyset$   
 $NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN$   
 $NAZIVN, SIFO, NAZIVO \rightarrow SIFO$   
 $NAZIVN, SIFO, NAZIVO \rightarrow NAZIVO$   
 $NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN, SIFO$   
 $NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN, NAZIVO$   
 $NAZIVN, SIFO, NAZIVO \rightarrow SIFO, NAZIVO$   
 $NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN, SIFO, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow \emptyset$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, NAZIVN$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, SIFO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN, SIFO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFO, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, NAZIVN, SIFO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, NAZIVN, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, SIFO, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow NAZIVN, SIFO, NAZIVO$   
 $SIFN, NAZIVN, SIFO, NAZIVO \rightarrow SIFN, NAZIVN, SIFO, NAZIVO$

Dobili smo da je jedini kandidat-ključ date šeme relacije SIFN.

Rezultat je iznenađujuće obiman – za slučaj 3 zavisnosti nad šemom relacije sa 4 atributa dobili smo da zatvarač  $F^+$  sadrži 159 zavisnosti. Ako bi iz svake grupe zavisnosti  $X \rightarrow Y$  sa istom desnom stranom  $X$  uzeli samo onu sa maksimalnim  $Y$  (sve druge iz grupe su izvodive iz nje po Armstrongovim pravilima) dobili bi netrivialni zatvarač  $F_n^+$  daleko umerenijeg obima, odnosno sa svega 15 zavisnosti:

SIFN $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO	NAZIVN, SIFO $\rightarrow$ NAZIVN, SIFO, NAZIVO
NAZIVN $\rightarrow$ NAZIVN	NAZIVN, NAZIVO $\rightarrow$ NAZIVN, SIFO, NAZIVO
SIFO $\rightarrow$ SIFO, NAZIVO	SIFO, NAZIVO $\rightarrow$ SIFO, NAZIVO
NAZIVO $\rightarrow$ SIFO, NAZIVO	SIFN, NAZIVN, SIFO $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO
SIFN, NAZIVN $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO	SIFN, NAZIVN, NAZIVO $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO
SIFN, SIFO $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO	SIFN, SIFO, NAZIVO $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO
SIFN, NAZIVO $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO	NAZIVN, SIFO, NAZIVO $\rightarrow$ NAZIVN, SIFO, NAZIVO
SIFN, NAZIVN, SIFO, NAZIVO $\rightarrow$ SIFN, NAZIVN, SIFO, NAZIVO	

### 7.3.4 Zatvarač skupa atributa i njegova primena

Prethodni primer je dobra ilustracija za kompleksnost zatvarača skupa funkcijskih zavisnosti. Možemo samo da zamislimo šta bi bilo u slučaju skupa od recimo 5 atributa i 4 zavisnosti u  $F$ .

Primer koji smo uradili otkriva i to da je pristup traženju izvedenih funkcijskih zavisnosti kombinatornog karaktera. Naime, mi formiramo redom kombinacije od jednog, dva, tri i četiri atributa sa leve strane i onda za takvu levu stranu izvodimo sve moguće zavisnosti sa različitim desnim stranama. Sama primena pravila izvođenja pri tom postupku je vrlo obiman i zamoran posao, a najgore je što odsustvo neke zavisnosti može da znači ne samo da ona stvarno ne postoji u  $F^+$ , nego i to da ona postoji a da nismo uspeli da je izvedemo. Usled toga pravila izvođenja imaju uglavnom teoretski značaj, u smislu da služe za dokazivanje valjanosti raznih algoritama u teoriji zavisnosti i nodmalnih formi.

Nalaženje  $F^+$  bi bilo znatno olakšano kada bi nekim algoritmom za zadati skup atributa  $X$  sa leve strane funkcijske zavisnosti uspeli da odredimo skup  $Y$  svih atributa sa desne strane, odnosno sve attribute koje jednoznačno određuje  $X$ .

Takav algoritam postoji i zasnovan je na pravilima izvođenja funkcijskih zavisnosti.

#### *Definicija*

Neka je  $R$  skup atributa,  $X$  neki njegov podskup, a  $F$  skup funkcijskih zavisnosti nad  $R$ . Zatvarač  $X^+$  skupa atributa  $X$  čini skup atributa  $Y$  koji odgovara desnoj strani zavisnosti  $X \rightarrow Y$  u  $F^+$  sa maksimalnim  $Y$  (za sve ostale zavisnosti u  $F^+$  oblika  $X \rightarrow Z$  važi  $Z \subset Y$ ).

#### *Primer*

Uvidom u funkcijske zavisnosti u  $F^+$  iz prethodnog primera dobijamo:

$$\text{SIFN}^+ = \text{SIFN}, \text{NAZIVN}, \text{SIFO}, \text{NAZIVO} \quad (\text{NAZIVO}, \text{SIFO})^+ = \text{NAZIV}, \text{SIFO}$$

Uočavamo da zatvarač nekog skupa atributa uvek sadrži i taj skup atributa, što je posledica reflektivnosti.

U daljem tekstu u ovom poglavlju izložićemo niz algoritama koristeći kao notaciju pseudo-jezik opisan u prilogu C.



Sledeći algoritam na osnovu datog skupa atributa  $X$  i skupa funkcijskih zavisnosti  $F$  nalazi skup atributa  $Y$  koji predstavlja zatvarač  $X^+$ .

```

ZatvaracX ( > X, > F )
: Inicijalizacija
: : XzatStaro =  $\emptyset$ 
: : Xzat      = X
: Ponavljanje do prolaza bez izmene zatvaraca
: : DO WHILE ( Xzat  $\neq$  XzatStaro )
: : | Pamcenje prethodnog zatvaraca
: : | : XzatStaro = Xzat
: : | Provera za sve zavisnosti
: : | : DO FOR EACH (  $Z \rightarrow W$  IN F )
: : | : | Provera da li je Z sadrzano u zatvaracu
: : | : | : IF (  $Z \subseteq$  Xzat )
: : | : | : Uvecanje zatvaraca sa W
: : | : | : Xzat := Xzat  $\cup$  W
: RETURN Xzat

```

Dokaz ispravnosti ovog algoritma zasniva se na Armstrongovim aksiomima.

*Primer*

Posmatrajmo šemu relacije

NASLOV ( SIFN, NAZIVN, SIFO, NAZIVO )

i skup funkcijskih zavisnosti

$$F = \{ \text{SIFN} \rightarrow \text{SIFN}, \text{NAZIVN}, \text{SIFO}, \text{NAZIVO} \quad \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{NAZIVO} \rightarrow \text{SIFO} \}$$

Na osnovu prethodnog algoritma dobijamo, pri čemu smo međurezultate razdvojili simbolom  $\mid$  a sa  $\surd$  označili konačni rezultat:

$$\text{SIFN}^+ = \text{SIFN} \mid \text{SIFN}, \text{NAZIVN}, \text{SIFO}, \text{NAZIVO} \surd$$

$$\text{SIFO}^+ = \text{SIFO} \mid \text{SIFO}, \text{NAZIVO} \surd$$

$$\text{NAZIVN}^+ = \text{NAZIVN} \surd$$

$$(\text{SIFO}, \text{NAZIVN})^+ = \text{SIFO}, \text{NAZIVN} \mid \text{SIFO}, \text{NAZIVN}, \text{NAZIVO} \surd$$

Dobili smo da je SIFN kandidat-ključ, pošto je njegov zatvarač cela šema relacije.

Algoritam *ZatvaracX* je sigurno najznačajniji algoritam u oblasti funkcijske zavisnosti podataka. Na njemu se zasniva većina drugih algoritama iz te oblasti, od kojih su neki navedeni u ovom poglavlju a neki u prilogu E. Pri tome ćemo smatrati da su nam na raspolaganju dve pomoćne funkcije:

- *SkupPodskupova* ( $> X, > n$ ) : funkcija koja kao rezultat daje skup čiji su elementi svi podskupovi formirani od  $n$  elemenata skupa  $X$  (u praksi, to će biti podskupovi od 1, 2 i više atributa šeme relacije  $R$ );
- *BrojElemenata* ( $> X$ ) : funkcija koja kao rezultat daje broj elemenata nekog skupa (u praksi, to će najčešće biti broj atributa u šemi relacije  $R$ ).

Prva primena algoritma *ZatvaracX* je u određivanju da li se neka funkcijska zavisnost  $X \rightarrow Y$  nalazi u zatvaraču  $F^+$  nekog skupa funkcijskih zavisnosti  $F$ . Ako dobijemo da važi  $Y \subseteq X^+$ , iz toga sledi  $(X \rightarrow Y) \in F^+$ , na osnovu Armstrongovih pravila reflektivnosti i tranzitivnosti. Sledeći algoritam na osnovu zadate funkcijske zavisnosti  $X \rightarrow Y$  i skupa funkcijskih zavisnosti  $F$  utvrđuje da li se ta zavisnosti nalazi u  $F^+$  ili ne:

```

ElementPlus (  $X \rightarrow Y$ ,  $F$  )
: Inicijalizacija
: :  $YuF = FALSE$ 
: Provera da li je  $Y$  sadržano u zatvaracu
: : IF (  $Y \subseteq \text{ZatvaracX} ( X, F )$ 
: : |  $YuF = TRUE$ 
: RETURN  $YuF$ 

```

Drugu primenu algoritma *ZatvaracX* predstavlja postupak za izračunavanja zatvarača  $F^+$  skupa funkcijskih zavisnosti  $F$  zadatih nad šemom relacije  $R=\{A_1, \dots, A_N\}$ . Suština tog postupka koji je kombinatorne prirode je u sledećem:

- zatvarač  $F^+$  je u početku prazan skup;
- redom se formiraju skupovi  $S$  svih podskupova  $WuS$ , sastavljenih prvo od jednog, zatim od dva i tako sve do svih atributa šeme relacije  $R$ ;

Za svaki takav podskup  $WuS$  u svakom  $S$  uradi se sledeće:

- odredi se zatvarač  $Y$ , odnosno podskup  $Y$  svih atributa šeme relacije  $R$  koji funkcijski zavise od  $WuS$ ;
- u  $F^+$  se doda zavisnost  $WuS \rightarrow \emptyset$ ;
- redom se formiraju skupovi  $T$  svih podskupova  $ZuT$ , sastavljenih prvo od jednog, zatim od dva i tako sve do svih atributa podskupa  $Y$ ;
- za svaki takav podskup  $ZuT$  u  $F^+$  se doda zavisnost  $WuS \rightarrow ZuT$ ; poslednje takvo dodavanje obuhvatiće zavisnost  $WuS \rightarrow Y$ .

Algoritam *ZatvaracF* glasi:

```

ZatvaracF ( > R, > F )
: Inicijalizacija
: : Fplus =  $\emptyset$ 
: : NatrR = BrojElemenata ( R )
: Ponavljanje onoliko puta koliko ima atributa u R
: : DO FOR ( i = 1 to NatrR )
: : | Formiranje skupa podskupova od i atributa iz R
: : | : S = SkupPodskupova ( R, i )
: : | Nalazenje svih zavisnosti od svih podskupova
: : | : DO FOR EACH ( WuS IN S )
: : | : | Nalazenje zatvaraca jednog podskupa
: : | : | : Y = ZatvaracX ( WuS, F )
: : | : | Dodavanje zavisnosti  $WuS \rightarrow \emptyset$  u Fplus
: : | : | : Fplus = Fplus  $\cup$  { $WuS \rightarrow \emptyset$ }
: : | : | Dodavanje svih ostalih zavisnosti u Fplus
: : | : | : NatrY = BrojElemenata ( Y )
: : | : | : DO FOR ( j = 1 to NatrY )
: : | : | : | Formiranje skupa podskupova od j atributa Y
: : | : | : | : T = SkupPodskupova ( Y, j )
: : | : | : | Dodavanje jedne zavisnosti u Fplus
: : | : | : | : DO FOR EACH ( ZuT IN T )
: : | : | : | : | Fplus = Fplus  $\cup$  {( $WuS \rightarrow ZuT$ )}
: RETURN Fplus

```

Treću i izuzetno značajnu primenu algoritma *ZatvaracX* predstavlja nalaženje skupova atributa koji su kandidat-ključevi, odnosno određivanje skupa kandidat-ključeva  $K$ . Suština algoritma je u sledećem:

- skup  $K$  je u početku prazan;
- redom se formiraju skupovi  $S$  svih podskupova  $XuS$ , sastavljenih prvo od jednog, zatim od dva i tako sve do svih atributa šeme relacije  $R$ ;

Za svaki takav podskup  $XuS$  koji nije već sadržan u ili jednak nekom elementu skupa  $K$  uradi se sledeće:

- odredi se zatvarač ;
- ako je zatvarač  $XuS$  cela šema relacije  $R$  , odnosno ako važi  $Y=R$ ,  $XuS$  se dodaje skupu kandidat-ključeva  $K$ .

Odgovarajući algoritam glasi:

```

KandidatKljucevi ( > R, > F )
: Inicijalizacija
: : KandK =  $\emptyset$ 
: : NatrR = BrojElemenata ( R )
: Ponavljanje onoliko puta koliko ima atributa u R
: : DO FOR ( i = 1 to NatrR )
: : | Formiranje skupa podskupova od i atributa iz R
: : | : S = SkupPodskupova ( R, i )
: : | Ponavljanje za svaki podskup atributa u S
: : | : DO FOR EACH ( XuS IN S )
: : | : | Provera da je XuS vec u KandK ili je superkljuc
: : | : | : Sadrzan = FALSE
: : | : | : DO FOR EACH ( Kljuc IN KandK )
: : | : | : | IF ( Kljuc  $\subseteq$  XuS )
: : | : | : | : Sadrzan = TRUE
: : | : | <:--|--|--EXIT
: : | : | Provera da li je XuS novi kandidat-kljuc
: : | : | : IF ( NOT Sadrzan )
: : | : | : | Nalazenje zatvaraca XuS
: : | : | : | : ZatlXuS = ZatvaracX ( XuS, F )
: : | : | : | : Dodavanje u KandK ako je kandidat-kljuc
: : | : | : | : : IF ( ZatlXuS == R )
: : | : | : | : : | KandK = KandK  $\cup$  XuS
: RETURN KandK

```

Poslednja primena algoritma *Zatvarac* koju ovde navodimo je algoritam *SuperKljučevi* ( $R, F$ ) koji kao rezultat daje skup svih super-ključeva  $S$  šeme relacije  $R$  na osnovu skupa funkcijskih zavisnosti  $F$ . Taj algoritam se razlikuje od prethodnog samo po tome što bezuslovno dodaje u  $S$  svako  $XuS$  čiji je zatvarač cela šema relacije  $R$ .

Odgovarajući algoritam glasi:

```

SuperKljučevi ( > R, > F )
: Inicijalizacija
: : SuperK =  $\emptyset$ 
: : NatrR = BrojElementa ( R )
: Ponavljanje onoliko puta koliko ima atributa u R
: : DO FOR ( i = 1 to NatrR )
: : | Formiranje skupa podskupova od i atributa iz R
: : | : S = SkupPodskupova ( R, i )
: : | Ponavljanje za svaki podskup atributa u S
: : | : DO FOR EACH ( XuS IN S )
: : | : | Nalazenje zatvaraca XuS
: : | : | : ZatlXuS = ZatvaracX ( XuS, F )
: : | : | Dodavanje u KandK ako je super-ključ
: : | : | : IF ( ZatlXuS == R )
: : | : | SuperK = SuperK  $\cup$  XuS
: RETURN SuperK

```

*Primer*

Posmatrajmo šemu relacije:

OBLAST ( SIFO, NAZIV )

i skup funkcijskih zavisnosti, zasnovan na okolnosti da su i nazivi oblasti unikatni:

$$F = \{ \text{SIFO} \rightarrow \text{NAZIV} \quad \text{NAZIV} \rightarrow \text{SIFO} \}$$

Algoritam *KandidatKljučevi* daje kao rezultat samo:

**SIFO**                      **NAZIV**

Primena algoritma *SuperKljučevi* daje sledeće podskupove atributa:

**SIFO**                      **NAZIV**                      **SIFO,NAZIV**



### 7.3.5 Dekompozicija skupa funkcijskih zavisnosti

Očuvanje atributa i podataka su bitni uslovi za valjanost dekompozicije neke polazne šeme relacije  $R$  u dve ili više novih šema  $R_i$ . Ako je nad polaznom šemom važio neki skup funkcijskih  $F$ , postavlja se pitanje kakvi će biti skupovi zavisnosti  $F_i$  nad šemama  $R_i$  posle dekompozicije.

Šta praktično znače funkcijske zavisnosti za neku relacionu bazu podataka? One su posledice ograničenja koja važe između nekih svojstava u sistemu koga ta baza podataka predstavlja, i ta ograničenja moraju uvek biti ispoštovana pri ažuriranju baze podataka. Ako važi  $X \rightarrow Y$ , onda u našoj bazi podataka svakoj vrednosti  $X$ , ma koliko puta se pojavila skupa sa  $Y$ , mora da odgovara samo jedna vrednosti  $Y$ . Pri tome su moguća dva slučaja u vezi načina zapisivanja  $X$  i  $Y$  u bazi podataka:

- $X$  i  $Y$  se nalaze u istoj šemi relacije, odnosno njihove vrednosti su uvek u istoj relaciji ;
- $X$  i  $Y$  se nalaze razdvojeni u dve šeme relacija, odnosno njihove vrednosti su u dve relacije, ali se mogu pojaviti u jednoj relaciji izvršavanjem upita koji spaja te dve relacije;

Razmotrimo prvo slučaj kada su  $X$  i  $Y$  u istoj šemi relacije. Neka nam kao ilustracija posluži raniji primer relacije **naslov**, odnosno torke označene sa (a) i (b), pri čemu je prvo uneta torka (a) a posle nje i torka (b):

```

naslov ( SIFN NAZIVN                                SIFO NAZIVO                                )
-----
      RBP0 Relacione baze podataka BP           Baze podataka
      RK00 Racunarske komunikacije RM          Racunarske mreze
  (a) PP00 PASCAL programiranje    PJ           Programski jezici
  (b) PJC0 Programski jezik C       PJ           Programski jezici
-----

```

U ovoj relaciji važi, pored ostalih, i funkcijska zavisnost  $SIFO \rightarrow NAZIVO$ .

Pri unosu torke (a) jedino je važno ograničenje da **SIFN** mora biti unikatno i ne-NULL. I pored zavisnosti  $SIFO \rightarrow NAZIVO$ , uz vrednost 'PJ' za **SIFO** mogli smo da unesemo potpuno proizvoljnu vrednost za **NAZIVO**, i to zato što se vrednost 'PJ' pojavljuje prvi put. Međutim, pri unosu torke (b) kada se ponovo unosi vrednost 'PJ' za **SIFO** važi dodatno ograničenje: pošto se u relaciji **naslov** već nalazi vrednost 'PJ', u torci (b) za **NAZIVO** *moramo* uneti vrednost 'Programski jezici', pošto bi u suprotnom narušili zavisnost  $SIFO \rightarrow NAZIVO$ . Proveru da li je to zadovoljeno možemo implementirati jednostavno - u definiciji tabele **naslov** treba da navedemo ograničenje tabele **UNIQUE(SIFO,NAZIVO)**.

Za ilustraciju slučaja kada su  $X$  i  $Y$  razdvojeni u dve relacije poslužiće nam dekompozicija prethodne relacije **naslov** na relacije **n1** i **n2**, pri čemu su naznačene torke koje odgovaraju torkama (a) i (b) u relaciji **naslov**:

n1 ( SIFN NAZIVN	NAZIVO	) n2( SIFN SIFO )
-----	-----	-----
RBP0 Relacione baze podataka	Baze podataka	RBP0 BP
RK00 Racunarske komunikacije	Racunarske mreze	RK00 RM
(a) PP00 PASCAL programiranje	Programski jezici	(a) PP00 PJ
(b) PJC0 Programski jezik C	Programski jezici	(b) PJC0 PJ
-----	-----	-----

Ova dekompozicija je bez gubitaka podataka, pošto je  $n1 \cap n2 \rightarrow n1$ . Posmatrajmo istu funkcijsku zavisnost kao i u prvom slučaju,  $SIFO \rightarrow NAZIVO$ .

Pri unosu para torki (a) u relacije **n1** i **n2** jedino je važno ograničenje da vrednost SIFN mora biti unikatna i različita od NULL, dok smo za atribut NAZIVO i SIFO mogli da unesemo bilo koji par vrednosti, s obzirom da se vrednost 'PJ' za SIFO pojavljuje prvi put. Međutim, pri unosu para torki (b) pojavljuje se vrednost 'PJ' za SIFO koja već postoji, pa za atribut NAZIVO *moramo* uneti vrednost 'Programski jezici', pošto bi svaka druga vrednost narušila funkcijsku zavisnost  $SIFO \rightarrow NAZIVO$ . Na primer, ako bi na NAZIVO uneli 'Baze podataka', upit koji spajanjem relacija **n1** i **n2** daje šifre i nazive oblasti dao bi rezultat koji narušava zavisnost  $SIFO \rightarrow NAZIVO$ :

SELECT N2.SIFO,N1.NAZIVO	BP	Baze podataka
FROM N2,N1	RM	Racunarske mreze
WHERE N2.SIFN=N1.SIFN ;	PJ	Programski jezici ?
	PJ	Baze podataka ?

Pošto se sada SIFO i NAZIVO nalaze razdvojeni u dve relacije, jedini način za proveru važenja zavisnosti  $SIFO \rightarrow NAZIVO$  je preko pravila opšteg integriteta

```
CREATE ASSERTION Zavisnost
CHECK ( NOT EXISTS ( SELECT      N2.SIFO
                        FROM        N2,N1
                        WHERE        N2.SIFN=N1.SIFN
                        GROUP BY     N2.SIFO,
                        HAVING      COUNT(DISTINCT N1.NAZIVO)>1 ))
```

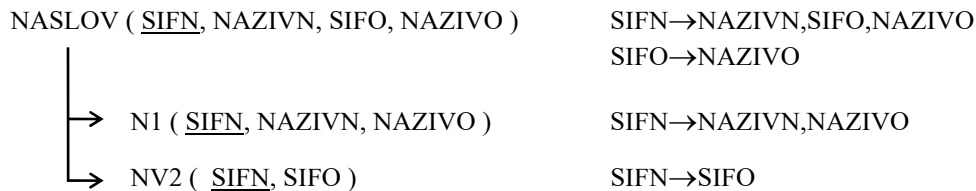
ili rečima: “u spoju **n2** i **n1** ni jedno SIFN ne sme da se javlja sa više od jednim SIFO”.

Dobar sistem za upravljanje relacionom bazom podataka ne sme da dozvoli situaciju narušavanja funkcijske zavisnosti kakva je prethodno opisana. Zbog toga se prilikom kreiranja baze podataka, odnosno opisa njene strukture, zadaju i sva dodatna ograničenja zasnovana na funkcijskim zavisnostima, a prilikom svake izmene sadržaja baze podataka sistem automatski proverava saglasnost i sa tim dodatnim ograničenjima. Međutim, tu su moguća dva slučaja, što smo već naveli:

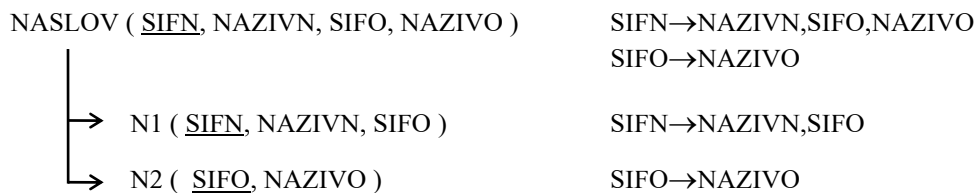
- za zadatu funkcijsku zavisnost u bazi podataka postoji bar jedna relacija koja sadrži sve njene atribute;
- za zadatu funkcijsku zavisnost u bazi podataka ne postoji ni jedna relacija koja sadrži sve njene atribute.

Prvi slučaj obezbeđuje efikasnu proveru, pošto se pristupa samo jednoj relaciji. U drugom slučaju može se desiti (ne mora uvek) da je provera moguća samo ako se izvrši spajanje dve ili više relacija u okviru pravila opšteg integriteta, što je daleko sporiji postupak provere.

Šta se ustvari dogodilo u našem prethodnom primeru? Polaznu relaciju **naslov** dekomponovali smo na dve relacije, **n1** i **n2**, što možemo prikazati šematski, uz navođenje skupova funkcijskih zavisnosti za relacije, kao:



Skupovi zavisnosti koji važe nad relacijama nastalim dekompozicijom neke polazne relacije nazivaju se projekcijama polaznog skupa zavisnosti. U našem primeru, imamo valjanu dekompoziciju koja zadovoljava uslove očuvanja podataka, ali je došlo do tzv. "gubitka zavisnosti", pošto zavisnost SIFO→NAZIVO nije primenjiva ni na jednu od nastalih relacija i može se proveravati samo spajanjem tih relacija. Sa druge strane, dekompozicija



je bolja, pošto ne dovodi do gubitka zavisnosti SIFO→NAZIVO.

*Definicija*

Neka je  $F$  skup funkcijskih zavisnosti  $\{X_i \rightarrow Y_i\}$  nad šemom relacije  $R$  i neka je  $S$  neki podskup atributa u  $R$ . Projekcija  $F$  po  $S$ , odnosno  $F[S]$ , je skup funkcijskih zavisnosti  $F'$  koji čine sve funkcijske zavisnosti oblika  $X_i \rightarrow Z_i$  za koje je zadovoljen uslov  $X_i \subseteq S \wedge Z_i = Y_i \cap S \wedge Z_i \neq \emptyset$ .

*Definicija*

Neka je  $F$  skup funkcijskih zavisnosti nad šemom relacije  $R$  i neka je data dekompozicija  $R$  na skup šema relacija  $\{R_1, \dots, R_N\}$ . Projekcija  $F$  po dekompoziciji  $\{R_1, \dots, R_N\}$ , odnosno  $F[\{R_1, \dots, R_N\}]$ , je skup funkcijskih zavisnosti  $F_D$  za koji važi

$$F_D = F[R_1] \cup \dots \cup F[R_N]$$

Navedimo prvo algoritam koji za dati skup funkcijskih zavisnosti  $F$  i podskup atributa  $S$  nalazi skup  $F'$  kao projekciju  $F$  po  $S$ :

```

ProjekcijaFpoS ( > F, > S )
: Inicijalizacija
: : FpoS =  $\emptyset$ 
: Projekcija svih zavisnosti
: : DO FOR EACH (  $X \rightarrow Y$  in  $F$  )
: : | Projekcija jedne zavisnosti
: : | : IF (  $X \subseteq S$  )
: : | : |  $Z := Y \cap S$ 
: : | : | IF (  $Z \neq \emptyset$  )
: : | : | | FpoS = FpoS  $\cup$  {  $X \rightarrow Z$  }
: RETURN FpoS

```

Na osnovu tog algoritma može se sastaviti algoritam koji za dati skup funkcijskih zavisnosti  $F$  i dekompoziciju  $D$  datu kao skup skupova atributa odnosno šema relacija  $\{R_1, \dots, R_N\}$  nalazi  $F_D$  kao projekciju po toj dekompoziciji:

```

ProjekcijaFpoD ( > F, > D )
: Inicijalizacija
: : FpoD =  $\emptyset$ 
: Projekcija po svim semama relacija
: : DO FOR EACH (  $R$  in  $D$  )
: : | Projekcija po jednoj semi relacije
: : | : FpoD = FpoD  $\cup$  ProjekcijaFpoS (  $F, R$  )
: RETURN FpoD

```

Funkcije *ProjekcijaFpoS* i *ProjekcijaFpoD* mogu se primeniti i za nalaženje odgovarajućih projekcija zatvarača  $F^+$  odnosno netrivialnog zatvarača  $F_n^+$ .

Pre nego što se ozbiljnije posvetimo problemu očuvanja funkcijskih zavisnosti pri dekompoziciji šeme relacije razmotrimo dva karakteristična primera. U oba slučaja poslužiće nam šema relacije

NASLOV ( SIFN, NAZIVN, SIFO, NAZIVO )

i skup funkcijskih zavisnosti

$$F = \{ \text{SIFN} \rightarrow \text{NAZIVN}, \text{SIFO}, \text{NAZIVO} \quad \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{NAZIVO} \rightarrow \text{SIFO} \}$$

*Primer*

Ako izvršimo dekompoziciju šeme relacije NASLOV na šeme relacija

NASLOV1 ( SIFN, NAZIVN, SIFO ) NASLOV2 ( SIFO, NAZIVO )

dobijamo po prethodnim algoritmima dekompoziciju F na dva skupa, F1 i F2 :

$$F1 = \{ \text{SIFN} \rightarrow \text{NAZIVN}, \text{SIFO} \} \quad F2 = \{ \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{NAZIVO} \rightarrow \text{SIFO} \}$$

Imamo da je  $F1 \cup F2 = F$ , odnosno ni jedna zavisnost iz F nije izgubljena.

*Primer*

Ako šemu relacije NASLOV dekomponujemo na drugi način

NASLOV1 ( SIFN, NAZIVN, NAZIVO ) NASLOV2 ( SIFN, SIFO )

dobijamo drugačija dva skupa, F1 i F2 :

$$F1 = \{ \text{SIFN} \rightarrow \text{NAZIVN}, \text{NAZIVO} \} \quad F2 = \{ \text{SIFN} \rightarrow \text{SIFO} \}$$

Sada imamo da je  $F1 \cup F2 \neq F$  i da su izgubljene dve zavisnosti:  $\text{SIFO} \rightarrow \text{NAZIVO}$  i  $\text{NAZIVO} \rightarrow \text{SIFO}$ , pošto ni na koji način ne mogu da se izvedu iz  $F1 \cup F2$ . Da li su stvarno izgubljene ?

Pri davanju odgovora na to pitanje treba imati na umu da smo dekomponovali samo zavisnosti koje su bile eksplicitno date u  $F$ , a ne i one koje se kao izvodiive iz njih nalaze u  $F^+$  odnosno u  $F_n^+$ . Da smo uključili i te zavisnosti, moglo bi se desiti da neka od izgubljenih zavisnosti bude izvodiiva iz njih i onih eksplicitno datih. Iz ovoga je jasno da možemo da govorimo o dve vrste gubitka zavisnosti:

- prividni gubitak: zavisnost iz  $F$  se eksplicitno ne nalazi u projekciji  $F$  po dekompoziciji niti je izvodiiva iz nje, ali se nalazi ili je izvodiiva iz projekcije  $F^+$  (odnosno  $F_n^+$ ) po dekompoziciji (u stvari, zavisnost je indirektno sačuvana);
- suštinski gubitak: zavisnost iz  $F$  se eksplicitno ne nalazi u projekciji  $F$  po dekompoziciji niti je izvodiiva iz nje, niti se nalazi i nije izvodiiva iz projekcije  $F^+$  (odnosno  $F_n^+$ ) po dekompoziciji.

Drugačije rečeno: za svaku zavisnost “izgublenu” pri dekompoziciji moramo proveriti da li prividno ili suštinski izgubljena. Ta provera se može sprovesti na dva načina:

- analitički: za polazni skup zavisnosti određuje se  $F^+$  (ili  $F_n^+$ ), projektuje se po dekompoziciji i proverava se da li se izgubljena zavisnost nalazi u dekompoziciji zavisnosti ili je izvodiiva iz nje;
- sintetički: nad svakom šemom  $R_i$  nastalom dekompozicijom se u odgovarajuće  $F_i$  kombinatornim postupkom dodaju zavisnosti koje nisu u njemu a mogu se izvesti iz polaznog skupa zavisnosti  $F$ .

U svim ovim razmatranjima pod “izvođenje” podrazumevamo primenu algoritma za nalaženje zatvarača skupa atributa.

Vratimo se sada na naš drugi primer kod koga su “izgubljene” zavisnosti  $SIFO \rightarrow NAZIVO$  i  $NAZIVO \rightarrow SIFO$ . Za taj slučaj polazne šeme relacije i skupa funkcijskih zavisnosti smo već ranije odredili netrivialni zatvarač  $F_n^+$ . Njegove projekcije po dekompoziciji  $\{ \text{NASLOV1 (SIFN, NAZIVN, NAZIVO)}, \text{NASLOV2 (SIFN, SIFO)} \}$  su, redom:

$$F1 = \{ SIFN \rightarrow SIFN, NAZIVN, NAZIVO \quad NAZIVN \rightarrow NAZIVN \quad NAZIVO \rightarrow NAZIVO \}$$

$$F2 = \{ SIFN \rightarrow SIFO \}$$

Ako nad ovakvim  $F1 \cup F2$  potražimo zatvarače atributa  $SIFO$  i  $NAZIVO$  dobijamo da su zavisnosti  $SIFO \rightarrow NAZIVO$  i  $NAZIVO \rightarrow SIFO$  neizvodiive, odnosno da su suštinski izgubljene.

Isti ishod bi dala i sintetička provera, što se prepušta čitaocima.



## **7.4 Funkcijske normalne forme i postupci normalizacije**

Već smo naglasiti da se normalizacija svodi na pogodnu dekompoziciju šeme relacije i relacije u cilju otklanjanja anomalija ažuriranja. Uzroci tih anomalija su usled prisustva neželjenih funkcijskih zavisnosti. Pod normalnim formama podrazumevamo određene kriterijume valjanosti neke šeme relacije, u smislu da funkcijske zavisnosti koje važe nad tom šemom moraju zadovoljavati određene uslove, koji mogu biti manje ili više strogi. Pre razmatranja te oblasti, neophodno je da se osvrnemo na neke specijalne slučajeve funkcijskih zavisnosti.

### 7.4.1 Specijalne funkcijske zavisnosti

Navedimo u formi definicija neke specijalne funkcijske zavisnosti koje su važne za definisanje određenih normalnih formi. Neka su pri tome  $R$  šema relacije, a  $X$ ,  $Y$  i  $Z$  podskupovi  $R$ .

#### *Definicija*

Funkcijska zavisnost  $X \rightarrow Y$  je superključna ako važi  $X \rightarrow R$ .

#### *Definicija*

Funkcijska zavisnost  $X \rightarrow Y$  je trivijalna ako važi  $X \subseteq Y$

#### *Definicija*

Funkcijska zavisnost  $X \rightarrow Y$  je totalna ako ne postoji ni jedan pravi podskup  $Z$  od  $X$  za koji važi  $Z \rightarrow Y$ , odnosno ako važi:

$$X \rightarrow Y \wedge \neg \exists Z (Z \subset X \wedge Z \rightarrow Y)$$

#### *Definicija*

Funkcijska zavisnost  $X \rightarrow Y$  je parcijalna ako postoji neki pravi podskup  $Z$  od  $X$  za koji važi  $Z \rightarrow Y$ , odnosno ako važi:

$$X \rightarrow Y \wedge \exists Z (Z \subset X \wedge Z \rightarrow Y)$$

#### *Definicija*

Funkcijska zavisnost  $X \rightarrow Y$  je tranzitivna ako postoji neki podskup atributa  $Z$  različit i od  $X$  i od  $Y$  za koji važi  $X \rightarrow Z$  i  $Z \rightarrow Y$ , odnosno ako važi:

$$X \rightarrow Y \wedge \exists Z (Z \neq X \wedge Z \neq Y \wedge X \rightarrow Z \wedge Z \rightarrow Y)$$

Definicije normalnih formi koje slede zasnivaju se upravo na specijalnim funkcijskim zavisnostima.

*Primer*

Posmatrajmo jednu krajnje nezgrapnu šemu relacije o pozajmicama

POZAJMICA ( SIFN, SIFC, DATUM, DANA, SIFK, NAZIVN, SIFO, NAZIVO )

kao i skup funkcijskih zavisnosti

$$F = \{ \begin{array}{l} \text{SIFN,SIFC,DATUM} \rightarrow \text{DANA} \quad \text{SIFN,SIFC,DATUM} \rightarrow \text{SIFK} \\ \text{SIFN,SIFC,DATUM} \rightarrow \text{NAZIVN} \quad \text{SIFN,SIFC,DATUM} \rightarrow \text{SIFO} \\ \text{SIFN,SIFC,DATUM} \rightarrow \text{NAZIVO} \quad \text{SIFK} \rightarrow \text{SIFN} \quad \text{SIFN} \rightarrow \text{SIFO} \\ \text{SIFN,SIFC} \rightarrow \text{SIFN} \quad \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{NAZIVO} \rightarrow \text{SIFO} \end{array} \}$$

Ovde imamo slučajeve svih specijalnih funkcijskih zavisnosti:

- superključne:  $\text{SIFN,SIFC,DATUM} \rightarrow \text{DANA}$        $\text{SIFN,SIFC,DATUM} \rightarrow \text{SIFK}$   
 $\text{SIFN,SIFC,DATUM} \rightarrow \text{NAZIVN}$        $\text{SIFN,SIFC,DATUM} \rightarrow \text{SIFO}$   
 $\text{SIFN,SIFC,DATUM} \rightarrow \text{NAZIVO}$  ;
- trivijalna:  $\text{SIFN,SIFC} \rightarrow \text{SIFN}$  ;
- totalne:  $\text{SIFN,SIFC,DATUM} \rightarrow \text{DANA}$        $\text{SIFN,SIFC,DATUM} \rightarrow \text{SIFK}$   
 $\text{SIFK} \rightarrow \text{SIFN}$        $\text{SIFN} \rightarrow \text{SIFO}$   
 $\text{SIFO} \rightarrow \text{NAZIVO}$        $\text{NAZIVO} \rightarrow \text{SIFO}$  ;
- parcijalne:  $\text{SIFN,SIFC,DATUM} \rightarrow \text{NAZIVN}$        $\text{SIFN,SIFC,DATUM} \rightarrow \text{SIFO}$  ;
- tranzitivna  $\text{SIFN,SIFC,DATUM} \rightarrow \text{NAZIVO}$ .

Superključne zavisnosti mogu ujedno biti i parcijalne ili tranzitivne.

### 7.4.2 Opšti postupak funkcijske normalizacije

Neka polazna šema relacije  $R$  nije u određenoj normalnoj formi ako u skupu funkciskih zavisnosti  $F$  koje važe nad tom šemom postoji bar jedna zavisnost oblika  $X \rightarrow Y$  koja narušava definiciju te normalne forme. Pri tome je prirodno da se postupak normalizacije do određene normalne forme sprovodi u koracima. Početna vrednost normalizovane dekompozicije  $D$  (skupa relacija  $R_i$  na koje je dekomponovana  $R$ ) je polazna šema relacije  $R$ , dok je početna vrednost projekcije  $F_D$  polaznog skupa zavisnosti  $F$  po dekompoziciji  $D$  taj polazni skup  $F$ .

Pri svakom koraku normalizacije posmatra se jedna šema  $R_i$  (na samom početku to je polazno  $R$ ) iz dekompozicije  $D$ . Za nju se prvo određuje skup svih kandidat-ključeva i nalazi se prva zavisnost  $X \rightarrow Y$  koja narušava željenu normalnu formu. Zatim se vrši dekompozicija posmatrane šeme relacije  $R_i$  tako što se neželjena zavisnost izdvaja u posebnu šemu  $R_{ik}(\underline{X}, Y)$  koja se dodaje dekompoziciji  $D$ , a  $R_i$  se redukuje tako što se iz nje izostavi skup atributa  $Y$ , odnosno svodi se na  $R_i - Y$ . Pri tome se vrši i dekompozicija odgovarajućeg skupa funkcijskih zavisnosti  $F_i$  na svoje projekcije  $F_i$  po redukovanom  $R_i$  i  $F_{ik}$  po novostanalom  $R_{ik}$  i  $F_{ik}$  se dodaje u  $F_D$ . Postupak se ponavlja sve dok se ne postigne da su sve šeme relacije u  $D$  u željenoj normalnoj formi.

Na osnovu toga, možemo formulisati sledeći opšti algoritam normalizacije u neku željenu normalnu formu N:

```

NormalizacijaN ( > R, > F, < D, < FpoD )
: Inicijalizacija
: : D      = {R}
: : FpoD   = {F}
: Normalizacija dok se ne postigne da su sve seme normalizovane
: : DO FOREVER
: : | Inicijalizacija brojaca izdvojenih zavisnosti
: : | : Izdvojeno = 0
: : | Normalizacija svih sema
: : | : DO FOR EACH ( Ri in D )
: : | : | Normalizacija jedne seme
: : | : | : Odredjivanje kandidat-kljuceva
: : | : | : : K = KandidatKljujevi ( Ri, Fi )
: : | : | : : Incijalizacija indikatora ispitivanja
: : | : | : : : Izdvojiti = FALSE
: : | : | : : Ispitivanje svih zavisnosti
: : | : | : : : DO FOR EACH ( X→Y IN Fi )
: : | : | : : : | Ispitivanje jedne zavisnosti
: : | : | : : : : IF ( Y ∉ X )
: : | : | : : : : : IF ( NOT ZavisnostN ( X→Y, Fi, K ) )
: : | : | : : : : : | Nadjena zavisnost za izdvajanje
: : | : | : : : : : : Izdvojiti = TRUE
: : | : | : : : : : : Izdvojeno = Izdvojeno + 1
: : | : | : : : <:--|--:--|--:--EXIT
: : | : : Redukcija i izdvajanje (ako treba)
: : | : : : IF ( Izdvojiti )
: : | : : : : ReduR   = Ri - Y
: : | : : : : NovoR   = X ∪ Y
: : | : : : : D = ( D - {Ri} ) ∪ ReduR ∪ NovoR
: : | : : : : ReduF   = ProjekcijaFPoS ( Fi, ReduR )
: : | : : : : NovoF   = ProjekcijaFPoS ( Fi, NovoR )
: : | : : : : FpoD = ( D - {Fi} ) ∪ ReduF ∪ Novo
: : | Kraj ako su sve seme normalizovane
: : | : IF ( Izdvojeno == 0 )
: <:--|--:--|--:-- EXIT

```

Pri tome, usvojili smo oznake

- R - polazna relacija;
- F - polazni skup funkcijskih zavisnosti;
- D - rezultat - dekompozicija  $\{R_1, \dots, R_N\}$  ;
- FpoD - rezultat - projekcija  $F[R_1] \cup \dots F[R_N]$  ;
- K - skup kandidat-ključeva relacije  $R_i$  ;
- ReduR - redukovana relacija  $R_i$  nastala izostavljanjem  $Y$  ;
- NovoR - nova relacija  $R_j$  nastala izdvajanjem  $X \rightarrow Y$  ;
- ReduF - projekcija  $F_i$  po  $RedR_i$  ;
- NovoF - projekcija  $F_i$  po  $NovR_j$  ;

Sa *ZavisnostN* smo označili funkciju koja ispituje da li je zadata netrivialna funkcijska zavisnost u normalnoj formi N. Ta funkcija će biti naknadno precizirana za svaku od normalnih formi koje razmatramo.

Suština ovog na izgled složenog algoritma je u sledećem:

- na početku, dekompoziciju D čini polazna relacija R, a projekciju FpoD polazni skup funkcijskih zavisnosti;
- koraci normalizacije se sprovode sve dok se ne postigne da su sve relacije  $R_i$  u dekompoziciji D u željenoj normalnoj formi;
- relacija  $R_i$  u dekompoziciji D je u željenoj normalnoj formi kada nad njom ne važi ni jedna funkcijska zavisnost koja narušava tu normalnu formu;
- svaka relacija  $R_i$  koja nije u željenoj normalnoj formi zbog neke zavisnosti  $X \rightarrow Y$  redukuje se tako što gubi iz svog sastava Y, a pri tome nastaje nova relacija  $R_j$  koju čine X i Y.

Algoritam *NormalizacijaN* daje dekompoziciju D bez gubitaka podataka, pošto je pri svakom koraku dekompozicije  $R_i$  na redukovano  $R_i$  i novo  $R_j$  zadovoljen uslov reverzibilnosti  $R_i \cap R_j \rightarrow R_j$ , s obzirom da je taj presek desna strana zavisnosti  $X \rightarrow Y$  koju izdvajamo u  $R_j$ , odnosno ključ je u  $R_j$ . Sa druge strane, ne postoji garancija da će pri tome biti očuvane i sve polazne funkcijske zavisnosti.

Treba naglasiti da je procedura *NormalizacijaN* navedena u pojednostavljenoj formi. Kompletan forma te procedure podrazumeva i određivanje zatvarača  $F^+$  polaznog skupa zavisnosti F i korišćenje zavisnosti iz  $F^+$  u postupku normalizacije.

### 7.4.3 Druga normalna forma

#### *Definicija*

Šema relacije  $R$  je u drugoj normalnoj formi ako nad njom ne postoji ni jedna funkcijska zavisnost po kojoj neki neključni atribut parcijalno zavisi od bilo kog kandidat-ključa.

Formulišimo sada algoritam *Zavisnost2* koji za dati skup kandidat-ključeva  $K$  ispituje da li je funkcijska zavisnost  $X \rightarrow Y$  u skladu sa definicijom druge normalne forme. Podrazumeva se da i  $X$  i  $Y$  pripadaju šemi relacije  $R$ .

```

Zavisnost2 ( > X→Y, > K )
: Inicijalizacija
: : XnepripadaK = TRUE
: : YpripadaK   = TRUE
: Ispitivanje
: : 1. ispitivanje za sve kandidat-kljuceve
: : : DO FOR EACH ( Z in K )
: : : | Ispitivanje da li je X deo kandidat-kljuca
: : : | : IF ( X ⊂ Z )
: : : | : | XnepripadaK = FALSE
: : : <:--|--:--|--EXIT
: : 2. ispitivanje za sve kandidat-kljuceve (ako treba)
: : : IF ( NOT XnepripadaK )
: : : | DO FOR EACH ( Z in K )
: : : | | Ispitivanje da li Y nije deo kandidat-kljuca
: : : | | : IF ( Y ⊄ Z )
: : : | | : | YpripadaK = FALSE
: : : <:--|--:--|--|--EXIT
: RETURN XnepripadaK ∨ YpripadaK

```

Ovde treba napomenuti: to što je leva strana  $X$  zavisnosti  $X \rightarrow Y$  pravi podskup nekog kandidat-ključa  $Z$  još uvek ne znači da je narušena druga normalna forma, pošto pri tome desna strana  $Y$  može da ne sadrži ni jedan neključni atribut.

*Primer*

Posmatrajmo šemu relacije kojom smo u odeljku 7.1 ilustrovali efekte loše strukture relacije:

AUTOR ( SIFA, SIFN, IME, KOJI )

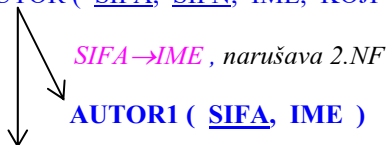
kao i skup funkcijskih zavisnosti nad njom

$F = \{ SIFA, SIFN \rightarrow IME, KOJI \mid SIFA \rightarrow IME \}$

Kandidat-ključ ove šeme je SIFA, SIFN. Zavisnost SIFA, SIFN  $\rightarrow$  IME je parcijalna, pošto IME zavisi od dela kandidat-ključa po zavisnosti SIFA  $\rightarrow$  IME.

Postupak normalizacije na drugu normalnu formu sastoji se u tome da zavisnost SIFA  $\rightarrow$  IME izdvojimo u posebnu šemu relacije a iz polazne šeme uklonimo desnu stranu te zavisnosti, odnosno IME. To daje sledeću dekompoziciju na šeme relacija i skupove funkcijskih zavisnosti:

AUTOR ( SIFA, SIFN, IME, KOJI )



AUTOR1 ( SIFA, IME )

$F1 = \{ SIFA \rightarrow IME \}$

AUTOR ( SIFA, SIFN, KOJI )

$F = \{ SIFA, SIFN \rightarrow KOJI \}$

Pri tome je uklonjena neželjena parcijalna funkcijska zavisnost.



#### 7.4.4 Treća normalna forma

##### *Definicija*

Šema relacije  $R$  je u trećoj normalnoj formi ako nad njom ne postoji ni jedna funkcijska zavisnost po kojoj neki neključni atribut tranzitivno zavisi od kandidat-ključa.

Umesto da navodimo kakve funkcijske zavisnosti nisu dozvoljene, možemo formulisati definiciju treće normalne forme u kojoj se navode dozvoljene funkcijske zavisnosti. To je pogodnije kod normalnih formi iznad druge, kod kojih je mnogo manje toga dozvoljeno nego što je zabranjeno.

##### *Definicija*

Šema relacije  $R$  je u trećoj normalnoj formi ako svaka funkcijska zavisnost  $X \rightarrow Y$  koja važi nad njom zadovoljava bar jedan od uslova:

- zavisnost je trivijalna, odnosno  $Y \subseteq X$  ;
- zavisnost je superključna, odnosno  $X \rightarrow R$  ;
- $Y$  je deo kandidat-ključa, odnosno  $\exists Z ( Y \subseteq Z \wedge Z \rightarrow R )$ .

Iz ove definicije i prethodne definicije druge normalne forme jasno je da je svaka šema relacije koja je u trećoj normalnoj formi istovremeno i u drugoj normalnoj formi: ni jedan od navedena tri uslova ne dozvoljava postojanje zavisnosti neključnog atributa od dela kandidat-ključa.

Algoritam *Zavisnost3*, koji za dati skup dati skup kandidat-ključeva  $K$  ispituje da li je funkcijska zavisnost  $X \rightarrow Y$  u skladu sa definicijom treće normalne forme, glasi:

```

Zavisnost3 ( > X→Y, > K )
: Inicijalizacija
: : Jeste = FALSE
: Ispitivanje
: : Provera da li je zavisnost trivijalna
: : : IF ( Y ⊆ X )
: : : | Jeste = TRUE
: <:---:--|--EXIT
: : Provera za sve kandidat-kljuceve
: : : DO FOR EACH ( Z IN K )
: : : | Provera da li Y deo kandidat-kjuca
: : : | : IF ( Y ⊆ Z )
: : : | : | Jeste = TRUE
: <:---:--|--: |--EXIT
: : : | Provera da li je X super-kljuc
: : : | : IF ( Z ⊆ X )
: : : | : | Jeste = TRUE
: <:---:--|--:--|--EXIT
: RETURN Jeste

```

*Primer*

Posmatrajmo relaciju o naslovima koja sadrži sve podatke o naslovima, oblastima i autorima:

NASLOV ( SIFN, SIFA, KOJI, NAZIVN, IME, SIFO, NAZIVO )

kao i odgovarajući skup funkcijskih zavisnosti:

$F = \{ \text{SIFN, SIFA} \rightarrow \text{KOJI, NAZIVN, IME, SIFO, NAZIVO}$   
 $\text{SIFN} \rightarrow \text{NAZIVN, SIFO} \quad \text{SIFA} \rightarrow \text{IME} \quad \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{NAZIVO} \rightarrow \text{SIFO} \}$

Ako primenimo ranije opisani univerzalni postupak normalizacije uz primenu kriterijuma za treću normalnu formu, dobijamo dekompoziciju:



Za dekompoziciju  $F'$  polaznog skupa funkcijskih zavisnosti  $F$  dobili smo:

$F = \{ \text{SIFO} \rightarrow \text{NAZIVO} \quad \text{SIFN} \rightarrow \text{NAZIVN, SIFO} \quad \text{SIFA} \rightarrow \text{IME} \quad \text{SIFN, SIFA} \rightarrow \text{KOJI} \}$

U ovom skupu u odnosu na polazni “izgubljena” je zavisnost

$\text{SIFN, SIFA} \rightarrow \text{KOJI, NAZIVN, IME, SIFO, NAZIVO}$

ali se to da je ona izvodiva iz  $F'$  može pokazati nalaženjem zatvarača  $(\text{SIFN, SIFA})^+$  nad skupom zavisnosti  $F'$  ili primenom pravila izvođenja. Drugim rečima, sve zavisnosti iz polaznog skupa  $F$ , odnosno njihovo važenje, su očuvani.

### 7.4.5 Bojs-Kodova normalna forma

Za prethodne dve normalne forme, lako uočavamo da je u oba slučaja uzročnik nevolja postojanje neželjenih tranzitivnih funkcijskih zavisnosti, pošto parcijalnu zavisnost možemo smatrati za specijalni slučaj tranzitivne: neključni atribut funkcijski zavisi od dela kandidat-ključa, a ovaj zavisi od celog kandidat-ključa.

U opštem slučaju, nad šemom relacije mogu da postoje četiri vrste tranzitivnih funkcijskih zavisnosti koje dovode do anomalija ažuriranja:

- zavisnost neključnog atributa posredstvom dela kandidat ključa;
- netrivialna zavisnost neključnog atributa posredstvom podskupa neključnih atributa;
- netrivialna zavisnost ključnog atributa posredstvom dela kandidat ključa;
- zavisnost ključnog atributa posredstvom podskupa neključnih atributa.

Druga i treća normalna forma iz dekompozicije neke šeme relacije eliminišu samo tranzitivne zavisnosti prve i druge vrste, pa se opravdano postavlja pitanje definisanja neke strožije normalne forme koja sasvim eliminiše tranzitivne zavisnosti.

#### *Definicija*

Šema relacije  $R$  je u Bojs-Kodovoj (Boyce-Codd) normalnoj formi ako svaka funkcijska zavisnost  $X \rightarrow Y$  koja važi nad njom zadovoljava jedan od uslova:

- zavisnost je trivialna, odnosno  $Y \subseteq X$  ;
- zavisnost je superključna, odnosno  $X \rightarrow R$  .

Razlika u odnosu na definiciju treće normalne forme je samo u tome što se ne dozvoljava treći slučaj (tranzitivna zavisnost dela kandidat-ključa), pa je šema relacije koja je u Bojs-Kodovoj normalnoj formi istovremeno i u trećoj normalnoj formi.

Sledeći algoritam za dati skup kandidat-ključeva  $K$  utvrđuje da li je funkcijska zavisnost  $X \rightarrow Y$  saglasna definiciji Bojs-Kodove normalne forme:

```

ZavisnostBC ( > X→Y, > K )
: Inicijalizacija
: : Jeste = FALSE
: Ispitivanje
: : Provera da li je zavisnost trivijalna
: : : IF ( Y ⊆ X )
: : : | Jeste = TRUE
: <:--:--|--EXIT
: : Provera za sve kandidat-kljuceve
: : : DO FOR EACH ( Z IN K )
: : : | Provera da li je zavisnost super-kljucna
: : : | : IF ( Z ⊆ X )
: : : | : | Jeste = TRUE
: <:--:--|--:--|--EXIT
: RETURN Jeste

```

Uočimo da se ovaj algoritam razlikuje od prethodnog algoritma *Zavisnost3* samo po tome što je uslov ispitivanja strožiji, bez dodatne mogućnosti  $Y \subset Z$ .

Bojs-Kodova normalna forma je najviša normalna forma koja se može postići na bazi funkcijskih zavisnosti.

Normalizacija u Bojs-Kodovu normalnu formu (i uopšte univerzalnim postupkom normalizacije) ima i svoju cenu: ne postoji garancija očuvanja funkcijskih zavisnosti. Šta više, redosled eliminisanja neželjenih funkcijskih zavisnosti može uticati na stepen gubitka zavisnosti. Sa druge strane, postoji specijalni algoritam normalizacije u treću normalnu formu koji garantuje očuvanje svih funkcijskih zavisnosti. Iz tih razloga, danas se treća normalna forma smatra za sasvim zadovoljavajuću kod relacionih baza podataka. Navedeni specijalni algoritam nećemo razmatrati, pošto uključuje pojmove minimalnog zatvarača i prstenastog pokrivača skupa funkcijskih zavisnosti koji nisu obuhvaćeni ovim udžbenikom.

*Primer*

Posmatrajmo raniju šemu relacije o pozajmicama koja sadrži podatke o naslovima, članovima i knjigama:

POZAJMICA ( SIFN, SIFC, DANA, NAZIVN, SIFK )

kao i odgovarajući skup funkcijskih zavisnosti:

$F = \{ \text{SIFN, SIFC, DANA} \rightarrow \text{NAZIVN, SIFK} \quad \text{SIFK} \rightarrow \text{SIFN} \quad \text{SIFN} \rightarrow \text{NAZIVN} \}$

Primenom algoritma normalizacije dobijamo sledeću dekompoziciju:



Prazan skup zavisnosti  $\{ \}$  označava da u odgovarajućoj relaciji važe samo trivijalne funkcijske zavisnosti. I u ovom primeru nemamo gubitak zavisnosti, što se lako pokazuje na način koji smo već opisali.

#### 7.4.6 Prva normalna forma

U jednom značajnom broju udžbenika iz oblasti baza podataka prva normalna forma razmatra se tek nakon svih ostalih normalnih formi na bazi funkcijskih zavisnosti. To nije bez razloga: u određenim situacijama, kada se prvo izvrši normalizacija u prvu normalnu formu, dolaze do izražaja nove vrste zavisnosti koje nisu funkcijskog karaktera.

U svim primerima do sada, i u poglavljima koja su se bavila jezicima relacionih baza podataka i u ovom poglavlju, podrazumevalo se da su vrednosti svih atributa skalari. To je prilikom formulisanja relacionog modela podataka pretpostavio i njegov tvorac Kod - otud ono "prva" u nazivu. Ovo predstavlja jednu vrstu implementacionog ograničenja nad tipovima atributa koja ne postoji u savremenim programskim jezicima. Primera radi još je programski PASCAL dozvolio ograničeni skupovni tip nad određenim baznim tipom, dok programski jezik Java dozvoljava skupove proizvoljnog prostog ili složenog tipa uključujući i skupove skupova.

*Primer*

Relacija **je\_autor** iz naše baze podataka BIBLIOTEKA može se napisati na način koji ukazuje na to da svakom naslovu, odnosno svakoj vrednosti atributa SIFN može da odgovara skup parova vrednosti atributa SIFA i KOJI:

```

je_autor ( SIFN ( SIFA KOJI )
-----
RBP0 {<AP0,1>,<JN0,2>}
RK00 {<DM0,1>}
PP00 {<ZP0,1>,<DM0,2>,<IT0,3>}
PJC0 {<AP1,1>,<ZP0,2>}
-----

```

Saglasno konceptu univerzalnog relacionog modela koji je izložen na kraju poglavlja 4, šema ovakve relacije zapisuje se u formi "relacije u relaciji"

JE\_AUTOR ( SIFN, ( SIFA, KOJI ) )

čime se naglašava da jednoj vrednosti SIFN može da odgovara više parova vrednosti atributa SIFA i KOJI. Pri tome, uočavamo da se te dve vrednosti ponavljaju vezano, dakle ne nezavisno jedna od druge.



Ovakve situacije su se često javljale u praksi i način na koji su razrešavane u klasičnim informacionim sistemima svodio se na nešto što bi se moglo opisati šemom relacije sa promenljivim brojem ponavljanja atributa:

JE\_AUTOR ( SIFN, SIFA<sub>1</sub>, KOJI<sub>1</sub>, SIFA<sub>2</sub>, KOJI<sub>2</sub>, ... )

Ovakvo rešenje podrazumeva promenljivu dužinu sloga, što se vrlo teško realizuje u uslovima rada sa direktnim pristupom slogovima. Sa druge strane, korišćenje fiksne dužine sloga zahteva da se unapred opredelimo za maksimalni mogući broj ponavljanja atributa. Za slučaj da je usvojeno maksimalno tri ponavljanja, imali bi šemu relacije

JE\_AUTOR ( SIFN, SIFA<sub>1</sub>, KOJI<sub>1</sub>, SIFA<sub>2</sub>, KOJI<sub>2</sub>, SIFA<sub>3</sub>, KOJI<sub>3</sub> )

a sama relacija **je\_autor** bi u tom slučaju glasila:

```
je_autor (  SIFN  SIFA1 KOJI1  SIFA2 KOJI2  SIFA3 KOJI3 )
-----
RBPO  AP0    1      JN0    2      null  null
RK00  DM0    1      null  null  null  null
PP00  ZP0    1      DM0    2      IT0    3
PJC0  AP1    1      ZP0    2      null  null
-----
```

Ovim smo postigli da u relaciji imamo onoliko torki koliko ima naslova, ali su mane ovakvog rešenja evidentne:

- ako se ikada pojavi naslov sa više od tri autora, nismo u mogućnosti da to unesemo u ovakvu relaciju;
- upiti i druge operacije nad ovakvom relacijom su izuzetno komplikovani (čitaocu se prepušta da sastavi na bilo kom jeziku upit koji daje šifre autora po naslovima, ili šifre naslova koje je napisao autor šifre DM0).

Situacija je još gora ako vrednosti koje se ponavljaju mogu nastajati i nestajati proizvoljan broj puta i proizvoljnim redosledom. Neka na osnovu pretpostavke da neki član biblioteke može kod sebe da drži najviše tri knjige imamo šemu relacije

DRZI ( SIFC, SIFK<sub>1</sub>, DATUM<sub>1</sub>, SIFK<sub>2</sub>, DATUM<sub>2</sub>, SIFK<sub>3</sub>, DATUM<sub>3</sub> )

Probleme sa upitima kod ovakvih šema relacija smo upravo opisali. Navedimo sada samo neke od problema ažuriranja:

- ako član koji je uzeo tri knjige vrati dve od njih, za vrednosti šifara tih knjiga i datuma moramo upisati NULL vrednosti;
- ako taj član, koji kod sebe drži jednu knjigu, uzme još jednu, šifru te knjige i datum uzimanja moramo upisati u relaciju, ali gde – na drugom ili trećem mestu ?

Iz ovih nekoliko primera zaključujemo: situacija da vrednosti nekog atributa u jednoj torci u relaciji odgovara skup vrednosti je neprihvatljiva, kako sa gledišta organizacije podataka tako i sa gledišta operacija nad takvim podacima.

### *Definicija*

Šema relacije  $R$  je u prvoj normalnoj formi ako je svaki njen atribut skalarnog domena.

U pitanju je ograničenje koje nije pominjano u definicijama druge, treće i Bojs-Kodove normalne forme, pa bi u tim uslovima svaku od tih defincija trebalo dopuniti na samom početku klauzulom "ako je u prvoj normalnoj formi".

Normalizacija, odnosno svođenje šeme relacije na prvu normalnu formu, krajnje je jednostavna. Za neku šemu relacije  $R$ , ta normalizacija se može formalno prikazati kao transformacija

$$R(X, (Y)) = R1(X, Y)$$

gde smo sa  $Y$  označili (jedini) skup atributa koji se zajedno ponavljaju, a sa  $X$  ostale attribute. U najnepovoljnijem slučaju, ključ šeme relacije  $R1$  će činiti ključ šeme relacije  $R$  zajedno sa  $Y$ , ali usled funkcijskih zavisnosti koje važe nad  $R1$  umesto celog  $Y$  u sastav ključa  $R1$  može ući samo jedan njegov deo.

Što se tiče relacija  $r$  i  $r1$  nad tim šemama, stanje je sledeće: od svake torke  $x_i, (y_i)$  u  $r$  nastaje onoliko torki  $x_i, y_k$  u  $r1$  koliko je bilo vrednosti u skupu  $(y_i)$ . Ako je  $(y_i)$  prazno, neće nastati ni jedna torka. Iz toga zaključujemo da od  $N(r)$  torki polazne relacije  $r$  u relaciji  $r1$  nastaje  $N(r1)=N_1+...+N_{N(r)}$  torki, gde je  $N_i$  broj vrednosti u skupovima  $(y_i)$ .

*Primer*

Posmatrajmo šemu relacije iz prethodnog primera

JE\_AUTOR ( SIFN, ( SIFA, KOJI ) )

nad kojom važi skup zavisnosti

$F = \{ SIFN \rightarrow (SIFA, KOJI), SIFN, SIFA \rightarrow KOJI \}$

gde sa  $X \rightarrow (Y)$  označavamo da svakoj vrednosti X odgovara jedan skup vrednosti Y.

Normalizacijom dobijamo šemu relacije

JE\_AUTOR1 ( SIFN, SIFA, KOJI )

pri čemu je skup zavisnosti nad njom sveden na

$F1 = \{ SIFN, SIFA \rightarrow KOJI \}$

što je uslovilo da KOJI ne uđe u sastav jedinog kandidat-ključa.

Normalizacijom relacije **je\_autor** dobija se relacija **je\_autor1**

```

je_autor1 ( SIFN  SIFA  KOJI )
-----
RBP0  AP0   1
RBP0  JN0   2

RK00  DM0   1

PP00  ZP0   1
PP00  DM0   2
PP00  IT0   3

PJC0  AP1   1
PJC0  ZP0   2
-----

```

pri čemu smo umesto slogovnog atributa <SIFA,KOJI> uveli elementarne attribute SIFA i KOJI, pa se u prikazu sadržaja relacije izgubilo "<>".

Do sada smo imali slučaj sa samo jednim skupom atributa  $Y$  koji se zajedno ponavljaju. U opštem slučaju, u šemi relacije  $R$  može biti prisutno više takvih skupova:

$$R ( X, (Y1), ..., (YM) )$$

Odmah treba naglasiti jednu bitnu stvar: skupovi  $(Y1), ..., (YM)$  su uzajamno potpuno nezavisni. To će biti vrlo jasno iz primera koji će uslediti.

Normalizacija ovakve šeme relacije formalno se može predstaviti kao transformacija:

$$R ( X, (Y1), ..., (YM) ) == R ( X, Y1, ..., YM )$$

Ključ konačne šeme relacije  $R$  činiće u najnepovoljnijem slučaju ključ šeme relacije  $R$  (neki podskup  $X$ ) i svi skupovi atributa  $Y_i$ , ali funkcijske zavisnosti nad  $R$  mogu usloviti i manji ključ.

Da bi bolje razumeli šta se dešava sa relacijom  $r$  zamislimo da normalizaciju sprovodimo u korak po korak, pri čemu svaki put razrešavamo jedan skup atributa  $(Y_i)$ . Radi preglednosti relaciju nastalu u  $i$ -tom koraku označićemo sa  $r_i$ , mada treba imati na umu da se sve dešava nad jednom relacijom. Neka je  $N$  broj torki u polaznoj relaciji  $r$  i neka su  $N_{i1}, ..., N_{iN(r)}$  redom ukupni brojevi vrednosti u skupovima  $(y_{i1}), ..., (y_{iN(r)})$ . U prvom koraku, imali bi kao rezultat normalizacije po  $(Y1)$  relaciju  $r1$  sa  $N(r1)$  torki

$$r ( X, (Y1), ..., (YM) ) == r1 ( X, Y1, (Y2), ..., (YM) ) \quad N(r1) = N_{11} + ... + N_{1N(r)}$$

dok bi drugi korak normalizacije po  $(Y2)$  dao

$$r1 ( X, Y1, (Y2), ..., (YM) ) == r2 ( X, Y1, Y2, (Y3), ..., (YM) ) \quad N(r2) = N_{21} + ... + N_{2N(r1)}$$

I bez navođenja daljih koraka je jasno šta se dešava: na kraju, naša potpuno normalizovana relacija  $rM$  može imati znatno veći broj torki od polazne.

*Primer*

Posmatrajmo šemu relacije NASLOV koja je rezultat nastojanja da se u jednoj relaciji uz naslove evidentiraju i podaci o autorima i članovima koji su pozajmljivali te naslove jednom ili više puta:

NASLOV ( SIFN, ( SIFA ), ( SIFC ) )

Skup funkcijskih zavisnosti nad tom šemom je:

$F = \{ \text{SIFN} \rightarrow (\text{SIFA}), \text{SIFN} \rightarrow (\text{SIFC}) \}$

Posle dva koraka normalizacije dobijamo šemu relacije

NASLOV ( SIFN, SIFA, SIFC )

i skup netrivialnih funkcijskih zavisnosti

$F2 = \{ \}$

Pošto je netrivialno F2 prazno, zaključili smo da su svi atributi šeme relacije NASLOV2 ključni.

Pogledajmo sada šta se dobija iz polazne relacije **naslov** čiji bi sadržaj, prema podacima iz naše baze podata BIBLIOTEKA, bio sledeći

<b>naslov</b>	<b>( SIFN (SIFA) (SIFC) )</b>
RBPO	{AP0,JN0} {PP0}
RK00	{DM0} {}
PP00	{ZP0,DM0,IT0} {PP0,JJ0,JJ1}
PJC0	{AP1,ZP0} {JJ0}

Obratimo pažnju na to da niko nije pozajmljivao naslov RK00 i da je usled toga odgovarajuća vrednost atributa SIFC prazan skup.

Neka je prvi korak normalizacije sproveden u odnosu na atribut SIFA. Relacija **naslov** će posle toga imati sadržaj:

<b>naslov</b>	<b>( SIFN SIFA (SIFC) )</b>
RBPO	AP0 {PP0}
RBPO	JN0 {PP0}
RK00	DM0 {}
PP00	ZP0 {PP0,JJ0,JJ1}
PP00	DM0 {PP0,JJ0,JJ1}
PP00	IT0 {PP0,JJ0,JJ1}
PJC0	AP1 {JJ0}
PJC0	ZP0 {JJ0}

Kod drugog koraka normalizacije suočavamo se sa specifičnom situacijom. Za naslov RK00 odgovarajući skup vrednosti SIFC je prazan, što ima za posledicu da u relaciji **naslov** ne nastaje ni jedna torka sa 'RK00' kao vrednošću atributa SIFN. Eventualni pokušaj da takvu torku ipak formiramo sa NULL kao vrednošću atributa SIFC bio bi u suprotnosti za uslovom da ni jedan deo ključa ne sme biti NULL vrednost (uslov egzistencijalnog integriteta).

Konačno normalizovana relacija **naslov2** glasi:

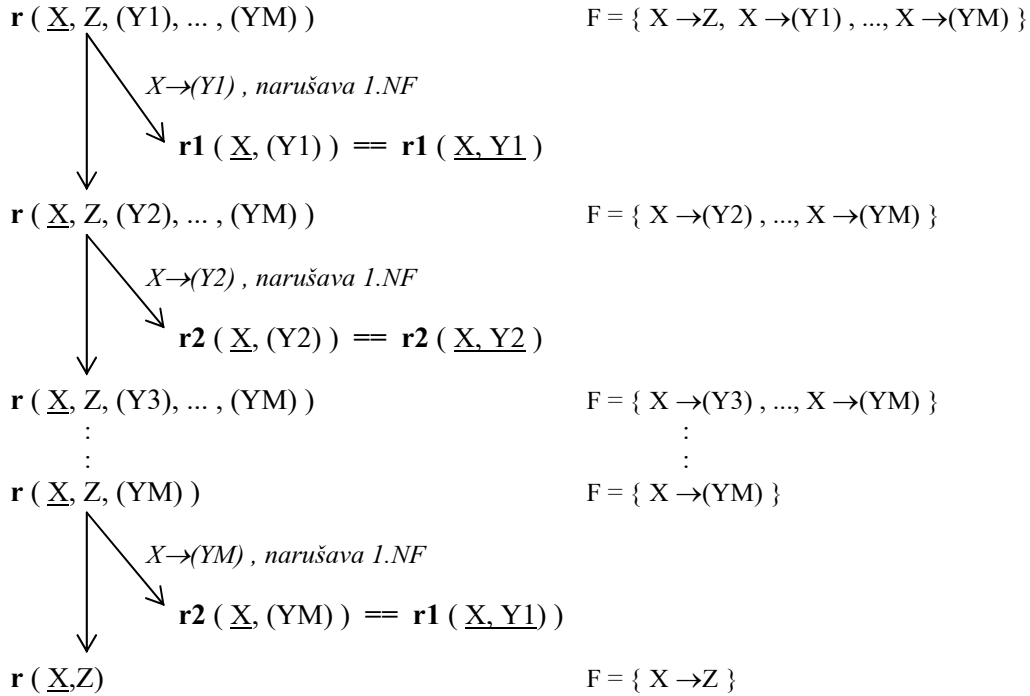
naslov ( SIFN SIFA SIFC )		
-----		
RBP0	AP0	PP0
RBP0	JN0	PP0
PP00	ZP0	PP0
PP00	ZP0	JJ0
PP00	ZP0	JJ1
PP00	DM0	PP0
PP00	DM0	JJ0
PP00	DM0	JJ1
PP00	IT0	PP0
PP00	IT0	JJ0
PP00	IT0	JJ1
PJC0	AP1	JJ0
PJC0	ZP0	JJ0
-----		

Konačna šema relacije **NASLOV** je sa gledišta funkcijskih zavisnosti maksimalno normalizovana i nalazi se u Bojs-Kodovoj normalnoj formi. Cela šema je ujedno i jedini kandidat-ključ, i nad njom važe samo trivijalne funkcijske zavisnosti. Međutim, ako je suditi po sadržaju relacije **naslov**, ova šema je daleko od idealne:

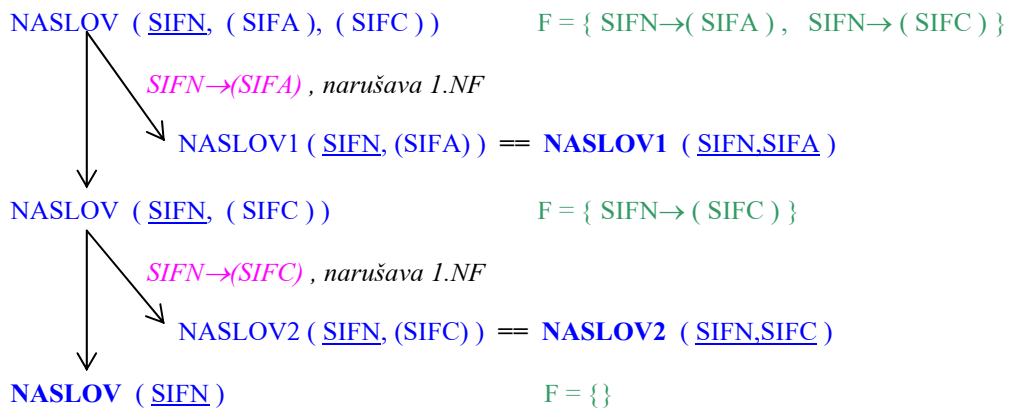
- prisutna je redundansa: unutar markirane grupe torki, podatak o tome da je neko autor nekog naslova mora da je prisutan onoliko puta koliko je članova pozajmljivalo taj naslov; obrnuto, podatak o tome da je neko pozajmljivao neki naslov prisutan je onoliko puta koliko taj naslov ima autora;
- prisutna je anomalija unošenja: podatke o tome ko su autori nekog naslova ne možemo uneti ako niko nije pozajmio taj naslov;
- prisutna je anomalija brisanja: ako uklonimo podatke o pozajmicama nekog naslova, gubimo i podatke o autorstvu tog naslova.

Razlog za navedene anomalije je u tome što smo normalizaciju u prvu normalnu formu sproveli "unarno", transformišući redom jednu te istu relaciju do željene forme kako bi zadovoljili ograničenje skalarnosti atributa koje je u samoj osnovi relacionog modela u tradiconalnom smislu. A kao posledica takvog pristupa vremenom su uvedene dodatne normalne forme - četvrta i peta - koje su komplikovane i potpuno nepotrebne u uslovima univerzalnog relacionog modela.

Uz prethodni postoji i drugi pristup normalizaciji u prvu normalnu formu koji se svodi na postepeno izdvajanje jedne po jedne zavisnosti  $X \rightarrow (Y_i)$ , slično onome što smo imali kod funkcijskih zavisnosti, pri čemu je zadovoljen uslov da dekompozicija bude bez gubitaka (sa  $Z$  smo označili ostale atribute koji su skalari):



U našem konkretnom slučaju šeme relacije NASLOV imali bi





Pri tome bi redom imali odgovarajuće krajnje relacije:

```
naslov ( SIFN (SIFA)          (SIFC)          )
```

```
-----
RBP0 {AP0,JN0}      {PP0}
RK00 {DM0}          {}
PP00 {ZP0,DM0,IT0}  {PP0,JJ0,JJ1}
PJC0 {AP1,ZP0}      {JJ0}
-----
```

↓

```
naslov1 ( SIFN (SIFA)          ) == naslov1( SIFN SIFA )
```

-----	-----
RBP0 {AP0,JN0}	RBP0 AP0
RK00 {DM0}	RBP0 JN0
PP00 {ZP0,DM0,IT0}	RK00 DM0
PJC0 {AP1,ZP0}	PP00 ZP0
-----	PP00 DM0
	PP00 IT0
	PJC0 AP1
	PJC0 ZP0
	-----

```
naslov ( SIFN (SIFC)          )
```

```
-----
RBP0 {PP0}
RK00 {}
PP00 {PP0,JJ0,JJ1}
PJC0 {JJ0}
-----
```

↓

```
naslov2 ( SIFN (SIFC)          ) == naslov2( SIFN SIFC )
```

-----	-----
RBP0 {PP0}	RBP0 PP0
RK00 {}	PP00 PP0
PP00 {PP0,JJ0,JJ1}	PP00 JJ0
PJC0 {JJ0}	PP00 JJ1
-----	PJC0 JJ0
	-----

```
naslov ( SIFN (SIFC)          )
```

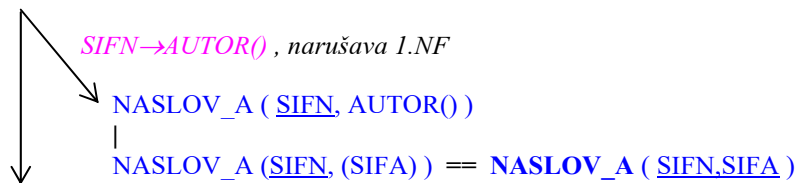
```
-----
RBP0 {PP0}
RK00 {}
PP00 {PP0,JJ0,JJ1}
PJC0 {JJ0}
-----
```

Ako bi želeli da prethodni postupak sprovedemo u skladu sa notacijom univerzalnog relacionog modela izloženom na kraju poglavlja 4, imali bi sledeću polaznu situaciju

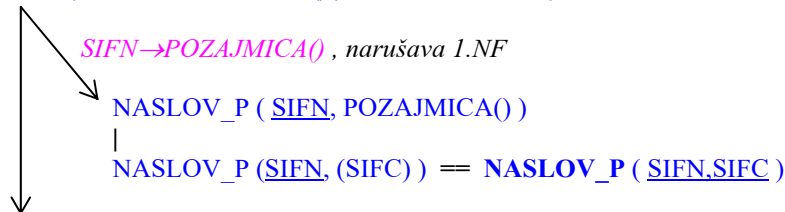
NASLOV ( <u>SIFN</u> , AUTOR(), POZAJMICA() )	F = { SIFN → AUTOR () ,POZAJMICA() }
AUTOR ( <u>SIFA</u> )	F = { }
POZAJMICA ( <u>SIFC</u> )	F = { }

i shodno tome dekompoziciju:

NASLOV ( SIFN, AUTOR(), POZAJMICA() )    F = { SIFN→AUTOR(),POZAJMICA() }



NASLOV ( SIFN, POZAJMICA() )    F = { SIFN→POZAJMICA() }



NASLOV ( SIFN )    F = { }

U ovom primeru atributi-relacije su bili jednostavni, sa samo jednim skalarnim atributom koji je ujedno bio i jedini kandidat-ključ, ali za jednu instancu relacije odnosno unutar jedne torke relacije **naslov**. Jedna vrednost SIFA može je da se pojavi samo jednom u svakoj torci te relacije, ali se može zato javiti u drugim torkama, odnosno unutar cele relacije **naslov** pojaviti više puta.

U slučaju složenih atributa-relacija, sastavljenih od više skalarnih atributa, unutar tog skupa atributa mogu postojati funkcijske zavisnosti, pri čemu su moguća dva slučaja:

- zavisnost je "lokalna":  $X \rightarrow Y$  važi unutar jedne instance relacije, odnosno unutar jedne torke obuhvatajuće relacije, a ne važi izvan toga;
- zavisnost je "globalna":  $X \rightarrow Y$  važi ne samo unutar jedne instance relacije, nego i u svim instancama obuhvatajuće relacije, odnosno u svim njenim torkama; jednoj vrednosti  $X$  u svim torkama obuhvatajuće relacije odgovara jedna vrednost  $Y$ .

Iz prethodnog proizilazi da je za svaki skup zavisnosti potrebno naznačiti ne samo za koju šemu relacije važi, nego i to koje zavisnosti su lokalne a koje globalne. Za lokalne zavisnosti usvajamo uobičajenu oznaku " $\rightarrow$ ", a za globalne " $\gamma \rightarrow$ ".

### 7.4.7 Univerzalni postupak normalizacije

Na osnovu do sada izloženog i usvajajući univerzalni relacioni model, možemo da formulišemo univerzalni postupak normalizacije neke šeme relacije sa atributima-relacijama. Neka je ta šema iz oblika

$$R ( Z, S_1\langle\rangle, \dots, S_j\langle\rangle, \dots, S_m\langle\rangle, R_1(), \dots, R_k(), \dots, R_n() )$$

gde je  $Z$  skup skalarnih atributa,  $S_j\langle\rangle$  su atributi-slogovi a  $R_k()$  atributi-relacije prvo prevedena u oblik

$$R ( X, R_1(), \dots, R_k(), \dots, R_p() )$$

gde  $X$  uz prvobitne skalarne attribute  $Z$  sadrži i skalarne attribute iz svih  $S_j\langle\rangle$ , a niz atributa-relacija  $R_1(), \dots, R_p()$  uz prvobitnih  $n$  sadrži i one koji su se nalazili u sastavu atributa-slogova. Neka nad tom šemom važi skup zavisnosti oblika

$$F = \{ Y_u \rightarrow Z_u \} \cup \{ Y_v \rightarrow R_v() \}$$

gde su  $Y_u$ ,  $Y_v$  i  $Z_u$  podskupovi  $X$ . Neka slično tome važi i za attribute-relacije

$$R_k ( X_k, R_{k1}(), \dots, R_{kj}(), \dots, R_{kp}() )$$

$$F_k = \{ Y_{ku} \rightarrow Z_{ku} \} \cup \{ Y_{kv} \rightarrow R_{kv}() \}$$

pri čemu se na kraju dolazi do relacija koje imaju samo skalarne attribute

$$R_{k..q} ( X_{k..q} )$$

$$F_{k..q} = \{ Y_{k..q} \rightarrow Z_{k..q} \}$$

Sam univerzalni postupak se sastoji u sledećem:

1. Primenom algoritma *KandidatKljučevi* za skup zavisnosti  $F$  određuje se skup kandidat-ključeva  $K$  za šemu  $R$  i bira jedan kao primarni ključ  $P$ .
2. Na osnovu datog skupa zavisnosti  $F$  i dobijenog skupa kandidat-ključeva vrši se normalizacija šeme  $R$  u željenu normalnu formu izdvajanjem svake zavisnosti  $Y_u \rightarrow Z_u$  koja je narušava, pri čemu za svaku takvu zavisnosti nastaje nova šema relacije  $R_{Nu}(\underline{Y}_u, Z_u)$  a iz šeme relacije  $R$  nestaje atribut  $Z_u$ .
3. Na osnovu datog skupa zavisnosti  $F$  i dobijenog skupa kandidat-ključeva vrši se normalizacija šeme  $R$  preostale posle koraka 2 u željenu normalnu formu izdvajanjem svake zavisnosti  $Y_v \rightarrow R_v()$  koja je narušava, pri čemu za svaku takvu zavisnosti nastaje nova šema relacije  $R_{Nv}(\underline{Y}_v, R_v())$  a iz šeme relacije  $R$  nestaje atribut-relacija  $R_v()$ .
4. Na osnovu datog skupa zavisnosti  $F$  i dobijenog skupa kandidat-ključeva vrši se normalizacija šeme  $R$  preostale posle koraka 3 u prvu normalnu formu izdvajanjem svake zavisnosti  $P \rightarrow R_v()$  koja je narušava, pri čemu za svaku takvu zavisnosti nastaje nova šema relacije  $R_{Nv}(\underline{P}, R_v())$  a iz šeme relacije  $R$  nestaje atribut-relacija  $R_v()$ .
5. Svaka šema relacije  $R_{Nv}(\underline{P}, R_v())$  nastala koracima 3 i 4 prevodi se u prvu normalnu formu do strukture  $(P, X_v, R_{v1}(), \dots, R_{vj}(), \dots, R_{vp}())$  pri čemu se odgovarajući skup zavisnosti  $F_v = \{ Y_{vu} \rightarrow Z_{vu} \} \cup \{ Y_{vw} \rightarrow R_{vw}() \}$  transformiše dopunom levih strana lokalnih zavisnosti sa  $Y_v$  odnosno  $P$ .
6. Sekvenca postupaka 1 do 4 se ponavlja za svaku novonastalu šemu relacije sve do sekvence bez efekta.

U svemu ovome bitan je redosled kod koga se u šemi relacije prvo izvrši sva normalizacija na osnovu funkcijskih zavisnosti a tek onda normalizacija u prvu normalnu formu. U suprotnom, neophodna je primena postupaka normalizacije za četvrtu i petu normalnu formu.

*Primer*

Posmatrajmo šemu relacije sa atributom-slogom i atributima-relacijama

R ( SIFN, NAZIVN, OBLAST<>, AUTOR(), POZAJMICA() )

OBLAST < SIFO, NAZIV >

AUTOR ( SIFA, IMEA, KOJI )

POZAJMICA ( SIFC, TELEFON() )

TELEFON ( BROJ )

gde su odgovarajući skupovi netrivialnih zavisnosti, na nivou naznačenih relacija

$F_R = \{ SIFN \rightarrow NAZIVN, OBLAST<>, AUTOR () , POZAJMICA () \}$

$F_{OBLAST} = \{ SIFO \rightarrow NAZIVO, NAZIVO \rightarrow SIFO \}$

$F_{AUTOR} = \{ SIFA \gamma \rightarrow IMEA, SIFA \rightarrow KOJI \}$

$F_{POZAJMICA} = \{ SIFC \gamma \rightarrow TELEFON() \}$

$F_{TELEFON} = \{ \}$

Kao prvo, ravojem jedinog atributa-sloga dobijamo šemu relacije

$R ( \underline{SIFN}, NAZIVN, SIFO, NAZIVO, AUTOR(), POZAJMICA() )$

i objedinjeni skup zavisnosti za tu šemu

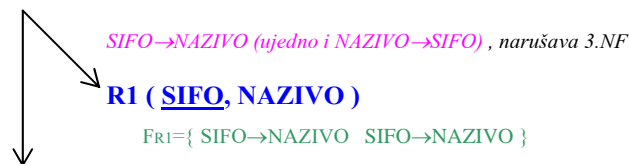
$FR = \{ SIFN \rightarrow NAZIVN, SIFO, NAZIVO, AUTOR(), POZAJMICA(), SIFO \rightarrow NAZIVO, NAZIVO \rightarrow SIFO \}$

Nakon toga, korakom 1 utvrđujemo da je jedini kandidat-ključ i ujedno primarni ključ šeme  $R$  atribut  $SIFN$ .

U okviru koraka 2 nad šemom  $R$  izdvajamo jedinu zavisnost koja narušava 3. a time i BC normalnu formu:

$R ( \underline{SIFN}, NAZIVN, SIFO, NAZIVO, AUTOR(), POZAJMICA() )$

$FR = \{ SIFN \rightarrow NAZIVN, SIFO, NAZIVO, AUTOR(), POZAJMICA(), SIFO \rightarrow NAZIVO, NAZIVO \rightarrow SIFO \}$



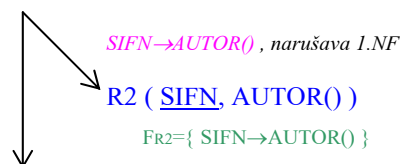
$R ( \underline{SIFN}, NAZIVN, SIFO, AUTOR(), POZAJMICA() )$

$FR = \{ SIFN \rightarrow NAZIVN, SIFO, AUTOR(), POZAJMICA() \}$

Ovim smo od polaznog  $R$  dobili dekompoziciju  $\{R, R_1\}$ . Korak 3 u ovom konkretnom slučaju nema efekata, dok korak 4 dovodi do izdvajanja svih zavisnosti oblika  $P \rightarrow R_v()$  u posebne šeme sa atributima-relacijama:

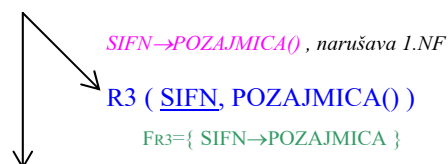
$R ( \underline{SIFN}, NAZIVN, SIFO, AUTOR(), POZAJMICA() )$

$FR = \{ SIFN \rightarrow NAZIVN, SIFO, AUTOR(), POZAJMICA() \}$



$R ( \underline{SIFN}, NAZIVN, SIFO, POZAJMICA() )$

$FR = \{ SIFN \rightarrow NAZIVN, SIFO, POZAJMICA() \}$



$R ( \underline{SIFN}, NAZIVN, SIFO )$

$FR = \{ SIFN \rightarrow NAZIVN, SIFO \}$

Sada imamo dekompoziciju  $\{R, R_1, R_2, R_3\}$ . Preostaje još da u okviru koraka 5 šeme  $R_2$  i  $R_3$  prevedemo u prvu normalnu formu:

$$R_2 ( \underline{\text{SIFN}}, \text{AUTOR}() ) == R_2 ( \text{SIFN}, \text{SIFA}, \text{IMEA}, \text{KOJI} )$$

$$FR_2 = \{ \text{SIFA} \rightarrow \text{IMEA}, \text{SIFN}, \text{SIFA} \rightarrow \text{KOJI} \}$$

$$R_3 ( \underline{\text{SIFN}}, \text{POZAJMICA} ) == R_3 ( \text{SIFN}, \text{SIFC}, \text{TELEFON}() )$$

$$FR_3 = \{ \text{SIFC} \rightarrow \text{TELEFON}() \}$$

Posle prvog prolaza kroz sve korake dobijena dekompozicija je  $\{R, R_1, R_2, R_3\}$ . Sada preostaje da prethodno izvedeni prolaz ponovimo za svaku šemu relacije iz dekompozicije. Kao prvo određujemo skupove kandidat-ključeva svih šema:

$$K = \{ \text{SIFN} \} \quad K_1 = \{ \text{SIFO}, \text{NAZIVO} \} \quad K_2 = \{ \text{SIFA}, \text{SIFN} \} \quad K_3 = \{ \text{SIFN}, \text{SIFC} \}$$

Na osnovu toga i odgovarajućih skupova zavisnosti uočavamo:

- šeme  $R$  i  $R_1$  su u prvoj i BC normalnoj formi i kao takve su konačne;
- šema  $R_2$  jeste u prvoj ali nije u BC normalnoj formi;
- šema  $R_3$  nije ni u prvoj ni u BC normalnoj formi.

Ponavljanjem koraka 2 do 5 (treba samo 2) za šemu  $R_2$  dobijamo:

$$R_2 ( \underline{\text{SIFN}}, \underline{\text{SIFA}}, \text{IMEA}, \text{KOJI} )$$

$$FR_2 = \{ \text{SIFA} \rightarrow \text{IMEA}, \text{SIFN}, \text{SIFA} \rightarrow \text{KOJI} \}$$

*SIFA → IMEA, narušava 2.NF*

$$R_{21} ( \underline{\text{SIFA}}, \text{IMEA} )$$

$$FR_{21} = \{ \text{SIFA} \rightarrow \text{IMEA} \}$$

↓

$$R_2 ( \underline{\text{SIFN}}, \underline{\text{SIFA}}, \text{KOJI} )$$

$$FR_2 = \{ \text{SIFN}, \text{SIFA} \rightarrow \text{KOJI} \}$$

Ponavljanje koraka 2 do 5 (treba 2, 4 i 5) daje:

$$R_3 ( \underline{\text{SIFN}}, \underline{\text{SIFC}}, \text{TELEFON}() )$$

$$FR_3 = \{ \text{SIFC} \rightarrow \text{TELEFON}() \}$$

*SIFC → TELEFON(), narušava 1. i 2.NF*

$$R_{31} ( \text{SIFC}, \text{TELEFON}() ) == R_{31} ( \underline{\text{SIFC}}, \underline{\text{BROJ}} )$$

$$FR_{31} = \{ \text{SIFC} \rightarrow \text{TELEFON}() \} \quad FR_{31} = \{ \}$$

↓

$$R_3 ( \underline{\text{SIFN}}, \underline{\text{SIFC}} )$$

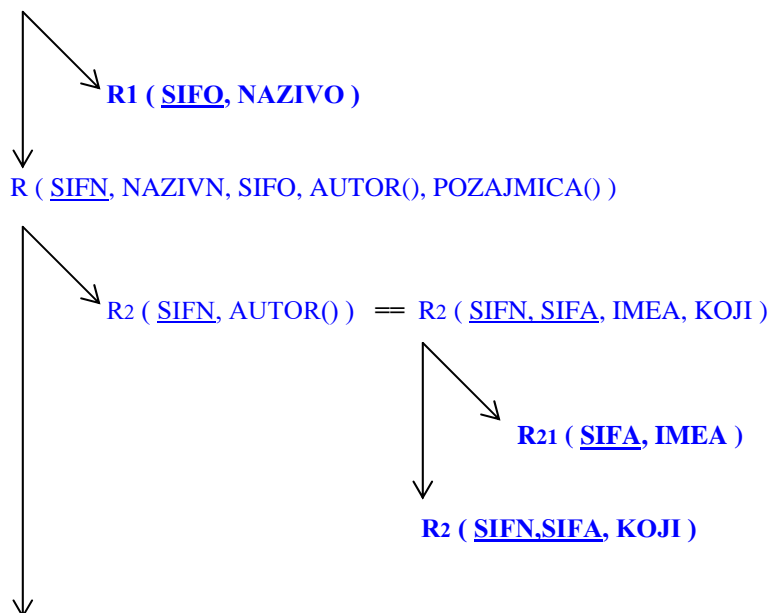
$$FR_3 = \{ \}$$



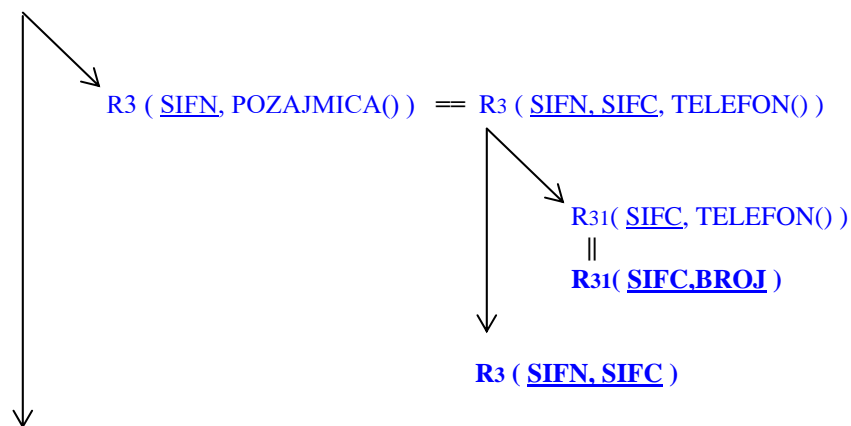
Kao rezultat svega ovoga dobili smo dekompoziciju  $\{R, R_1, R_2, R_{21}, R_3, R_{31}\}$ . Uvidom u skupove zavisnosti i skupove kandidat-ključeva zaključujemo da naredni prolazi nizu potrebni, pošto su sve šeme u dekompoziciji i u BC i u prvoj normalnoj formi.

Na kraju ovog primera, prikažimo čitav postupak normalizacije i integralno:

$R ( \underline{SIFN}, NAZIVN, SIFO, NAZIVO, AUTOR(), POZAJMICA() )$



$R ( \underline{SIFN}, NAZIVN, SIFO, POZAJMICA() )$

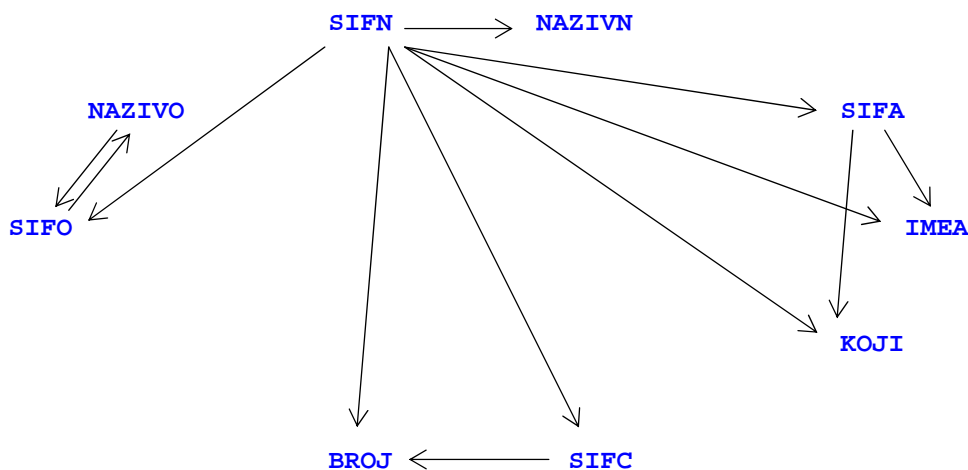


$R ( \underline{SIFN}, NAZIVN, SIFO )$

Preostaje da razmotrimo još jedno pitanje u okviru univerzalnog pristupa problemu normalizacije. Kako bi bilo potpuno jasno o čemu se radi, konstruišimo za prethodni primer sa razvijenim atributima-relacijama graf zavisnosti na sledeći način:

- svaki atribut šeme relacije  $R$  postaje čvor;
- za svaku zavisnost is skupa  $F \rightarrow X \rightarrow Y$  gde  $Y$  ili njegovi delovi mogu biti i relacije od svakog atributa sadržanog u  $X$  povlačimo poteg ka svakom atributu sadržanom u  $Y$ .

Navedenim postupkom dobijamo za naš primer



Na osnovu ovog dijagrama uočavamo da su svi čvorovi obuhvaćeni jednim grafom zavisnosti, odnosno da ne postoje izolovani podgrafovi i čvorovi. U tom smislu, “univerzalna” polazna relacija  $R$  je imala opravdanja i mogla je odmah da se normalizuje, pošto su svi atributi u njenom sastavu bili nekako povezani, jednoznačno ili višeznačno, direktno ili indirektno.

Posmatrajmo sada slučaj izmenjene polazne šeme relacije  $R$

$R ( \underline{SIFN}, NAZIVN, SIFO, NAZIVO, ( SIFA, IMEA, KOJI ), ( SIFC, (BROJ) ) )$

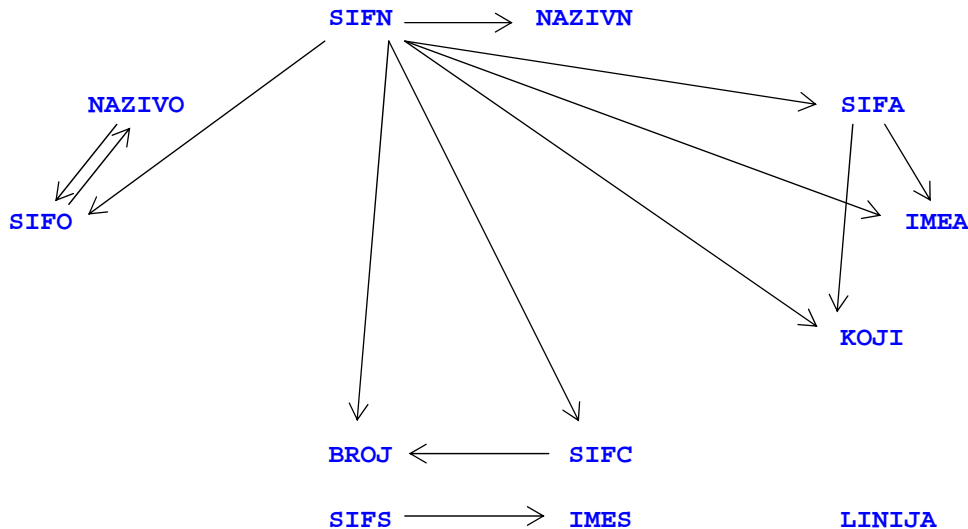
kojoj dodajemo još i attribute šifre škole, naziva škole i autobuske linije

$SIFS, NAZIVS$  i  $LINIJA$

čime “po prirodi stvari” dodajemo u  $F$  i zavisnost

$SIFS \rightarrow NAZIVS$

Ako sada za takav primer sastavimo graf zavisnosti dobijamo:



Graf zavisnosti sada ima tri izolovana podgrafa.

Nije teško zaključiti da u ovom slučaju sa polaznom šemom relacije R nešto nije u redu. Naime, jednom te istom šemom relacije obuhvatili smo podatke biblioteke, podatke o školama koje sa time nemaju nikakve veze i podatke o autobuskim linijama koje ni sa jednim od prethodnog nemaju nikakve veze. Posledica takve loše strukture su brojne anomalije. Primera radi, pošto bi u praksi sigurno bilo najviše naslova pa autobuskih linija pa škola, višak redova u odnosu na autobuske linije i škole morali bi da popunimo NULL-vrednostima. U slučaju da umesto autobuskih linija imamo autobuse moglo bi se dogoditi da vremenom njihov broj postane jednak broju naslova i od tog trenutka ne bi mogli da evidentiramo ni jedan novi autobus pošto bi time narušili integritet kandidat-ključa SIFN (ili bi za naslove morali da unesemo NULL-vrednosti ili bi morali da ponovimo vrednosti za neki naslov).

Šta možemo da zaključimo iz prethodnog razmatranja? Kao prvo, u jednu šemu relacije ne treba veštački stavljati podatke koji nemaju nikakve uzajamne veze. Kao drugo, kod normalizacije neke šeme relacije neophodno je prvo primeniti postupak tzv. “prednormalizacije” kojim će se polazna šema relacije na osnovu grafa zavisnosti dekomponovati na odgovarajući broj šema relacije. Kod primera kojeg smo upravo razmatrali, dobili bi tri šeme relacije:

**R1 ( SIFN, NAZIVN, SIFO, NAZIVO, ( SIFA, IMEA, KOJI ), ( SIFC,BROJ ) )**

**R2 ( SIFS, NAZIVS)    R3 ( LINIJA )**

# 8

## ***PROJEKTOVANJE RELACIONE BAZE PODATAKA***

---

Iz sadržaja dosadašnjih poglavlja uverili smo se u valjanost šeme relacione baze podataka BIBLIOTEKA (Prilog A) koja nam je do sada služila kao osnovni primer. Niti smo nailazili na smetnje pri formulisanju upita nad tom bazom podataka, niti je u njoj prisutna i jedna od anomalija koje smo naveli u prethodnom poglavlju. Sa druge strane, pojedine izmenjene šeme relacija iz te baze podataka poslužile su nam za ilustraciju tih anomalija. Jednom rečju, šema relacione baze podataka BIBLIOTEKA je savršena u svakom pogledu.

Pitanje koje se prirodno nameće je: *kako* je nastala šema relacione baze podataka BIBLIOTEKA? Da li na osnovu osećaja i slobodne procene ili na neki drugi način? Kako u praksi nastaju isto tako savršene šeme relacionih baza podataka, ali sa stotinama ili više relacija?

Odgovor na sva ta pitanja je kratak i jasan i glasi: savršenost šeme relacione baze podataka postiže se pažljivim postupkom projektovanja, pri čemu se striktno poštuju određena pravila. Sam postupak projektovanja svakog informacionog sistema je dvojakog karaktera i obuhvata:

- projektovanje podataka, odnosno šeme relacione baze podataka;
- projektovanje postupaka, odnosno procesa za održavanje i korišćenje podataka.

Pri tome je od najvećeg značaja prvi vid projektovanja, iz razumljivih razloga: nikakvo projektovanje postupaka ne može da otkloni nedostatke i anomalije loše šeme relacione baze podataka, dok se posledice lošeg projektovanja postupaka nad valjanom šemom mogu naknadno otkloniti.

U ovom poglavlju razmatraćemo oba vida projektovanja relacione baze podataka odnosno informacionog sistema zasnovanog na njoj. Predmet našeg interesovanja biće prvo metod projektovanja podataka poznat pod nazivom "model objekata i odnosa". Ovaj model čine: skup grafičkih simbola, skup pravila korektnosti i skup pravila prevođenja u model nižeg reda koji se direktno može prevesti u odgovarajuće strukture podataka. Nakon toga osvrnućemo se na projektovanje postupaka metodom funkcionalne dekompozicije i algoritamske specifikacije.

## 8.1 Projektovanje podataka

Kao što je već navedeno u poglavlju 3, model objekata i odnosa spada u grupu modela koji su semantički višeg reda, i predstavlja mešavinu modela sistema i modela podataka. Preciznije rečeno, model objekata i odnosa se nalazi "iznad" relacionog modela podataka, pa kao takav sadrži i pravila prevođenja u relacioni model. Važno je imati na umu da se ovim modelom predstavljaju *klase* objekata i odnosi između njih, a ne pojedinačne instance objekata.

Razmatranje modela objekata i odnosa koje sledi sprovedeno je u skladu sa podelom entiteta na specijalne slučajeve objekata i veza. Na kraju, izložen je postupak prevođenja tog modela u relacioni model i dat je primer projektovanja relacione baze podataka BIBLIOTEKA.

### 8.1.1 Nezavisne klase objekata

#### *Definicija*

Nezavisna klasa objekata u posmatranom sistemu je svaka klasa objekata čije instance egzistiraju nezavisno od instanci drugih klasa objekata u sistemu.

Navedimo dva primera za naš sistem BIBLIOTEKA:

- klasa CLAN je nezavisna, pošto članovi kao osobe egzistiraju potpuno nezavisno od instanci drugih klasa;
- klasa NASLOV je takođe nezavisna: na prvi pogled, egzistencija naslova uvek zavisi od autora, ali to nije tako - primer za to je naslov koga izdavač prvo osmisli a zatim nalazi autore koji će da ga napišu.

Grafički simbol za nezavisnu klasu objekata je pravougaonik iscertan jednostrukom linijom. Unutar tog pravougaonika u pregrađenom delu pri vrhu ispisujemo velikim slovima naziv klase, a u delu ispod toga ispisujemo jedan ispod drugog nazive klasifikacionih svojstava, pri čemu podvlačimo identifikator (klasifikaciono svojstvo koje svojom vrednošću identifikuje svaku instancu u klasi).

Pravilo prevođenja nezavisne klase objekata u odgovarajući koncept relacionog modela je jednostavno:

- od nezavisne klase objekata nastaje šema relacije istog naziva i sa atributima koji odgovaraju klasifikacionim svojstvima, pri čemu je atribut koji odgovara identifikatoru primarni ključ.

#### *Primer*

Za klasu CLAN u našem primeru biblioteke imamo sledeći simbol modela objekata i odnosa i odgovarajuću šemu relacionog modela:



## 8.1.2 Zavisne klase objekata

### *Definicija*

Zavisna klasa objekata u posmatranom sistemu je svaka klasa objekata čije instance egzistiraju zavisno od konstantnog broja instanci drugih klasa objekata u sistemu.

Iz prethodne definicije zaključujemo da modelom objekata i odnosa ne može da se predstavi zavisnost od promenljivog broja instanci. Tako, u našem primeru biblioteke nismo u mogućnosti da klasu NASLOV predstavimo zavisnom od klase AUTOR, pošto pojedini naslovi imaju različiti broj autora.

Sistem BIBLIOTEKA naveden u poglavlju 1 je suviše jednostavan i kao takav ne može da posluži kao primer za zavisne klase objekata, ali ga u tu svrhu možemo dopuniti (što je i urađeno u prilogu A) praćenjem sledećih događaja:

- POZAJMICA: skup svojstava koji govori o tome da je određeni član pozajmio određenu knjigu određenog naslova i određeni broj dana;
- REZERVACIJA: skup svojstava koji govori o tome da je određeni član rezervisao određeni naslov određenog datuma.

Navedena dve klase su zavisne, i to iz sledećih razloga:

- ni jedna pozajmica ne može da egzistira nezavisno od člana koji ju je izvršio i knjige koja je pozajmljena;
- ni jedna rezervacija ne može da egzistira nezavisno od člana koji ju je izvršio i naslova koji je rezervisan.

Uz to, za obe klase važi da su zavisne od po dve klase objekata. To ne mora biti uvek slučaj: zavisnost može biti od jedne, dve ili više klasa.

Zavisnost klase objekata može biti dvojaka, prema stepenu zavisnosti:

- egzistencijalna: instance zavisne klase poseduju identifikaciono svojstvo, a zavisna je samo njihova egzistencija;
- identifikaciona: instance zavisne klase ne poseduju identifikaciono svojstvo, i u njihovoj identifikaciji učestvuju i identifikatori klasa od kojih zavise.

Identifikaciona zavisnost je u većini slučajeva nepoželjna, pošto nameće problem određivanja skupa identifikacionih svojstava a uz to može biti uzrok problema u obezbeđivanju dinamičkih uslova integriteta u bazi podataka. Takva zavisnost se uvek može izbeći time što se za zavisnu klasu objekata veštački uvodi identifikaciono svojstvo u vidu rednog broja (brojača nastanka) instance.



Zavisna klasa objekata se predstavlja pravougaonikom iscrtanim dvostrukom linijom. Unutar njega u pregrađenom delu ispisujemo velikim slovima naziv klase, a u delu ispod toga ispisujemo jedan ispod drugog nazive klasifikacionih svojstava. Ako postoji identifikator, podvlačimo ga, kao i kod nezavisne klase objekata. Ako postoji jedno ili više svojstava koji uz identifikatore uslovljavajućih klasa učestvuju u identifikaciji, podvlačimo ih isprekidano. Na kraju, usmerenim isprekidanim linijama spajamo takav pravougaonik sa pravougaonicima koji odgovaraju svim klasama od kojih postoji zavisnost. Uz te linije naznačujemo vrste zavisnosti: E ili ništa za egzistencijalnu, I za identifikacionu.

Zavisnost između klasa objekata je tipičan slučaj neposrednog osnosa klasa, a kako su klase skupovi instanci objekata taj odnos možemo posmatrati i kao odnos preslikavanja između dva skupa. Na osnovu toga, mogu se za svaku zavisnost od neke klase definisati dve vrste kardinalnosti preslikavanja:

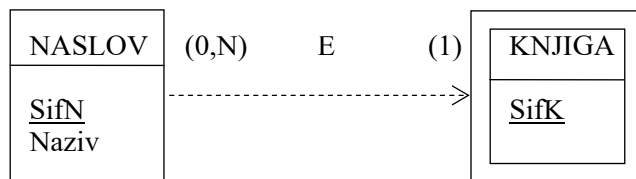
- kardinalnost uslovljavanja: par brojeva  $(m,n)$  koji se zapisuje uz klasu od koje je zavisnost i koji ima sledeće značenje: svaka instanca te klase mora da uslovljava bar  $m$  a može najviše  $n$  instanci zavisne klase;
- kardinalnost uslovljenosti: broj  $(m)$  koji se zapisuje uz zavisnu klasu i koji ima sledeće značenje: svaka instanca te klase mora biti zavisna od tačno  $m$  instanci klase od koje zavisi.

Pravilo prevođenja za zavisnu klasu objekata je znatno složenije nego za slučaj nezavisne klase:

- pre prevođenja zavisne klase objekata u odgovarajući koncept relacionog modela, mora biti sprovedeno takvo prevođenje za sve klase od kojih postoji zavisnost;
- od zavisne klase objekata nastaje šema relacije istog naziva, a attribute šeme pored klasifikacionih svojstava te klase čine i primarni ključevi šema relacija svih klasa od kojih postoji zavisnost; svaki primarni ključ uzima se onoliko puta kolika je kardinalnost odgovarajuće uslovljenosti, uz eventualnu promenu naziva;
- ako je zavisnost egzistencijalna (sve zavisnosti od drugih klasa su egzistencijalne), identifikator zavisne klase objekata postaje primarni ključ nastale šeme relacije; u suprotnom, za primarni ključ se uzima ona minimalna kombinacija atributa koja se po prirodi stvari može javiti samo jednom.

*Primer:*

U proširenom sistemu biblioteke, koji je poslužio kao osnova za bazu podataka BIBLIOTEKA, klasa KNJIGA je zavisna od klase NASLOV, s obzirom na to da ne postoji "prazna" knjiga i da svaka knjiga sadrži tačno jedan naslov. Ako sa SifK označimo identifikator (inventarni broj knjige), imamo:



⇓

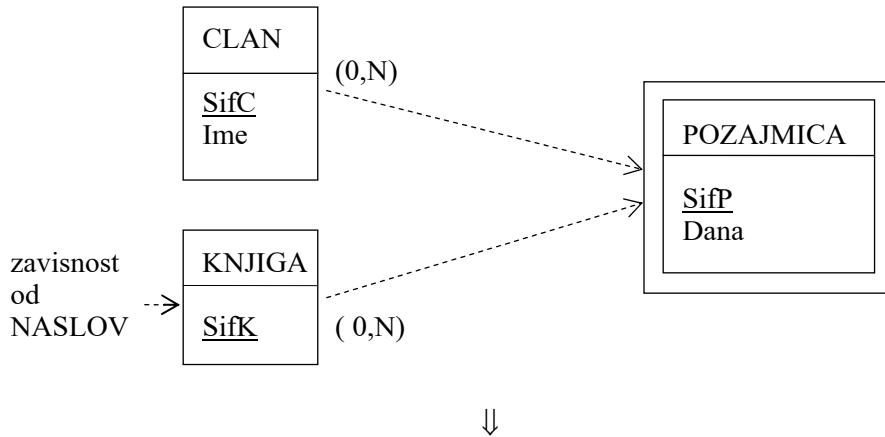
NASLOV ( SifN, Naziv )

KNJIGA ( SifK, SifN )

Uobičajeno je da se egzistencijalna zavisnost i kardinalnost uslovljenosti (1) posebno ne naznačuju, pa ćemo se toga pridržavati u narednim primerima.

Primer:

Za već opisanu situaciju zavisne klase POZAJMICA imamo:



CLAN ( SifC, Ime )

POZAJMICA ( SifP, SifC, SifK, Dana )

KNJIGA ( SifK, SifN )

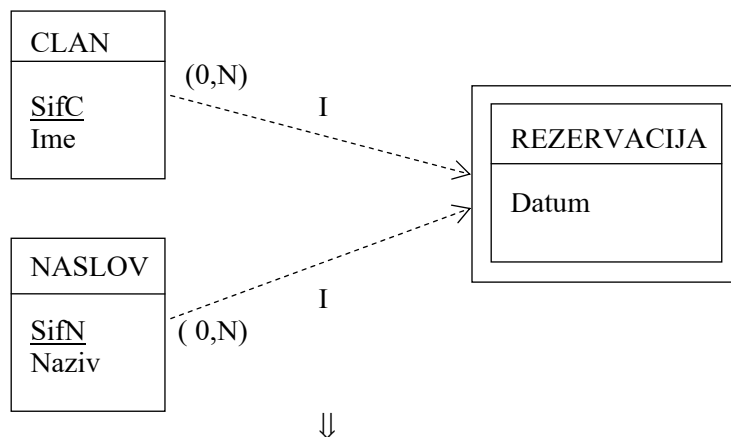
Ovde su neophodne određene napomene:

- šema relacije KNJIGA sadrži atribut SifN na osnovu zavisnosti od klase NASLOV, što nije u celini prikazano;
- ni jedna kombinacija atributa SifC, SifK i Dana nema osobinu unikatnog pojavljivanja vrednosti, pa nije mogla da posluži kao primarni ključ (moguće je, mada malo verovatno, da isti član pozajmi istu knjigu u trajanju od istog broja dana, pogotovo ako je to jedini primerak određenog naslova);
- prethodna situacija je prevaziđena tako što smo za klasu POZAJMICA uveli identifikator SifP sa značenjem rednog broja pozajmice.

Prethodna dva primera odnosila su se na egzistencijalne zavisnosti.

*Primer:*

Za zavisnu klasu objekata REZERVACIJA koju smo već opisali imamo:



CLAN ( SifC, Ime )

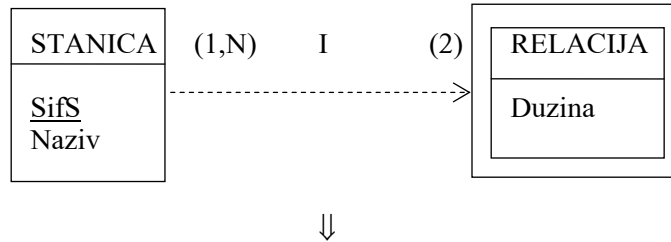
REZERVACIJA ( SifC, SifN, Dana )

NASLOV ( SifN, Naziv )

Ovde imamo primer identifikacione zavisnosti, a unikatnost kombinacije atributa SifC i SifN je jasna: niko ne rezerviše isti naslov dva puta.

*Primer:*

Posmatrajmo deo sistema železnice, u kome postoje stanice, a između po dve stanice relacije sa svojstvom dužine:



STANICA ( SifS, Naziv )

RELACIJA ( Duzina, SifSPoc, SifSZad )

Ovde imamo identifikacionu zavisnost sa kardinalnošću uslovljenosti 2, pa se zato primarni ključ šeme STANICA javlja dva puta u šemi RELACIJA. Pri tome smo bili u obavezi da promenimo naziv za bar jedno pojavljivanje ta dva atributa, ali smo promenili oba radi preglednosti, da bi naglasili šta je početna a šta zadnja stanica.



### 8.1.3 Specijalizacija i generalizacija

*Definicija:*

Za neku klasu objekata (podklasu) kažemo da predstavlja specijalizaciju ako predstavlja specijalan slučaj neke druge klase objekata (nadklase) po bar jednom od sledeća dva kriterijuma:

- ima specifična klasifikaciona svojstva;
- ima specifične odnose sa drugim klasama objekata.

Pored značenja specijalnog slučaja neke nadklase objekata, specijalizacija kao pojam ima i značenje postupka pri kome uočavamo da neka klasa objekata, nadklasa, ima po bilo kom od navedenih kriterijuma specijalne slučajeve. Pri tome, nadklasa će imati sva klasifikaciona svojstva koja su zajednička za sve specijalne slučajeve, dok će podklase imati, ako je to osnov specijalizacije, samo specifična svojstva.

Specijalizacija se u okviru modela objekata i odnosa predstavlja tako što se i za nadklasu i za podklase koristi dosadašnji način označavanja pomoću pravougaonika odgovarajućeg tipa i sadržaja, s tim što se od nadklase povlači puna linija koja preko trougla vodi do jedine podklase ili se grana do svih podklasa, ako ih ima više. Za nadklasu se navodi identifikator i sva svojstva koja su zajednička za sve podklase. Za svaku podklasu navode se samo njoj specifična svojstva, ako ih ima.

I specijalizacija predstavlja slučaj neposrednog odnosa klasa, odnosno preslikavanja instanci nadklase na instance podklasa, pa se i ovde definiše odgovarajuća kardinalnost:

- kardinalnost specijalizacije: par brojeva  $(m,n)$  koji se zapisuje uz nadklasu koja specijalizuje i koji ima sledeće značenje: svaka instanca nadklase specijalizira na po jednu instancu u najmanje  $m$  i najviše  $n$  podklasa.

Jasno je da  $n$  ne može biti veće od broja podklasa specijalizacije. Prema kardinalnosti specijalizacije mogu se izvesti dve podele. Prva je po  $m$  i obuhvata dva slučaja:

- parcijalna (neobavezna) specijalizacija: slučaj kada je  $m=0$ , odnosno kada u nadklasi mogu da postoje instance koje ne učestvuju u specijalizaciji;
- totalna (obavezna) specijalizacija: slučaj kada je  $m>0$  (a najčešće 1), odnosno kada svaka instanca u nadklasi mora da specijalizuje u instance bar  $m$  podklasa.

Druga podela je na osnovu  $n$  i obuhvata takođe dva slučaja:

- ekskluzivna (isključujuća) specijalizacija: slučaj kada je  $n=1$ , odnosno kada svaka instanca nadklase može da specijalizuje u instancu najviše jedne podklase;
- inkluzivna (uključujuća) specijalizacija: slučaj kada je  $n>1$ , odnosno kada svaka instanca nadklase može da specijalizuje u instance najviše  $n$  podklasa.

**Pravilo prevođenja specijalizacije** u odgovarajuće koncepte relacionog modela je jednostavno:

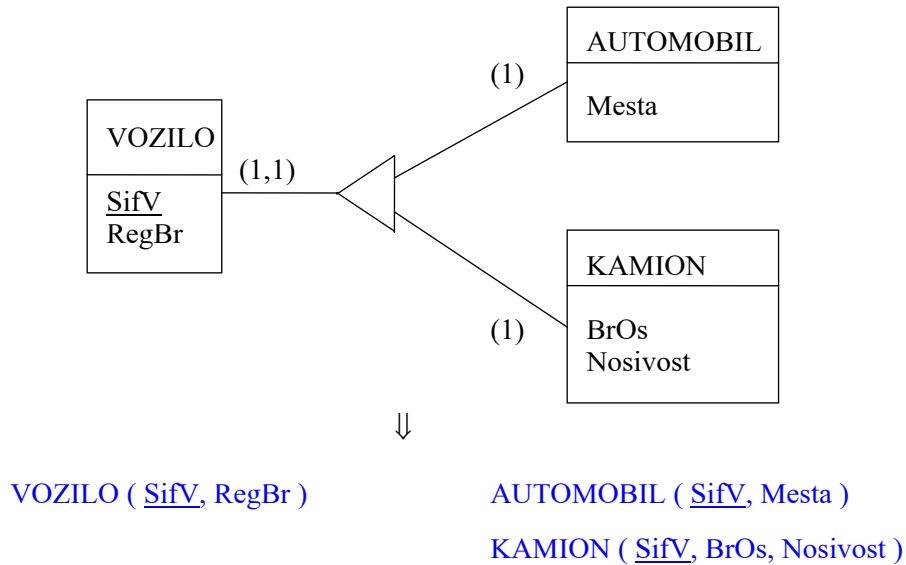
- od nadklase objekata koja specijalizuje nastaje šema relacija istog naziva i sa atributima koji odgovaraju klasifikacionim svojstvima, pri čemu je atribut koji odgovara identifikatoru primarni ključ;
- od svake podklase objekata nastaje šema relacije istog naziva, a attribute te šeme čine klasifikaciona svojstva podklase i primarni ključ šeme koja odgovara nadklasi, koji postaje i primarni ključ šeme podklase.

Jasno je da pre formiranja šema relacija podklasa moramo doći do šeme nadklase. U retkim slučajevima, moguća je i specijalizacija zavisnih klasa objekata. Isto tako, mogu postojati lanci specijalizacije sa dva i više nivoa.



*Primer:*

Posmatrajmo klasu vozila sa svojstvima šifre i registarskog broja, koja kao specijalne slučajeve sadrži automobile sa dodatnim svojstvom broja mesta i kamione sa dodatnim svojstvima broja osovina i nosivosti. Za opisanu situaciju imamo sledeće:

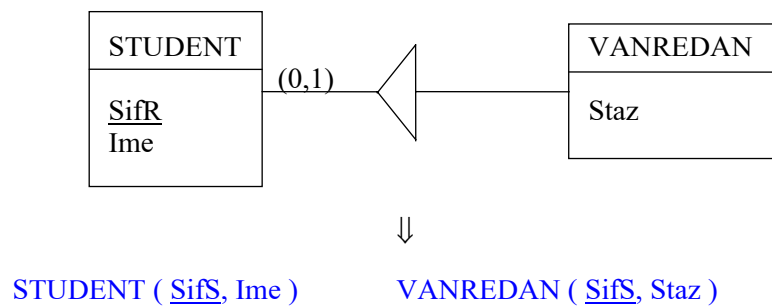


Ovo je primer totalne i ekskluzivne specijalizacije, što je i naznačeno kardinalnošću (1,1), iz razumljivih razloga: svako vozilo mora biti ili automobil ili kamion, ali nikako oboje istovremeno. Kardinalnost preslikavanja sa strane podklasa je uvek (1) i podrazumeva se, pa je više nećemo navoditi.



*Primer:*

Ovaj primer se odnosi na okolnost da su neki od studenata vanredni i da kao zaposleni imaju svojstvo radnog staza, a specijalizacija je parcijalna i vrši se na samo jednu podklasu:



Generalizacija je postupak koji je obrnut u odnosu na specijalizaciju i daje isti krajnji rezultat, odnosno istu nadklasu i podklase.

*Definicija:*

Za neku klasu objekata (nadklasu) kažemo da predstavlja generalizaciju ako predstavlja opšti slučaj jedne ili više drugih klasa objekata (podklase) po zajedničkim klasifikacionim svojstvima.

Pored toga što predstavlja opšti slučaj nekih podklasa objekata, generalizacija kao pojam ima i značenje postupka pri kome uočavamo da neke klase objekata, podklase, imaju po navedenom kriterijumu nešto zajedničko što se može pridružiti jednoj nadklasi. Pri tome, nadklasa će imati sva klasifikaciona svojstva koja su zajednička za sve podklase, a podklase će imati samo specifična svojstva.

U poređenju sa specijalizacijom, **pravilo prevođenja generalizacije** u odgovarajuće komponente relacionog modela ima na samom početku dva dodatna koraka:

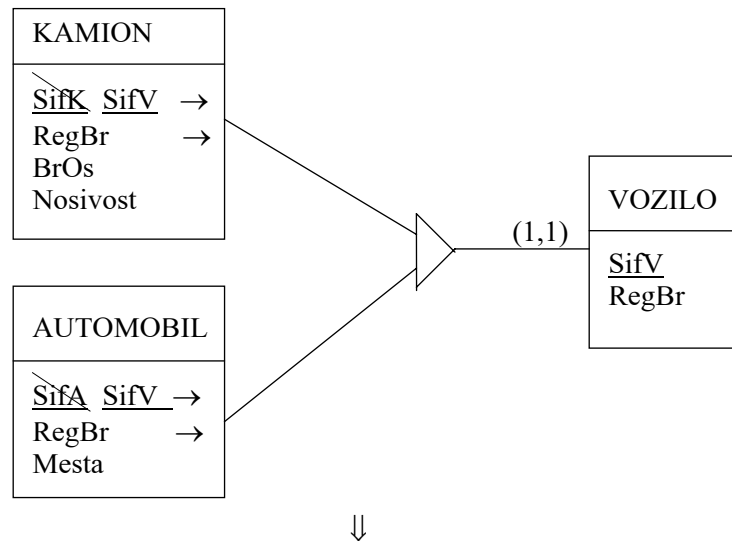
- u svim podklasama vrši se usaglašavanje klasifikacionih svojstava po nazivu, ako su zatečeni nazivi različiti;
- sva klasifikaciona svojstva koja su zajednička za sve podklase postaju zajednička svojstva nadklase, a podklase zadržavaju samo specifična svojstva;
- od svake podklase objekata nastaje šema relacije istog naziva, a atribute te šeme čine klasifikaciona svojstva podklase i primarni ključ šeme koja odgovara nadklasi, koji postaje i primarni ključ šeme podklase;
- od nadklase objekata koja specijalizuje nastaje šema relacija istog naziva i sa atributima koji odgovaraju klasifikacionim svojstvima, pri čemu je atribut koji odgovara identifikatoru primarni ključ.

S obzirom da je krajnji rezultat u modelu nekog sistema isti, sve ostalo što je navedeno za specijalizaciju važi i za generalizaciju.

Kategoriju, najsloženiji koncept modela entiteta i odnosa, nećemo razmatrati.

*Primer:*

Za generalizaciju će nam poslužiti raniji primer vozila. Neka su u našem modelu entiteta i odnosa prvo nastale klase objekata AUTOMOBIL i KAMION, i neka smo tek tada uočili da za njih postoji generalizacija u vidu nadklase VOZILO:



AUTOMOBIL ( SifV, Mesta )

VOZILO ( SifV, RegBr )

KAMION ( SifV, BrOs, Nosivost )

Redosled nastanka modela je s leva na desno. Precrtavanjem smo označili promenu naziva klasifikacionog svojstva, a strelicom prebacivanje u nadklasu.

### 8.1.4 Predstavljanje klasa veza

Veze (ili asocijacije) predstavljaju posredan odnos između klasa objekata.

#### *Definicija*

Klasa veza u posmatranom sistemu je svaka klasa čija instanca predstavlja odnos konstantnog broja instanci (najmanje dve) iz jedne ili više klasa objekata, pri čemu taj odnos može imati i određena svojstva.

Treba naglasiti da neka instanca u klasi veze nastaje kada se između odgovarajućih instanci objekata uspostavi odnos koji ta veza predstavlja, a nestaje kada se taj odnos raskine.

Bitna razlika ovog odnosa klasa objekata i odnosa zavisnosti i specijalizacije je u tome što taj odnos nije neposredan nego se ostvaruje posredstvom dodatne klase - klase veze.

Navedimo dva primera za prošireni sistem biblioteke:

- klasa veze PRIPADA između klasa objekata NASLOV i OBLAST, koja nema svojstva;
- klasa veze DRZI između klasa objekata CLAN i KNJIGA, koja ima svojstvo Datum (datum od kada član drži knjigu kod sebe).

Klasa veze se u modelu objekata i odnosa predstavlja rombom, unutar koga se velikim slovima upisuje naziv. Od ivica romba se povlače linije do svake klase objekata koji stupaju u vezu. Ako veza ima svojstva, umesto romba se koristi simbol dobijen kombinacijom romba i pravougaonika i u raspoloživi prostor se upisuju nazivi svojstava veze.

Treba naglasiti da veze nisu ograničene samo na nezavisne klase objekata. Moguće su veze između nezavisnih i zavisnih klasa objekata, kao i samo između zavisnih klasa objekata. Uz to, objekti koji su u vezi mogu biti i učesnici u specijalizaciji.

I u slučaju klase veze može se govoriti o preslikavanju, doduše posrednom, između klasa objekata, pa se za svaku klasu objekata koja učestvuje u vezi može definisati odgovarajuća kardinalnost preslikavanja:

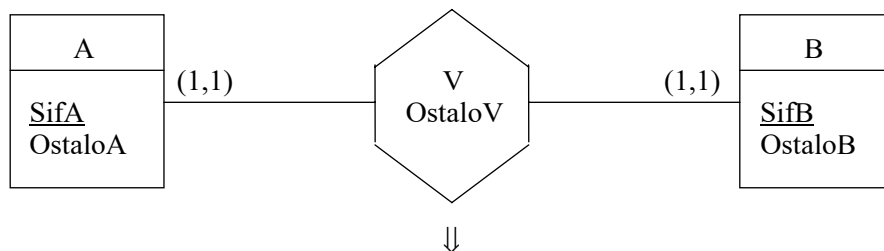
- kardinalnost veze: par brojeva (m,n) koji se zapisuje uz klasu od koja je u vezi i koji ima sledeće značenje: svaka instanca te klase mora da bude učesnik u bar m a može u najviše n instanci veze.

Prema kardinalnosti veze može se sprovesti sledeća podela:

- parcijalna (neobavezna) veza: slučaj kada je  $m=0$ , odnosno kada u odgovarajućoj klasi objekata mogu da postoje instance koje ne učestvuju u vezi;
- totalna (obavezna) veza: slučaj kada je  $m>0$  (a najčešće 1), odnosno kada svaka instanca u odgovarajućoj klasi objekata mora da učestvuje u bar m veza.

**Pravilo prevođenja veze** u odgovarajući koncept relacionog modela sadrži niz varijanti, zavisno od kardinalnosti veze. Radi jasnoće, ovde je izloženo pravilo za slučaj veze sa dva učesnika, uz ilustraciju svakog slučaja:

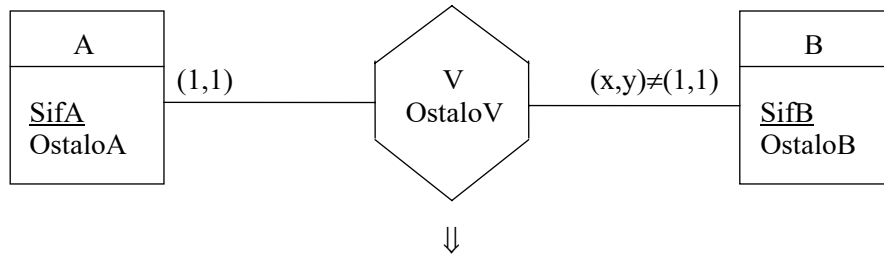
- **pre prevođenja klase veze u odgovarajući koncept relacionog modela, mora biti sprovedeno takvo prevođenje za sve klase objekata koje učestvuju u vezi;**
- **ako je kardinalnost veze za obe klase objekata (1,1), ne nastaje posebna šema relacije veze; u pitanju je greška u projektovanju koja je dovela do razdvajanja jedne klase objekata na dve; u tom slučaju, šema relacije jedne od klasa objekata se ukida, a šema relacije druge klase objekata dopunjuje se atributima ukinute šeme i atributima koji odgovaraju svojstvima klase veze, ako postoje; primarni ključ šeme relacije veze je bilo koji od primarnih ključeva šema objekata:**



A ( SifA, OstaloA, SifB, OstaloB, OstaloV )  
 ili  
 A ( SifA, OstaloA, SifB, OstaloB, OstaloV )

/ jedno od SifA ili SifB se  
 može izostaviti ako nije  
 svojstvo sa značenjem /

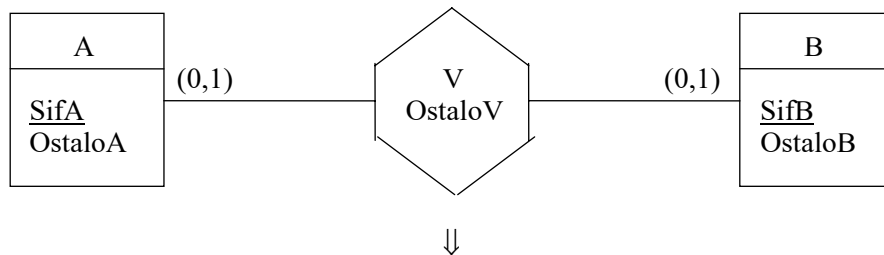
- ako je kardinalnost veze za jednu klasu objekata (1,1) a za drugu bilo šta osim (1,1), ne nastaje posebna šema relacije veze; u tom slučaju, šema relacija koja odgovara klasi objekata sa (1,1) strane dopunjuje se primarnim knjučem šeme relacije druge klase objekata i atributima koji odgovaraju svojstvima klase veze, ako postoje:



$A ( \underline{\text{SifA}}, \text{OstaloA}, \text{SifB}, \text{OstaloV} )$

$B ( \underline{\text{SifB}}, \text{OstaloB} )$

- ako je kardinalnost veze za obe klase objekata (0,1), nastaje posebna šema relacije veze; u tom slučaju, šemu relacije veze čine primarni ključevi šema relacija klase objekata i atributi koji odgovaraju svojstvima klase veze, ako postoje; primarni ključ šeme relacije veze je bilo koji od primarnih ključeva šema relacija objekata; izbor je po principu “čvrstine” (“po prirodi stvari” je jače) pa onda dužine:



$A ( \underline{\text{SifA}}, \text{OstaloA} )$

$V ( \underline{\text{SifA}}, \underline{\text{SifB}}, \text{OstaloV} )$

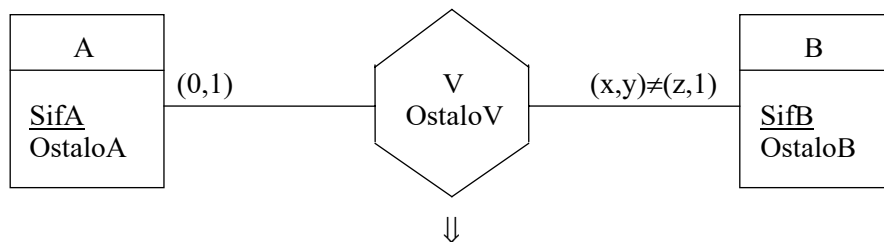
$B ( \underline{\text{SifB}}, \text{OstaloB} )$

ili

$V ( \text{SifA}, \underline{\text{SifB}}, \text{OstaloV} )$



- ako je kardinalnost veze za jednu klasu objekata (0,1) a za drugu bilo šta osim (1,1) i (0,1), nastaje posebna šema relacije veze; u tom slučaju, šemu relacije veze čine primarni ključevi šema relacija objekata i atributi koji odgovaraju svojstvima klase veze, ako postoje; primarni ključ šeme relacije veze je primarni ključ šeme relacije objekta sa (0,1) strane:

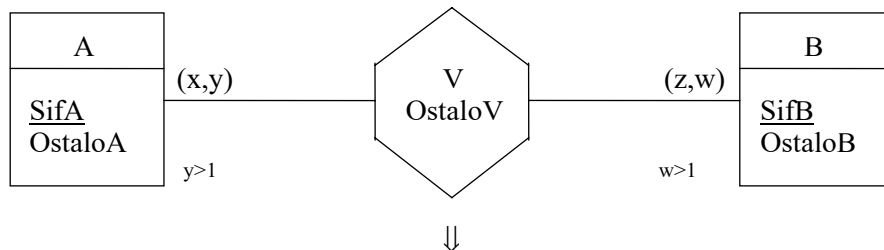


A ( SifA, OstaloA )

V ( SifA, SifB, OstaloV )

B ( SifB, OstaloB )

- u svim ostalim slučajevima, nastaje posebna šema relacije veze, koju čine primarni ključevi šema relacija klasa objekata i atributi koji odgovaraju svojstvima klase veze, ako postoje; primarni ključ šeme relacije veze čine dodati primarni ključevi zajedno:



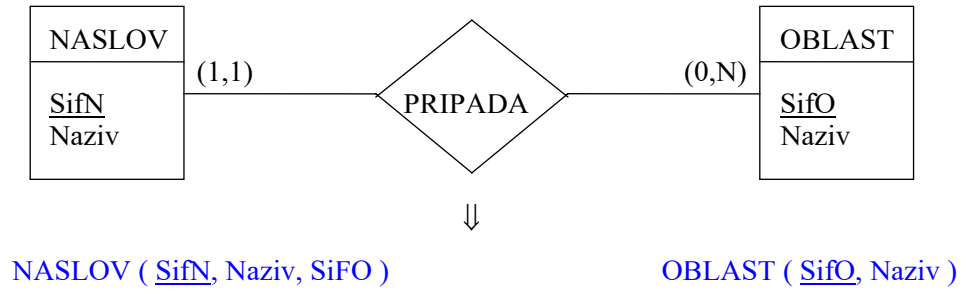
A ( SifA, OstaloA )

V ( SifA, SifB, OstaloV )

B ( SifB, OstaloB )

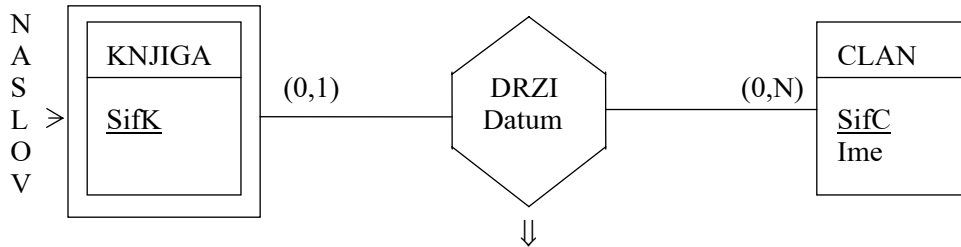
*Primer:*

U proširenom sistemu biblioteke, na koji se odnosi baza podataka BIBLIOTEKA (prilog A), imamo klasu veze PRIPADA između klasa objekata NASLOV i OBLAST. Veza je bez svojstava, pa je prikazujemo romбом.



*Primer:*

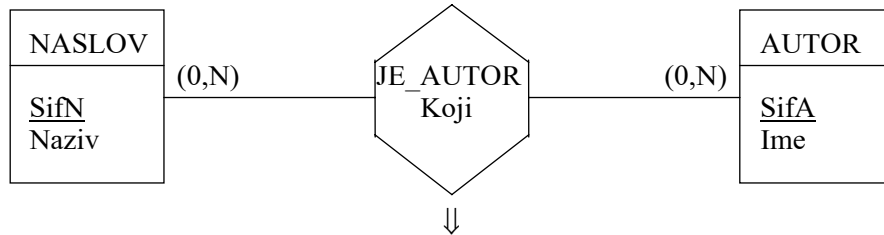
U istom sistemu imamo između klasa objekata CLAN i KNJIGA vezu drži koja ima svojstvo Datum:



Šema relacije KNJIGA je proširena atributom na osnovu zavisnosti od klase objekata NASLOV koja je samo naznačena. Kardinalnost (0,1) je jasna sama po sebi: jedna knjiga može biti u nekom trenutku biti izdata samo jednom članu.

*Primer:*

U sistemu biblioteke klase objekata NASLOV i AUTOR su povezane preko klase veze JE\_AUTOR koja ima dodatno svojstvo Koji, pa imamo:



$\text{NASLOV ( } \underline{\text{SifN}}, \text{ Naziv ) } \text{ JE\_AUTOR ( } \underline{\text{SifN}}, \underline{\text{SifA}}, \text{ Koji ) } \text{ AUTOR ( } \underline{\text{SifA}}, \text{ Ime )}$

Kardinalnosti veza (0,N) sa strane obe klase objekata smo odabrali kako bi bili u mogućnosti da evidentiramo naslove i autore a da pri tome odmah ne moramo da evidentiramo i autorstva.

### 8.1.5 Predstavljanje agregacije

*Definicija:*

Agregacija je klasa veze koja se ponaša kao klasa objekata na taj način što može da učestvuje u vezama.

U modelu objekata i odnosa agregacija se predstavlja simbolom veze uokvirenim pravougaonikom.

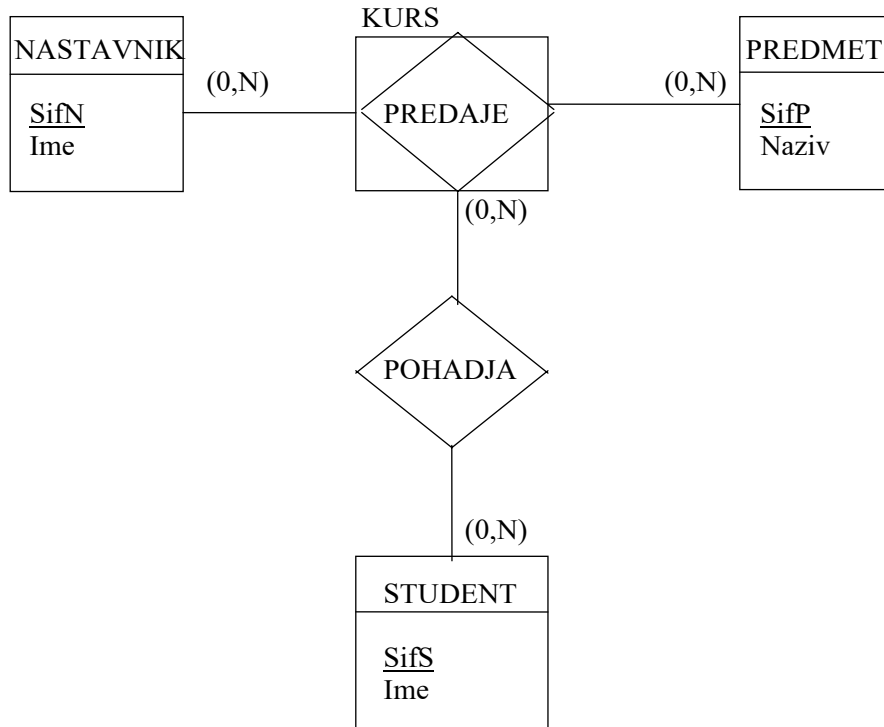
Prevođenje agregacije u odgovarajući koncept relacionog modela je jednostavno:

- agregacija se posmatra kao klasa objekata i na osnovu toga tretira na ranije opisane načine.

Pojasnimo ovaj pojam pogodno odabranim primerom koji se odnosi na deo sistema koga čini neki fakultet.

*Primer:*

Neka između klasa objekata NASTAVNIK i PREDMET postoji klasa veze PREDAJE. Studenti na fakultetu slušaju predavanja iz određenih predmeta, ali kod određenih nastavnika. To možemo da predstavimo tako što klasu veze PREDAJE tretiramo kao agregaciju koju možemo nazvati KURS, a zatim između te agregacije i klase objekata STUDENT uspostavimo klasu veze POHADJA:



Prvo vršimo prevođenje u relacioni model za klase objekata. Dobijamo:

NASTAVNIK ( SifN, Ime )   STUDENT ( SifS, Ime )   PREDMET ( SifP, Naziv )

Nakon toga, formiramo šemu relacije za klasu veze PREDAJE:

PREDAJE ( SifN, SifP )

Ova šema relacije se u uslovima agregacije ponaša kao šema relacije za klasu objekata koju smo radi jasnoće označili sa KURS, pa nam preostaje da po pravilu za veze formiramo šemu relacije za klasu veza POHADJA:

POHADJA ( SifN, SifP, SifS )

Uočimo da se pomoćna oznaka KURS nigde ne pojavljuje u relacionom modelu. Uz to, primetimo da u ovom specifičnom slučaju dobijeni primarni ključ nije minimalan zato što “po prirodi stvari” jedan student samo jednom pohađa jedan predmet, odnosno u POHADJA važi funkcijska zavisnosti  $SIFS, SIFP \rightarrow SIFN$ . Drugim rečima, u ovakvim situacijama uvek treba proveriti minimalnost primernog ključa. U našem konkretnom slučaju dobijamo:

POHADJA ( SifN, SifP, SifS )

### 8.1.6 Prevođenje u relacioni model

Na osnovu svega do sada izloženog možemo formulisati celoviti postupak prevođenja modela objekata i odnosa u relacioni model. Redosled je sledeći:

- prvo se na osnovu pravila izloženog u odeljku 8.1.1 vrši prevođenje za sve nezavisne klase objekata, a po pravilu iz odeljka 8.1.3 vrši se prevođenje i za sve njihove specijalizacije;
- posle toga se na osnovu pravila izloženih u odeljku 8.1.2 vrši prevođenje za sve zavisne klase objekata, a po pravilu iz odeljka 8.1.3 i za njihove specijalizacije;
- zatim se prema pravilu navedenom u odeljku 8.1.4 i 8.1.5 vrši prevođenje za sve veze koje su istovremeno i agregacije;
- na kraju, prema pravilima za veze iz odeljka 8.1.4 vrši se prevođenje za sve preostale veze.

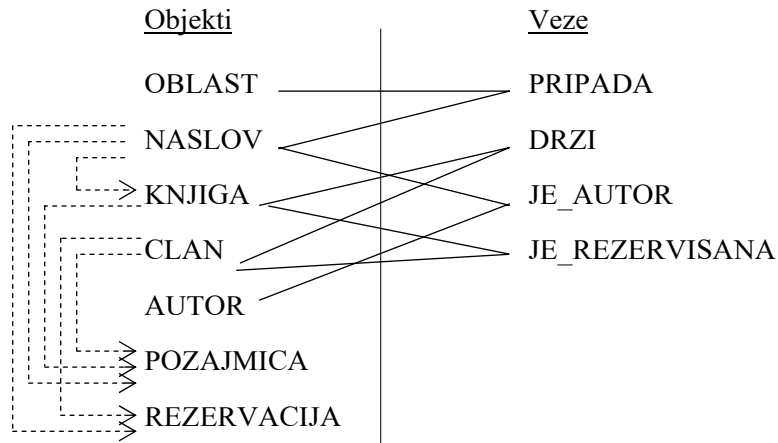
Navedeni postupak ćemo ilustrovati sa jednim primerom.



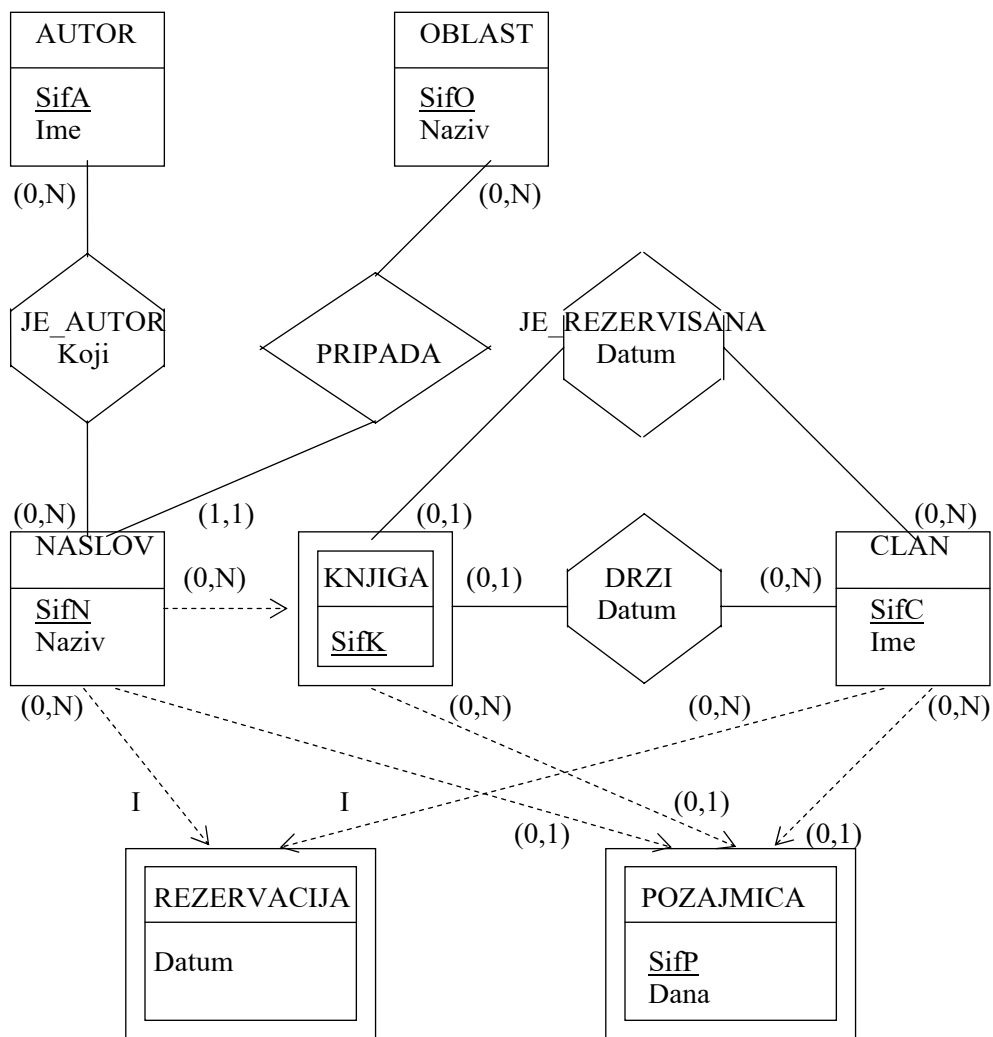
*Primer:*

Neka je posmatrani sistem naša biblioteka. Pokazaćemo kako je dobijena šema relacione baze podataka BIBLIOTEKA data u prilogu A.

Postupcima abstrakcije opisanim u poglavlju 3 dolazimo do šematskog prikaza sledećih klasa objekata i veza od interesa u posmatranom sistemu:



Ovaj šematski prikaz je koristan i preporučuje se kao prvi korak u razvoju modela podataka, pošto iz njega sagledavamo koje klase su najmanje opterećene u smislu zavisnosti i učešća u vezama. Svakoј takvoj klasi pripada periferno mesto u modelu objekata i odnosa. U našem slučaju to su AUTOR i OBLAST, pa dobijamo model:



Na osnovu ovog modela dobija se relacioni model kao i u prilogu A, na kome je prikazana dopuna šeme objekta na odnosu pravila za (1,1) vezu:

AUTOR ( <u>SifA</u> , Ime )	DRZI ( <u>SifK</u> , SifC, Datum )
CLAN ( <u>SifC</u> , Ime )	JE_AUTOR ( <u>SifA</u> , <u>SifN</u> , Koji )
OBLAST ( <u>SifO</u> , Naziv )	JE_REZERVISANA ( <u>SifK</u> , SifC, Datum )
NASLOV ( <u>SifN</u> , Naziv <del>/</del> , SifO )	< dopunjeno zbog veze PRIPADA
KNJIGA ( <u>SifK</u> , SifN )	
REZERVACIJA ( <u>SifN</u> , <u>SifC</u> , Datum )	
POZAJMICA ( <u>SifP</u> , SifC, SifK, SifN, Dana )	

U vezi prethodnog modela i izvedene relacije {eme neophodne su određene napomene o odnosu zavisnosti između klasa objekata. Konkretno, radi se o klasi POZAJMICA gde smo u odnosu na ranija razmatranja uveli i zavisnost od klase NASLOV tako da imamo sledeću situaciju zavisnosti:



Na prvi pogled, zavisnost klase POZAJMICA od klase NASLOV se čini suvišnom pošto se podatak o tome koji naslov je pozajmljen može izvesti posredstvom zavisnosti od klase KNJIGA. Postoje dva razloga protiv takvog zaključka:

- konceptualno posmatrano, član je tražio i pozajmio naslov a sa time obavezno i knjigu kao njegovog fizičkog nosioca;
- ako ne bi evidentirali koji naslov je pozajmljen, u slučaju brisanja knjige iz evidencije (usled gubitka ili oštećenja) izgubili bi podatke o pozajmicama naslova sa te knjige.

U ovom konkretnom slučaju radi se o gubitku podataka usled nestanka posrednika u lancu zavisnosti, ali u posebnim okolnostima posledice mogu biti još gore, što najbolje ilustruje primer video-kluba gde za razliku od biblioteke kaseta kao fizički nosilac filma može presnimavanjem promeniti sadržaj. Bez zavisnosti pozajmice od filma imali bi sledeću situaciju:

FILM ( SIFF, NAZIV ) KASETA ( SIFK ) SADRZI ( SIFK, SIFF ) CLAN ( SIFC, IME )  
 POZAJMICA ( SIFP, SIFC, SIFK, DANA )

U slučaju presnimavanja kasete, odnosno promene atributa SIFF u relaciji SADRZAJ za tu kasetu, kao efekat bi imali ne gubitak podataka nego pogrešne podatke: sve pozajmice filma koji se ranije nalazio na kaseti bile bi pripisane novom filmu.

Zaključak koji sledi iz ovih primera je jasan: pre nego što se neka zavisnost između klasa objekata proglasi za suvišnu i odbaci treba pažljivo analizirati alternativni lanac zavisnosti odnosno posrednike.

U vezi šeme relacije POZAJMICA postoji još jedna okolnost koja je vredna pažnje. U nameri da minimiziramo gubitak podataka u slučaju brisanja iz evidencije člana, knjige ili naslova, za strane ključeve SIFC, SIFK i SIFN usvojili smo za dinamičku specifikaciju referencijalnog integriteta **SET NULL** za operaciju **DELETE** (poglavlje 6, naredbe kreiranja tabela). To nameće izvesno prilagođavanje pojma zavisnosti između klasa objekata:

- u suštini, može se govoriti o dva aspekta odnosa zavisnosti; prvi je zavisnost nastanka koji podrazumeva da instanca slabog objekta ne može da nastane nezavisno od svojih uslovitelja; drugi je zavisnost postojanja koji podrazumeva da instanca slabog objekta ne može da postoji nezavisno od svojih uslovitelja;
- kardinalnost uslovljenosti je u slučaju da je u pitanju samo zavisnost nastanka par brojeva (0,N).

### 8.1.7 Model objekata i odnosa i zavisnosti između atributa

Osnovna namena modela objekata i odnosa je da posle pažljive analize nekog sistema dodamo do dobro ustrojene šeme relacione baze podataka za taj sistem. Striktno govoreći, ovaj model se sastoji samo iz strukturne komponente, pošto se integritetska komponenta podrazumeva, s obzirom da strani ključevi u šemama relacija nastaju isključivo primenom opisanih pravila prevođenja.

Treba naglasiti da projektovanje relacione baze podataka pomoću modela objekata i odnosa ne isključuje analizu mogućih zavisnosti između atributa i dodatnu transformaciju dobijene šeme relacione baze podataka postupcima normalizacije, o čemu je bilo reći u poglavlju 7. Naime, među odabranim svojstvima klasa mogu postojati neželjene zavisnosti, pa to treba ispitati u završnoj fazi projektovanja relacione baze podataka. Konkretno, radi se o sledećem:

- ako se izbegava identifikaciona zavisnost sve šeme relacije objekata nastale iz modela imaju proste identifikatore pa je time automatski ispoštovana druga normalna forma; takva garancija ne postoji za treću normalnu formu i stoga za svaku šemu relacije objekata treba ispitati da li između neključnih atributa postoje tranzitivne zavisnosti;
- za svaku šemu relacije veze “više prema više” koja ima složeni primarni ključ i neključne attribute treba ispitati da li između neključnih atributa ima tranzitivnih zavisnosti i da li ima neključnih atributa koji zavise od dela primarnog ključa;
- za svaku šemu relacije veze koja je sa jedne strane “jedan prema jedan” ili “jedan prema više” koja ima prost primarni ključ i neključne attribute treba ispitati da li između neključnih atributa ima tranzitivnih zavisnosti.

U slučaju da nastupi bilo koja od navedenih neželjenih situacija sprovodi se postupak normalizacije kako je opisano u poglavlju 7.

### 8.1.8 Kompromisi u šemi relacione baze podataka

Kompromisi u šemi relacione baze podataka se bez obzira na vrstu sprovode iz jednog jedinog razloga – poboljšanja performansi rada sa bazom podataka. Pri tome je osnovni cilj da se kod upita izbegnu zahtevne operacije spajanja tabela kao i razna obimna svođenja podataka. Postoje dve vrste kompromisne izmene šeme relacione baze podataka:

- redukcija šeme relacione baze podataka;
- redudansa šeme relacione baze podataka.

Kako i sam naziv govori, suština redukcije šeme relacione baze podataka jeste u tome da se smanji broj šema relacija. Postoje tri situacije kada je to moguće:

- tretman veze kardinalnosti (0,1) po pravilu za kardinalnost (1,1): posebna šema relacije veze se ukida a primarni ključ šeme relacije drugog učesnika u vezi i eventualna svojstva veze se dodaju šemi relacije učesnika sa (0,1) strane; u toj relaciji vrednosti dodatih atributa će biti NULL za one instance koje nisu u vezi;
- implozija odnosa specijalizacije/generalizacije: šeme relacija specijalnih objekata se ukidaju a svi atributi iz njih se dodaju šemi relacije generalnog objekta; svaka instanca u toj šemi će imati vrednosti NULL za one attribute koji nisu njeni; radi lakšeg raspoznavanja koja instanca je koji specijalni slučaj, jedinstvenoj šemi relacije se može dodati selektorski atribut u slučaju ekskluzivne specijalizacije, odnosno odgovarajući broj indikatorskih atributa u slučaju inkluzivne specijalizacije;
- implozija odnosa zavisnosti kardinalnosti uslovljavanja (0,1): šema relacije slabog objekta se ukida a svi neključni atributi iz nje se dodaju šemi relacije objekta-uslovitelja; ovo je moguće samo kod slabih objekata koji zavise od jedne klase uslovitelja; svaka instanca objekta-uslovitelja koja nije uslovila instancu slabog objekta imaće vrednost NULL za one attribute koji nisu njegovi.

U našem primeru biblioteke nema mogućnosti primene poslednja dva pravila, ali se zato može primeniti prvo. Time umesto šema relacija

KNJIGA ( SIFK, SIFN )

DRZI ( SIFK, SIFC, DATUM )    JE\_REZERVISANA ( SIFK, SIFC, DATUM )

ukidanjem šema relacija veza DRZI i JE\_REZERVISANA dobijamo

KNJIGA ( SIFK, SIFN, SIFCDRZ, DATUMDRZ, SIFCREZ, DATUMREZ )

Ovim se značajno pojednostavljuje postupak obrade zahteva člana za naslovom. Umesto da se sprovodi upit nad relacijama KNJIGA, DRZI, JE\_REZERVISANA, sada se sve odvija nad dopunjenom relacijom KNJIGA.

Redudansa šeme relacione baze podataka se može sprovesti u dva vida:

- replikativna redudansa: suština je u tome da se izbegne spajanje tabela; atribut iz jedne šeme relacije se dodaje-replicira u drugoj šemi relacije; pri tome replika mora biti sinhronizovana sa originalom – u slučaju izmene originala treba izmeniti i sve njene replike; na izvestan način, replicirani podatak se ponaša kao strani ključ za koji je za operaciju **UPDATE** specificirano **CASCADE**;
- svodna redudansa: suština je u tome da se izbegne svođenje podataka (brojanje, sabiranje itd.) u tabelama; šemi relacije se dodaje atribut koji predstavlja svodni podatak; pri tome, svaki put kada se ažuriraju tabele iz kojih je izveden svodni podatak mora se ažurirati i svodni podatak.

Kao primer za replikativnu redudansu za naš slučaj biblioteke uzmimo dopunjenu šemu relacije o pozajmicama koja kao takva omogućava da se podaci o ukupnom broju i trajanju pozajmica po oblastima dobiju bez spajanja sa tabelom NASLOV:

POZAJMICA ( SIFP, SIFC, SIFK, SIFN, DANA, SIFO )

Svodnu redudansu možemo ilustrovati primerom dopunjene šeme relacije NASLOV koja obezbeđuje brzo dobijanje podatka o ukupnom broju i trajanju pozajmica naslova:

NASLOV ( SIFN, NAZIV, SIFO, **BROJPOZ**, **DANAPOZ** )

Cena koju smo platili za tu brzinu jeste složeniji postupak održavanja podataka: svaki put kada se vraća pozajmljena knjiga treba dodatno ažurirati podatke BROJPOZ i DANAPOZ u relaciji NASLOV. Savremeni SQL dozvoljava mogućnost da se nad tabelom definiše tzv. “okidač” (TRIGGER) koji reaguje na izmenu u tabeli tako što izvršava specificirani niz naredbi. To u mnogome olakšava održavanje svodnih redundantnih podataka,



Poslednji primer koji ilustruje gotovo spektakularne efekte svodne redudanse odnosi se na jednu banku koja posluje sa štedišama tako što im omigučava da imaju otvorene račune određenih vrsta (tekući, žiro, oročeni itd.) za koje mogu vršiti uplate i isplate. To se bez redudanse može predstaviti sledećim skupom šema relacija:

VRSTA\_RACUNA ( SIFV, NAZIV )    RACUN ( SIFR, STEDISA, SIFV )

PROMET ( SIFR, REDBR, DATUM, UPLATA, ISPLATA )

Pošto se evidentiraju sve uplate i isplate, iz ovih šema se spajanjem i sumiranjem za svaki račun svakog štediša mogu dobiti ukupna uplata, ukupna isplata i stanje (saldo). Isto to se može dobiti i za svaku vrstu računa, a kada se sumira za sve vrste računa i za celu banku. Problem je samo u tome što tipična banka ima desetak vrsta računa i u okviru njih stotinak hiljada računa štediša od kojih svaki mesečno ima na desetine stavki prometa. Drugim rečima, da bi se došlo do podataka koliko banka ima ukupno na svim vrstama računa treba prethodno sumirati iznose svih stavki prometa čiji se broj kreće na milione. To u uslovima brzog donošenja odluka u savremenom poslovanju nije prihvatljivo. Rešenje navedenog problema je u uvođenju redudantnog skupa šema relacija:

VRSTA\_RACUNA ( SIFV, NAZIV, UPLATA, ISPLATA )

RACUN ( SIFR, STEDISA, SIFV, UPLATA, ISPLATA )

PROMET ( SIFR, REDBR, DATUM, UPLATA, ISPLATA )

Ovim smo obezbedili najbrže moguće odgovore na upite koliko banka ima sredstava ukupno ili po vrstama računa, ali smo zato opteretili obradu svake uplate i isplate ažuriranjem svodnih podataka za račune i vrste računa što se u radu praktično ne oseća s obzirom da je unos podataka o uplatama i isplatama manuelan.

## 8.2 Projektovanje postupaka

Kao što je već naglašeno, projektovanje postupaka podrazumeva dva koraka. Prvo što se sprovodi jeste funkcionalna dekompozicija posmatranog sistema. Kada se taj postupak obavi do određenog nivoa detalja stižu se uslovi za sprovođenje narednog koraka, a to je algoritamska specifikacija.

## 8.2.1 Funkcionalna dekompozicija

Sušтина postupka funkcionalne dekompozicije jeste da se funkcionalnost posmatranog sistema dekomponuje na sastavne podfunkcije i da se to sukcesivno ponavlja na dobijene podfunkcije sve do dostizanja željenog nivoa detalja kada se može pristupiti algoritamskoj specifikaciji. U savremenoj informatici postoje četiri oblika funkcionalne dekompozicije od kojih će ovde biti navedena samo prva dva:

- funkcionalna dekompozicija prve vrste: jedino što je vidljivo jeste hijerarhijski odnos funkcionalnosti, odnosno to koje funkcije su sastavljene iz kog skupa podfunkcija; ne postoji nikakva naznaka podataka preko kojih funkcije komuniciraju, redosleda izvršavanja funkcija, uslovljavanja njihovog izvršavanja ili ponavljanja;
- funkcionalna dekompozicija druge vrste: pored hijerarhijskog odnosa funkcionalnosti, postoji i naznaka podataka preko kojih funkcije komuniciraju, redosleda izvršavanja funkcija, uslovljavanja njihovog izvršavanja i ponavljanja izvršavanja.

Sprovedimo prvo funkcionalnu dekompoziciju prve vrste za sistem BIBLIOTEKA kako je opisan u odeljku A3 priloga A. Pri tome će naglasak biti na održavanju podataka, s obzirom da se korišćenje podataka svodi na pogodno formulisane upite. Nizom transformacija dobijamo redom:

### BIBLIOTEKA

Održavanje maticnih podataka  
Obrada prometa

### BIBLIOTEKA

Održavanje maticnih podataka  
Održavanje podataka o oblastima  
Održavanje podataka o naslovima  
Održavanje podataka o autorima  
Održavanje podataka o knjigama  
Održavanje podataka o članovima  
Obrada prometa  
Obrada traženja naslova  
Obrada vraćanja knjige  
Obrada gubitka knjige

**BIBLIOTEKA**

- Odrzavanje maticnih podataka
  - Odrzavanje podataka o oblastima
    - Dodavanje nove oblasti
    - Izmena postojece oblasti
    - Brisanje postojece oblasti
  - Odrzavanje podataka o naslovima
    - Dodavanje novog naslova
    - Izmena postojeceg naslova
    - Brisanje postojeceg naslova
  - Odrzavanje podataka o autorima
    - Dodavanje novog autora
    - Izmena postojeceg autora
    - Brisanje postojeceg autora
  - Odrzavanje podataka o knjigama
    - Dodavanje nove knjige
    - Izmena postojece knjige
    - Brisanje postojece knjige
  - Odrzavanje podataka o clanovima
    - Dodavanje novog clana
    - Izmena postojeceg clana
    - Brisanje postojeceg clana
- Obrada prometa
  - Obrada trazanja naslova
    - Ocitavanje neophodnih podataka
    - Provera da li clan ima kod sebe trazeni naslov
    - Provera da li ima slobodne knjige
    - Evidentiranje izdavanja knjige
    - Evidentiranje rezervacije naslova
  - Obrada vracanja knjige
    - Ocitavanje neophodnih podataka
    - Evidentiranje vracanja
    - Evidentiranje pozajmice
    - Provera ima li rezervacije
    - Opsluzivanje rezervacije
  - Obrada gubitka knjige
    - Ocitavanje neophodnih podataka
    - Evidentiranje gubitka

Ovde treba naglasiti da se situacije izmene matičnih podataka odnose na unos naknadnih izmena podataka ili ispravku pogrešno unetih podataka.

Funkcionalnu dekompoziciju druge vrste navešćemo samo za poslednju od prethodnih dekompozicija. Pri tome je naglasak stavljen na obradu prometa:

## BIBLIOTEKA

## Odrzavanje maticnih podataka

## Odrzavanje podataka o oblastima

Dodavanje nove oblasti ( I:OBLAST )  
 Izmena postojece oblasti ( U:OBLAST )  
 Brisanje postojece oblasti ( D:OBLAST )

## Odrzavanje podataka o naslovima

Dodavanje novog naslova ( I:NASLOV )  
 Izmena postojeceg naslova ( U:NASLOV )  
 Brisanje postojeceg naslova ( D:NASLOV )

## Odrzavanje podataka o autorima

Dodavanje novog autora ( I:AUTOR )  
 Izmena postojeceg autora ( U:AUTOR )  
 Brisanje postojeceg autora ( D:AUTOR )

## Odrzavanje podataka o knjigama

Dodavanje nove knjige ( I:KNJIGA )  
 Izmena postojece knjige ( U:KNJIGA )  
 Brisanje postojece knjige ( D:KNJIGA )

## Odrzavanje podataka o clanovima

Dodavanje novog clana ( I:CLAN )  
 Izmena postojeceg clana ( U:CLAN )  
 Brisanje postojeceg clana ( D:CLAN )

## Obrada prometa

## Obrada trazanja naslova

1 Ocitanje neophodnih podataka  
 ( < SifC, < SifN )  
 2 Provera da li clan ima kod sebe trazeni naslov  
 ( > SifC, > SifN, < Ishod, S:DRZI,KNJIGA )  
 ?3 Provera da li ima slobodne knjige / Ishod  
 ( > SifN, < Ishod, < SifK, S:KNJIGA,DRZI,JE\_REZERVISANA )  
 ?4e Evidentiranje izdavanja knjige / Ishod  
 ( > SifK, > SifC, I:DRZI )  
 ?4e Evidentiranje rezervacije naslova / Ishod  
 ( > SifC, > SifN, I:REZERVACIJA )

## Obrada vraćanja knjige

1 Ocitanje neophodnih podataka  
 ( < SifK, < SifN, < SifC, < Datum, S:DRZI,KNJIGA )  
 2 Evidentiranje vraćanja  
 ( > SifK, D:DRZI )  
 2 Evidentiranje pozajmice  
 ( > SifC, > SifK, > SifN, > Datum, I:POZAJMICA )  
 3 Provera ima li rezervacije  
 ( > SifN, < Ishod, < SifC, S:REZERVACIJA )  
 ?4 Opsluživanje rezervacije / Ishod  
 ( > SifK, > SifC, I:JE\_REZERVISANA )

## Obrada gubitka knjige

Ocitanje neophodnih podataka  
 ( < SifK )  
 Evidentiranje gubitka  
 ( > SifK, D:DRZI,KNJIGA )

Ovde je u pogledu korišćene notacije neophodan niz napomena:

- uslovljenost neke funkcije se označava znakom `?` i navođenjem uslovljavajućeg podatka iza naziva funkcije i odvojeno znakom `/` (na primer, to je funkcija `Opsluzivanje rezervacije`);
- redosled izvršavanja funkcija se označava rednim brojevima; za funkcije bez redosleda ili sa istim redosledom sekvenca njihovog izvršavanja je nebitna (to važi za funkcije `Evidentiranje vratanja` i `Evidentiranje pozajmice`);
- uslovljene funkcije koje se uzajamno isključuju označavaju se istim brojem redosleda i sa `e` iza toga (to je slučaj kod funkcija `Evidentiranje izdavanja knjige` i `Evidentiranje rezervacije naslova`);
- u zagradama iza ili ispod naziva funkcije navode se kvalifikatorima `>`, `<` ili `<>` ulazni, izlazni i ulazno-izlazni podaci;
- unutar istih zagrada navode se operacije funkcije nad entitetima modela podataka odnosno relacijama u bazi podataka, i to: `S`-očitavanje, `I`-ubacivanje, `U`-izmena i `D`-brisanje.

## 8.2.2 Algoritamska specifikacija

Sušтина algoritamske specifikacije je u tome da se funkcionalna dekompozicija druge vrste za posmatrani sistem dopuni dodatnim detaljima do te mere da je njena implementacija svedena na jednoznačni rutinski postupak.

Kao primer algoritamske specifikacije navešćemo onu za funkciju **Obradatrazenjanaslova**. U vezi takvog izbora treba naglasiti da algoritamska specifikacija nije vezana isključivo za najniže sastavne funkcije nego da se može primeniti i na funkcije višeg nivoa.

Kao osnova za sastavljanje navedene algoritamske specifikacije poslužiće nam opis rada biblioteke koji je izložen u odeljku A.3 priloga A. Prema onome što je tamo navedeno, kada član traži neki naslov prvo se proverava da li član već ima kod sebe knjigu sa traženim naslovom. Ako ima, ne izdaje mu se druga knjiga, a ako nema proverava se da li je za njega već odvojena kao rezervisana knjiga sa tim naslovom. Ako jeste, ta knjiga mu se izdaje, a samo ako nije traži se da li ima slobodne knjige sa tim naslovom. Ako takva knjiga postoji ona se izdaje članu, a ako ne postoji član se izjašnjava da li želi da rezerviše traženi naslov. Ako želi, to se i evidentira.

Na osnovu svega prethodno navedenog za funkciju **Obradatrazenjanaslova** sledi i formalna algoritamska specifikacija. Radi preglednosti, kao komentari su navedene sve podfunkcije. Prvo je izložena specifikacija po notaciji koja je nešto izmenjena u odnosu na onu izloženu u prilogu B i poglavlju 4, a zatim sledi i još detaljnija algoritamska specifikacija koja koristi i elemente programskog SQL jezika izložene u Prilogu C.

```

ObradaTrazenjaNaslova
:  Ocitanje neophodnih podataka i inicijalizacija
:  :  OUTPUT ( "Unesite sifre clana i naslova" )
:  :  INPUT  ( vSifC, vSifN )
:  :  vIshod = 'NEMA_KNJIGE'
:  Provera da li vec ima naslov kod sebe
:  :  DO FOR EACH ( vDrzi IN DRZI )
:  :  :  WHERE ( vDrzi.SIFC == vSifC )
:  :  :  WHILE ( vIshod == 'NEMA_KNJIGE' )
:  :  :  DO FOR FIRST ( vKnjiga IN KNJIGA )
:  :  :  :  WHERE ( vKnjiga.SIFK == vDrzi.SIFK AND
:  :  :  :  :  vKnjiga.SIFN == vSifN )
:  :  :  :  vIshod = 'VEC_IMA'
:  :  :  OUTPUT ( "Clan ima knjigu sa trazanim naslovom" )
:  Provera ima li slobodne knjige
:  :  IF ( vIshod == 'NEMA_KNJIGE' )
:  :  :  Provera da li je za clana rezervisana knjiga
:  :  :  :  DO FOR EACH ( vJeRezerv IN JE_REZERVISANA )
:  :  :  :  :  WHERE ( vJeRezerv.SIFC == vSifC )
:  :  :  :  :  WHILE ( vIshod == 'NEMA_KNJIGE' )
:  :  :  :  :  DO FOR FIRST ( vKnjiga IN KNJIGA )
:  :  :  :  :  :  WHERE ( vKnjiga.SIFK == vJeRezerv.SIFK AND
:  :  :  :  :  :  :  vKnjiga.SIFN == vSifN )
:  :  :  :  :  :  vIshod = 'IMA_KNJIGE'
:  :  :  :  :  :  vSifK = vKnjiga.SIFK
:  :  :  Provera da li ima knjige koja nije kod clana
:  :  :  :  IF ( vIshod == 'NEMA_KNJIGE' )
:  :  :  :  :  DO FOR EACH ( vKnjiga IN KNJIGA )
:  :  :  :  :  :  WHERE ( vKnjiga.SIFN == vSifN )
:  :  :  :  :  :  WHILE ( vIshod == 'NEMA_KNJIGE' )
:  :  :  :  :  :  vSlobodna = 'DA'
:  :  :  :  :  :  DO FOR FIRST ( vDrzi IN DRZI )
:  :  :  :  :  :  :  WHERE ( vDrzi.SIFK == vKnjiga.SIFK )
:  :  :  :  :  :  :  vSlobodna = 'NE'
:  :  :  :  :  :  DO FOR FIRST ( vJeRezerv IN JE_REZERVISANA )
:  :  :  :  :  :  :  WHERE ( vJeRezerv.SIFK == vKnjiga.SIFK )
:  :  :  :  :  :  :  :  WHILE ( vSlobodna = 'DA' )
:  :  :  :  :  :  :  :  vSlobodna = 'NE'
:  :  :  :  :  :  :  IF ( vSlobodna == 'DA' )
:  :  :  :  :  :  :  :  vIshod = 'IMA_KNJIGE'
:  :  :  :  :  :  :  :  vSifK = vKnjiga.SIFK
:  :  IF ( vIshod == 'IMA_KNJIGE' )
:  :  :  Evidentiranje izdavanja knjige
:  :  :  :  OUTPUT ( 'Izdati clanu knjigu', vSifK )
:  :  :  :  INSERT DRZI ( vSifK, vSifC, Datum() )
:  +- ( vIshod == 'NEMA_KNJIGE' )
:  :  Evidentiranje rezervacije naslova
:  :  :  OUTPUT ( "Unesite da li clan zeli rezervaciju" )
:  :  :  INPUT  ( vZeliRezervaciju )
:  :  :  IF ( vZeliRezervaciju == 'DA' )
:  :  :  :  INSERT REZERVACIJA ( vSifN, vSifC, DatumVreme() )

```



U vezi korišćene notacije može se kao prvo konstatovati da je u pitanju specifična forma relacionog računa n-torki, s obzirom da u petljama oblika

```
DO FOR EACH ( Varijabla IN Relacija )
```

promenljiva **Varijabla** varira nad n-torkama relacije **Relacija**. Što se tiče izmena u odnosu na notaciju iz priloga B i poglavlja 4, one su bazirane na osnovu saznanja da se često javljaju algoritamske konstrukcije tipa

```
DO FOR EACH ( Var IN SkuP )      DO FOR EACH ( Var IN Skup )
| IF ( Uslov )                  | IF ( Uslov )
| | Postupak                    | | Postupak
                                <|--EXIT
```

odnosno petlja nad elementima skupa koja izvršava **Postupak** za sve elemente koji zadovoljavaju **Uslov** i petlja nad elementima skupa koja izvršava **Postupak** samo za prvi element koji zadovolji **Uslov** a zatim prekida dalji rad. Za navedene dve situacije usvojene su respektivno algoritamske konstrukcije

```
DO FOR EACH ( Var IN SkuP )      DO FOR FIRST ( Var IN Skup )
| WHERE ( Uslov )                | WHERE ( Uslov )
| Postupak                      | Postupak
```

čime je povećana preglednost specifikacije i smanjena dubina ugnežđjavanja. Ovde uslov u **WHERE** klauzuli predstavlja uslov izbora elementa. Ako se želi zadavanje uslova “dokle god” kojim se iteriranje prekida kada taj uslov postane **FALSE**, to se postiže dodatnom klauzulom **WHILE**. Tako se dobijaju konstrukcije

```
DO FOR EACH ( Var IN SkuP )      DO FOR FIRST ( Var IN Skup )
| WHERE ( UslovIzbora )          | WHERE ( UslovIzbora )
| WHILE ( UslovRada )           | WHILE ( UslovRada )
| Postupak                     | Postupak
```

Klauzule **WHERE** i **WHILE** su opcione. Inače, za sve varijante iteriranja nad relacijama sa i bez ugnježdjavanja mogu se formulisati pravila za prevođenje u odgovarajuće SQL konstrukcije (upite i kursore) što obezbeđuje rutinski i jednoznačni postupak implementacije.

Algoritamska specifikacija zasnovana na elementima programskog SQL jezika je još detaljnija od prethodne i jednostavnija je za implementaciju:

#### ObradaTrazenjaNaslava

```

:  Ocitanje neophodnih podataka i inicijalizacija
:  :  OUTPUT ( "Unesite sifre clana i naslova" )
:  :  INPUT  ( vSifC, vSifN )
:  :  vIshod = 'NEMA_KNJIGE'
:  Provera da li vec ima naslov kod sebe
:  :  SELECT SIFK
:  :  INTO :vSifK
:  :  FROM DRZI D
:  :  WHERE SIFC = :vSifC
:  :  AND EXISTS ( SELECT SIFK
:  :  :  FROM KNJIGA
:  :  :  WHERE SIFK = D.SIFK
:  :  :  AND SIFN = :vSifN ) ;
:  :  IF ( SQLCODE != NOT_FOUND )
:  :  |  vIshod = 'VEC_IMA'
:  :  |  OUTPUT ( "Clan ima knjigu sa traženim naslovom" )
:  Provera ima li slobodne knjige
:  :  Provera da li je za clana rezervisana knjiga
:  :  :  IF ( vIshod == 'NEMA_KNJIGE' )
:  :  :  |  SELECT SIFK
:  :  :  :  INTO :vSifK
:  :  :  :  FROM JE_REZERVISANA JR
:  :  :  :  WHERE SIFC = :vSifC
:  :  :  :  AND EXISTS ( SELECT SIFK
:  :  :  :  :  FROM KNJIGA
:  :  :  :  :  WHERE SIFK = JR.SIFK
:  :  :  :  :  AND SIFN = :vSifN ) ;
:  :  :  |  IF ( SQLCODE != NOT_FOUND )
:  :  :  |  |  vIshod = 'IMA_KNJIGE'
:  Provera da li ima knjige koja nije kod clana
:  :  IF ( vIshod == 'NEMA_KNJIGE' )
:  :  |  SELECT SIFK
:  :  |  INTO :vSifK
:  :  |  FROM KNJIGA
:  :  |  WHERE SIFN = :vSifN
:  :  |  AND SIFK NOT IN ( SELECT SIFK
:  :  |  :  FROM DRZI
:  :  |  :  UNION
:  :  |  :  SELECT SIFK
:  :  |  :  FROM JE_REZERVISANA ) ;
:  :  |  IF ( SQLCODE != NOT_FOUND )
:  :  |  |  vIshod = 'IMA_KNJIGE'

```

nastavlja se

nastavak

```

: IF ( vIshod == 'IMA_KNJIGE' )
: | Evidentiranje izdavanja knjige
: | : OUTPUT ( 'Izdati clanu knjigu", vSifK )
: | : INSERT INTO DRZI
: | : VALUES ( :vSifK, :vSifC, Datum() ) ;
: +- ( vIshod == 'NEMA_KNJIGE' )
: | Evidentiranje rezervacije naslova
: | : OUTPUT ( "Unesite da li clan zeli rezervaciju" )
: | : INPUT ( vZeliRezervaciju )
: | : IF ( vZeliRezervaciju == 'DA' )
: | : | INSERT INTO REZERVACIJA
: | : | VALUES ( :vSifN, :vSifC, DatumVreme() )

```

Ovakva algoritamska specifikacija je korak do implementacije: ako se formalizuje, moguće ju je automatski prevesti na izabrani programski jezik uz minimalne dodatne intervencije programera.

### 8.3 Zaključne napomene

U prethodnim odeljcima razmotrena su oba vida projektovanja relacione baze podataka: projektovanje podataka i projektovanje postupaka. Ostalo je otvoreno pitanje njihovog redosleda, odnosno da li se prvo u celini obavlja projektovanje podataka pa u celini projektovanje postupaka, ili je obrnuto.

Odgovor glasi: ni jedno ni drugo. U pitanju je objedinjeni postupak projektovanja postupaka i podataka kod koga projektovanje postupaka počinje prvo i uvek u izvesnoj meri prednjači u odnosu na projektovanje podataka, pri čemu se vrši postepena razrada “korak po korak”. To je razumljivo samo po sebi: da bi projektant mogao da sagleda koji su podaci potrebni u bazi podataka neophodno je da ima bar neku predstavu o tome kako budući sistem treba da funkcioniše. Sa svakim korakom razrade pojaviće se nove funkcije nižeg reda koje će zahtevati uvođenje novih podataka. Pri tome, pogodno sredstvo za objedinjavanje projektovanja postupaka i podataka predstavlja matrica “postupci – operacije” koju smo u pojednostavljenoj formi tabele već izložili u poglavlju 4.

Uz prethodno, treba naglasiti i to da je vrlo važno da se na samom početku projektovanja sagleda šta budući korisnici očekuju od baze podataka. To se sve može formulisati u vidu odgovarajućih upita iz kojih proizilazi i potreba za odgovarajućim podacima u bazi podataka.

Takođe, sastavni deo projektovanja relacione baze podataka jeste formulacija dinamičkih uslova referencijalnog integriteta. To je za primer BIBLIOTEKA već urađeno u poglavljima 4 i 6 i stoga neće biti ponavljano ovde.

Prilog

*A*

***RELACIONA BAZA PODATAKA 'BIBLIOTEKA'***

## A.1 Šema relacione baze podataka

Relaciona baza podataka BIBLIOTEKA je jednostavan ali realan primer: Vodi se evidencija o oblastima, naslovima, autorima i autorstvu, knjigama i članovima, a pored toga što se prati koji članovi drže koje knjige kod sebe prate se i svi podaci o prošlim pozajmicama i budućim rezervacijama.

OBLAST ( <u>SIFO</u> , NAZIV )	- oblasti iz kojih su naslovi
NASLOV ( <u>SIFN</u> , NAZIV, SIFO )	- naslovi i oblasti iz kojih su
AUTOR ( <u>SIFA</u> , IME )	- autori
KNJIGA ( <u>SIFK</u> , SIFN )	- knjige i naslovi koje sadrže
CLAN ( <u>SIFC</u> , IME )	- članovi
POZAJMICA ( <u>SIFP</u> , SIFC, SIFK, SIFN, DANA )	- pozajmice
REZERVACIJA ( <u>SIFN</u> , <u>SIFC</u> , DATUM )	- rezervacije
JE_AUTOR ( <u>SIFA</u> , <u>SIFN</u> , KOJI )	- autorstva naslova
DRZI ( <u>SIFK</u> , SIFC, DATUM )	- knjige koje drže članovi
JE_REZERVISANA ( <u>SIFK</u> , SIFC, DATUM )	- knjige koje su rezervisane

## A.2 Sadržaj relacione baze podataka

oblast ( <u>SIFO</u> NAZIV )	naslov ( <u>SIFN</u> NAZIV SIFO )
BP Baze podataka	RBP0 Relacione baze podataka BP
RM Računarske mreže	RK00 Računarske komunikacije RM
PJ Programski jezici	PP00 PASCAL programiranje PJ
	PJC0 Programski jezik C PJ

autor ( <u>SIFA</u> IME )	knjiga ( <u>SIFK</u> SIFN )	clan ( <u>SIFC</u> IME )
AP0 A.Popović	001 RBP0	JJ0 J.Janković
IT0 I.Todorović	002 RBP0	PP0 P.Petrović
AP1 A.Petrović	003 RK00	JJ1 J.Jovanović
JN0 J.Nikolić	004 PJC0	MM0 M.Marković
DM0 D.Marković	005 PJC0	
ZP0 Z.Petrović	006 PJC0	
	007 PP00	
	008 PP00	
	009 PP00	

pozajmica ( <u>SIFP</u> SIFC SIFK SIFN DANA )	rezervacija ( <u>SIFN</u> SIFC DATUM )
1 JJ0 004 PJC0 5	RBP0 JJ1 18.10.95
2 PP0 007 PP00 2	RBP0 MM0 20.10.95
3 JJ1 005 PJC0 6	
4 JJ0 008 PP00 7	
5 PP0 002 RBP0 4	
6 JJ1 009 RBP0 3	

je_autor ( <u>SIFA</u> <u>SIFN</u> KOJI )	drzi ( <u>SIFK</u> SIFC DATUM )
AP0 RBP0 1	001 JJ0 10.10.95
JN0 RBP0 2	002 PP0 15.10.95
DM0 RK00 1	004 JJ0 18.10.95
ZP0 PP00 1	
DM0 PP00 2	
AP1 PJC0 1	
IT0 PP00 3	
ZP0 PJC0 2	

je\_rezervisana ( SIFK SIFC DATUM )

### A.3 Opis rada biblioteke

Osnovni zadatak biblioteke jeste da opslužuje svoje članove kojima pozajmljuje knjige. U tu svrhu, pored evidencije o članovima postoji i evidencija o oblastima, naslovima, autorima i pojedinačnim knjigama. Pri tome, u evidenciji mogu biti i naslovi za koje trenutno ne postoji ni jedan primerak knjige, oblasti iz kojih nema naslova i autori za koje trenutno ne postoje naslovi. Međutim, za svaki naslov u evidenciji postoji evidencija o tome ko je njegov autor, odnosno ko su autori ako ih ima više i ko je koji autor (prvi, drugi itd.). Takođe se kao osnovna vodi evidencija koje knjige i od kog datuma se nalaze pozajmljene kod kojih članova.

Uz navedene evidencije, postoji i evidencija o svim pozajmicama iz prošlosti koja treba da obezbedi uvid u čitanost pojedinih naslova, promet pojedinih knjiga i aktivnost pojedinih članova. Kada god neki član vrati neku knjigu evidentira se koji je to član, koja knjiga, koji naslov i koliko dana je trajala pozajmica.

U situacijama kada članovi traže naslove za koje trenutno ne postoji ni jedan slobodni primerak knjige predviđena je mogućnost rezervacije. U svakoj takvoj situaciji i ako član to zatraži, evidentira se da on ima rezervaciju za traženi naslov kao i trenutak rezervacije – datum i vreme. Vreme se evidentira u satima, minutima i sekundama, kako bi se u slučajevima kada više članova ima rezervaciju za isti naslov od istog datuma moglo rešiti pitanje prioriteta.

Opisana mogućnost rezervacije podrazumeva da se prilikom vraćanja svake knjige proverava da li za njen naslov postoji rezervacija za nekog člana. Ako postoji, knjiga se odvaja u stranu i istovremeno se evidentira da je rezervisana za dotičnog člana i brišu se podaci o rezervaciji naslova kao opsluženoj. Ako više članova ima rezervaciju, opslužuje se ona najranija. Rezervacija knjige je vremenski oročena: ako je član ne podigne u određenom roku od datuma kada je rezervisana, rezervacija se poništava i knjiga postaje slobodna.

U navedenim okolnostima, kada član traži neki naslov prvo se proverava da li on već ima knjigu sa tim naslovom (tada mu se ne izdaje nova knjiga), zatim da li je za njega kao rezervisana odvojena knjiga sa tim naslovom i tek onda se proverava da li ima slobodnu knjigu sa tim naslovom.

Članovi mogu ako to zažele da istupe iz članstva odnosno da se brišu iz evidencije pod uslovom da kod sebe ne drže ni jednu pozajmljenu knjigu. U suprotnom, moraju prethodno da vrate knjige koje drže ili da ih plate ako su ih izgubili.

Knjiga se briše iz evidencije ako ju je član izgubio ili ako je vremenom postala oštećena i neupotrebljiva.



Prilog

*B*

***SINTAKSNA I ALGORITAMSKA NOTACIJA***

## B.1 Sintaksna notacija sa zagradama

U osnovi svake sintaksne notacije nalaze se sledeći elementi:

- sintaksni pojam: leksička konstrukcija koja se definiše;
- metasimbol: element sintaksne notacije koji služi za konstrukciju definicije sintaksnog pojma;
- terminalni simbol: znak ili ključna reč kao najniži i finalni element u sastavu definicije sintaksnog simbola.

U ovom odeljku razmotrićemo kao najprikladniju sintaksnu notaciju sa zagradama. Kao prvo, naglasimo da se u toj notaciji terminalni simboli pišu podvučeno a sintaksni pojmovi obično. Na primer, BEGIN i ; su terminalni simboli dok Naziv to nije. Drugo, definicija bilo kakvog sintaksnog pojma se formira primenom pogodno odabranog skupa metasimbola čije objašnjenje sledi:

- ::=** ima značenje “definiciono jednako”; svaka definicija je forme **Pojam ::= ... ;**
- [ ]** ono što se nalazi između zagrada može se pojaviti jednom ili nijednom;
- { }** ono što se nalazi između zagrada mora se pojaviti jednom;
- |** uzima se ono levo ili ono desno od znaka **|** koji označava izbor;
- ...** ono što se nalazi levo od **...** uzima se 0 ili više puta.

Između navedenih metasimbola ne postoje nikakvi prioriteti. Željeni “redosled” primene ovih simbola postiže se grupisanjem delova definicije između para vitičastih ili uglastih zagrada. Sledi nekoliko ilustrativnih primera:

*Definicija označenog celog broja*

```
OznaceniCeoBroj ::= [ + | - ] Cifra Cifra ...
Cifra ::= 0|1|2|3|4|5|6|7|8|9
```

*Definicija naziva varijable - simbola*

```
Simbol ::= Slovo { Slovo | Cifra | - } ...
```

*Definicija C sekvence naredbi*

```
SekvencaNaredbi ::= { Naredba ; { Naredba ; } ... }
```

*Definicija C prototipa funkcije*

```
PrototipFunkcije ::=
  Tip Funkcija ( Tip Simbol [ , Tip Simbol ] ... ) ;
```

Situacija da se neki sintaksni pojam pojavljuje jednom ili više puta odvojeno zarezima je toliko česta da je za to uveden posebni metasimbol:

,... sve što se nalazi levo od ,... uzima se jednom ili više puta odvojeno zarezima kao terminalnim simbolima.

Sa takvom notacijom, definicija prototipa C funkcije je mnogo jednostavnija:

```
PrototipFunkcije ::= Tip Funkcija ( { Tip Simbol } ,... ) ;
```

## B.2 Algoritamska notacija akcionih dijagrama

Notacija akcionih dijagrama predstavlja jednostavno sredstvo za strukturiranu specifikaciju algoritama. Prednost te notacije u odnosu na grafičke strukturirane dijagrame i blok-dijagrame toka je dvojaka:

- u osnovi se radi o tekstuelnoj notaciji za koju nije potreban poseban editor;
- postupak postepenog razvoja algoritma ostaje vidljiv na finalnoj specifikaciji.

Razradu nekog postuka **Postupak** na podpostupke **Postupak1** i **Postupak2** predstavljamo pomoću znaka dvotačke i uvlačenjem, kako je prikazano:

```

Postupak
->      Postupak
        : Postupak1
        : Postupak2

```

Time se ujedno formira i sekvenca postupaka. Podrazumevani redosled u sekvenci je od gore na dole i, ako je u jednom redu navedeno više postupaka, s leva na desno.

Uslovno grananje **IF** u svim varijantama predstavlja se na sledeći način:

```

IF ( LogickiIzraz )      IF ( LogickiIzraz )      IF ( LogickiIzraz1 )
| PostupakIf             | PostupakIf             | PostupakIf1
                        +-
                        | PostupakElse             +- ( LogickiIzraz2 )
                        |                           | PostupakIf2
                        |                           +-
                        |                           | PostupakElse

```

Ponavljjanje **DO** u svim varijantama predstavlja se kako je prikazano:

while petlja “dokle god”

```
DO WHILE ( LogickiIzraz )
|   Postupak
```

until petlja “sve dok ne”

```
DO
|   Postupak
+-- UNTIL ( LogickiIzraz )
```

for petlja “za svako i”

```
DO FOR ( i = Poc,Zad,Korak )
|   Postupak
```

for petlja “za svaki element u skupu”

```
DO FOR EACH ( Element IN Skup )
|   Postupak
```

for petlja “zauvek” (podrazumeva se da u telu postoji iskok)

```
DO FOREVER
|   Postupak
```

Iskok iz proizvoljno ugnježđenog dela algoritma predstavlja se kao

```
<---EXIT
```

pri čemu se vrh strelice povlači do nivoa do koga se iskače. Sam iskok ima smisla samo ako je uslovljen, odnosno ako se nalazi neposredno unutar nekog **IF**.

Napomenimo i to da se kvalifikacija argumenata podprograma kao ulaznih, izlaznih i ulazno-izlaznih postiže oznakama **>**, **<** i **<>** s leva, kao i da se ulaz podataka sa standardnog ulaza i izlaz podataka na standardni izlaz predstavljaju pomoću procedura **INPUT()** i **OUTPUT()** gde se između zagrada odvojeno zarezima navodi šta se učitava odnosno ispisuje.

Na kraju, neka nam kao ilustrativni primer posluži postupak nalaženja nule funkcije metodom bisekcije. Polazni interval  $[A, B]$  se polovi sve dok ne nastupi jedna od sledećih situacija:

- sredina intervala je egzaktna nula;
- sredina intervala je približna nula koja zadovoljava uslov  $B-A < \text{Eps}$  ;
- ni tačna ni približna nula nisu dostignuti za  $\text{Maxit}$  iteracija.

Postupak kao rezultat vraća ishod (kako je završen), dostignute granice intervala i dostignutu nulu funkcije. Specifikacija algoritma je sprovedena postupno. Radi konciznosti nisu navedene deklaracije varijabli niti simboličkih konstanti.

Korak 1.

```
Bisekcija ( < Ishod, <> A, <> B, < X0, > Eps, > Maxit )
```

Korak 2.

```
Bisekcija ( < Ishod, <> A, <> B, < X0, > Eps, > Maxit )
: Inicijalizacija
: Obrada
: Finalizacija
```

Korak 3.

```
Bisekcija ( < Ishod, <> A, <> B, < X0, > Eps, > Maxit )
: Inicijalizacija
: : Postavljanje pocetnog ishoda
: Obrada
: : DO FOR ( i = 1, Maxit, 1 )
: : | Jedna iteracija
: Finalizacija
```

Korak 4.

```
Bisekcija ( < Ishod, <> A, <> B, < X0, > Eps, > Maxit )
: Inicijalizacija
: : Postavljanje pocetnog ishoda
: : : Ishod = NEDOVOLJNO_ITERACIJA
: Obrada
: : DO FOR ( i = 1, Maxit, 1 )
: : | Jedna iteracija
: : | : Odredjivanje sredisne tacke
: : | : Test na egzaktnu nulu
: : | : Test na približnu nulu
: : | : Odredjivanje novog intervala
: Finalizacija
```

Korak 5 - konačno.

```

Bisekcija ( < Ishod, <> A, <> B, < X0, > Eps, > Maxit )
: Inicijalizacija
: : Postavljanje pocetnog ishoda
: : : Ishod = NEDOVOLJNO_ITERACIJA
: Obrada
: : DO FOR ( i = 1, Maxit, 1 )
: : | Jedna iteracija
: : | : Odredjivanje sredisne tacke
: : | : : X0 = (B+A)/2.
: : | : Test na egzaktnu nulu
: : | : : IF ( F(X0) == 0. )
: : | : : | Ishod = EGZAKTNA_NULA
: <:--|--:--:--|--EXIT
: : | : Test na pribliznu nulu
: : | : : IF ( B-A < Eps )
: : | : : | Ishod = PRIBLIZNA_NULA
: <:--|--:--:--|--EXIT
: : | : Odredjivanje novog intervala
: : | : : IF ( F(A)*F(X0) < 0 )
: : | : : | B = X0
: : | : : +-
: : | : : | A = X0

```

U ovom primeru, ispostavilo se da je postupak **Finalizacija** prazan pa je zato izpstavljen iz finalne specifikacije.

Prilog

*C*

***UGRAĐENI PROGRAMSKI SQL***



Sve SQL naredbe koje smo obradili u poglavlju 6 mogu se koristiti u interaktivnom režimu rada sa bazom podataka. U takvom režimu rada, korisnik zadaje jednu po jednu SQL naredbu, a sistem za upravljanje bazom podataka je izvršava ili odbacuje ako je po bilo kom osnovu neispravna. Pri tome, korisnik može da uradi sve što je u granicama njegovih prava.

Ono što odmah možemo da uočimo u vezi interaktivnog režima rada jeste nemogućnost formiranja procedura koje bi automatizovale kompleksne manipulacije nad bazom podataka.

Uz prethodno, ozbiljna mana interaktivnog rada sa bazom podataka jeste i to što od svakog korisnika traži poznavanje SQL jezika. U tradicionalnim informacionim sistemima to nije bio slučaj: korisnik je pristupao podacima posredstvom posebno izrađenih programa, i njegovo je bilo samo da vrši izbor opcija, unos podataka i uvid u razne preglede, bez ulaženja u programirane detalje manipulacije podacima.

Navedene okolnosti uslovile su da SQL jezik od početka sadrži naredbe koje omogućavaju njegovo korišćenje u programima pisanim na standardnim proceduralnim programskim jezicima. Način na koji se to ostvaruje već smo opisali u poglavlju 2:

- u izvornom programu na nekom standardnom programskom jeziku na određenim mestima se ubacuju standardni SQL delovi teksta;
- SQL preprocesor prevodi SQL delove teksta u pozive posebnih funkcija i procedura, što daje izvorni program po standardu za određeni programski jezik;
- standardni izvorni program se prevodi i povezuje u izvršnu celinu, prilikom čega se uz standardnu koristi i SQL biblioteka posebnih funkcija i procedura.

Na opisani način dobijaju se izvršni programi koji kombinuju sve prednosti tradicionalnih programskih jezika i SQL jezika.

U narednim odeljcima izložene su osnove upotrebe SQL jezika u programima pisanim na standardnim programskim jezicima, pri čemu se ograničavamo na programski jezik C. Radi preglednosti, sve programske SQL konstrukcije u C programskom tekstu koji sledi naglašene su tamnijim slovima i znakovima, a nebitni delovi programa kao što su unošenje i ispisivanje podataka navedeni su kao **<opis>**.

## C.1 Deklaracija programskih varijabli

Prilikom sastavljanja programa na nekom standardnom programskom jeziku za rad sa bazom podataka, prvo što moramo definisati jesu varijable koje služe kao sprega sa podacima u bazi podataka. Te varijable se koriste i u standardnim i u SQL delovima programa, a deklaracija koju prepoznaje SQL preprocesor je sledeće forme:

```
DeklaracijaSpreznihVarijabli ::=
  EXEC SQL BEGIN DECLARE SECTION ;
  DeklaracijeVarijabliProgramskogJezika
  EXEC SQL END DECLARE SECTION ;
```

Pri tome, korespondencija između tipova podataka SQL i C jezika je sledeća:

SQL tip	C tip
-----	-----
INTEGER	int
REAL	float
CHARACTER	char
CHARACTER [n]	char [n]
CHARACTER VARYING [n]	char [n+1] organizovan kao string

*Primer*

U C programu za rad sa tabelom Pozajmica deklaracija bi glasila:

```
EXEC SQL BEGIN DECLARE SECTION ;  
    int  SifP ;  
    char SifC [3] ;  
    char SifK [3] ;  
    char SifN [4] ;  
    int  Dana ;  
EXEC SQL END DECLARE SECTION ;
```

## C.2 Programski SELECT i INSERT za jedan red

Pod programskim **SELECT** i **INSERT** naredbama za jedan red podrazumevamo naredbe koje se odnose isključivo na jedan red tabele, odnosno:

- **SELECT** koji kao rezultat daje jedan red;
- **INSERT** koji ubacuje jedan red u tabelu.

Navedena **SELECT** naredba se po izvesnim dodacima razlikuje od one koja se koristi u interaktivnom režimu rada. Njena sintaksna definicija glasi:

```
ProgramskiSELECTJedanRed ::=
    EXEC SQL SELECT      { R-Izraz | G-Izraz } ...
                        INTO      { :Varijabla } ...
                        FROM      { Tabela [ Nadimak ] } ...
    [ WHERE      R-Predikat ]
    [ GROUP BY  Kolona ...
    [ HAVING      G-Predikat ] ] ;
```

Jedino ograničenje koje važi jeste to da se uvek dobija rezultat u formi jednog i samo jednog reda. Unutar tog ograničenja, dozvoljeno je sve što smo do sada naveli za **SELECT** naredbu: svodenje, spajanje, skupovne operacije, podupiti, pogledi. U **R-Predikat** se mogu koristiti i programske varijable, sa dvotačkom ispred naziva. U **INTO** klauzuli navode se, sa dvotačkom ispred, jedna ili više programskih varijabli odvojenih zarezima. Te varijable moraju odgovarati po broju, redosledu i tipu elementima iza **SELECT** klauzule.

Navedena naredba **INSERT** za programski režim rada je forme:

```
ProgramskiINSERTJedanRed ::=
    EXEC SQL INSERT INTO Tabela [ ( Kolona ... ) ]
    VALUES ( { { :Varijabla } | Konstanta } ... ) ;
```

Razlika u odnosu na interaktivnu formu **INSERT** naredbe jeste ta što se kao vrednosti pored konstanti mogu navesti i programske varijable, a sve to mora po broju, redosledu i tipu biti saglasno sa deklaracijom kolona, ili ako ona nije data sa **CREATE TABLE** naredbom za tabelu **Tabela**.

*Primeri*

Sledeći programski segment na C jeziku za datu postojeću šifru naslova prikazuje naziv naslova i naziv oblasti:

```

...
EXEC SQL BEGIN DECLARE SECTION ;
      char [4]   SifN ;
      char [20]  NazivN ;
      char [20]  NazivO ;
EXEC SQL END DECLARE SECTION ;
...
{
  < učitavanje SifN > ;
  EXEC SQL SELECT N.Naziv, O.Naziv
            INTO   :NazivN, :NazivO
            FROM   Naslov N, Oblast O
            WHERE  N.sifo = O.sifo
            AND    N.sifN = :SifN ;
  < ispisivanje NazivN i NazivO > ;
}

```

Programski segment na C jeziku kojim se unosi podatak o novom članu glasi:

```

...
EXEC SQL BEGIN DECLARE SECTION ;
      char [3]   SifC ;
      char [15]  Ime ;
EXEC SQL END DECLARE SECTION ;
...
{
  < učitavanje SifC i Ime > ;
  EXEC SQL INSERT INTO Clan
            VALUES ( :SifC, :Ime ) ;
}

```

### C.3 Programski DELETE i UPDATE

Za naredbe programskog ažuriranja **UPDATE** i **DELETE** ne postoje ograničenja na jedan red tabele, i njima se može postići ažuriranje ni jednog, jednog ili više redova, uključujući i sve.

Sintaksna definicija programske naredbe uklanjanja **DELETE** u neposrednoj formi (posredna će biti izložena kasnije) glasi:

```
ProgramskiDELETE ::=
    EXEC SQL DELETE FROM Tabela [ Nadimak ]
    [ WHERE R-Predikat ] ;
```

gde za **R-Predikat** važi sve ranije rečeno, uz dodatne napomene:

- mogu se koristiti i programske varijable, sa dvotačkom ispred naziva;
- nije dozvoljen podupit nad tabelom iz koje uklanjamo redove.

Izvršenjem ove naredbe uklanjaju se iz tabele **Tabela** oni redovi za koje je zadovoljen **R-Predikat** ili svi redovi ako **R-Predikat** nije zadat.

Za programsku naredbu izmene **UPDATE** u neposrednoj formi imamo sledeću definiciju

```
ProgramskiUPDATE ::=
    EXEC SQL UPDATE Tabela [ Nadimak ]
    SET { Kolona = R-Izraz } ...
    [ WHERE R-Predikat ] ;
```

uz sledeće napomene:

- u **R-Izraz** i **R-Predikat** se mogu koristiti i programske varijable, sa dvotačkom ispred naziva;
- u **R-Predikat** nije dozvoljen podupit nad tabelom u kojoj menjamo redove.

Ovakva naredba vrši naznačene izmene u onim redovima tabele **Tabela** za koje je zadovoljen **R-Predikat** ili u svim ako **R-Predikat** nije zadat.

I pored jednostavnosti navedenih formi naredbi, one se retko koriste u praksi. Umesto toga, koristi se posredna forma koja se kao i druga forma **SELECT** naredbe zasniva na prethodnom definisanju skupa redova koji su predmet manipulacije i potom iterativnoj obradi tog skupa “red po red”.

*Primeri*

Sledeći programski segment uklanja podatke o pozajmicama zadate knjige:

```
...
EXEC SQL BEGIN DECLARE SECTION ;
      char [3] SifK ;
EXEC SQL END DECLARE SECTION ;
...
{
  < učitavanje SifK > ;
  EXEC SQL DELETE FROM  Pozajmica
                        WHERE SifK = :SifK ;
}
```

Programski segment za ispravku pogrešno unete šifre oblasti naslova glasi

```
...
EXEC SQL BEGIN DECLARE SECTION ;
      char [4] SifN ;
      char [2] SifO ;
EXEC SQL END DECLARE SECTION ;
...
{
  < učitavanje SifN i SifO > ;
  EXEC SQL UPDATE Naslov
      SET      SifO = :SifO
      WHERE SifN = :SifN ;
}
```

## C.4 Kursor i operacije nad njim

Navedimo dve bitne razlike između interaktivnog jezika SQL i standardnih proceduralnih programskih jezika:

- interaktivni SQL jezik ne raspolaže sa konstrukcijama grananja i ponavljanja.
- proceduralni jezici su u radu sa datotekama slogovno orijentisani, u tom smislu da se podaci obrađuju “slog po slog” i da uvek postoji tekući slog kome se pristupa; nasuprot tome, SQL jezik je tabelarno orijentisan - predmet manipulacija su cele tabele i rezultati su u opštem slučaju cele tabele;

Prva razlika se premoštava kombinacijom standardnih i programskog SQL jezika, dok nam druga do sada nije predstavljala problem s obzirom na jednostavnost naših primera. Sve programske **SELECT** naredbe koje smo do sada naveli odnosile su se na samo jedan red rezultatnih podataka i takav rezultat se mogao preuzeti **INTO** klauzulom u programske varijable.

Za slučajeve upita koji kao rezultat daju više redova programski SQL predviđa sledeće dodatne konstrukcije:

- deklaraciju skupa rezultatnih redova preko definicionog upita; takav skup se u SQL terminologiji naziva kursor
- otvaranje kursora, što podrazumeva izvršenje definicionog upita i nastanak rezultatnog skupa redova; nakon toga, nad njim se mogu obavljati operacije “red po red”, što odgovara slogovnoj orijentaciji proceduralnih programskih jezika;
- zatvaranje kursora, kada više nije potreban.



Razmotrimo prvo naredbu deklaracije kursora, čija je sintaksna definicija:

```
DeklaracijaKursora ::=
    EXEC SQL DECLARE CURSOR Kursor
    FOR R-Upit ;
```

Ovde **R-Upit** može biti bilo koja od formi koje smo obradili u odeljcima o **SELECT** naredbi u poglavlju 6. U slučaju da nad redovima rezultata želimo da vršimo ažuriranje, na snazi su ograničenja koja smo definisali za ažurabilnost pogleda. Unutar istog programa možemo deklarirati više kursora sa različitim nazivima **Kursor** koji se formiraju po pravilima za nazive programskih varijabli.

Prethodna naredba predstavlja samo deklaraciju. Da bi skup redova koji čine kursor bio dostupan za obradu, kursor treba otvoriti naredbom čija je forma:

```
OtvaranjeKursora ::=
    EXEC SQL OPEN CURSOR Kursor ;
```

Po izvršenju naredbe otvaranja kursora, prvi rezultatni red (ako ga ima) u skupu nastalom izvršenjem definicionog upita postaje tekući red.

Na kraju programa ili kada nam kursor više ne treba, kursor zatvaramo naredbom sledeće sintakse:

```
ZatvaranjeKursora ::=
    EXEC SQL CLOSE CURSOR Kursor ;
```

Naredba učitavanje podataka iz tekućeg reda kursora je forme:

```
CitanjeRedaKursora ::=
    EXEC SQL FETCH Kursor
    INTO :Varijabla ,... ;
```

Navedena lista programskih varijabli mora biti saglasna po broju, redosledu i tipu sa listom kolona iz definicionog upita kursora. Posle izvršenja ove naredbe, naredni red u kursoru postaje tekući red.

Naredba izmene tekućeg reda ažurabilnog kursora je nešto složenija. Njena sintaksna definicija glasi:

```
IzmenaRedaKursora ::=
    EXEC SQL UPDATE Tabela
    SET { Kolona = R-Izraz } ,...
    WHERE CURRENT OF Kursor ;
```

Ovde su neophodne sledeće napomene:

- svakom **UPDATE** treba da prethodi **FETCH** iz istog kursora;
- **Tabela** je naziv tabele navedene u definicionom upitu kursora;
- **R-Izraz** može da sadrži i programske varijable sa dvotačkom ispred naziva;
- **WHERE CURRENT OF** klauzula znači da se izmena vrši u redu koji je prethodno pročitao sa **FETCH**.

Izvršenje ove naredbe ne pomera indikator tekućeg reda.

Preostaje još da razmotrimo naredbu uklanjanja tekućeg reda kursora, za koju važi sledeća sintaksna definicija:

**UklanjanjeRedaKursora ::=**

```
EXEC SQL DELETE FROM Tabela  
WHERE CURRENT OF Kursor ;
```

pri čemu važe napomene kao i za **UPDATE** naredbu. Nakon izvršenja ove naredbe, tekući red kursora postaje red koji se nalazio iza uklonjenog reda

Na kraju, napomenimo da se uslov za izmenu ili uklanjanje tekućeg reda kursora ili mora ugraditi u definicioni upit kursora ili razrešiti u programu uslovljavanjem izvršenja odgovarajuće SQL naredbe.

*Primeri*

Navešćemo primere za upit i operacije izmene i uklanjanja nad kursorom. Neka za sve navedene primere važe sledeće deklaracije varijabli i kursora:

Varijable:

```

...
EXEC SQL BEGIN DECLARE SECTION ;
      char [4]   SifN ;
      char [20]  Naziv ;
      char [2]   SifO ;
      char [2]   SifOZad ;
      char [2]   SifOSta ;
      char [2]   SifONov ;
EXEC SQL END DECLARE SECTION ;
...

```

*Deklaracija kursora:*

```

...
EXEC SQL DECLARE CURSOR SviNaslovi
                        FOR SELECT *
                        FROM Naslov ;
...

```

*Upit*

Programski segment za prikaz naziva svih naslova zadate šifre oblasti:

```
...
{
EXEC SQL OPEN CURSOR SviNaslovi ;
< učitavanje SifoZad > ;
< ispisivanje SifoZad > ;
do while (1)                                // Objasnjenje naknadno
{
EXEC SQL FETCH SviNaslovi
INTO :SifN, :Naziv, :Sifo ;
if ( < NistaNijeUcitano > ) // Objasnjenje naknadno
break ;
if ( Sifo == SifoZad )
< ispisivanje Naziv > ;
}
EXEC SQL CLOSE CURSOR Svi Naslovi ;
}
```

*Izmena*

Programski segment kojim se stara šifra oblasti naslova menja u novu:

```

...
{
EXEC SQL OPEN CURSOR SviNaslovi ;
< učitavanje SifOSta i SifONov > ;
do while (1)
{
EXEC SQL FETCH SviNaslovi
            INTO :SifN, :Naziv, :SifO ;
if ( < NistaNiJeUcitano > )
    break ;
if ( SifO == SifOSta )
    EXEC SQL  UPDATE Naslov
                SET SifO = :SifONov
                WHERE CURRENT OF SviNaslovi ;
}
EXEC SQL CLOSE CURSOR SviNaslovi
}

```

*Brisanje*

Programski segment za uklanjanje naslova zadane šifre oblasti.

```

...
{
EXEC SQL OPEN CURSOR SviNaslovi ;
< učitavanje SifoZad > ;
do while (1)
{
EXEC SQL FETCH SviNaslovi
            INTO :SifN, :Naziv, :Sifo ;
if ( < NistaNijeUcitano > )
    break ;
if ( Sifo == SifoZad )
    EXEC SQL DELETE FROM Naslov
            WHERE CURRENT OF SviNaslovi ;
}
EXEC SQL CLOSE CURSOR SviNaslovi ;
}

```

Uz prethodna tri primera neophodna su određena objašnjenja. U sva tri slučaja korišćena je безусловna programska petlja `do while (1)` iz koje se iskače kada SQL naredba čitanja reda iz kursora ne vrati ništa pošto nema više redova za čitanje. To podrazumeva da programski SQL podržava ispitivanje ishoda operacija.

## C.5 Varijabla SQLCODE i naredba WHENEVER

U prethodnim primerima ostalo je nerešeno pitanje prepoznavanja kraja kursora, odnosno situacije kada **FETCH** naredba više nema šta da čita. Isto tako, u ranijim jednostavnim primerima ostavili smo nerešene situacije traženja reda koji ne postoji, ubacivanja reda sa vrednošću primarnog ključa koja već postoji u tabeli, i slično. Sve te situacije razrešene su u programskom SQL jeziku preko posebne sistemske varijable **SQLCODE** koja se u programu mora deklarirati kao varijabla tipa **INTEGER** i koja se posle izvršenja svake programske SQL naredbe postavlja na vrednost koja govori o ishodu, u smislu uspešnog izvršenja ili neke greške.

Vrednosti na koje se postavlja **SQLCODE** posle izvršenja svake programske SQL naredbe pokrivaju sve moguće ishode, odnosno sledeće situacije:

- komanda je izvršena bez greške i imala je efekta, odnosno nije nastupila ni jedna od niže navedenih situacija:  
**SQLCODE** se postavlja na vrednost 0 ;
- komanda je izvršena bez greške ali nije imala nikakvog efekta, odnosno: **SELECT** nije vratio ni jedan red rezultata, **FETCH** nema šta da učita (već je učitao poslednji red kursora ili njegov definicioni upit nije dao ni jedan red), za neposredni **UPDATE** ili **DELETE** ni jedan red tabele ne zadovoljava **R-Predikat** u **WHERE** klauzuli:  
**SQLCODE** se postavlja na 100;
- prilikom izvršenja komande nastupila je neka greška; svi dotadašnji efekti komande se poništavaju:  
**SQLCODE** se postavlja na neku negativnu vrednost koja bliže identifikuje nastalu grešku; konkretna vrednost zavisi od implementacije.

Varijablu ishoda `SQLCODE` možemo u programu koristiti u uslovnim grananjima i petljama i tako obraditi svaki mogući ishod, ali i pored toga u SQL jeziku postoji posebna naredba grananja (u stvari obrade izuzetka) čija je forma:

**DefinicijaObradeIzuzetka ::=**

EXEC SQL WHENEVER Ishod CONTINUE | { GOTO Labela } ;

**Ishod ::=**

Konstanta | { { SQLWARNING | NOT FOUND } | SQLERROR }

čije je značenje je sledeće: ako posle neke operacije `SQLCODE` vrati vrednost `Ishod`, ili se izvršava sledeća programska naredba ako je navedena akcija `CONTINUE` ili se vrši skok na programsku naredbu sa labelom `Labela` ako je navedena `GOTO` akcija. Ishod se osim brojnom konstantom može specificirati i navođenjem jedne od standardnih simboličkih konstanti za `SQLCODE`:

- `SQLWARNING` ili `NOT FOUND` kao ekvivalent vrednosti 100;
- `SQLERROR` za slučaj neke greške, odnosno kao ekvivalent negativne vrednosti.



## C.6 Primer složenog održavanja baze podataka

Kao primer složenog održavanja baze podataka poslužiće nam situacija kada član vraća pozajmljenu knjigu, o čemu je već bilo reči na kraju poglavlja 4. Neka nam kao polazna osnova za to posluži ranije navedena algoritamska specifikacija postupka `VracanjeKnjige`, i to u varijanti koraka 2:

```
VracanjeKnjige ( > pSifK )
: Inicijalizacija
: Obrada
: : Uvid u neophodne podatke
: : : Uvid koji je clan i kada uzeo knjigu
: : : Uvid kojeg je naslova knjiga
: : Evidentiranje vracanja knjige
: : Evidentiranje obavljene pozajmice
: : : Odredjivanje nove vrednosti primarnog kljuca
: : : Odredjivanje koliko je trajala pozajmica
: : : Evidentiranje podataka o pozajmici
: : Obrada eventualne rezervacije
: : : Provera da li ima rezervacije za vracenu knjige
: : : Opsluzivanje rezervacije (ako treba)
: : : : Nalazenje prioritete rezervacije
: : : : Evidentiranje da je rezervacija opsluzena
: : : : Evidentiranje da je knjiga rezervisana
: Finalizacija
```

Sledi odgovarajuća funkcija na programskom jeziku C sa ubačenim programskim SQL naredbama. Prethodno je potrebno da definišemo određene pomoćne funkcije koje su neophodne s obzirom da se u bazi podataka BIBLIOTEKA kao tip većine atributa koristi niz znakova umesto stringa:

```
void KopijaNiza ( char *NizU, char *NizIz, int n ) ;
```

Kopira n znakova iz znakovog niza **NizIz** u znakovni niz **NizU**;

```
int BrojDana ( char *Datum ) ;
```

za datum u obliku znakovnog niza **Datum** dužine 8 vraća broj dana između tog datuma i tekućeg datuma;

```
void TekuciDatum ( char *Datum ) ;
```

vraća tekući datum u obliku znakovnog niza **Datum** dužine 8'.

Sama funkcija **VracanjeKnjige** glasi:

```
#include "sql.h" ;
int VracanjeKnjige ( char *pSifK )
{
    /* Inicijalizacija */
    {
        enum ( NE_REZERVISANA, JE_REZERVISANA ) ;
        int vIshod = NE_REZERVISANA ;
        EXEC SQL BEGIN DECLARE SECTION ;
            char vSifC [3] ;
            char vSifN [4] ;
            char vSifK [3] ;
            char vDatum [8] ;
            char vDatumVreme [14] ;
            int vSifP ;
            int vDana ;

        EXEC SQL END DECLARE SECTION ;

        KopijaNiz ( vSifK, pSifK, 3 ) ;
    }

    /* Obrada */
    {
        /* Uvid u neophodne podatke */
        {
            EXEC SQL SELECT SIFC, DATE_CHAR ( DATUM )
                INTO :vSifC, :vDatum
                FROM DRZI
                WHERE SIFK = :vSifK ;

            EXEC SQL SELECT SIFN
                INTO :vSifN
                FROM KNJIGA
                WHERE SIFK = :vSifK ;
        }
    }
    /* Evidentiranje vracanja knjige */
    {
        EXEC SQL DELETE FROM DRZI
            WHERE SIFK = :vSifK ;
    }
}
```

nastavlja se

```

nastavak
/*      Evidentiranje obavljene pozajmice
    {
        EXEC SQL SELECT MAX ( SIFP )
            INTO :vSifp
            FROM POZAJMICA ;

        vSifP ++ ;
        vDana = BrojDana ( vDatum ) ;

        EXEC SQL INSERT INTO POZAJMICA
            VALUES ( :vSifP, :vSifC,
                :vSifK, :vSifN, :vDana ) ;
    }
/*      Obrada eventualne rezervacije * /
    {
        EXEC SQL SELECT MIN ( DATETIME_CHAR ( DATUM ) )
            INTO :vDatumVreme
            FROM REZERVACIJA
            WHERE SIFN = :vSifN ;

        if ( !SQLCODE )
        {
            vIshod = JE_REZERVISANA ;

            EXEC SQL SELECT SIFC
                INTO :vSifC
                FROM REZERVACIJA
                WHERE DATUM = CHAR_DATETIME ( :vDatumVreme ) ;

            EXEC SQL DELETE FROM REZERVACIJA
                WHERE DATUM = CHAR_DATETIME ( :vDatumVreme ) ;

            TekuciDatum ( vDatum ) ;

            EXEC SQL INSERT INTO JE_REZERVISANA
                VALUES ( :vSifK, :vSifC,
                    CHAR_DATE ( vDatum ) ;
        }
    }
/*      Finalizacija */
    {
        return vIshod ;
    }
}

```

Pri tome smo koristili sledeće funkcije konverzije koje pod istim ili drugačijim nazivima podržava većina SQL implementacija:

<code>DATE_CHAR</code>	konvertuje tip <code>DATE</code> u niz znakova;
<code>DATETIME_CHAR</code>	konvertuje tip <code>TIMESTAMP</code> u niz znakova;
<code>CHAR_DATE</code>	konvertuje niz znakova u tip <code>DATE</code> ;
<code>CHAR_DATETIME</code>	konvertuje niz znakova u tip <code>TIMESTAMP</code> .

# ***LITERATURA***

*ACM Portal, Digital Library, Transactions on Database Systems*,  
[www.acm.org/portal](http://www.acm.org/portal).

Batini C., Ceri S., Navathe S.B.: *"Conceptual Database Design"*,  
Benjamin-Cummings, 1992.

Connolly T., Begg C.: *"Database Systems: A Practical Approach to Design, Implementation, and Management"*, 4th Ed.,  
Addison-Wesley, 2005.

Date C.J.: *"An Introduction to Database Systems"*, 8th Ed.,  
Pearson /Addison-Wesley, 2004.

Date C.J., Darween H.: *"A Guide to the SQL Standard"*, 4th Ed.,  
Addison-Wesley, 1997.

Elmasri R., Navathe S.B.: *"Fundamentals of Database Systems"*, 4th Ed.,  
Pearson /Addison-Wesley, 2003.

Garcia-Molina H., Ullman J.D., Widom J.: *"Database systems: The Complete Book"*,  
Prentice-Hall, 2002.

Melton J., Simon A.R.: *"SQL 99: Understanding Relational Language Concepts"*,  
Academic Press, 2002.

Silberschatz A., Korth H.E., Sudarshan S.: *"Database System Concepts"*, 5th Ed.,  
Mc Graw-Hill, 2006.

Tkalac S.: *"Relacijski model podataka"*,  
Informator, 1988.