Uvod u relacione baze podataka

Gordana Pavlović-Lažetić

Sadržaj

0

Uvod

Tradicionalni pristup razvoju programskih sistema za čuvanje i obradu podataka, do pojave baza podataka krajem šezdesetih godina, bio je u suštini *ad hoc*: polazeći od definicije aplikacije, kreirala bi se datoteka ili skup datoteka podataka potrebnih za definisanu aplikaciju, i pisali bi se programi koji obrađuju podatke tih datoteka u skladu sa aplikacijom ([52]). Sva proširenja definisane aplikacije realizovala bi se dodavanjem novih datoteka i pisanjem dodatnih programa. Ovakav pristup je brzo pokazao sve svoje nedostatke:

- Ponavljanje istih podataka uz različite aplikacije. Na primer, u jednom bibliotečkom okruženju, aplikacije za obradu knjiga, reversa i čitalaca mogle bi da koriste odgovarajuće datoteke – KNJIGE, REVERSI, ČITAOCI, redom. pri tom bi, s obzirom na potrebe ovih aplikacija, datoteka REVERSI sadržala sve podatke o knjizi i čitaocu koji su već sadržani i u datoteci KNJIGE odnosno ČITAOCI.

- Nekonzistentnost podataka. Zbog ponavljanja istih podataka u raznim datotekama, može se dogoditi da se promena vrednosti primercima istog podatka ne izvede dosledno, tj. da se jednom primerku istog podatka vrednost promeni (namerno ili nenamerno), a da se to ne učini sa ostalim primercima tog podatka. Na primer, adresa istog čitaoca može da bude različito zapisana u datotekama ČITAOCI i REVERSI, što će proizvesti nekonzistentnost (nekorektnost) podataka.

- Programi za obradu podataka zavise od načina struktuiranja podataka. Tako će ti programi biti različiti u slučajevima različite organizacije datoteka (sekvencijalna, indeks-sekvencijalna, direktna, indeksirana, itd). Zato promena strukture podataka zahteva promenu programa.

- Obrada podataka je skupa, s obzirom na nekonzistentnost i zavisnost programa od organizacije podataka.

- Korišćenje istih podataka od strane većeg broja korisnika je otežano. Na primer, istovremeni pokušaj dva ili više korisnika da promene sadržaj datoteke REVERSI završiće se, u mnogim slučajevima, pamćenjem promena samo onog korisnika koji je poslednji završio rad sa datotekom. Ostali korisnici ni na koji način neće biti obavešteni o tome da njihove promene nisu upamćene. U drugim slučajevima višekorisničkog pristupa podacima smeštenim u sistem datoteka, kada se pristup vrši istoj kopiji datoteke, mogući su i teži primeri narušavanja korektnosti podataka.

- Neadekvatna realizacija oporavka od pada sistema. U slučaju pada sistema (npr. nestanka električnog napajanja), aktivni poslovi nemaju mogućnost poništavanja svojih delimičnih izvršenja (ako su ona deo jedinstvene logičke celine), a često, po uspostavljanju sistema, ni evidenciju o svom delimičnom izvršenju. Na primer, ako bi svim čitaocima trebalo produžiti (na reversu) rok zaduženja za po 10 dana, i ako bi usred izvršavanja tog posla došlo do pada sistema, po uspostavljanju rada ne bi postojala informacija o tome koji su reversi obrađeni a koji još nisu.

Da bi se mogle iskoristiti pogodnosti računara za čuvanje podataka, neophodno je obezbediti i programske alate prenosa podataka, upravljanja podacima i komunikacije sa korisnikom. Zato nova organizacija podataka, nazvana baza podataka, zajedno sa programskim alatima koji je podržavaju, treba da obezbedi:

- nezavisnost struktura podataka od programa koji ih obrađuju, i programa od struktura podataka (izmena podataka ne utiče na programe);

- minimalnost ponavljanja podataka;

- da obrada podataka nije vezana za programski jezik opšte namene, već za viši "upitni" jezik;

- korišćenje baze podataka od strane većeg broja korisnika.

Pojam *baza podataka* pojavio se krajem šezdesetih godina i označavao je skup međusobno povezanih podataka koji se čuvaju zajedno, i među kojima ima samo onoliko ponavljanja koliko je neophodno za njihovo optimalno korišćenje pri višekorisničkom radu. Podaci se pamte tako da budu nezavisni od programa koji ih koriste, i struktuiraju se tako da je omogućen porast baze. Posle svake aktivnosti nad bazom, koja predstavlja logičku celinu posla, stanje baze mora biti *konzistentno* (valjano); to znači da podaci u bazi i odnosi među njima moraju zadovoljavati unapred zadate uslove koji odslikavaju deo realnosti modelirane podacima u bazi. Za efikasan rad sa podacima i održavanje konzistentnog stanja baze koristi se specifični programski proizvod – sistem za upravljanje bazama podataka (kraće SUBP – DBMS, Data Base Management System). Baze podataka zajedno sa SUBP čine *sistem baza podataka*. U širem smislu, u sistem baza podataka spada i odgovarajući hardver, svi programski alati koji se nadgrađuju na SUBP i olakšavaju rad

sa bazama, kao i raznorodni korisnici baza podataka (od krajnjih korisnika, preko aplikativnih programera do administratora baza podataka).

U razvoju sistema baza podataka može se uočiti nekoliko generacija SUBP, koje su ili koegzistirale na tržištu ili smenjivale jedna drugu. Fundamentalna razlika između sistema ovih generacija je razlika u modelu podataka, koja se u implementaciji odgovarajućih SUBP reflektuje na efikasnost pristupa podacima i obrade podataka, produktivnost korisnika, funkcionalnost sistema i podršku raznovrsnim aplikacijama. Tako se u prve dve generacije svrstavaju tzv. mrežni (CODASYL) sistemi i hijerarhijski sistemi, koji su gotovo u potpunosti prevaziđeni, osamdesetih godina, relacionom tehnologijom kao trećom generacijom SUBP (Relacioni SUBP – RSUBP). Sve tri generacije namenjene su pre svega poslovno-orijentisanim aplikacijama.

Ograničenja relacione tehnologije odnose se na neadekvatnu podršku aplikacijama nad podacima kompleksnijim od podataka u poslovnoj obradi, npr. nad prostornim ili tekstuelnim podacima. Zato se relacioni model podataka proširuje konceptima objektnog modela, koji na adekvatniji način apstrahuje svojstva kompleksnih objekata; istovremeno, relaciona tehnologija se integriše sa objektnom tehnologijom gradeći tako novu, četvrtu generaciju SUBP. Objektna tehnologija je još uvek u istraživačkoj fazi, mada postoje i prototipske i komercijalne implementacije sistema za upravljanje bazama podataka. Teorijske osnove objektne tehnologije, za razliku od relacione, još nisu postavljene; ovo važi i za sam objektni model.

Ova knjiga odnosi se pre svega na relacionu tehnologiju baza podataka, počevši od osnovnih postavki relacionog modela i relacionih upitnih jezika, preko teorije logičkog projektovanja relacione baze podataka i semantičkog modeliranja, pa do problema i rešenja vezanih za implementaciju pune funkcionalnosti RSUBP.

Osnovna komponenta sistema za upravljanje relacionim bazama podataka koju korisnik vidi jeste relacioni upitni jezik. To je sredstvo kojim korisnik ostvaruje komunikaciju sa relacionom bazom podataka, kojim izražava i zadovoljava sve zahteve vezane za podatke u bazi. Zato se, sa korisničke tačke gledišta, upitni jezik često identifikuje sa kompleksnim sistemom za upravljanje bazama podataka, koji u stvari te zahteve izvršava, korektno i efikasno. Jedan od upitnih jezik koji se u ovoj knjizi detaljno izlaže i kojim se ilustruje funkcionalnost RSUBP jeste SQL. Bez obzira na sve nedostatke, SQL je standard relacionih upitnih jezika, i daleko je najzastupljeniji u relacionim sistemima; zato je njegovo detaljno poznavanje neophodno za efikasno korišćenje tih sistema. Kako se varijante SQL-a u različitim sistemima razlikuju, ovde se, zbog preciznosti, izlaže jedna od tih varijanti – SQL sistema IBM DB2 UDB (Universal DataBase) Verzija 5, bez specifičnosti tesno vezanih za ovaj RSUBP.

Sredstvo kojim sistem za upravljanje bazama podataka omogućuje većem broju korisnika istovremeni pristup podacima, uz punu bezbednost i korektnost podataka, jeste transakcija. Transakcija je logička jedinica posla i može da se sastoji od jedne ili većeg broja radnji nad bazom podataka, pri čemu obezbeđuje uslov da podaci i odnosi među njima, posle završetka transakcije, korektno odražavaju realnost koja

se tim podacima modelira. Upravljanje transakcijama omogućuje da se svi zahtevi korisnika korektno izvršavaju, a realizuje se posebnim komponentama koje koristi sistem za upravljanje bazama podataka.

Funkcija SUBP kojom se regulišu prava pristupa pojedinih korisnika pojedinim objektima (podacima i drugim resursima), kao i prava izvršenja raznih operacija nad tim objektima, zove se autorizacija. Ona se delom realizuje mehanizmom pogleda, ali i specifičnim podsistemom autorizacije i prava korisnika. Načini realizacije autorizacije predstavljeni su u ovoj knjizi.

Poseban značaj u radu sa relacionim bazama podataka ima struktuiranje i organizacija samih podataka. Podatke je na nivou korisnika potrebno struktuirati, a na fizičkom nivou organizovati tako da njihovo održavanje bude najlakše, a operisanje njima najefikasnije. Skup postupaka kojima se dolazi do dobro struktuiranih podataka u bazi podataka naziva se metodama logičkog projektovanja baze podataka. Skup postupaka kojima se podaci fizički organizuju u bazi, tako da im je pristup i održavanje najefikasnije, naziva se metodama fizičkog projektovanja, tj. metodama fizičke organizacije podataka. Zato metode logičkog projektovanja i fizičke organizacije imaju posebno mesto u ovoj knjizi.

Značajan segment relacione tehnologije predstavljaju distribuirane baze podataka. Mada su sistemi za upravljanje distribuiranim bazama podataka takoďe relacioni, problemi koji se u njima moraju rešavati, kao i metode za njihovo rešavanje, znatno su složeniji nego u centralizovanim sistemima. U ovoj knjizi razmatraju se posebno karakteristični problemi u realizaciji pune funkcionalnosti distribuiranih sistema. Takoďe se izlaže i specijalni slučaj arhitekture distribuiranih sistema – široko rasprostranjena klijent-server arhitektura.

Pored relacione tehnologije kojoj je ovaj udžbenik posvećen, izuzetno aktuelnu temu u oblasti baza podataka predstavljaju objektno orijentisane baze podataka. Objektna tehnologija nema tendenciju potiskivanja starije – relacione tehnologije. Umesto toga, integracija koncepata relacione i objektne tehnologije vidi se kao osnov za izgradnju modelâ i sistemâ baza podataka nove generacije. Zato je konceptima objektno orijentisanih modela i nekim od ključnih komponenti objektno orijentisanih sistema posvećen dodatak ovog udžbenika.

Deo I

Relaciona tehnologija

Deo **I** sastoji se od dve glave:

- Glava 1, **Relacioni model podataka**, prikazuje relacioni model podataka i određuje njegovo mesto u arhitekturi sistema baza podataka. Relacioni model se definiše preko tri bitne komponente modela; to su strukturni, manipulativni i integritetni deo modela. Strukturni i integritetni deo relacionog modela izlažu se detaljno u ovoj glavi, dok je manipulativni deo samo naznačen. Zbog značaja i kompleksnosti manipulativnih formalizama relacionog modela, njihovo razmatranje izdvojeno je u posebnu, drugu glavu.

- U glavi 2, **Manipulativni formalizmi relacionog modela**, izlažu se dva osnovna formalizma za manipulisanje podacima u relacionom modelu, relaciona algebra i relacioni račun. S obzirom na značajna odstupanja postojećih relacionih implementacija od osnovnih postavki relacionog modela, pre svega u upitnim jezicima, aktuelizovano je istraživanje implikacijâ manipulativnih formalizama relacionog modela na razvoj novih sistema.

1

Relacioni model podataka

1.1 O modelima podataka

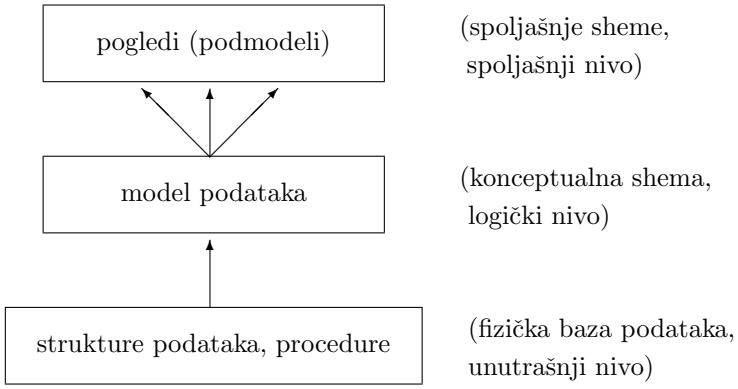
Arhitektura najvećeg broja sistema baza podataka odgovara predlogu ANSI/SPARC studijske grupe Američkog nacionalnog instituta za standarde, i poznata je kao ANSI arhitektura ([6]). Ova arhitektura predstavljena je hijerarhijom apstrakcija, pri čemu svaki nivo hijerarhije uključuje specifični način predstavljanja, *reprezentaciju*, objekata, odnosa među objektima i operacija nad objektima. Hijerarhijska arhitektura omogućuje prirodnu dekompoziciju i efikasni razvoj sistema za upravljanje bazama podataka.

Najniži nivo ANSI arhitekture je *unutrašnji* nivo. On je najbliži fizičkoj reprezentaciji baze podataka, koja u računarskom sistemu jedina zaista postoji. Zbog toga se unutrašnji nivo često i zove "nivo fizičke baze podataka". Sledeći nivo ANSI arhitekture je *konceptualni* (logički) i predstavlja način na koji se podaci iz fizičke baze podataka predstavljaju korisniku u opštem slučaju. Najviši nivo ANSI arhitekture je *spoljašnji* nivo koji predstavu o podacima iz baze prilagođava potrebama specifičnih korisnika ili grupa korisnika.

Globalna ANSI arhitektura sistema baza podataka može se predstaviti shemom na slici 1.1. Objasnimo nešto detaljnije njenu strukturu.

Reprezentacija koja se nalazi na "srednjem", konceptualnom nivou ANSI hijerarhije zove se *model podataka*. Modelom podataka predstavlja se logička struktura svih podataka u bazi i skup operacija koje korisnik može izvršiti nad tim podacima. To znači da se na konceptualnom nivou mogu "videti" svi podaci iz fizičke baze podataka, samo što je njihova reprezentacija pogodnija za korisnika od fizičke (na višem je nivou apstrakcije).

Pojedini korisnici ili grupe korisnika mogu imati svoja sopstvena specifična gledanja na model podataka (npr. iz razloga zaštite ili udobnosti), pa se *pogledi* (podmodeli, spoljašnji nivo hijerarhije) nalaze iznad modela u hijerarhiji apstrakcija. Isti podaci iz fizičke baze podataka (i sa konceptualnog nivoa), na ovom nivou



Slika 1.1: ANSI arhitektura sistema baza podataka

mogu se raznim korisnicima predstaviti na razne načine, dok se postojanje nekih podataka može od nekih korisnika i sakriti. Zato postoji tačno jedan model podataka u sistemu, koji se odnosi na celokupnost baze podataka, i veći broj spoljašnjih pogleda, od kojih se svaki sastoji od apstraktne slike dela baze podataka.

Na primer, baza podataka o studentima može da sadrži podatke iz dosijea studenata, sa ličnim podacima i podacima o uspehu (predmetima, ocenama, datumima polaganja, nagradama, kaznama) svakog studenta. Na konceptualnom nivou podaci mogu biti predstavljeni tabelama DOSIJE, NASTAVNI_PLAN, NAGRADE_I_KAZNE i POLAGANJE. Na spoljašnjem nivou, korisnicima statističkih obrada ova baza podataka može se predstaviti kao da sadrži samo podatke o uspehu (npr. u virtuelnoj tabeli USPEH_PO_PREDMETIMA); ostali, lični podaci koji mogu da identifikuju studenta, skrivaju se od statističkih aplikacija. Različitim podgrupama korisnika mogu odgovarati još apstraktnije predstave o podacima, koje odgovaraju višim nivoima podmodela.

Korisnik svoje zahteve za operacijama nad bazom podataka postavlja na konceptualnom ili spoljašnjem nivou, i to interaktivno, na posebnom "jeziku podataka", ili programski, kada je jezik podataka ugnježden u neki programski jezik opšte namene (tzv. matični jezik, v. odeljak 3.4).

Na nivou fizičke baze podataka – unutrašnjem nivou, model podataka predstavlja se specifičnim, fizičkim strukturama podataka (datotekama sa sekvencijalnim, indeksnim ili direktnim pristupom), i procedurama za fizičku realizaciju operacija koje korisnik zadaje na višem nivou.

Primarni cilj modela podataka u kontekstu baza podataka jeste da obezbedi formalni sistem za predstavljanje podataka i manipulisanje podacima u bazi podataka.

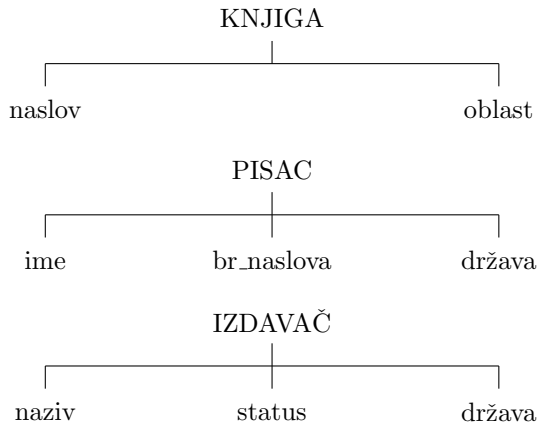
Model podataka se može posmatrati kao kolekcija sledeća tri skupa:

a) skup tipova objekata
b) skup operacija nad objektima definisanih tipova
c) skup pravila integriteta.

Opišimo ukratko svaki od ovih skupova.

a) Skup tipova objekata čini *strukturni deo modela*. Tipovima objekata opisuju se entiteti (objekti, stvari, lica, pojave, događaji od značaja koji se mogu jednoznačno identifikovati) koji su relevantni za korisnike baze (objekat i entitet će se nadalje uglavnom upotrebljavati sinonimno, osim ako se naglasi drugačije). Sami tipovi entiteta karakterišu se svojstvima.

   Na primer, izdavačka baza podataka može da sadrži, između ostalih, tipove entiteta KNJIGA, PISAC i IZDAVAČ, sa odgovarajućim svojstvima (slika 1.2). Značenje svojstava je uglavnom očigledno: svaka knjiga ima naslov i oblast kojoj pripada, uz pisca se kao značajna svojstva navode njegovo ime, broj objavljenih naslova i država čiji je državljanin, dok izdavača karakterišu njegov naziv, status (u funkciji broja izdatih naslova) i država u kojoj je registrovan. Specifični model (npr. relacioni, mrežni, hijerarhijski) izabraće specifičan način za opisivanje ovih tipova entiteta i njihovih svojstava.



Slika 1.2: Tipovi entiteta izdavačke baze podataka

b) Skup operacija čini *manipulativni deo modela*. Operacijama se zadaju *upiti* i *radnje* nad entitetima, tj. pretraživanje i menjanje (ažuriranje) podataka o entitetima. Na primer, radnje nad pojedinačnim entitetima mogu biti:
   – registrovati novu knjigu
   – rashodovati knjigu

– izmeniti državljanstvo pisca

– izmeniti status izdavača.

Primer operacije kojom se zadaje upit nad pojedinačnim tipom entiteta može biti:

– štampati spisak knjiga iz zadate oblasti.

c) Skup pravila integriteta čini *integritetni deo modela*. Pravila integriteta definišu uslove integriteta podataka u bazi podataka, tj. svojstva podataka ili odnosa među podacima, koji moraju biti zadovoljeni da bi stanje baze bilo valjano.

Na primer, uslovi integriteta pridruženi pojedinačnim tipovima entiteta mogu biti:

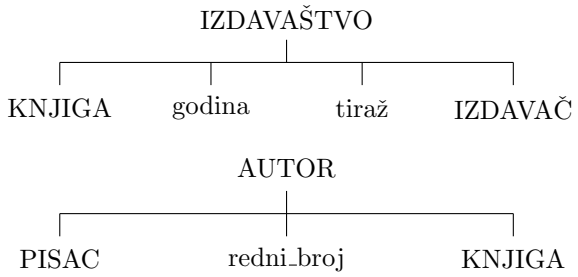
– dva razna izdavača ne mogu imati isti naziv

– pri promeni broja naslova pisca, novi broj naslova mora biti veći od prethodnog (starog).

Osim upita, radnji i uslova integriteta koji se odnose na pojedinačne tipove entiteta, kao u prethodnim primerima, neki upiti i radnje mogu se odnositi na više tipova entiteta, kao na primer:

– štampati spisak svih knjiga datog pisca
– promeniti izdavača za datu knjigu.

I upiti i radnje ove vrste (nad više tipova entiteta) zadaju se nad *odnosima* među tipovima entiteta. Na primer, IZDAVAŠTVO se može definisati kao odnos između tipova entiteta IZDAVAČ i KNJIGA, dok se AUTOR može definisati kao odnos između tipova entiteta PISAC i KNJIGA. Ti odnosi se mogu posmatrati i kao *apstraktni tipovi entiteta* u modelu (sa svojim svojstvima). Svojstva odnosa IZDAVAŠTVO i AUTOR mogu se prikazati slikom 1.3. Redni_broj mogao bi da označava da je dati pisac naveden kao prvi (drugi, treći, itd.) autor date knjige.



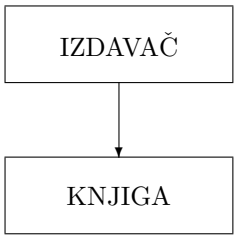
Slika 1.3: Tipovi odnosa izdavačke baze podataka

Nad odnosima se mogu zadavati i uslovi integriteta, na primer:

– dva razna izdavača ne mogu u istoj godini izdati istu knjigu.

Najpoznatiji klasični modeli podataka su *hijerarhijski*, *mrežni* i *relacioni* model. Za hijerarhijski i mrežni model karakteristično je da su nastali u procesu apstrakcije ili indukcije iz postojećih implementacija SUBP, a ne kao apstraktni modeli. Sami mrežni i hijerarhijski sistemi su prerelacioni i karakterišu se nižim nivoom apstrakcije nego relacioni sistemi, pre svega slogovnom prirodom podataka i manipulisanja podacima. Detaljni opis hijerarhijskog i mrežnog modela kao i elementi njihovih najpoznatijih implementacija, IMS i IDMS redom, mogu se naći u [22], i u ovoj knjizi neće biti razmatrani. Umesto toga, ilustrujmo sledećim primerom način na koji se u hijerarhijskom, mrežnom i relacionom sistemu realizuju delovi odgovarajućih modela.

Neka među entitetima KNJIGA i IZDAVAČ postoji funkcionalni odnos, tj. neka važi da jednu knjigu može da izdaje samo jedan izdavač, dok jedan izdavač može da izdaje veći broj knjiga. Strukturni deo hijerarhijskog modela u ovom slučaju sastoji se od dva tipa entiteta – KNJIGA i IZDAVAČ sa pripadnim svojstvima, i jednog tipa funkcionalne veze – KNJIGA → IZDAVAČ (slika 1.4). U ovoj funkcionalnoj vezi tip entiteta IZDAVAČ je tzv. roditeljski tip, a tip entiteta KNJIGA je tzv. tip deteta. Jedan tip deteta može imati najviše jedan roditeljski tip.



Slika 1.4: Funkcionalna veza u hijerarhijskom modelu

U hijerarhijskom SUBP IMS definicija baze uključuje naziv baze i, za svaki tip entiteta, definiciju tog tipa (u IMS terminologiji – segment, SEGM). Definicija tipa entiteta uključuje ime entiteta, dužinu odgovarajućeg segmenta u bajtovima, ime roditeljskog tipa entiteta (PARENT) i imena atributa (u IMS terminologiji – polja, FIELD), a za svako polje – dužinu u bajtovima, adresu prvog bajta polja u okviru segmenta, i eventualnu uređenost segmenata po tom polju (engl. sequencing, SEQ). Deo definicije hijerarhijske baze u našem slučaju mogao bi da ima sledeći oblik (polja I_SIF, K_SIF predstavljaju šifru izdavača odnosno šifru knjige):

```
1  DBD    NAME = IZDAVASTVO

2  SEGM   NAME = IZDAVAC,        BYTES=44
3  FIELD  NAME = (I_SIF, SEQ),   BYTES=2,  START=1
```

```
4  FIELD  NAME = NAZIV,          BYTES=20, START=3
5  FIELD  NAME = STATUS,         BYTES=2,  START=23
6  FIELD  NAME = DRZAVA,         BYTES=20, START=25

7  SEGM   NAME = KNJIGA, PARENT = IZDAVAC, BYTES=52
8  FIELD  NAME = (K_SIF, SEQ),   BYTES=2,  START=1
9  FIELD  NAME = NASLOV,         BYTES=40, START=3
10 FIELD  NAME = OBLAST,         BYTES=10, START=43
```

Ova definicija hijerarhijske baze uključuje i integritetni deo u smislu da u bazi ne može postojati informacija o knjizi ako ne postoji informacija o njenom izdavaču, jer se entitet tipa KNJIGA definiše u okviru "roditeljskog" entiteta tipa IZDAVAČ (uslov koji je u relacionom modelu poznat kao referencijalni integritet).

Manipulisanje podacima u hijerarhijskom modelu je u znatnoj meri proceduralno i podrazumeva precizno navoďenje hijerarhijskog puta ka traženom segmentu. Tako bi, na primer, zahtev za nalaženjem prve knjige – romana izdavača 'Prosveta' morao da sadrži i precizno uputstvo kako se do odgovarajućeg segmenta o toj knjizi dolazi, i pojednostavljenom sintaksom jezika za manipulisanje podacima sistema IMS izrazio bi se u obliku

```
GET UNIQUE  IZDAVAC WHERE NAZIV  = 'Prosveta'
            KNJIGA  WHERE OBLAST = 'roman'
```

(redosled navoďenja entiteta je veoma bitan i mora da odgovara redosledu roditelj – dete iz definisane funkcionalne veze).

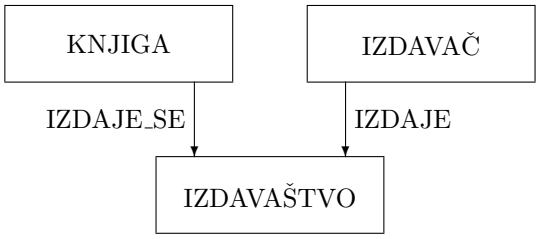
Ovakav iskaz ne može se zadavati interaktivno, već isključivo iz programa na programskom jeziku opšte namene (npr. COBOL, PL/I).

U slučaju da meďu tipovima entiteta IZDAVAČ i KNJIGA ne postoji funkcionalna veza, hijerarhijski model pokazuje svoje slabosti (jer ne može da definiše više od jednog tipa roditelja za jedan tip entiteta). Mrežni model dopušta da jedan tip deteta može imati veći broj roditeljskih tipova entiteta u okviru različitih (imenovanih) funkcionalnih veza. Svaki tip entiteta – domen funkcionalne veze zove se tip člana te funkcionalne veze, a kodomen funkcionalne veze – tip vlasnika te funkcionalne veze. Tako se naš primer u mrežnom modelu može izraziti pomoću tri tipa entiteta – IZDAVAČ, KNJIGA i "veznog" tipa entiteta IZDAVAŠTVO, i pomoću dve imenovane funkcionalne veze:

IZDAJE_SE: IZDAVAŠTVO → KNJIGA
IZDAJE: IZDAVAŠTVO → IZDAVAČ (slika 1.5).

U mrežnom SUBP IDMS definicija baze uključuje naziv baze, definiciju tipa entiteta za svaki tip entiteta (u IDMS terminologiji – slog, RECORD) i definiciju imenovanog tipa veze za svaki tip veze (u terminologiji IDMS – skup, SET).

Manipulisanje podacima u mrežnim sistemima, slično hijerarhijskim, proceduralno je orijentisano i izvršava se paketno – operacijama umetnutim u programski



Slika 1.5: Imenovane funkcionalne veze – skupovi u mrežnom modelu

jezik opšte namene, npr. COBOL. Pri tome sintaksa operacija omogućuje precizno navoďenje puta kojim se pristupa pojedinom slogu, uključujući navoďenje skupa, vlasnika skupa, prvog, sledećeg, poslednjeg člana skupa za datog vlasnika, itd. Tako bi se zahtev za nalaženjem prve knjige – romana izdavača 'Prosveta' izrazio proceduralno, prvo nalaženjem prvog člana (IZDAVAŠTVO) skupa IZDAJE u kome je izdavač 'Prosveta' vlasnik, zatim pronalaženjem vlasnika (KNJIGA) skupa IZDAJE_SE čiji je član – slog naďen u prvom koraku i, najzad, ponavljanjem prethodna dva koraka sve dok se ne doďe do primerka sloga KNJIGA koji je roman. Izražen pojednostavljenom sintaksom, ovaj zahtev ima sledeći oblik:

```
MOVE 'Prosveta' TO NAZIV IN IZDAVAC
FIND IZDAVAC
FIND FIRST IZDAVASTVO WITHIN IZDAJE
PERFORM
     FIND OWNER WITHIN IZDAJE_SE
     GET KNJIGA
     IF OBLAST IN KNJIGA = 'roman'
          izdati  NASLOV tekućeg sloga  KNJIGA;
          izaći iz PERFORM petlje
     END-IF
     FIND NEXT IZDAVASTVO WITHIN IZDAJE
END-PERFORM
```

Hijerarhijski i mrežni model ne ostvaruju nezavisnost nivoa predstavljanja podataka ANSI arhitekture. Na logičkom nivou prisutna je velika količina informacija o fizičkoj reprezentaciji i grupisanju podataka. Nasuprot njima, relacioni model podataka na logičkom nivou predstavlja i entitete i odnose meďu njima na jedinstven način – relacijama tabelarnog izgleda. Jezik za definisanje podataka i manipulisanje podacima je neproceduralan i ne uključuje informaciju o putu kojim se fizički pristupa podacima. Tako bi se entiteti i odnosi iz prethodnog primera u relacionom modelu prikazali pomoću tri tabele:

```
KNJIGA      (K_SIF, NASLOV, OBLAST)
IZDAVAC     (I_SIF, NAZIV,  STATUS, DRZAVA)
```

```
IZDAVASTVO  (I_SIF, K_SIF, GODINA, TIRAZ)
```

a zahtev za nalaženjem svih romana izdavača 'Prosveta' izrazio bi se relacionim jezikom, na primer SQL-om, na sledeći način:

```
SELECT NASLOV
FROM KNJIGA, IZDAVAC, IZDAVASTVO
WHERE  OBLAST = 'roman' AND KNJIGA.K_SIF = IZDAVASTVO.K_SIF AND
       NAZIV = 'Prosveta' AND IZDAVAC.I_SIF = IZDAVASTVO.I_SIF
```

Redosled navoďenja relacija u FROM liniji je nebitan jer sve relacije imaju ravnopravan tretman; isto važi i za redosled poreďenja u WHERE liniji. Prethodni relacioni iskaz može se izvršavati i interaktivno i u okviru aplikativnog programa.

Za izlaganje najznačajnijih aspekata sistema baza podataka koristiće se relacioni model. Razlozi za ovaj izbor mogu se naslutiti iz prethodnog primera, a u potpunosti se mogu sagledati tek detaljnim upoznavanjem svih modela (v. [22]). Ti razlozi su mnogobrojni i prirodni, i nalaze se u svim bitnim karakteristikama relacionog modela. Najvažnije meďu njima su:

- strukturna jednostavnost
- formalno i strogo zasnivanje
- ekonomični upitni jezici
- razgraničenje nivoa predstavljanja podataka.

Relacioni model će biti izložen u narednim odeljcima u verziji koja je poslužila kao osnov za kasnija proširenja (RM/T, RM/2), i koju je autor relacionog modela Edgar Codd (Edgar Kôd) prikazao u [18], [19].

1.2 Strukturni deo relacionog modela

Dva osnovna tipa objekata koji karakterišu strukturu relacionog modela su *domen* i *relacija*. Domen je skup vrednosti istog tipa, kao što je skup naslova knjiga, skup imena gradova ili skup datuma nekih dogaďaja. Domen je *jednostavan* ako su mu sve vrednosti atomične, tj. ako SUBP ne može da ih razloži u komponente sa specifičnim značenjem; u suprotnom, domen je *kompozitan*. U daljem tekstu će se podrazumevati da su svi domeni jednostavni (osim ako se naglasi drugačije).

Neka su $D_1, D_2, \ldots, D_n$ ($n \in \mathbf{N}$) domeni (ne obavezno različiti). *Dekartov proizvod* tih domena, u oznaci $D_1 \times D_2 \times \cdots \times D_n$, jeste skup n-torki $(v_1, v_2, \ldots, v_n)$ takvih da je $v_i \in D_i$ za sve $i = 1, 2, \ldots, n$. *Relacija R stepena n*, definisana na domenima $D_1, D_2, \ldots, D_n$, je proizvoljni konačni podskup navedenog Dekartovog proizvoda. Ova relacija, za razliku od matematičkog pojma relacije, je dinamičkog

sadržaja, jer se neke njene n-torke[1] tokom "života" relacije brišu, neke se dodaju a neke se menjaju.

Ako se umesto skupa indeksa domenâ $\{1, 2, \ldots, n\}$ uzme proizvoljni skup (različitih) imenovanih indeksa $\{A_1, A_2, \ldots, A_n\}$, onda je svaki element v_i n-torke relacije R karakterisan ne samo domenom D_i već i indeksom A_i. Različiti imenovani indeksi $A_i\,(i = 1, 2, \ldots, n)$ zovu se *atributi* relacije R i označavaju različite uloge (ne nužno različitih) domena nad kojima je izgrađena relacija R. Sada se svaka n-torka relacije R može posmatrati kao skup n parova $(A_i, v_i), i = 1, 2, \ldots, n$, gde je v_i vrednost iz domena D_i.

Relacijom R predstavlja se neki tip entiteta E, dok se n-torkama relacije R predstavljaju pojedinačni entiteti tipa E. Zato se svaki atribut A_i relacije R može interpretirati kao preslikavanje tipa entiteta E u odgovarajući domen D_i, tj.

$$A_i : \; E \rightarrow D_i.$$

Slično, svaki podskup $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ skupa atributa relacije R može se interpretirati kao preslikavanje

$$(A_{i_1}, A_{i_2}, \ldots, A_{i_k}) : \; E \rightarrow D_{i_1} \times D_{i_2} \cdots \times D_{i_k}.$$

Mada je skup D_i kodomen preslikavanja (tj. atributa) A_i, u terminologiji relacionog modela on se zove *domen atributa* A_i, što se označava sa $dom(A_i)$. Slično, sa $dom(A_{i_1}, A_{i_2}, \ldots, A_{i_k})$ označava se domen skupa atributa $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$, tj. skup $D_{i_1} \times D_{i_2} \cdots \times D_{i_k}$.

Neka je e entitet tipa E. Tada je element $v_i \in D_i$ *vrednost atributa* A_i entiteta e (tj. odgovarajuće n-torke) ako je $A_i(e) = v_i$. Slično, element $(v_{i_1}, v_{i_2}, \ldots, v_{i_k}) \in D_{i_1} \times D_{i_2} \cdots \times D_{i_k}$ je *vrednost skupa atributa* $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ entiteta e (tj. odgovarajuće n-torke) ako je $(A_{i_1}, A_{i_2}, \ldots, A_{i_k})(e) = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$.

Ako su svi domeni atributa relacije R jednostavni, onda relacija R ima tabelarni izgled i nalazi se u *prvoj normalnoj formi*, u oznaci 1NF, tj. *normalizovana* je (o višim normalnim formama biće reči u delu o logičkom projektovanju baze podataka). Kolone tabele odgovaraju atributima relacije, a vrste – n-torkama relacije.[2]

Takva tabela ima sledeće karakteristike:

- nema dupliranih vrsta (jer je relacija skup)
- redosled vrsta je nebitan (jer je relacija skup)
- redosled kolona je nebitan (jer atributi relacije čine skup)
- sve vrednosti u tabeli su atomične.

[1] Kada je iz konteksta jasno kojoj relaciji pripada neka n-torka, ili kada to nije od značaja, n-torka će se jednostavno zvati *torka*; pri tome se za dimenziju torke uzima stepen odgovarajuće relacije.

[2] U daljem tekstu parovi termina relacija/tabela, atribut/kolona i n-torka/vrsta upotrebljavaće se kao sinonimi; uobičajeno je da se u razmatranju samog relacionog modela upotrebljavaju prve komponente ovih parova, a u kontekstu implementacija relacionog modela – druge.

Oznaka $R(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$ koristi se za predstavljanje relacije R čiji atribut A_1 uzima vrednost iz domena D_1, atribut A_2 iz domena D_2, itd. i zove se *relacijska shema* relacije R. Kada je jasno o kojim se domenima radi, ili kada domeni (odnosno atributi) nisu bitni za izlaganje, za relaciju R upotrebljava se oznaka $R(A_1, A_2, \ldots, A_n)$ (odnosno samo R). Umesto oznake za nabrajanje atributa $R(A_1, A_2, \ldots, A_n)$, često se, u identičnom značenju, koristi i oznaka sa dopisivanjem atributa, $R(A_1A_2 \ldots A_n)$. Takoďe, ako su od značaja dva istaknuta disjunktna podskupa X, Y skupa atributa relacije R (čija je unija jednaka celom skupu), ona se obeležava i sa $R(X, Y)$.

Relaciona baza podataka je skup skupova podataka koji se na logičkom nivou mogu posmatrati kao normalizovane relacije odgovarajućih stepena i dinamičkog sadržaja. Skup relacijskih shema relacione baze podataka je *shema relacione baze podataka*. *Bazne relacije* su relacije koje su definisane nezavisno od drugih relacija u bazi podataka, i koje se ne mogu izvesti iz drugih baznih relacija. One su na fizičkom nivou predstavljene odgovarajućim strukturama podataka. *Izvedene relacije* su relacije koje se mogu izvesti iz baznih relacija primenom skupa operacija. One obično nemaju fizičku reprezentaciju, i predstavljaju različite *poglede* koje razni korisnici mogu imati na istu bazu podataka.

Neke druge definicije relacionog modela ne uključuju pojam domena u osnovne tipove objekata strukture relacionog modela. Meďutim, ovaj pojam ima svoj operacioni značaj u tome što domeni odreďuju skup operacija koje se nad njihovim vrednostima mogu izvršavati. Tako se vrednosti atributa nad istim domenima mogu porediti (a vrednosti atributa nad različitim domenima ne mogu), pa se nad atributima, definisanim nad istim domenima, mogu izvršavati i sve operacije koje uključuju poreďenje. Većina upitnih jezika, kao što je SQL, ne poznaje semantički aspekt domena, pa je moguće, na primer, postavljati upite kojima se porede naslovi knjiga sa imenima država (kao niske simbola).

Sva naredna razmatranja biće ilustrovana jednom aproksimacijom izdavačke relacione baze sa sledećom shemom:

```
P  (P_SIF,  IME,  BR_NASLOVA,  DRZAVA)
I  (I_SIF,  NAZIV,  STATUS,  DRZAVA)
K  (K_SIF,  NASLOV,  OBLAST)
KP (K_SIF,  P_SIF,  R_BROJ)
KI (K_SIF,  I_SIF,  IZDANJE,  GODINA,  TIRAZ)
```

Prve tri relacije predstavljaju tipove entiteta PISAC, IZDAVAČ i KNJIGA redom, i imaju sličnu strukturu kao i odgovarajući tipovi entiteta. Nov je prvi atribut u svakoj od relacija, koji predstavlja šifru, tj. jedinstveni identifikator pisca, izdavača i knjige. Na primer, šifra knjige, K_SIF, može biti ISBN (International Standard Book Number), P_SIF i I_SIF mogu biti matični broj pisca i izdavača, redom. Relacija KP predstavlja apstraktni tip entiteta AUTOR, tj. odnos izmeďu tipova entiteta KNJIGA i PISAC, dok relacija KI predstavlja apstraktni tip entiteta IZDAVAŠTVO, tj. odnos izmeďu tipova entiteta KNJIGA i IZDAVAČ.

Neka je sadržaj[3] tih relacija (koji će se podrazumevati u svim primerima) sledeći:

P

P_SIF	IME	BR_NASLOVA	DRZAVA
p1	B.Ćopić	2	Jugoslavija
p2	M.Benson	1	Engleska
p3	B.Šljivić-Šimšić	1	Jugoslavija
p4	D.Maksimović	2	Jugoslavija
p5	C.J.Date	1	Amerika

I

I_SIF	NAZIV	STATUS	DRZAVA
i1	Prosveta	30	Jugoslavija
i2	Addison Wesley Publ. Comp.	20	Amerika
i3	Dečje novine	10	Jugoslavija
i4	Matica srpska	30	Jugoslavija

K

K_SIF	NASLOV	OBLAST
k1	Osma ofanziva	roman
k2	Nemam više vremena	poezija
k3	Pionirska trilogija	roman
k4	Srpskohrvatsko-engleski rečnik	leksikografija
k5	An Introduction to Database Systems	računarstvo
k6	Tražim pomilovanje	poezija

KP

K_SIF	P_SIF	R_BROJ
k1	p1	1
k2	p4	1
k3	p1	1
k4	p2	1
k4	p3	2
k5	p5	1
k6	p4	1

KI

K_SIF	I_SIF	IZDANJE	GODINA	TIRAZ
k1	i1	2	1965	10000
k2	i1	2	1974	7000
k3	i1	1	1975	10000
k4	i1	2	1979	10000
k5	i2	4	1986	5000
k6	i3	1	1966	3000
k6	i4	3	1988	5000

[3]Stvarni sadržaj uključio bi samo simbole iz dopuštenog skupa.

1.3 Manipulativni deo relacionog modela

Manipulativni deo relacionog modela predstavlja formalizam za definisanje *relacionog izraza* opšteg oblika, čija je vrednost proizvoljni podskup skupa podataka iz baze.

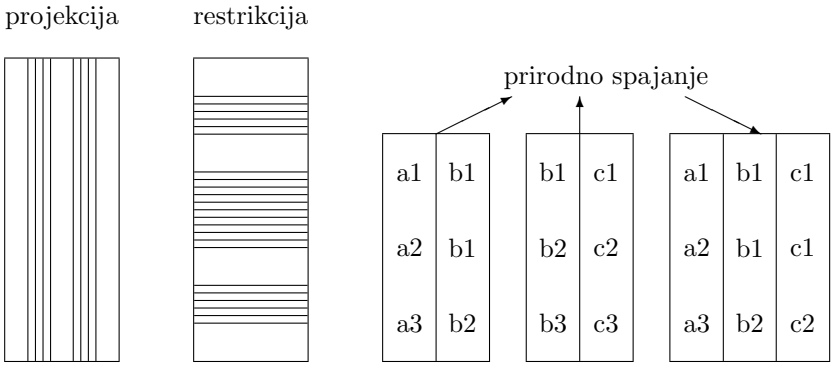
Definicija relacionog modela uključuje dva formalna jezika za rad sa relacijama, koja imaju traženo svojstvo: jedan algebarski u suštini – *relaciona algebra*, i drugi, zasnovan na predikatskom računu prvog reda – *relacioni račun* ([14]). Dokazana je ekvivalentnost ovih formalizama u pogledu izražajne moći ([17]). Sami formalizmi kao i skica dokaza njihove ekvivalentnosti biće izloženi u sledećoj glavi. Da bismo ilustrovali mogućnosti ovih formalizama, ovde ćemo samo neformalno uvesti pojam relacione algebre.

Relaciona algebra je skup operacija nad relacijama. Relacioni izraz u relacionoj algebri sastoji se od niza operacija nad odgovarajućim relacijama. Među važnijim operacijama relacione algebre su projekcija, restrikcija i prirodno spajanje (slika 1.6, [27]):

- neka je X podskup skupa atributa relacije R; projekcija $R[X]$ relacije R na atribute X je relacija koja sadrži (samo) kolone relacije R koje odgovaraju atributima X (v. i tačku 2.1.1);
- restrikcija $R[X = x]$, za $x \in dom(X)$, jeste relacija koja sadrži (samo) n-torke relacije R čija je vrednost skupa atributa X jednaka x (v. i tačku 2.1.1);
- prirodno spajanje $R * S$ je dopisivanje torki relacije S na torke relacije R ako imaju iste vrednosti na zajedničkim atributima, uz eliminaciju po jednog atributa iz svakog para zajedničkih atributa (v. i tačku 2.1.2).

Mada se operacije relacione algebre definišu nad relacijama, one se primenjuju i na pojedinačne n-torke, jer su i n-torke – (jednočlane) relacije; na primer, projekcija n-torke $(v_1, v_2, \ldots, v_n)$ na prva dva atributa (A_1, A_2) je $(v_1, v_2, \ldots, v_n)[A_1A_2] = (v_1, v_2)$ (v. i tačke 2.1.1, 2.1.2).

Specifični sistemi relacionih baza podataka (kraće – relacioni sistemi) implementiraju manipulativni deo relacionog modela *relacionim jezicima* za definisanje podataka i manipulisanje podacima, specifične sintakse. Da bi jezik za definisanje podataka bio relacioni, mora da ima konstrukcije za definisanje osnovnih koncepata strukture relacionog modela (ali i pravila integriteta, opštih i eventualno posebnih, v. odeljak 1.4). Da bi jezik za manipulisanje podacima bio relacioni, mora da bude *relaciono kompletan*, tj. da ima izražajnu moć bar kolika je izražajna moć relacionog računa ili relacione algebre. Osim toga, jezik za manipulisanje podacima jednog relacionog sistema mora da ima još neka svojstva. Naime, formalizam relacionog modela ima mogućnost *pretraživanja* podataka, tj. pronalaženja podataka koji predstavljaju vrednost relacionog izraza. Pored ove mogućnosti, relacioni jezik treba da omogući i *ažuriranje* podataka u najširem smislu, tj. unošenje novih i brisanje i izmenu postojećih podataka u bazi. Zahtev za pretraživanjem zove se *upit*, dok se zahtev za ažuriranjem zove *radnja*.



Slika 1.6: Operacije projekcije, restrikcije i prirodnog spajanja

Jezik za definisanje podataka i jezik za manipulisanje podacima zajedno čine *upitni jezik*. Osnova za izgradnju upitnog jezika, s obzirom na potrebu za njegovom relacionom kompletnošću, uvek leži u jednom od pomenutih formalizama.

1.3.1 Nedostajuće vrednosti

U bazi podataka može postojati potreba za registrovanjem semantički različitih vrsta nedostajućih vrednosti, kao što su trenutno nepoznata vrednost (npr. status izdavača), ili neprimenljivo svojstvo (npr. vrednost atributa IME_SUPRUŽNIKA za osobu koja nije u braku). Semantika nedostajuće vrednosti može biti i drugačija ([46]) (npr. nepoznata vrednost koja je u poznatom intervalu), ali se prve dve navedene vrste semantički najviše razlikuju i najčešće pojavljuju u realnosti.

Da bi se relacioni model proširio ovim dvema vrstama nedostajućih vrednosti, neophodno je definisati proširene relacije poređenja nad vrednostima iz domena i različitim nedostajućim vrednostima, kao i proširene relacije pripadnosti skupu i inkluzije. Rezultat poređenja sada može biti, osim tačno ili netačno, i drugačije okvalifikovan (npr. 'ne zna se', 'možda', itd). Tako se dolazi do viševalentnih logika u relacionom modelu. Kako se definicije operacija relacione algebre baziraju na poređenju vrednosti atributa n-torki, a rezultat tog poređenja može biti tačan, netačan ili 'možda', potrebno je za svaku operaciju relacione algebre definisati više od jedne operacije u proširenom modelu. Pored toga, i ostali koncepti relacionog modela koji su bazirani na poređenju vrednosti (npr. funkcionalne, višeznačne zavisnosti) dobijaju kompleksniju formu, što dosta usložnjava sam model. Stoga postojeći sistemi za upravljanje bazama podataka, ako uopšte podržavaju nedostajuće vrednosti, podržavaju samo jednu vrstu takve vrednosti. Jedno proširenje relacione algebre kao osnove za upitne jezike koji manipulišu i nedostajućim vrednostima dao je Codd u radu [18]; ovo proširenje biće izloženo u tački 2.1.3 – Proširena relaciona algebra.

1.4 Integritetni deo relacionog modela

U integritetni deo relacionog modela spadaju uslovi koje treba da zadovolje podaci u bazi da bi stanje baze bilo valjano. Pri svakom ažuriranju baze podataka sistem za upravljanje bazom podataka proverava eksplicitno ili implicitno zadate uslove integriteta. Svaki uslov integriteta je neko *tvrđenje* (engl. assertion), tj. istinitosna formula čija vrednost treba da bude 'tačno' u svakom valjanom stanju baze. Uslov integriteta može da se odnosi na pojedinačni atribut, domen ili relaciju, kao i na celu bazu podataka.

Uslovi integriteta mogu biti *specifični* (odnose se na zadati atribut, relaciju, bazu), pri čemu se zadaju eksplicitno, ili *opšti*, koji se mogu primeniti na svaku relaciju ili bazu podataka, i koji su zato implicitni.

Specifični uslovi integriteta mogu biti dvojaki. Prva vrsta tih uslova odnosi se na karakteristike pojedinačnih atributa ili njihovih domena, ili na vezu između specifičnih vrednosti atributâ n-torki jedne ili više relacija. Na primer,

- u relaciji K, svaka vrednost iz domena atributa K_SIF je oblika slova k za kojim sledi jednocifren do trocifren ceo broj (karakteristika pojedinačnog domena);
- status jugoslovenskih izdavača mora biti manji ili jednak 30 (veza između specifičnih vrednosti atributâ n-torki jedne relacije);
- tiraž izdanja izdavača 'Prosveta' ne može biti manji od 5000 (veza između specifičnih vrednosti atributâ n-torki dve relacije).

Druga vrsta specifičnih uslova integriteta su logičke veze koje postoje između atributâ jedne relacije. Na primer,

- u relaciji I, svaka šifra izdavača, I_SIF, jednoznačno određuje vrednost atributa NAZIV, STATUS, DRZAVA (sadržaj relacije I zadovoljava ovaj uslov);
- u relaciji I, vrednost atributa DRZAVA jednoznačno određuje vrednost atributa STATUS (sadržaj relacije I ne zadovoljava ovaj uslov, ali ako bi on bio eksplicitno zadat, svi izdavači iz iste države morali bi da imaju isti status).

Opšti uslovi integriteta (koji se mogu primeniti na svaku relaciju) biće ilustrovani kroz sledeća dva primera:

- u relaciji I atribut I_SIF mora imati definisanu i određenu vrednost (iz odgovarajućeg domena) u svakoj n-torci, tj. njegova vrednost ne sme biti nedostajuća; razne n-torke relacije I moraju imati različite vrednosti atributa I_SIF (*integritet entiteta*);
- u relaciji KI ne sme postojati n-torka čiji atribut I_SIF ima vrednost koja ne postoji na atributu I_SIF relacije I (*integritet obraćanja*, *referencijalni integritet*).

Opšti uslovi integriteta biće vezani za koncepte primarnog odnosno stranog ključa, koji će precizno biti definisani u narednim tačkama. Ovi uslovi integriteta moraju da važe uvek, u svakoj relaciji, za njen primarni odnosno strani ključ (ako relacija ima strani ključ).

Među specifičnim uslovima integriteta, uslovi druge vrste (logičke veze) mogu se formalizovati kroz koncept zavisnosti među atributima. Razne vrste zavisnosti i njihove implikacije na logičko projektovanje baze biće razmotrene u glavama 8 i 9. Ovde će biti opisan najčešće korišćeni oblik zavisnosti – funkcionalna zavisnost koja se koristi i u definiciji primarnog ključa.

Definicija 1.1 Neka su $X, Y \subseteq \{A_1, A_2, \ldots, A_n\}$ neprazni podskupovi skupa atributa relacije $R(A_1, A_2, \ldots, A_n)$. Označimo sa $R[XY]$ projekciju relacije R na uniju atributa X, Y (eventualni presečni atributi uzimaju se samo jednom). Tada, između skupova atributa X i Y postoji *funkcionalna zavisnost*, u oznaci $X \rightarrow Y$, ako u svakom trenutku postojanja relacije R, $R[XY]$ jeste funkcija $R[X] \rightarrow R[Y]$, tj. ako za svaki par n-torki $t_1, t_2 \in R$, takav da je $t_1[X] = t_2[X]$, važi da je i $t_1[Y] = t_2[Y]$. U tom slučaju kaže se da skup atributa X *funkcionalno određuje* atribute skupa Y, tj. da skup atributa Y *funkcionalno zavisi* od atributa skupa X.

Napomena 1.1 Uslov koji definiše funkcionalnu zavisnost $X \rightarrow Y$ ekvivalentan je uslovu da $R[X = x][Y]$ ima najviše jednu n-torku za proizvoljno $x \in dom(X)$ ($R[X = x][Y]$ je projekcija na atribute Y restrikcije relacije R po uslovu $X = x$).

Primeri funkcionalnih zavisnosti su:

I_SIF	$\rightarrow$	NAZIV
I_SIF	$\rightarrow$	STATUS
I_SIF	$\rightarrow$	DRZAVA
DRZAVA	$\rightarrow$	STATUS

Prva funkcionalna zavisnost čita se kao "I_SIF funkcionalno određuje NAZIV (izdavača)", ili "NAZIV (izdavača) funkcionalno zavisi od I_SIF". Analogno je i čitanje ostalih funkcionalnih zavisnosti.

1.4.1 Primarni ključ

Definicija 1.2 Neka je $X \subseteq \{A_1, A_2, \ldots, A_n\}$ podskup skupa atributa relacije $R(A_1, A_2, \ldots, A_n)$. Kaže se da je X *ključ* (ili *kandidat za ključ*) relacije R ako su ispunjena sledeća dva uslova:

a) X funkcionalno određuje sve atribute relacije R, tj. $X \rightarrow A_i$ za $i = 1, 2, \ldots, n$;

b) nijedan pravi podskup od X nema tu osobinu, tj. za svaki pravi podskup $X' \subset X$ postoji $1 \leq j \leq n$ tako da $X' \not\rightarrow A_j$.

Nadključ je svaki skup atributa za koji važi samo svojstvo a).

Na primer, atribut I_SIF je ključ relacije I.

Svaka relacija ima bar jedan ključ, jer u svakoj relaciji $R(A_1, A_2, \ldots, A_n)$ važi funkcionalna zavisnost $\{A_1, A_2, \ldots, A_n\} \rightarrow A_i$, za svako $1 \leq i \leq n$. Dakle, skup

$\{A_1, A_2, \ldots, A_n\}$ svih atributa relacije R je nadključ. Ako ne postoji pravi podskup $X \subset \{A_1, A_2, \ldots, A_n\}$ sa tim svojstvom, onda je skup $\{A_1, A_2, \ldots, A_n\}$ ključ. Ako takav pravi podskup X postoji, ključ je najmanji podskup od X sa tim svojstvom.

Jedna relacija može imati više ključeva. (Na primer, relacija I može imati i ključ NAZIV ako važi funkcionalna zavisnost NAZIV $\rightarrow$ I_SIF, što bi postojeći sadržaj relacije I zadovoljio). Jedan od ključeva bira se za *primarni ključ*, koji jedinstveno identifikuje entitete predstavljene n-torkama relacije. Primarni ključ može se sastojati od jednog ili više atributa. Svaka relacija u relacionom modelu mora imati primarni ključ.

Prvo opšte pravilo integriteta, integritet entiteta, koje implicitno mora da važi za primarni ključ svake relacije, iskazuje se, korišćenjem prethodne definicije, sledećim zahtevima:

- dve razne n-torke jedne relacije ne mogu imati identične vrednosti primarnog ključa;
- nijedan od atributa primarnog ključa relacije ne sme imati nedostajuću vrednost ni u jednoj n-torci relacije.

1.4.2 Strani ključ

Na osnovu primarnog ključa jedne relacije moguće je definisati koncept stranog ključa druge relacije.

Definicija 1.3 Skup STK jednog ili više atributa bazne relacije T_2 je njen *strani ključ* koji se odnosi na baznu relaciju T_1 ako su ispunjena sledeća dva uslova:
a) svaka vrednost skupa atributa STK n-torke iz T_2 je ili u potpunosti nedostajuća ili u potpunosti nije nedostajuća (element je odgovarajućeg Dekartovog proizvoda);
b) bazna relacija T_1 ima primarni ključ PRK takav da je svaka vrednost skupa atributa STK n-torke iz T_2, koja nije nedostajuća, identična PRK-vrednosti neke n-torke iz T_1.
Za relaciju T_2 kaže se da je *zavisna* od relacije T_1.

Na primer, atribut I_SIF u relaciji KI je strani ključ relacije KI koji se odnosi na relaciju I.

Drugo opšte pravilo integriteta, referencijalni integritet ili integritet obraćanja, definisano je upravo uslovom b) iz definicije 1.3. Ono mora implicitno da važi u svakoj relaciji za koju je definisan strani ključ, jer obezbeđuje njegove korektne vrednosti.

Ovo pravilo integriteta može imati i drugu, ekvivalentnu formulaciju. Ona koristi semantički aspekt pojma domen (v. odeljak 1.2), i zahteva da svaki atribut jedne relacije koji je definisan na domenu na kome je definisan atribut primarnog ključa druge relacije ima svojstva koja karakterišu strani ključ.

Relacije T_1, T_2 ne moraju biti različite. Na primer, relacija RADNIK može sadržati, pored ostalih, atribute MATIČNI_BROJ i MATIČNI_BROJ_ŠEFA; prvi navedeni atribut je primarni ključ, dok je drugi navedeni atribut – strani ključ koji se odnosi na primarni ključ iste relacije, jer šef može biti samo neko ko je radnik tog preduzeća.

Problem održavanja integriteta entiteta i referencijalnog integriteta javlja se pri svakoj operaciji izmene sadržaja baznih relacija. Posebno, referencijalni integritet može biti narušen pri unošenju novih n-torki u relaciju nad kojom je definisan strani ključ, pri izmeni vrednosti neke n-torke na atributu iz primarnog ili stranog ključa, kao i pri brisanju n-torki iz relacije na koju se odnosi strani ključ druge relacije. Ovi problemi i načini njihovog rešavanja biće detaljnije ilustrovani primerima u tački 3.3.14 o operacijama ažuriranja upitnog jezika SQL.

Svi implementirani relacioni sistemi obezbeđuju u izvesnoj meri podršku uslovima integriteta, pre svega opštim uslovima integriteta. Međutim, to nije uvek slučaj sa specifičnim uslovima integriteta. Prvi kompletni relacioni prototipski sistem, System R IBM-ove istraživačke laboratorije iz San Hozea u Kaliforniji, imao je iskaz za zadavanje specifičnih uslova integriteta koji se odnose bilo na stanje baze (vrednosti atributa, sadržaj relacije ili cele baze), bilo na promenu stanja baze. Taj iskaz može da sadrži proizvoljno tvrđenje upitnog jezika. Na primer, uslov integriteta koji se nameće pri promeni statusa izdavača je da novi status mora biti veći od starog, i u sintaksi System R zadaje se iskazom

```
ASSERT ON UPDATE TO I: New STATUS >= Old STATUS
```

Izmena podataka koja narušava uslov integriteta se odbija. Uslov integriteta može da se odnosi na pojedinačnu n-torku ili na skup n-torki (kada je u tvrđenju prisutna agregatna funkcija, v. tačku 3.3.7).

Komercijalni proizvodi razvijeni iz prototipskog sistema System R ne podržavaju na prethodni, deklarativni način, specifične uslove integriteta. Tako, na primer, neke verzije DB2 sistema nude mogućnost programiranja procedurâ za održavanje proizvoljnih (specifičnih) uslova integriteta, ali je te procedure obavezno pisati u Assembler 370 jeziku, što ih čini praktično neupotrebljivim za krajnjeg korisnika, i svakako veoma udaljenim od ideje deklarativne specifikacije. Sistem DB2 UDB Verzija 5 ima mogućnost deklarativnog zadavanja specifičnih ograničenja koja se odnose na jednu ili više kolona jedne tabele, i koja mora da zadovolji svaka vrsta te tabele. Ta ograničenja zadaju se u okviru CONSTRAINT – CHECK opcije iskaza za kreiranje tabele (CREATE) odnosno iskaza za izmenu strukture tabele (ALTER TABLE). Na primer, u tabeli KI (izdavaštvo), ograničenje da svako izdanje starije od 1980. godine mora imati tiraž bar 5000, u DB2 UDB iskazuje se CONSTRAINT opcijom pomenutih iskaza oblika

```
CONSTRAINT GODTIR CHECK (GODINA >= 1980 OR TIRAZ >=5000)
```

Opcija CONSTRAINT omogućuje zadavanje "statičnih" ograničenja, tj. ograničenja koja se odnose na vrednosti pojedinih kolona u svakoj pojedinačnoj vrsti

tabele, u svakom trenutku, i koja ne mogu izraziti odnos između starih i novih vrednosti kolona pri ažuriranju, kakav je bio prethodni primer ograničenja pri ažuriranju kolone STATUS. Ova druga mogućnost u DB2 UDB sistemu ostvarena je kroz opštiji koncept – koncept trigera kojima je posvećena sledeća tačka.

1.4.3 Trigeri

Svi dosad pomenuti oblici uslova integriteta uključuju, eksplicitno ili implicitno, pored tvrđenja, i akciju koju sistem preduzima u slučaju da tvrđenje nije tačno. Na primer, kod specifičnih uslova integriteta koji se odnose na strukturu domena, pokušaj da se unese vrednost atributa koja nije iz odgovarajućeg domena odbija se. To odbijanje da se registruje vrednost predstavlja akciju koja se preduzima u slučaju da uslov integriteta nije zadovoljen. Pokušaj da se izbriše n-torka iz jedne relacije, sa vrednošću primarnog ključa koja postoji kao vrednost stranog ključa n-torke zavisne relacije, odbija se, ili proizvodi kaskadno brisanje odgovarajućih n-torki iz zavisne relacije, odnosno postavljanje markera nedostajućih vrednosti stranog ključa u odgovarajuće n-torke zavisnih relacija (o ovim efektima brisanja biće više reči u odeljku o SQL podršci brisanju).

Opštija forma uslova integriteta, kojom se može zadati proizvoljna akcija u slučaju proizvoljnog (navedenog) događaja nad objektom u bazi, zove se *triger* (okidač, engl. trigger). Triger je proceduralni mehanizam koji specifičnoj operaciji modifikovanja podataka – *triger operaciji*, nad specifičnom baznom relacijom, pridružuje niz SQL iskaza, *triger proceduru*, koja se "aktivira" tj. izvrši kadgod se izvrši triger operacija.

Triger se može koristiti za validaciju ulaznih podataka, za automatsko generisanje vrednosti unesene vrste, za čitanje iz ili upis u druge tabele u cilju uspostavljanja međusobnih veza, za podršku uzbunjivanju (alarmiranju) kroz poruke elektronske pošte. Korišćenje trigera omogućuje brži razvoj aplikacija, globalno nametanje ograničenja (uslova integriteta) i lakše održavanje aplikacija i podataka.

U sistemima koji podržavaju mehanizam trigera uobičajeno je da za svaku od operacija unošenja, izmene i brisanja postoje po dve vrste trigera kojima se definišu procedure koje ta operacija aktivira: triger čija se triger procedura aktivira *pre* izvršenja same operacije (pre-triger) i triger čija se triger procedura aktivira *posle* izvršenja operacije (posle-triger). Pre-trigeri koriste se najčešće za proveru nekih uslova pre izvršenja triger procedure. Posle-trigeri koriste se za propagiranje vrednosti prema potrebi, ili za izvođenje drugih zadataka kao što je slanje poruke, koji mogu biti deo triger operacije.

Primer 1.1 Za relacije P (pisac) i KP (autor), sa pripadnim atributima, u DB2 UDB sistemu mogao bi da definiše sledeći triger vezan za registraciju novog autorskog dela, tj. za unošenje n-torke u relaciju KP:

```
CREATE TRIGGER KP_UNOS
       AFTER INSERT ON KP
```

```
        REFERENCING NEW AS NKP
        FOR EACH ROW MODE DB2SQL
            UPDATE P
               SET BR_NASLOVA = BR_NASLOVA + 1
               WHERE P_SIF = NKP.P_SIF
```

(ovaj se triger odnosi na automatsko uvećanje broja naslova autora za 1, a aktivira se kadgod se unese podatak o novoj knjizi tog autora; dakle, triger operacija je operacija unošenja – INSERT – u tabelu KP, a triger procedura sastoji se od jednog jedinog SQL iskaza – ažuriranja – UPDATE – tabele P; uvećanje u tabeli P izvršava se *posle* – AFTER – izvršenog unosa u tabelu KP; nova uneta vrsta u tabelu KP dobija "ime" NKP opcijom REFERENCING; triger se aktivira za svaku novu unetu vrstu u tabelu KP – FOR EACH ROW, dok je DB2SQL jedini podržani vid trigera u ovom sistemu).

Na sličan način mogu se definisati i trigeri pri brisanju, odnosno izmeni podatka o autorskom delu:

```
CREATE TRIGGER KP_BRIS
       AFTER DELETE ON KP
       REFERENCING OLD AS OKP
       FOR EACH ROW MODE DB2SQL
           UPDATE P
              SET BR_NASLOVA = BR_NASLOVA - 1
              WHERE P_SIF = OKP.P_SIF)


CREATE TRIGGER KP_AZUR
       AFTER UPDATE OF P_SIF ON KP
       REFERENCING NEW AS NKP OLD AS OKP
       FOR EACH ROW MODE DB2SQL
       BEGIN ATOMIC
             UPDATE P
                SET BR_NASLOVA = BR_NASLOVA - 1
                WHERE P_SIF = OKP.P_SIF;
             UPDATE P
                SET BR_NASLOVA = BR_NASLOVA + 1
                WHERE P_SIF = NKP.P_SIF
       END
```

(pošto se triger procedura poslednjeg trigera sastoji od dva SQL iskaza, neophodno je "zagraditi" ih BEGIN ATOMIC – END konstrukcijom, koja ukazuje da se ili oba iskaza moraju izvršiti uspešno ili nijedan).

Uslov integriteta koji kontroliše odnos starih i novih vrednosti kolona pri ažuriranju može se sada izraziti trigerom čija se triger procedura sastoji od iskaza generisanja greške u slučaju da odnos nije valjan, i izveštavanja o toj grešci.

Primer 1.2 Zahtev da pri promeni statusa izdavača (atribut STATUS tabele I), novi status mora biti veći od starog, inače se promena ne izvršava već se izdaje poruka o grešci, može biti realizovan sledećim trigerom u sistemu DB2:

```
CREATE TRIGGER N_STATUS
     NO CASCADE BEFORE UPDATE OF STATUS ON I
     REFERENCING NEW AS NI OLD AS OI
     FOR EACH ROW MODE DB2SQL
     WHEN (NI.STATUS < OI.STATUS)
        SIGNAL SQLSTATE '85000' ('Novi status manji od starog')
```

(triger se aktivira pre izvršenja operacije ažuriranja – BEFORE – a triger procedura ne proizvodi aktiviranje drugih trigera – NO CASCADE; kôd greške – niskovna konstanta '85000' bira se prema utvrđenim sintaksnim pravilima).

Korišćenjem trigera mogu se izraziti i opšti uslovi integriteta, integritet entiteta i referencijalni integritet. Na primer, referencijalni integritet nad relacijama I, KI sa kaskadnim brisanjem, ukoliko zavisne n-torke postoje u zavisnoj relaciji, može se izraziti sledećim trigerom u sistemu DB2:

```
CREATE TRIGGER REFINT
       AFTER DELETE ON  I
       REFERENCING OLD AS OI
       FOR EACH ROW MODE DB2SQL
           DELETE FROM KI
              WHERE KI.I_SIF = OI.I_SIF
```

Ipak, zbog značaja koji opšti uslovi integriteta imaju, neuporedivo je bolje imati posebnu, jednostavnu i deklarativnu (neproceduralnu) formu za njihovo zadavanje.

1.5 Pitanja i zadaci

1. Definisati sledeće pojmove:

   ANSI arhitektura
   logička reprezentacija
   konceptualna shema
   fizička reprezentacija
   pogled
   model podataka
   baza podataka
   sistem za upravljanje bazama podataka
   sistem baza podataka
   strukturni deo modela
   manipulativni deo modela
   upitni jezik
   integritetni deo modela
   relacioni model podataka
   relaciona baza podataka
   strukturni deo relacionog modela
   atribut
   domen

relacija	normalizovana relacija
bazna relacija	relacijska shema
izvedena relacija	shema relacione baze podataka

2. U čemu se razlikuje sistem za upravljanje bazama podataka od datotečkog sistema?
3. Čime je predstavljen manipulativni deo relacionog modela?
4. Šta je to relaciona kompletnost?
5. Objasniti različite vrste uslova integriteta. Kako se one predstavljaju u relacionom modelu?
6. Definisati sledeće pojmove:

funkcionalna zavisnost	integritet entiteta
primarni ključ	referencijalni integritet
nadključ	triger
strani ključ	uslov i procedura trigera

7. Navesti primer specifičnog uslova integriteta.
8. Navesti sve funkcionalne zavisnosti, primarne i strane ključeve relacija I, K, P, KI, KP.
9. Navesti primer trigera.
10. Razmotriti primer relacione baze podataka o studentima. Koje se relacije i koje funkcionalne zavisnosti među njihovim atributima mogu uočiti?
11. Navesti primer upita i radnji nad relacionom bazom iz primera 10.
12. Koje su najvažnije karakteristike relacionog modela?
13. Koji su tipovi objekata u hijerarhijskom a koji u mrežnom modelu?
14. Uporediti relacioni, hijerarhijski i mrežni model.
15. Šta su to nedostajuće vrednosti?
16. Pored raznih definicija "klasičnog" relacionog modela, postoji čitav niz modela koji predstavljaju specifične varijante relacionog modela. Takvi su, na primer, binarni relacioni modeli koji se zasnivaju na konceptu binarne relacije, nesvodivi relacioni modeli koji se baziraju na relacijama koje se nikako ne mogu dekomponovati bez gubljenja informacije, funkcionalni modeli, bazirani na unarnim relacijama i funkcijama među njima ([21]), itd. Kako bi u binarnom relacionom modelu izgledala shema baze podataka iz primera 10?

2

Manipulativni formalizmi relacionog modela

Kao što je već pomenuto u odeljku 1.3, formalizmi za manipulisanje podacima, koji su sastavni deo relacionog modela, jesu relaciona algebra i relacioni račun. Upitni jezici u sastavu SUBP imaju svoj osnov u jednom od ova dva formalizma, ili u njihovoj kombinaciji.

Skup svojstava koja treba da podrži svaki upitni jezik biće u sledećem odeljku prikazan kroz formalizam relacione algebre, jer je ovaj formalizam, u odnosu na relacioni račun, jednostavniji za razumevanje pri prvom susretu sa pojmovima, i olakšava objašnjenje koncepata kao što su, na primer, uslovi integriteta, mehanizmi autorizacije ili algoritmi ažuriranja pogleda. Uprkos pogodnosti relacione algebre u prezentaciji i razjašnjavanju koncepata, relacioni račun pokazuje prednosti u implementaciji koncepata kroz relacioni upitni jezik. Zato će u odeljku 2.2 biti izložen i formalizam relacionog računa.

2.1 Relaciona algebra

Prema originalnoj Codd-ovoj definiciji, relacionu algebru čini skup od devet operacija nad relacijama, izabranih tako da se njihovim komponovanjem može pristupiti svakom podatku u bazi. Kompozicija operacija zajedno sa relacijama na koje se primenjuju obrazuje relacioni izraz u relacionoj algebri ili *izraz relacione algebre*. Vrednost takvog izraza je uvek relacija. Operacije relacione algebre su relacioni analogoni skupovnih operacija unije, preseka, razlike i Dekartovog proizvoda, kao i specifične relacione operacije: unarne – projekcija i restrikcija, i binarne – prirodno spajanje, slobodno spajanje i relaciono deljenje.

Upotrebljena notacija za operacije relacione algebre je jedna od mogućih notacija i koristi se za ilustrovanje elemenata i koncepata formalizma. Skup operacija treba interpretirati kao jednu definiciju moći koju svaki relacioni jezik mora da poseduje.

Osnovne operacije relacione algebre biće definisane na sledećim dvema (apstraktnim) relacijama: $R(A_1, A_2, \ldots, A_n)$, $S(B_1, B_2, \ldots, B_m)$.

2.1.1 Unarne operacije

Pod unarnim relacionim operacijama podrazumevaju se operacije koje se izvršavaju nad pojedinačnom relacijom. To su operacije projekcije i restrikcije.

Osim operanda – imena relacije, uz ove operacije se navodi još po jedan operand tipa skupa atributa (u slučaju operacije projekcije), odnosno logičkog izraza (u slučaju operacije restrikcije).

Projekcija. Neka je X podskup skupa atributa $\{A_1, A_2, \ldots, A_n\}$ relacije R, a Y – komplement skupa X u odnosu na ceo skup atributa $\{A_1, A_2, \ldots, A_n\}$. Tada se promenom redosleda atributa relacija R može predstaviti kao $R(X, Y)$[1]. Operacija projekcije relacije R na skup atributa X obeležava se sa $R[X]$ i definiše na sledeći način:

$$R[X] = \{x \mid x \in dom(X) \text{ i postoji } y \in dom(Y) \text{ takav da } (x, y) \in R(X, Y)\}.^2$$

Dakle, rezultat operacije projekcije relacije R na skup atributa X je relacija koja uključuje samo atribute iz skupa X. Drugim rečima, operacijom projekcije se iz tabele kojom je predstavljena relacija R biraju samo kolone koje odgovaraju atributima iz skupa X, i to bez ponavljanja dupliranih torki.

Operacija projekcije može se primeniti i na pojedinačnu torku, s obzirom da se torka može tretirati kao relacija koja sadrži tačno tu torku. Tako, ako je $r(A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n)$ – n-torka, onda se operacija projekcije n-torke r na skup atributa $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ definiše sa:

$$r[A_{i_1}, A_{i_2}, \ldots, A_{i_k}] = (v_{i_1}, v_{i_2}, \ldots, v_{i_k}).$$

Analogno se i ostale operacije relacione algebre, definisane nad relacijama, mogu primeniti i na pojedinačne torke.

Primer 2.1 Rezultat operacije projekcije relacije K na atribut OBLAST, u oznaci K[OBLAST], jeste relacija koja ima jedan atribut (OBLAST) i koja sadrži sve različite oblasti iz kojih su knjige registrovane u bazi:

OBLAST
roman
poezija
leksikografija
računarstvo

[1] Umesto oznake $R(X, Y)$ koristi se i oznaka $R(XY)$.
[2] Umesto oznake torke (x, y), koristiće se i oznaka xy.

Pokazuje se da je relaciona projekcija algebarski analogon egzistencijalnog kvantifikatora koji se primenjuje u relacionom računu. Intuitivno, ako za relaciju $R(X, Y)$ sa dva disjunktna podskupa skupa atributa X, Y važi da je $x \in R[X]$, onda u relaciji R **postoji** n-torka r takva da je $r[X] = x$.

Restrikcija. Neformalno govoreći, operacijom restrikcije biraju se n-torke (u celosti) zadate relacije koje zadovoljavaju logički izraz naveden u oznaci operacije. Argumenti logičkog izraza su poređenja vrednosti dva atributa zadate relacije (definisana nad istim domenima), ili poređenja jednog atributa te relacije sa konstantom iz odgovarajućeg domena; operacije logičkog izraza su logičke operacije NOT, OR i AND (uz uobičajena pravila prioriteta).

Za formalno definisanje operacije restrikcije koristi se uži pojam Θ-restrikcije, koji se uvodi na sledeći način. Neka je Θ relacijska operacija iz skupa $\{=, \neq, <, \leq, >, \geq\}$, i neka su A_i, A_j – atributi relacije R sa zajedničkim domenom nad kojim je definisana operacija Θ; neka je a – vrednost iz tog domena. Tada zapis $R[A_i \Theta A_j]$ (ili $R[A_i \Theta a]$) označava operaciju Θ-restrikcije, definisanu sa:

$$R[A_i \Theta A_j] = \{x \mid x \in R \text{ i } x[A_i] \Theta x[A_j]\} \text{ (slično za } R[A_i \Theta a]).$$

Poređenja $A_i \Theta A_j,\ A_i \Theta a$ zovu se *termi poređenja*.

Relacioni logički izraz je:

- term poređenja;
- NOT (relacioni logički izraz);
- (relacioni logički izraz) AND (relacioni logički izraz);
- (relacioni logički izraz) OR (relacioni logički izraz).

Neka je P relacioni logički izraz. Tada se izrazu P može pridružiti preslikavanje koje svaku vrstu relacije R preslikava u 'tačno' ili 'netačno'. Operacija restrikcije (ili *proširene* Θ*-restrikcije*, [19]) relacije R po relacionom logičkom izrazu[3] P obeležava se sa $R[P]$ i definiše se na sledeći način:

$$R[P] = \{x \mid x \in R \text{ i } P(x)\}.$$

Primer 2.2 Rezultat restrikcije relacije K po uslovu OBLAST = 'roman', u oznaci K[OBLAST = 'roman'], jeste relacija sa istom shemom kao i relacija K, i sa dve trojke:

K_SIF	NASLOV	OBLAST
k1	Osma ofanziva	roman
k3	Pionirska trilogija	roman

[3]U daljem tekstu, umesto "relacioni logički izraz" koristiće se i termin "logički izraz". U istom značenju koristi se i termin "logički uslov".

2.1.2 Binarne operacije

Prvi skup binarnih operacija relacione algebre predstavljaju relacioni analogoni klasičnih skupovnih operacija unije, preseka, razlike i Dekartovog proizvoda.

Relaciona unija, presek, razlika i Dekartov proizvod. Operacija relacione unije je specifična u odnosu na tu operaciju u matematici. Ova operacija u matematici dopušta uniranje raznorodnih skupova, kao što su skupovi knjiga, izdavača i autora, dok relaciona operacija unije dopušta uniranje skupa knjiga sa drugim skupom knjiga, skupa izdavača sa drugim skupom izdavača, skupa autora sa drugim skupom autora. Relaciona operacija unije treba da obezbedi predstavljanje jednom relacijom svih objekata (n-torki) iz dve odabrane relacije, pod uslovom da su objekti u tim relacijama iste vrste. Objekti u dve relacije su iste vrste ako su te relacije *unijski kompatibilne*, pri čemu se unijska kompatibilnost definiše na sledeći način: relacije R i S su unijski kompatibilne ako su istog stepena ($m = n$), i za bar jedno $1 - 1$ preslikavanje skupa atributa relacije R u skup atributa relacije S važi da odgovarajući parovi atributa relacija R, S uzimaju vrednosti iz istih domena. Pri relacionoj operaciji unije, redosled atributa u relaciji R i redosled njihovih slika u relaciji S, pri jednom takvom preslikavanju, mora biti implicitno ili eksplicitno naznačen, nezavisno od principa da je redosled atributa relacije nebitan. Pod tim uslovima,

$$R \cup S = \{t \mid t \in R \text{ ili } t \in S\}.$$

Treba naglasiti da, kao što i skupovna definicija kaže, operacija uniranja podrazumeva i eliminaciju duplikata, kako bi rezultat bio relacija.

Primer 2.3 Ako su atributi DRZAVA u relacijama P (pisac) i I (izdavač) definisani nad istim domenom, onda se može izračunati unija projekcijâ ovih relacija na zajednički atribut:

P[DRZAVA] $\cup$ I[DRZAVA] =

DRZAVA
Jugoslavija
Engleska
Amerika

Pod istim uslovima kao i za operaciju unije, definišu se operacije *preseka* i *razlike* na sledeći način:

$$R \cap S = \{t \mid t \in R \text{ i } t \in S\} \text{ (presek)},$$
$$R \setminus S = \{t \mid t \in R \text{ i } t \notin S\} \text{ (razlika)}.$$

Dekartov proizvod relacija R_1 i R_2 sa po n_1, n_2 atributa, redom, jeste relacija $R_1 \times R_2$ sa $n_1 + n_2$ atributa čije su vrste dobijene dopisivanjem vrsta

relacije R_2 na vrste relacije R_1. Ako je $t^1 = (t^1_1, \ldots, t^1_{n_1})$ proizvoljna n_1-torka relacije R_1 a $t^2 = (t^2_1, \ldots, t^2_{n_2})$ proizvoljna n_2-torka relacije R_2, tada je $(t^1, t^2) = (t^1_1, t^1_2, \ldots, t^1_{n_1}, t^2_1, t^2_2, \ldots, t^2_{n_2})$ odgovarajuća $n_1 + n_2$-torka relacije $R_1 \times R_2$.

Dekartov proizvod k relacija ($k > 2$) $R_1, R_2, \ldots, R_k$ definiše se kao $(R_1 \times R_2 \times \cdots \times R_{k-1}) \times R_k$. Ova operacija se uglavnom uvodi iz koncepcijskih razloga, a veoma retko zaista i izvršava, s obzirom na veličinu relacije (broj i dužinu vrsta) koju proizvodi kao rezultat.

Primer 2.4 Rezultat Dekartovog proizvoda relacija K i I je sledeća relacija sa 7 atributa i 24 sedmorke:

K_SIF	NASLOV	OBLAST	I_SIF	NAZIV	STAT.	DRZAVA
k1	Osma ofan.	roman	i1	Prosveta	30	Jugoslav.
k1	Osma ofan.	roman	i2	Addison...	20	Amerika
k1	Osma ofan.	roman	i3	Dečje...	10	Jugoslav.
k1	Osma ofan.	roman	i4	Matica...	30	Jugoslav.
k2	Nemam v.v.	poezija	i1	Prosveta	30	Jugoslav.
k2	Nemam v.v.	poezija	i2	Addison...	20	Amerika
k2	Nemam v.v.	poezija	i3	Dečje...	10	Jugoslav.
k2	Nemam v.v.	poezija	i4	Matica...	30	Jugoslav.
...	...	...	...	...	...	...
k6	Tražim pom.	poezija	i4	Matica...	30	Jugoslav.

(skraćenja oznaka su iz prostornih razloga).

Slobodno (Θ) spajanje. Operacija slobodnog spajanja (ili Θ-spajanja) primenjuje se na dve relacije, R i S, i proizvodi kao rezultat relaciju sa $m + n$ atributa. Ta relacija sadrži n-torke relacije R na koje su dopisane m-torke relacije S, ali samo ako su vrednosti naznačenih atributa A_i odnosno B_j tih torki u relaciji Θ.

Neka je Θ relacijska operacija iz skupa $\{=, \neq, <, \leq, >, \geq\}$, i neka su A_i, B_j atributi relacija R, S, redom. Tada se operacija slobodnog (Θ) spajanja relacija R, S po atributima A_i, B_j, redom, obeležava sa $R[A_i \Theta B_j]S$ i definiše na sledeći način:

$$R[A_i \Theta B_j]S = \{(r, s) \mid r \in R \text{ i } s \in S \text{ i } r[A_i] \Theta s[B_j]\}.$$

Prirodno spajanje. Operacija prirodnog spajanja je specijalni (i najčešći) slučaj operacije slobodnog spajanja, a to je spajanje po jednakosti. Ovo spajanje se može izvesti po jednakosti podskupova atributa dveju relacija (a ne samo po jednakosti pojedinačnih atributa), pri čemu se iz rezultata isključuje jedan od tih podskupova.

Neka su za relacije R, S uočeni podskupovi atributa $X \subset \{A_1, A_2, \ldots, A_n\}$ i $Y \subset \{B_1, B_2, \ldots, B_m\}$, koji imaju jednak broj elemenata a odgovarajući parovi (prvih, drugih, itd.) atributa imaju iste domene. Tada se komplementi skupova

atributa X odnosno Y, u odnosu na skup svih atributa relacije R odnosno S, mogu obeležiti sa Z odnosno W. Permutacijom skupova atributa, relacije R, S mogu se predstaviti sa $R(Z, X)$, $S(Y, W)$. Tada se operacija prirodnog spajanja relacija R, S po skupovima atributa X, Y, redom, obeležava sa $R[X * Y]S$ i definiše na sledeći način:

$$R[X * Y]S = \{(z, x, w) \mid (z, x) \in R \text{ i } (y, w) \in S \text{ i } x = y\}.$$

Kada su X i Y – svi zajednički atributi relacija R i S, operacija prirodnog spajanja se obeležava sa $R * S$.

Primer 2.5 Rezultat operacije prirodnog spajanja relacija P i I (spajanje po jednakosti atributâ DRZAVA) je sledeća relacija sa 7 atributa i 10 sedmorki od kojih se svaka odnosi na pisca i izdavača iz iste zemlje:

P_SIF	IME	BR_N.	DRZAVA	I_SIF	NAZIV	STATUS
p1	B.Ćopić	2	Jugoslav.	i1	Prosveta	30
p1	B.Ćopić	2	Jugoslav.	i3	Dečje novine	10
p1	B.Ćopić	2	Jugoslav.	i4	Matica srpska	30
p3	B.Š.Š.	1	Jugoslav.	i1	Prosveta	30
p3	B.Š.Š.	1	Jugoslav.	i3	Dečje novine	10
p3	B.Š.Š.	1	Jugoslav.	i4	Matica srpska	30
p4	D.Maks.	2	Jugoslav.	i1	Prosveta	30
p4	D.Maks.	2	Jugoslav.	i3	Dečje novine	10
p4	D.Maks.	2	Jugoslav.	i4	Matica srpska	30
p5	C.J.Date	1	Amerika	i2	Addison Wesley	20

(skraćenja oznaka su iz prostornih razloga).

Deljenje. Relaciona operacija deljenja relacije $R(X, Y)$ po zadatom skupu atributa Y drugom relacijom T proizvodi rezultat (analogan celobrojnom delu količnika pri deljenju celih brojeva) koji se sastoji od projekcijâ n-torki relacije R na atribute X, i to samo onih projekcija n-torki koje se, (Dekartovski) pomnožene svakom vrstom relacije T, sadrže u relaciji R.

Neka je, kao i do sada, $R(A_1, A_2, \ldots A_n)$ relacija, i neka je $Y = \{Y_1, Y_2, \ldots, Y_k\} \subset \{A_1, A_2, \ldots, A_n\}$, a $X = \{A_1, A_2, \ldots, A_n\} \setminus Y$. Neka relacija $T(Z_1, Z_2, \ldots, Z_k)$ ima isti broj atributa kao skup Y, pri čemu su parovi odgovarajućih atributa definisani nad istim domenima. Označimo simbolom Z skup atributa relacije T. Tada se operacija deljenja relacije R po skupu atributa Y relacijom T obeležava sa $R[Y \div Z]T$ i definiše sa:

$$R[Y \div Z]T = \{x \mid x \in R[X] \text{ i } \{x\} \times T(Z) \subseteq R(X, Y)\}.$$

Dakle, rezultat operacije deljenja je maksimalni podskup projekcije $R[X]$ čiji je Dekartov proizvod sa $T(Z)$ sadržan u $R(X, Y)$.

Primer 2.6 Neka je TEMP – relacija koja sadrži sve šifre knjiga Branka Ćopića, tj.

((P[IME = 'B.Ćopić']) ∗ KP) [K_SIF] = TEMP

K_SIF
k1
k3

(o mogućnosti dodeljivanja vrednosti relacionog izraza nekoj relaciji biće reči na kraju izlaganja o relacionoj algebri).

Šifre izdavača koji su štampali sve knjige Branka Ćopića mogu se dobiti deljenjem, relacijom TEMP, projekcije relacije KI na atribute K_SIF, I_SIF:

(KI[K_SIF, I_SIF]) [K_SIF ÷ K_SIF] TEMP =

I_SIF
i1

Dakle, jedini takav izdavač je 'Prosveta' sa šifrom i1.

Za dve proizvoljne relacije R, S važi da je $(R \times S) \div S = R$, tj. operacija relacionog deljenja je inverzna operaciji množenja (Dekartovog proizvoda). Otuda i naziv "deljenje" za razmatranu operaciju.

Pokazuje se da je operacija deljenja algebarski analogon univerzalnog kvantifikatora. Intuitivno, ako za relaciju $R(X, Y)$ sa dva disjunktna podskupa atributa X, Y i relaciju T(Z), važi da $x \in R[Y \div Z]T$, onda **za svaku** vrednost $z \in T$ važi da je $(x, z) \in R(X, Y)$.

Primer 2.7 Primer izraza relacione algebre, koji uključuje nekoliko njenih operacija, je nalaženje naziva jugoslovenskih izdavača koji su štampali bar jednu knjigu engleskog pisca, kao i imenâ takvih pisaca i naslova njihovih knjiga:

((I[DRZAVA = 'Jugoslavija'] × P[DRZAVA = 'Engleska']) ∗ KI ∗ KP ∗ K)
[NAZIV, IME, NASLOV]

Osim operacija relacione algebre koje je uveo Codd, relaciona algebra se, udobnosti radi, često proširuje sa dve dodatne operacije, redovno podržane i u upitnim jezicima ([22]):

- definisanje aliasa – drugog imena za relaciju, što se posebno koristi pri izračunavanju Dekartovog proizvoda relacije sa samom sobom, da se ne bi ponovila imena atributa u relaciji rezultatu:

  DEFINE ALIAS ime-relacije-1 FOR ime-relacije-2

- dodela, kojom se rezultat relacionog izraza dodeljuje novoj relaciji:

  relacija-1 := relacioni_izraz.

Relaciona algebra, kako je ovde izložena, u izvesnom smislu je maksimalna: ona sadrži sve operacije relacione algebre. Može se, međutim, dokazati, da taj skup

operacija nije minimalan, u smislu da se neke od operacija mogu isključiti a da preostali skup operacija zadrži istu izražajnu moć. Jedan minimalni, ekvivalentni skup operacija relacione algebre čine operacije unije, razlike, Dekartovog proizvoda, projekcije i restrikcije ([64]).

2.1.3 Proširena relaciona algebra

Proširenje relacione algebre koje se u ovoj tački razmatra odnosi se na podršku jednoj vrsti nedostajuće vrednosti, kao i na šire definisanje operacija unije, preseka, razlike i operacija spajanja. Izlažemo Codd-ovo proširenje relacionog modela i relacione algebre ([18]).

Domen svakog atributa proširen je jednom oznakom nedostajuće vrednosti, koja se može upotrebiti za oznaku bilo koje (svake) nedostajuće vrednosti, bez obzira na njenu semantiku. Tada se i skup istinitosnih vrednosti {T, F} proširuje "nepoznatom" istinitosnom vrednošću (označimo je sa ω), i postaje {T, F, ω}. Primetimo da je semantika nepoznate istinitosne vrednosti različita od semantike nedostajuće vrednosti za atribut (mada mogu biti isto označene), jer je nepoznata istinitosna vrednost – vrednost iz domena istinitosnih vrednosti, dok nedostajuća vrednost atributa to nije.

Proširena relacija jednakosti (u oznaci $\doteq$) je sada trovalentna, tj.

$$\tau(a \doteq b) \in \{T, F, \omega\},$$

a takve su i druge relacije poređenja, pripadnosti skupu i inkluzije. Specijalno, $\tau(\omega \dot{\in} S) = \omega$ i $\tau(\{\omega\} \dot{\subseteq} S) = \omega$, za svaki neprazni skup S i nedostajuću vrednost istog tipa, ω.

Da bi se definisao logički izraz u ovoj trovalentnoj logici, potrebno je definisati i odgovarajuće (trovalentne) logičke operacije, koje se zadaju sledećim tablicama:

AND	T	F	ω
T	T	F	ω
F	F	F	F
ω	ω	F	ω

OR	T	F	ω
T	T	T	T
F	T	F	ω
ω	T	ω	ω

NOT	
T	F
F	T
ω	ω

Prošireni logički izraz dobija se primenom proširenih logičkih operacija na proširena poređenja. Pri tom, on treba da ima istinitosnu vrednost ω ako i samo ako su ispunjena sledeća dva uslova:

- svako pojavljivanje nedostajuće vrednosti ω u izrazu može se zameniti određenom vrednošću iz odgovarajućeg domena tako da izraz dobije vrednost T;
- svako pojavljivanje nedostajuće vrednosti ω u izrazu može se zameniti određenom vrednošću iz odgovarajućeg domena tako da izraz dobije vrednost F.[4]

[4] Uočeno je da postoji niz problema u interpretaciji nedostajućih vrednosti, posebno trovalentne

Na primer, $\tau(\omega \doteq a) = \omega$ (a je vrednost iz nekog domena; kada se ω kao argument poređenja zameni sa a, vrednost poređenja postaje T; kada se ω zameni sa b, za $b \neq a$, vrednost poređenja postaje F).

Proizvoljni upit sa zadatim logičkim izrazom sada se interpretira kao – "naći sve podatke *za koje se zna* da je vrednost logičkog izraza 'tačno' (a ne "naći sve podatke za koje je vrednost logičkog izraza 'tačno' "), pa ne moraju da važe sve tautologije dvovalentne logike.

Razlike u definiciji proširenih logičkih izraza u trovalentnoj logici (u odnosu na odgovarajuću definiciju onih u dvovalentnoj logici) reflektuju se i na razlike u definiciji operacija proširene relacione algebre. Naime, za svaku operaciju relacione algebre, s obzirom da u svoju definiciju uključuje neku od proširenih relacija (poređenje, pripadnost skupu, inkluzija), uvode se po dve nove operacije: "TAČNA" i "MOŽDA" operacija, čiji su rezultati, redom, skup podataka za koje se zna da pripadaju rezultatu operacije, i skup podataka za koje se ne zna da li pripadaju rezultatu operacije. Tako se TAČNA restrikcija relacije R po logičkom uslovu P, u oznaci $R[P_T]$, može definisati na sledeći način:

$$R[P_T] = \{x \mid x \in R \text{ i } \tau(P) = T\}.$$

Na primer, TAČNA restrikcija relacije R po Θ relacijskoj operaciji na atributima A_i i A_j, u oznaci $R[A_i \Theta_T A_j]$, definiše se kao

$$R[A_i \Theta_T A_j] = \{x \mid x \in R \text{ i } \tau(x[A_i] \dot{\Theta} x[A_j]) = T\}.$$

Slično, MOŽDA restrikcija relacije R po logičkom uslovu P, u oznaci $R[P_\omega]$, može se definisati na sledeći način:

$$R[P_\omega] = \{x \mid x \in R \text{ i } \tau(P) = \omega\}.$$

Na primer, MOŽDA restrikcija relacije R po Θ relacijskoj operaciji na atributima A_i i A_j, u oznaci $R[A_i \Theta_\omega A_j]$, definiše se kao

$$R[A_i \Theta_\omega A_j] = \{x \mid x \in R \text{ i } \tau(x[A_i] \dot{\Theta} x[A_j]) = \omega\}.$$

Slično važi i za operacije spajanja (Θ i prirodnog), kao i za operaciju deljenja.

Uvođenje nedostajuće vrednosti u relacioni model omogućuje i proširenje operacija unije, preseka i razlike, sa unijski kompatibilnih relacija na relacije koje to

logike ([25]). Tako će neki logički izraz, striktnom primenom proširenih logičkih operacija, dobiti vrednost ω, mada jedan od navedenih uslova nije ispunjen ([27]). Na primer, ako relacija R ima samo jedan atribut, A, i jednu "n-torku" – vrednost a, a relacija S – dva atributa, B, A, i jednu "n-torku" – par (b, ω), tada će prošireni logički izraz $\tau(R.A \doteq S.A)$ AND $\tau(S.A \doteq' a'_1)$, primenom proširenih logičkih operacija, dobiti vrednost ω (ω AND $\omega = \omega$), mada prema navedenim uslovima izraz treba da ima vrednost F. Zbog ovakvih problema neki autori sugerišu da se u relacione sisteme i ne uključuju nedostajuće vrednosti, dok drugi preporučuju pažljivu preradu komponenti RSUBP, posebno optimizatora, da bi se ovakvi problemi rešili.

nisu. Tako se za relacije $R(A,B)$ i $S(B,C)$ (A,B,C su skupovi atributa) operacija *spoljašnje unije*, u oznaci $R \overset{\circ}{\cup} S$, definiše kao

$$R \overset{\circ}{\cup} S = R_1 \cup S_1, \text{ gde je } R_1 = R \times (C:\omega)^5 \text{ i } S_1 = (A:\omega) \times S.$$

Slično se definišu i proširene – *spoljašnje* operacije preseka i razlike.

Spoljašnje slobodno spajanje relacija R, S definiše se tako da rezultujuća relacija sadrži sve podatke tih relacija, čak i kada njihove projekcije na atribute spajanja nisu jednake. Tako, za relacije $R(A,B)$ i $S(C,D)$ spoljašnje Θ-spajanje po paru atributa B i C, u oznaci $R[B \overset{\circ}{\Theta} C]S$, definiše se kao unija tri skupa:

$$R[B \overset{\circ}{\Theta} C]S = T \cup (R_1 \times (C:\omega, D:\omega)) \cup ((A:\omega, B:\omega) \times S_1);$$

relacija T je rezultat običnog Θ-spajanja, a relacije R_1 i S_1 su delovi relacija R i S, redom, koji se ne mogu spojiti Θ-spajanjem sa drugom relacijom, tj.

$$\begin{array}{lcl} T & = & R[B\Theta C]S, \\ R_1 & = & R \setminus T[A,B], \\ S_1 & = & S \setminus T[C,D]. \end{array}$$

Spoljašnje prirodno spajanje definiše se na sličan način kao spoljašnje slobodno spajanje; sada se za Θ uzima relacijska operacija $=$, a iz rezultata se uklanjanju kolone dupliranih vrednosti:

$$R[B \overset{\circ}{*} C]S = T[A,B,D] \cup (R_1 \times (D:\omega)) \cup ((A:\omega) \times S_1)$$

(relacije T, R_1 i S_1 su kao i ranije, samo sa '$=$' umesto 'Θ').

Ovako definisana spoljašnja spajanja zovu se i *puna* spajanja (engl. FULL JOIN), za razliku od običnih, *unutrašnjih* spajanja (engl. INNER JOIN). Ako se iz prethodnih definicionih izraza izostavi druga odnosno treća komponenta ($R_1 \times \ldots$, odnosno $\cdots \times S_1$), dolazi se do definicije *desnog* (engl. RIGHT) odnosno *levog* (engl. LEFT) spoljašnjeg spajanja.

Detaljne informacije o proširenoj trovalentnoj logici, 'tačnim', 'možda' i spoljašnjim operacijama relacione algebre, kao i primeri trovalentnih logičkih izraza i proširenih operacija, mogu se naći u [18], [25], [26].

2.2 Relacioni račun

Drugi formalizam kojim se može opisati manipulativni deo relacionog modela je relacioni račun. Dok relaciona algebra daje skup eksplicitnih *operacija za izgradnju* relacije rezultata, relacioni račun pruža *notaciju za opisivanje svojstava* relacije rezultata. Tako relaciona algebra predstavlja recepturni, proceduralni pristup, a relacioni račun – deskriptivni, neproceduralni.

[5] Oznaka $(X:\omega)$ označava relaciju sa jednim atributom – X i jednom "vrstom" – ω; slično važi i za par $(X:\omega, Y:\omega)$.

Na primer, zahtev za nalaženjem naslova knjiga koje je izdalo izdavačko preduzeće 'Prosveta', mogao bi se, u terminima operacija relacione algebre, preciznim navoďenjem redosleda izvršavanja operacija, izraziti na sledeći način :

– izvršiti restrikciju relacije I po uslovu NAZIV='Prosveta';

– izvršiti prirodno spajanje dobijenog meďurezultata i relacije KI;

– izvršiti prirodno spajanje dobijenog meďurezultata i relacije K;

– projektovati meďurezultat na atribut NASLOV.

Isti zahtev bi se, manje proceduralnom notacijom relacionog računa, izrazio opisom sledećih svojstava rezultata:

– naći naslove knjiga za koje postoje izdanja izdavača čiji je naziv 'Prosveta'.

Relacioni račun je baziran na predikatskom računu prvog reda. Dve varijante ovog računa definisao je, kao i relacioni model, Edgar Codd; jedna je relacioni račun n-torki, a druga – relacioni račun domena.

2.2.1 Relacioni račun n-torki

Kao što relaciona algebra predstavlja skup operacija nad relacijama, kojima se može izgraditi izraz relacione algebre, tako relacioni račun n-torki predstavlja formalizam za izgradnju relacionog izraza, *izraza relacionog računa*.

Bitno svojstvo relacionog računa n-torki predstavlja n-torna promenljiva. To je promenljiva koja je definisana nad specifičnom relacijom za koju je *vezana*, tj. promenljiva koja kao svoje vrednosti uzima n-torke te relacije, i samo njih. Relacija za koju je vezana n-torna promenljiva zove se i *dijapazon* te n-torne promenljive. S obzirom da je relacioni račun formalni koncept, razne implementacije usvajaju različite sintakse za opis njegovih konstrukcija, i način na koji se jednoj n-tornoj promenljivoj dodeljuje njen dijapazon.

Jedna varijanta originalne Codd-ove definicije i sintakse izraza relacionog računa n-torki (*alfa-izraza*) može se opisati sledećim elementima ([17], [64]).

Izraz relacionog računa n-torki je oblika

$$\{(t_1, t_2, \ldots, t_k) : f\},$$

pri čemu važi:

1. Promenljive $t_1, t_2, \ldots, t_k$ su n-torne promenljive (npr. r, s, t) ili indeksirane n-torne promenljive (npr. $r[1], s[3], t[2]$), i njihov skup se zove *ciljna lista*.

2. f je formula (*kvalifikacioni izraz*) koja se gradi od *atoma* i operacija na sledeći način:

   a1) ako je R ime relacije, a s – n-torna promenljiva vezana za relaciju R, onda oznaka $R(s)$ označava to vezivanje i zove se *atom pripadnosti*;

   a2) ako su s, u n-torne promenljive, a – konstanta, a Θ – operacija poređenja $(=, \neq, <, \leq, >, \geq)$, onda oznaka $s[i]\Theta u[j]$ (odnosno $s[i]\Theta a$) označava poređenje i-te komponente promenljive s sa j-tom komponentom promenljive u (odnosno konstantom a) i zove se *atom poređenja*;

   f1) svaki atom je *formula*; sve n-torne promenljive atoma su slobodne promenljive u toj formuli;

   f2) ako su f, g formule, onda su i f AND g, f OR g i NOT f – formule; pojavljivanje promenljive je slobodno ili vezano u f AND g, f OR g, NOT f, tačno onakvo kakvo je odgovarajuće pojavljivanje te promenljive u f odnosno u g;

   f3) ako je f formula, onda su i $(\exists s)(f)$ i $(\forall s)(f)$ formule; n-torna promenljiva s ima definisan dijapazon u formuli f; ona je slobodna u formuli f i vezana egzistencijalnim (odnosno univerzalnim) kvantifikatorom u formuli $(\exists s)(f)$ (odnosno u formuli $(\forall s)(f)$);

   f4) ako je f formula, onda je i (f) formula;

   f5) ništa drugo nije formula relacionog računa n-torki.

3. Slobodne promenljive u formuli f su one i samo one n-torne promenljive koje su sadržane u ciljnoj listi.

Primer 2.8 Neka su r, s, t imena n-tornih promenljivih, i neka su K(r), P(t), KP(s) atomi pripadnosti relacijama K, P, KP, redom. Označimo sa $r[i]$ – i-ti atribut promenljive r (analogno značenje imaju $s[j], t[k]$; pretpostavlja se da je skup atributa relacije uređen). Tada su primeri izraza relacionog računa:

$\{r[2] : \text{K}(r)\}$ (naslovi svih knjiga)

$\{r[2] : \text{K}(r) \text{ AND } r[3] = \text{'roman'}\}$ (naslovi svih romana)

$\{(r[2], r[1]) : \text{K}(r) \text{ AND } (\exists s)(\exists t) \text{ (KP}(s) \text{ AND P}(t) \text{ AND } s[1] = r[1] \text{ AND } s[2] = t[1] \text{ AND } t[4] = \text{'Jugoslavija'})\}$
(naslovi i šifre knjiga čiji bar jedan autor ima jugoslovensko državljanstvo).

Jedna specifična sintaksa zadavanja izraza relacionog računa n-torki, koja je bliska sintaksi upitnih jezika zasnovanih na relacionom računu, opisana je u [22] i biće ovde ukratko prikazana.

Za deklaraciju n-torne promenljive i njeno vezivanje za relaciju, u ovoj sintaksi koristi se iskaz oblika

RANGE OF <ime-promenljive> IS <ime- relacije>

(npr. RANGE OF k IS K).

Upit se u ovoj sintaksi izražava izrazom relacionog računa (kraće: izrazom) koji u Bekusovoj notaciji ima sledeći oblik:

<izraz>	::=	<ciljni-atributi>
		| <ciljni-atributi> WHERE <formula>
<ciljni-atributi>	::=	<promenljiva>.<atribut>{,<promenljiva>.<atribut>}
<formula>	::=	<poređenje>
		| (<formula>)
		| NOT <formula>
		| <formula> AND <formula>
		| <formula> OR <formula>
		| EXISTS <promenljiva> (<formula>)
		| FORALL <promenljiva> (<formula>)
<poređenje>	::=	<promenljiva>.<atribut> ($=$ | $\neq$ | $<$ | $\leq$ | $>$ | $\geq$)
		(<promenljiva>.<atribut> | <konstanta>)

(pod "promenljiva" podrazumeva se ime n-torne promenljive; "atribut" je ime atributa).

Dakle, izraz relacionog računa je oblika

$$\mathrm{X}_1.\mathrm{A}_1,\ \mathrm{X}_2.\mathrm{A}_2,\ \ldots,\ \mathrm{X}_k.\mathrm{A}_k \text{ WHERE } f,$$

pri čemu su $\mathrm{X}_1,\ \mathrm{X}_2, \ldots,\ \mathrm{X}_k$ – n-torne promenljive, $\mathrm{A}_1,\ \mathrm{A}_2,\ \ldots, \mathrm{A}_k$ su atributi relacija – dijapazona promenljivih $\mathrm{X}_1,\ \mathrm{X}_2,\ \ldots,\ \mathrm{X}_k$, redom, a f je formula koja koristi tačno $\mathrm{X}_1,\ \mathrm{X}_2,\ \ldots,\ \mathrm{X}_k$ kao slobodne promenljive. Neka $rel(\mathrm{X})$ označava relaciju – dijapazon n-torne promenljive X. Tada vrednost izraza relacionog računa jeste projekcija na atribute $\mathrm{A}_1,\ \mathrm{A}_2,\ \ldots,\ \mathrm{A}_k$ onog podskupa Dekartovog proizvoda

$$rel(\mathrm{X}_1) \times rel(\mathrm{X}_2) \times \cdots \times rel(\mathrm{X}_k)$$

za koji f dobija vrednost 'tačno'.

Primer 2.9 Izrazi relacionog računa iz primera 2.8 u ovoj sintaksi imaju sledeći oblik:

```
RANGE OF k    IS K
RANGE OF kp   IS KP
RANGE OF p    IS P

k.NASLOV                                          (naslovi svih knjiga)

k.NASLOV WHERE k.OBLAST ='roman'                  (naslovi svih romana)

k.NASLOV, k.K_SIF WHERE  EXISTS kp  (EXISTS p (kp.K_SIF = k.K_SIF
     AND kp.P_SIF = p.P_SIF AND p.DRZAVA = 'Jugoslavija'))
(naslovi i šifre knjiga čiji bar jedan autor ima jugoslovensko državljanstvo).
```

2.2.2 Relacioni račun domena

Relacioni račun domena liči na relacioni račun n-torki, s tim što je promenljiva tog računa deklarisana nad domenom atributa (*domenska promenljiva*) a ne nad relacijom.

Prema Codd-ovoj definiciji relacionog računa, izraz relacionog računa domena ima oblik

$$\{x_1 x_2 \dots x_k | \, f\},$$

pri čemu važi:

1. $x_1, x_2, \dots, x_k$ su domenske promenljive;
2. f je formula relacionog računa domena čije su slobodne promenljive upravo promenljive $x_1, x_2, \dots, x_k$; pri tome se ta formula definiše na isti način kao i formula relacionog računa n-torki, osim što sada atom može imati nešto modifikovan oblik:

   a1') ako je R relacija stepena n, a $y_1, y_2, \dots, y_n$ – konstante ili domenske promenljive nad domenima odgovarajućih atributa, onda je $R(y_1 y_2 \dots y_n)$ atom; značenje ovog atoma je da vrednosti od y_i (za one y_i koji su domenske promenljive) moraju biti izabrane tako da $y_1 y_2 \dots y_n$ predstavlja n-torku relacije R;

   a2') ako su x, y – domenske promenljive, a – konstanta, a Θ – operacija poređenja, onda su $x\Theta y$ i $x\Theta a$ atomi; značenje ovih atoma je da par vrednosti x, y (odnosno vrednost x) mora biti izabran tako da je poređenje Θ tačno.

Primer 2.10 Izrazi relacionog računa domena su:

a) Naći šifru i status izdavača čiji je naziv 'Prosveta':

   $\{xy | \, (\exists z)(\mathrm{I}(x\text{'Prosveta'}yz))\}$

b) Naći imena pisaca jugoslovenskih državljana:

   $\{x | \, (\exists y)(\exists z)(\mathrm{P}(yxz\text{'Jugoslavija'}))\}$

c) Naći ime i šifru autora 'Osme ofanzive':

$\{xy|\ (\exists z)(\exists u)(\exists v)(\exists w)(\exists t)$

(K(z'Osma ofanziva'u) AND KP(zyv) AND P($yxwt$))}

Specifična sintaksa relacionog računa n-torki, opisana u [22] i prikazana u prethodnoj tački, može se modifikovati tako da obuhvati i relacioni račun domena. Formula obuhvata sada još jedan oblik "poređenja" – tzv. *uslov pripadnosti*. Naime, ako je $R(A_1, \ldots, A_n)$ relacija sa atributima $A_1, \ldots, A_n$, onda je $R(A_{i_1} : a_{i_1}, \ldots, A_{i_k} : a_{i_k})$ – uslov pripadnosti, gde je a_{i_j} ili domenska promenljiva nad domenom atributa A_{i_j} ili konstanta iz domena atributa A_{i_j}. Za n-torku $r \in R$ vrednost uslova pripadnosti je 'tačno' ako su konstante iz uslova pripadnosti jednake vrednostima odgovarajućih atributa n-torke r. Uslov pripadnosti eliminiše navođenje egzistencijalno kvantifikovanih domenskih promenljivih koje nisu od interesa za dati upit (npr. u prethodnom primeru promenljive u, v, w, t).

U izrazu relacionog računa domena, domenske promenljive navode se umesto ciljnih atributa (*ciljne domenske promenljive*), a takođe i umesto atributa u poređenju. Ostali elementi sintakse formule (i izraza) relacionog računa n-torki i relacionog računa domena su isti.

Primer 2.11 Ako su KX, PX, IX, STATUSX, IMEX, NASLOVX domenske promenljive deklarisane nad domenima atributa K_SIF, P_SIF, I_SIF, STATUS, IME, NASLOV, redom, onda izrazi relacionog računa domena iz prethodnog primera u ovoj sintaksi imaju sledeći oblik:

a) Naći šifru i status izdavača čiji je naziv 'Prosveta':

IX, STATUSX, WHERE I(I_SIF:IX, STATUS:STATUSX, NAZIV:'Prosveta');

b) Naći imena pisaca jugoslovenskih državljana:

IMEX WHERE P(IME:IMEX, DRZAVA:'Jugoslavija');

c) Naći ime i šifru autora 'Osme ofanzive':

IMEX, PX WHERE EXISTS KX

(K(K_SIF: KX, NASLOV:'Osma ofanziva') AND

KP(K_SIF:KX, P_SIF:PX) AND P(P_SIF:PX, IME:IMEX)).

Relacioni račun domena je mehanizam ekvivalentan relacionom računu n-torki, u smislu da za svaki izraz relacionog računa domena postoji izraz u relacionom računu n-torki koji ima istu vrednost, i obratno (dokaz u [64]). Osim toga, može se dokazati da je relaciona algebra ekvivalentna, u istom smislu, sa relacionim računom n-torki. Dokaz svodljivosti izraza relacione algebre na izraz relacionog računa n-torki, izraza relacionog računa n-torki na izraz relacionog računa domena, a izraza relacionog računa domena na izraz relacione algebre, može se naći, za različite

sintakse relacione algebre i relacionog računa, u [64], [48]. Otuda sledi, posebno, ekvivalentnost relacione algebre i relacionog računa (u bilo kojoj od dve varijante). Ovde će biti skiciran algoritam svoďenja izraza relacionog računa n-torki (direktno) na izraz relacione algebre, koji je Codd dao još 1972. godine ([17]).

Algoritam 1 (svoďenje izraza relacionog računa n-torki na izraz relacione algebre)

Algoritamski koraci:

1. Za svaku n-tornu promenljivu izraza relacionog računa, pretražiti odgovarajuću relaciju, uz izvoďenje operacije restrikcije ako postoji uslov restrikcije nad tom relacijom.

2. Napraviti Dekartov proizvod relacija dobijenih u koraku 1.

3. Izvršiti restrikciju Dekartovog proizvoda dobijenog u koraku 2 prema uslovu spajanja iz formule, tj. prema poreďenju različitih n-tornih promenljivih.

4. Primeniti kvantifikatore zdesna ulevo na sledeći način:

   – za kvantifikator EXISTS T, ako je T n-torna promenljiva nad relacijom R, eliminisati atribute iz R iz relacije – meďurezultata;

   – za kvantifikator FORALL T, podeliti relaciju – meďurezultat relacijom R (tj. onim njenim delom koji je dobijen posle eventualne primene restrikcije u koraku 1).

5. Projektovati dobijenu relaciju na skup atributa iz ciljne liste.

Primer 2.12 Ilustrujmo rad algoritma 1 na izdavačkoj bazi podataka. Upit "naći šifre i nazive jugoslovenskih izdavača čiji je status veći od 10, i koji su 1979. godine štampali sve knjige[6] nekog inostranog pisca" može se preformulisati u upit "naći šifru i naziv svakog jugoslovenskog izdavača čiji je status veći od 10, i za kojeg postoji inostrani pisac takav da za svaku izdatu knjigu važi sledeće: ako je autor knjige taj pisac, onda je tu knjigu 1979. godine izdao taj izdavač". Ovaj se upit može u relacionom računu n-torki formulisati sledećim izrazom (o sistematičnoj obradi implikacije "ako – onda" v. tačku 3.3.6):

[6] "Sve knjige" ovde znači – sve knjige o kojima postoji informacija u bazi, tj. primenjuje se pretpostavka o *zatvorenoj interpretaciji baze*.

```
RANGE OF   i    IS I
RANGE OF   p    IS P
RANGE OF   ki   IS KI
RANGE OF   kp   IS KP

i.I_SIF, i.NAZIV WHERE
    i.DRZAVA = 'Jugoslavija' AND i.STATUS> 10 AND
    EXISTS p (p.DRZAVA ≠ 'Jugoslavija' AND
      FORALL kp (kp.P_SIF ≠ p.P_SIF OR
        EXISTS ki (ki.GODINA = 1979 AND
                   ki.K_SIF = kp.K_SIF AND ki.I_SIF = i.I_SIF)))
```

Kada se algoritam primeni na ovaj primer, njegovi pojedini koraci daju sledeći niz međurezultata, tj. rezultat.

Posle koraka 1 pretražena je kompletna relacija KP i sledeći delovi relacija I, P, KI:

I

I_SIF	NAZIV	STATUS	DRZAVA
i1	Prosveta	30	Jugoslavija
i4	Matica srpska	30	Jugoslavija

P

P_SIF	IME	BR_NASLOVA	DRZAVA
p2	M.Benson	1	Engleska
p5	C.J.Date	1	Amerika

KI

K_SIF	I_SIF	IZDANJE	GODINA	TIRAZ
k4	i1	2	1979	10000

Posle koraka 2 dobija se relacija Rez_2 koja ima 16 atributa i $2{\times}2\ {\times}1{\times}7 = 28$ 16-torki (oznake su maksimalno skraćene zbog prostornog ograničenja):

I_SIF	NAZIV	...	P_SIF	IME	...	K_SIF	I_SIF	...	K_SIF	P_SIF	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k1	p1	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k2	p4	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k3	p1	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k4	p2	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k4	p3	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k5	p5	...
i1	Prosveta	...	p2	Benson	...	k4	i1	...	k6	p4	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
i4	Matica s.	...	p5	Date	...	k4	i1	...	k6	p4	...

Korak 3 predstavlja restrikciju dobijenog Dekartovog proizvoda po uslovu spajanja relacija I, P, KI, KP. Uslov spajanja u ovom primeru je implikacija tipa: ako su pisac (P) i autor knjige (KP) isti, onda ta knjiga treba da je izdanje (KI) izdavača I, tj. u terminima atributa n-torki,

P.P_SIF = KP.P_SIF ⇒KP.K_SIF = KI.K_SIF AND KI.I_SIF = I.I_SIF,

ili kako je zapisano u formulaciji upita:

kp.P_SIF ≠ p.P_SIF OR (ki.K_SIF = kp.K_SIF AND ki.I_SIF = i.I_SIF).

Zato se posle koraka 3 dobija relacija Rez_3, u kojoj su prisutne sve 16-torke koje zadovoljavaju prvi član disjunkcije (pisac i autor knjige nisu isti), i sve 16-torke koje zadovoljavaju drugi član disjunkcije, tj. 16-torke kod kojih su pisac i autor knjige isti a knjiga je u izdanju navedenog izdavača. Dakle, relacija Rez_3 dobija se iz relacije Rez_2 izbacivanjem samo onih vrsta koje ne zadovoljavaju ni jedan član disjunkcije, tj. za koje važi: P.P_SIF = KP.P_SIF AND (KP.K_SIF ≠ KI.K_SIF OR KI.I_SIF ≠ I.I_SIF) (pisac i autor knjige su isti, ali knjiga nije u izdanju izdavača iz te 16-torke); vrste koje se u koraku 3 izbacuju su:

(i1, Prosveta, ..., p5, Date, ..., k4, i1, ..., k5, p5, ...)
(i4, Matica S., ..., p2, Benson, ..., k4, i1, ..., k4, p2, ...)
(i4, Matica S., ..., p5, Date, ..., k4, i1, ..., k5, p5, ...)

Relacija Rez_3 ima 16 atributa i dvadeset pet 16-torki.

Posle prve primene koraka 4, koja odgovara primeni egzistencijalnog kvantifikatora na n-tornu promenljivu tabele KI (EXISTS ki), dobija se relacija REZ_4 analogna relaciji REZ_3, samo bez atributâ relacije KI:

REZ_4

I_SIF	NAZIV	...	P_SIF	IME	...	K_SIF	P_SIF	...
i1	Prosveta	...	p2	Benson	...	k1	p1	...
i1	Prosveta	...	p2	Benson	...	k2	p4	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
i4	Matica s.	...	p5	Date	...	k6	p4	...

Ova tabela ima 11 atributa i dvadeset pet 11-torki.

Posle druge primene koraka 4, koja odgovara primeni univerzalnog kvantifikatora na n-tornu promenljivu tabele KP (FORALL kp), dobija se relacija sa 8 atributa i samo jednom osmorkom. Ona predstavlja izdavača (Prosvetu) koji zadovoljava traženi uslov da je izdao sve knjige inostranog pisca, zajedno sa svim takvim piscima (u ovom slučaju samo M.Benson):

$\text{REZ}_{4'}$

I_SIF	NAZIV	ST	DRZAVA	P_SIF	IME	BR_N	DRZAVA
i1	Prosveta	30	Jugoslavija	p2	Benson	1	Engleska

Posle treće primene koraka 4, koja odgovara primeni egzistencijalnog kvantifikatora na n-tornu promenljivu tabele P (EXISTS p), dobija se relacija sa 4 atributa i jednom četvorkom:

$REZ_{4''}$

I_SIF	NAZIV	STATUS	DRZAVA
i1	Prosveta	30	Jugoslavija

Najzad, posle koraka 5 dobija se relacija sa dva (izlazna) atributa i jednim uređenim parom, koji predstavlja i jedini par vrednosti koje zadovoljavaju početni upit:

REZ_5

I_SIF	NAZIV
i1	Prosveta

Ovo je i krajnji rezultat (vrednost) polaznog izraza relacionog računa. Redosled izvršavanja odgovarajućih operacija relacione algebre rezultuje sledećim izrazom relacione algebre, dobijenim transformacijom izraza relacionog računa:

$$
\begin{array}{ll}
(((((& \text{I[DRZAVA = 'Jugoslavija' AND STATUS} > 10] \quad) \\
\times & \text{(P[DRZAVA} \neq \text{'Jugoslavija']) } \times \text{ (KI[GODINA = 1979]) } \times \text{ KP} \quad) \\
& \text{[KP.P\_SIF} \neq \text{P.P\_SIF OR} \\
& \qquad \text{(KI.K\_SIF = KP.K\_SIF AND KI.I\_SIF = I.I\_SIF)]} \quad) \\
& \text{[I.I\_SIF, NAZIV, STATUS, DRZAVA, P.P\_SIF, IME,} \\
& \qquad \text{BR\_NASLOVA, DRZAVA, KP.K\_SIF, KP.P\_SIF, R\_BROJ]} \quad) \\
\div & \text{KP} \quad) \\
& \text{[I.I\_SIF, I.NAZIV]}
\end{array}
$$

Dokaz korektnosti algoritma svođenja Codd bazira na svodljivosti logičkih operacija AND, OR, NOT u formulama relacionog računa na odgovarajuće skupovne operacije $\cap, \cup, \setminus$, kao i na analogiji koja postoji između operacije projekcije i egzistencijalnog kvantifikatora, odnosno između operacije deljenja i univerzalnog kvantifikatora.

I relaciona algebra i relacioni račun su suviše formalni jezici za krajnjeg korisnika, pa se prirodno javlja potreba za izgradnjom jednostavnijih i prirodnijih za upotrebu upitnih jezika baza podataka. Jedan od opisana dva formalizma (ili njihova kombinacija) koristi se kao osnov za izražajnije upitne jezike. Za upitni jezik zahteva se da bude *relaciono kompletan*, tj. da se u tom jeziku mogu izraziti bar svi izrazi relacionog računa. Prethodna tvrđenja ustanovljavaju relacionu kompletnost relacione algebre. Dok je relaciona algebra pogodna za izlaganje koncepata relacionog modela, relacioni račun pokazuje prednosti u implementaciji koncepata kroz relacioni upitni jezik. Relacioni račun je neproceduralan i pogodniji je (od relacione algebre) za izražavanje složenih upita jednim jedinstvenim izrazom. Ovo svojstvo čini upitni jezik zasnovan na relacionom računu pogodnijim za optimizaciju, što je veoma značajno u realizaciji upitnog jezika. Naime, optimizatori postojećih SUBP

proizvoda nisu u mogućnosti da optimizuju istovremeno veći broj izraza (tj. iskaza), pa je njihova produktivnost veća ako se više aktivnosti "upakuje" u jedan jedini izraz. To je i razlog zbog kojeg Codd preporučuje upotrebu "logičkog" jezika – relacionog računa, u odnosu na "algebarski" jezik – relacionu algebru, kao osnove za implementaciju upitnog jezika ([19]).

2.3 Pitanja i zadaci

1. Definisati sledeće pojmove:

relaciona algebra	proširena relaciona algebra
projekcija	spoljašnja unija
restrikcija	spoljašnje slobodno spajanje
relaciona unija, presek, razlika	spoljašnje prirodno spajanje
Dekartov proizvod	relacioni račun
slobodno (Θ) spajanje	relacioni račun n-torki
prirodno spajanje	n-torna promenljiva
relaciono deljenje	relacioni račun domena

2. Kako se izraz relacionog računa n-torki svodi na izraz relacione algebre? Navesti primer.

3. Dokazati komutativnost i asocijativnost operacija relacione unije, preseka i prirodnog spajanja.

4. Izraziti operacije prirodnog spajanja, relacionog preseka i deljenja pomoću operacija relacione unije, razlike, Dekartovog proizvoda, projekcije i restrikcije.

5. Navesti primere operacija relacione algebre.

6. Da li su operacije Dekartovog proizvoda i deljenja relacija inverzne? Da li definicija operacije deljenja dopušta deljenje praznom relacijom? Objasniti.

7. Izračunati rezultat relacionog izraza P$*$KP$*$KI$*$I. Po kojim atributima se izvršava svaka od operacija spajanja?

8. Objasniti razloge zbog kojih su potrebna proširenja relacione algebre. Navesti primere operacija proširene relacione algebre.

9. Kako biste u relacionom modelu sa nedostajućim vrednostima definisali funkcionalnu zavisnost?

10. Izrazima relacione algebre, relacionog računa n-torki i relacionog računa domena formulisati sledeće upite:

- Naći sve informacije o piscima.
- Naći šifre izdavača koji izdaju knjigu sa šifrom k1.
- Naći sva izdanja sa tiražem između 4000 i 8000.
- Naći parove (šifra izdavača, šifra pisca) iz iste države.
- Naći naslove knjiga jugoslovenskih pisaca.
- Naći naslove knjiga engleskih pisaca u izdanju jugoslovenskih izdavača.
- Naći parove država takve da izdavač iz prve države izdaje knjigu pisca iz druge države.
- Naći sve parove šifara knjiga koje izdaje isti izdavač.
- Naći šifre pisaca kojima je bar jednu knjigu izdao neki jugoslovenski izdavač.
- Naći šifre izdavača koji nisu izdali ni jedan roman jugoslovenskog pisca.
- Naći imena pisaca čije je sve knjige objavio izdavač sa šifrom i1.
- Naći naslove knjiga čiji je bar jedan autor jugoslovenski državljanin, ili ih je objavio jugoslovenski izdavač.
- Naći nazive izdavača koji su izdali bar sve knjige koje je izdao i izdavač sa šifrom i3.

Deo II

Relacioni upitni jezici

Relacioni upitni jezici su praktični rezultat formalnih istraživanja relacionog modela i njegovih manipulativnih formalizama. Zasnovani su na relacionoj algebri ili relacionom računu, ali poseduju znatno veće mogućnosti za izražavanje opštijih radnji nad relacijama baze. Način formulacije upita prilagoden je širokom krugu korisnika baza podataka i, posebno, korisnicima ograničenih znanja iz matematike i programiranja.

Jezici za komunikaciju sa bazom podataka (upitni jezici u širem smislu) mogu se klasifikovati u jezike dobijene nadgradnjom programskih jezika opšte namene, jezike izgrađene po modelu relacionog računa (n-torki ili domena), jezike zasnovane na relacionoj algebri, i SQL-olike jezike, koji uključuju konstrukcije kako relacione algebre, tako i relacionog računa.

Najveći broj implementacija relacionih upitnih jezika 80-tih godina 20. veka bio je zasnovan na upitnom jeziku SQL (Structured Query Language), proizvoda istraživačke IBM laboratorije u San Hozeu (San Jose) u Kaliforniji. Bez obzira na izvesne nedostatke i odstupanja od principa relacionog modela, aktuelni standard relacionog upitnog jezika je takoďe SQL upitni jezik.

Gotovo svi komercijalni RSUBP proizvodi podržavaju i SQL jezik. Meďu onima koji su razvijani kao SQL sistemi značajniji su, na primer, SQL/DS, DB2, MRS, Oracle, AIM/RDB, INFORMIX.

Većina relacionih upitnih jezika ima svoju *interaktivnu* i *aplikativnu* varijantu. Prva omogućuje postavljanje *ad hoc* upita nad relacionom bazom, dok druga omogućuje ugradnju upitnog jezika u matični programski jezik i razvoj složenih aplikacija nad relacionom bazom. Princip dualnosti, koji se najčešće poštuje, obezbeďuje da se sve radnje interaktivne varijante upitnog jezika mogu izraziti i u aplikativnoj (i da obratno, naravno, ne važi).

Deo **II** sastoji se od 5 glava:

- Glava 3, **Upitni jezik SQL**, prikazuje evoluciju SQL-a i njegove mogućnosti u definisanju podataka, manipulisanju podacima i definisanju uslova integriteta, kako u interaktivnoj tako i u aplikativnoj varijanti. Ova glava uključuje i mogućnosti SQL-a u definisanju autorizacije, dodeli i oduzimanju prava korisnicima za izvršavanje odreďenih operacija (i posebno SQL iskaza) nad odreďenim podacima u bazi podataka (ili bazom podataka u celini). Izloženi su bitni aspekti autorizacije i kontrole prava korisnika kao jedne od funkcija SUBP.

- Glava 4, **Pogledi**, odnosi se na spoljašnji nivo ANSI arhitekture sistema baza podataka. Pogledi se predstavljaju prvo u obliku u kome je njihovo definisanje i manipulisanje njima podržano SQL-om, a zatim se izlaže jedan novi pristup pogledima koji na dosledniji način tretira ovaj koncept.

- U glavi 5, **Standardi SQL-a**, izlaže se evolucija standardâ SQL-a, i posebno specifičnosti aktuelnog standarda SQL2.

- Glava 6, **Upitni jezik QUEL**, sadrži prikaz osnovnih elemenata relacionog upitnog jezika QUEL. Mada daleko manje prisutan od SQL-a, ovaj jezik predstavlja, po mišljenju većine stručnjaka, superiorniji upitni jezik od SQL-a po nizu svojstava.

- Najzad, glava 7, **Optimizacija upita**, posvećena je strategijama i tehnikama optimizovanja izvršavanja upita u relacionim upitnim jezicima.

3

Upitni jezik SQL

3.1 Istorijat

Upitni jezik SQL predstavlja završnu fazu evolucije klase jezika razvijene u IBM istraživačkoj laboratoriji u San Hozeu, Kalifornija. Ovu evoluciju upitnih jezika karakteriše odstupanje od jezika relacione algebre i približavanje jezicima relacionog računa. Svi jezici ove klase su skupovno orijentisani, u značajnoj meri neproceduralni, i omogućuju zadavanje onoga *šta* se želi dobiti, bez preciznog navođenja načina *kako* će se željeni rezultat dobiti.

Prvi od jezika u ovoj klasi bio je upitni jezik SQUARE (Specifying QUeries As Relational Expressions) ([9]), implementiran u prvom potpunom prototipskom relacionom sistemu SYSTEM R ([4]). Sâm SQUARE ima mogućnosti koje ne poznaje relaciona algebra, kao što je poređenje skupova inkluzijom ili imenovanje n-torki relacije (*slobodne promenljive*). Jezik je relaciono kompletan i ima prepoznatljivu sintaksu za sve operacije relacione algebre. Osim toga, SQUARE ima agregatne funkcije COUNT, AVG, MIN, MAX i SUM kojima se skup vrednosti nekog atributa relacije preslikava u numeričku vrednost ili vrednost iz odgovarajućeg domena. Tako, u jeziku SQUARE, zapis

$$_{A_1,\ldots,A_n}R_{B_1,\ldots,B_m}(\Theta_1 b_1,\ldots,\Theta_m b_m)$$

označava projekciju na atribute $A_1,\ldots,A_n$ restrikcije relacije R po logičkom izrazu $B_1\Theta_1 b_1$ AND ... AND $B_m\Theta_m b_m$ ($A_1,\ldots,A_n,B_1,\ldots,B_m$ su neki od atributa relacije R); kompozicija ovakvih operacija (u oznaci $\circ$) omogućuje realizaciju operacije spajanja, odnosno osnovnih operacija relacione algebre. Na primer, zapis

$$_{A,B,C}R_{C\circ C}S_D(=d_1)$$

označava projekciju na atribute A,B,C rezultata prirodnog spajanja relacije R i restrikcije relacije S po uslovu $D=d_1$.

S druge strane, zapis

$$(t_1)_{\alpha_1} \in R_1,\ \ldots,\ (t_k)_{\alpha_k} \in R_k\ :\ \psi$$

u jeziku SQUARE može se smatrati analogonom izraza relacionog računa n-torki i označava projekciju na atribute α_i torki t_i relacijâ R_i $(1 \leq i \leq k)$, koje zadovoljavaju logički uslov ψ. Promenljive t_i su *slobodne promenljive*, a uslov ψ uključuje, kao operande, konstante, promenljive t_i, ili neke projekcije tih promenljivih. Ovaj zapis ujedno predstavlja okvir za istovremenu obradu (u jednom upitu) većeg broja relacija.

Neprivlačnu matematičku sintaksu jezika SQUARE trebalo je da popravi njegov sledbenik SEQUEL (Structured English QUEry Language) ([11]). "English" (engleski) u njegovom imenu odnosi se na uvoďenje rezervisanih reči engleskog jezika za označavanje uloge koju pojedini atributi relacije imaju u upitu, dok "structured" (struktuirani) označava postojanje blokova od kojih se gradi (struktuira) upit, umetanjem jednog u drugi. Tako prvi SQUARE zapis u SEQUEL-u ima oblik

SELECT $A_1, \ldots, A_n$
FROM R
WHERE $B_1 \Theta_1 b_1$ AND ... AND $B_m \Theta_m b_m$.

Odgovarajuća sintaksa za slobodnu promenljivu t relacije R u SEQUEL-u je

SELECT ... FROM R t WHERE ... .

Operisanje većim brojem relacija postiže se navoďenjem imena tih relacija u FROM liniji, npr.

SELECT $*$ FROM R, S.

I SEQUEL je relaciono kompletan jezik koji podržava iste agregatne funkcije kao i SQUARE.

Upitni jezik SEQUEL je kasnije, uz izvesne izmene (preko jezika SEQUEL/2, implementiranog 1976. [12], i *de facto* podržanog u poslednjoj verziji System R), i iz pretežno pravnih razloga, preimenovan u SQL (Structured Query Language). Pod tim imenom je doživeo niz verzija i standarda.

Do 1986. godine nije bilo standarda upitnog jezika baza podataka, ni internacionalnog, ni domaćeg. Prvi takav standard objavila je komisija X3H2 za baze podataka Američkog instituta za standarde. Bio je to ANSI SQL/86, koji se pojavio kao reakcija na postojeće stanje na tržištu ("reakcioni" standard, [43]), i koji je sadržavao zajedničke komponente raznih realizovanih verzija SQL jezika. Ovaj standard bio je praćen i odgovarajućim ISO standardom. Standard SQL/86 obezbeďivao je samo tabelarni model podataka, nezavisan od jezika, definiciju sheme, poglede i kursore za sumeďe (engl. interface) ka slogovno orijentisanim programskim jezicima.

Standard SQL-a koji je objavljen 1989, SQL/89, bio je široko prihvaćen na tržištu. On je funkcionalno kompletniji od SQL/86, obezbeďuje referencijalni integritet i ugnježďenje SQL-a u programske jezike Ada, C, COBOL, FORTRAN, Pascal i PL/I.

Aktuelni standard SQL jezika je SQL/92 (SQL2, SQL-92) ili, pod punim nazivom, "Međunarodni standardni jezik baza podataka (1992)" ("International Standard Database Language SQL (1992)", [39]). SQL2 je anticipatorni standard, što označava njegovu pretenziju da vodi i usmerava tržište. Uključuje, između ostalih, i sledeća svojstva:

- jezik za definisanje podataka
- dinamički SQL kroz PREPARE i EXECUTE iskaze ([24])
- spoljašnje spajanje
- kaskadno ažuriranje i brisanje
- privremene tabele
- skupovne operacije unije, preseka i razlike
- definiciju domena u shemi
- nove ugrađene tipove podataka
- nivoe konzistentnosti transakcija
- odloženu proveru uslova integriteta
- SQL dijagnostiku, itd.

SQL2 standardu posvećena je glava 5.

Za 1995. godinu planirano je bilo objavljivanje novog standarda SQL-a, SQL3, kompatibilnog sa SQL2, ali obogaćenog objektnim aspektom podataka (o objektnom modelu podataka biće reči u Dodatku), izračunljivo kompletnog (sa kontrolnim strukturama, bez potrebe ugnježdenja u programske jezike opšte namene). O statusu pojedinih komponenti SQL3 standarda v. glavu 5.

U preostalom delu ove glave biće detaljno opisana sintaksa i semantika SQL-a. Radi konkretizacije jezika i mogućnosti programiranja na njemu (bilo interaktivnog ili aplikativnog), izlaganje će se odnositi na jednu implementaciju SQL-a, za RSUBP DB2 UDB (Universal Data Base) Verzija 5.2 (pod operativnim sistemima Windows 95, Windows NT, OS/2). Bez obzira na to, najveći deo razmatranja odnosi se na SQL uopšte, nezavisno od implementacije.

3.2 Definisanje podataka

Osnovni iskazi definisanja podataka u SQL-u su iskazi za kreiranje i iskazi za uklanjanje baznih tabela, indeksa i pogleda, kao i iskazi za izmenu strukture baznih tabela i pogleda. Pored ovih objekata, novije verzije SQL-a (uključujući i standard SQL2, v. 5.1), podržavaju koncept *sheme*, različit od koncepta relacijske sheme kako je definisana relacionim modelom – v. 1.2, a sličan konceptu sheme relacione baze podataka, kao i iskaze za definisanje, izmenu i uklanjanje sheme.

Mogućnost izvršenja ovih iskaza u bilo kom trenutku, a ne samo na početku razvoja aplikacije, čini SQL sisteme vrlo fleksibilnim; proces projektovanja baze podataka je oslobođen potrebe da celokupni posao logičkog i fizičkog projektovanja (definisanja tabela i pristupnih puteva kao što su indeksi) mora biti urađen odjednom i perfektno.

S obzirom na specifičnost virtuelnih tabela – pogleda, koje fizički ne postoje, već postoji samo njihova definicija sačuvana u sistemskom katalogu (v. odeljak 7.1 – Katalog), definisanju pogleda, kao i ostalim operacijama nad pogledima biće posvećena glava 4. Zato će u ovom odeljku biti opširnije razmatrani samo iskazi za definisanje shema, baznih tabela i indeksa.

3.2.1 Shema

Shema je kolekcija imenovanih objekata baze podataka, koja obezbeđuje logičku klasifikaciju tih objekata. Neki od objekata baze podataka koje shema može da sadrži jesu tabele, pogledi, trigeri, paketi, itd.

I sama shema se može posmatrati kao objekat baze podataka. Shema se kreira izvršavanjem CREATE SCHEMA iskaza, kojim se mogu kreirati i drugi objekti u toj shemi i dodeliti korisnicima prava pristupa tim objektima (v. 3.6). Iskaz za kreiranje sheme ima sledeći oblik:

```
 CREATE SCHEMA  ime-sheme [ AUTHORIZATION  ime-vlasnika-sheme]
[ CREATE TABLE  iskaz {, CREATE TABLE  iskaz }]|
[ CREATE VIEW  iskaz {, CREATE VIEW  iskaz }]  |
[ CREATE INDEX  iskaz {, CREATE INDEX  iskaz}] |
[ GRANT  iskaz {, GRANT  iskaz } ]
```

Ako se AUTHORIZATION opcija ne navede, podrazumevani vlasnik kreirane sheme je korisnik koji izvršava CREATE SCHEMA iskaz.

Pri kreiranju nekog objekta izvan CREATE SCHEMA iskaza, ime objekta se može kvalifikovati imenom sheme. Tada objekat ima dvodelno ime, pri čemu je prvi deo imena – ime sheme kojoj se objekat dodeljuje. Ako se ime sheme ne navede, objekat se dodeljuje podrazumevanoj (engl. default) shemi čije je ime – identifikacija korisnika koji izvršava iskaz kreiranja objekta. Slično, ako se pri navođenju objekta (npr. u SELECT iskazu) izostavi ime sheme, podrazumeva se shema imenovana imenom korisnika koji taj iskaz zadaje. Ovo svojstvo biće korišćeno u svim primerima SQL-a.

Shema se može i ukloniti iskazom oblika

```
DROP SCHEMA  ime-sheme RESTRICT
```

pri čemu pravilo RESTRICT zabranjuje uklanjanje sheme ako ona sadrži bilo koji objekat baze podataka.

3.2.2 Bazne tabele

Iskaz kreiranja. Osnovni oblik iskaza za kreiranje bazne tabele je

```
CREATE TABLE  ime-bazne-tabele
      ( def-kolone {, def-kolone}
      [, def-prim-ključa]
      [, def-str-ključa {, def-str-ključa}]
      [, uslov-ograničenja {, uslov-ograničenja})
      [ drugi-parametri]
```

Ime bazne tabele je identifikator (maksimalne dužine 18 znakova u DB2), čiji je prvi znak slovo a ostali znaci su slova, cifre ili podvlaka ('_'). Ovako definisano ime je *nekvalifikovano*, a može se kvalifikovati navođenjem lokacije (sheme) i vlasnika tabele.

Definicija kolone *def-kolone* ima oblik:

ime-kolone *tip-podataka* `[NOT NULL] [[WITH] DEFAULT [` *vrednost*`]]`

gde *tip-podataka* može biti:[1]

- INTEGER ili INT, 32-bitni ceo broj;
- SMALLINT, 16-bitni ceo broj;
- BIGINT, 64-bitni ceo borj;
- DECIMAL(p,q) ili DEC(p,q), decimalni broj; p je ukupni broj cifara decimalnog broja, dok je q broj cifara razlomljenog dela ($1 \leq p \leq 31$, $0 \leq q \leq p$);
- FLOAT(p), broj u pokretnom zarezu, jednostruke ($1 \leq p \leq 24$) ili dvostruke ($25 \leq p \leq 53$) tačnosti;
- CHARACTER(m) ili CHAR(m), niska znakova fiksne dužine m ($1 \leq m \leq 254$).

Mogu se koristiti sledeće skraćenice:

DECIMAL(p)	$\equiv$	DECIMAL(p,0),
DECIMAL	$\equiv$	DECIMAL(5),
FLOAT	$\equiv$	FLOAT(53),
DOUBLE [PRECISION]	$\equiv$	FLOAT,
REAL	$\equiv$	FLOAT(24),
CHAR	$\equiv$	CHAR(1),
CHARACTER	$\equiv$	CHAR(1).

[1] Skup tipova podataka sistema DB2 je bogatiji i uključuje i tipove datuma, vremena (DATE, TIME), niski znakova promenljive dužine (VARCHAR, LONG VARCHAR), tipove grafičkih niski (GRAPHIC, VAR GRAPHIC), binarnih i znakovnih velikih objekata (BLOB, CLOB, DBCLOB), korisnički definisane tipove bazirane na ugrađenim SQL tipovima ([37]).

Opcija "[NOT NULL] [[WITH] DEFAULT [*vrednost*]]" odnosi se na kolonu koja ne dozvoljava nedostajuće – NULL vrednosti, tj. na mogućnost supstitucije nedostajuće vrednosti navedenom (DEFAULT) vrednošću iz domena (na primer, *vrednost* može biti konstanta 0 ili 1 za slučaj kolone tipa INTEGER). Ako se *vrednost* u opciji DEFAULT ne navede, nedostajuća vrednost se sistemski zamenjuje podrazumevanom (DEFAULT) vrednošću za odgovarajući domen (na primer, belina tj. blank za tip CHAR, 0 za tip INTEGER, itd). Opcija DEFAULT može da se koristi i samostalno (bez NOT NULL) i to u slučaju kolone koja dopušta NULL vrednosti; pri tome se kao podrazumevana vrednost može navesti i NULL.

Definicije primarnog i stranog ključa predstavljaju posebne oblike uslova ograničenja, pa im je i sintaksa prilagođena opštoj sintaksi ovih uslova.

Definicija primarnog ključa je oblika

```
[CONSTRAINT ime] PRIMARY KEY ( ime-kolone {, ime-kolone})
```

pri čemu svaka kolona čije je ime navedeno u ovoj definiciji mora biti eksplicitno definisana kao NOT NULL.

Primarni ključ koji se sastoji od jedne kolone može se zadati i u samoj definiciji te kolone, navođenjem opcije [CONSTRAINT ime] PRIMARY KEY.

U sistemu DB2, tabela ne mora imati definisan primarni ključ (što predstavlja odstupanje od principa relacionog modela), ali je veoma poželjno da ga ima jer on obezbeđuje jedinstvenost vrsta.

Definicija stranog ključa je oblika

```
 [CONSTRAINT ime] FOREIGN KEY ( kolona {, kolona})
REFERENCES odnosna-tabela
[ON DELETE efekat] [ON UPDATE efekat]
```

gde je *odnosna-tabela* – bazna tabela u kojoj ovde navedene kolone čine primarni ključ. Tabela u kojoj je definisan strani ključ je *zavisna* od odnosne tabele. Ne zahteva se postojanje indeksa po kolonama stranog ključa, ali je to često pogodno iz razloga performansi (npr. kod operacije spajanja nad primarnim i stranim ključem ili kod održavanja referencijalnog integriteta).

Strani ključ koji se sastoji od jedne kolone može se zadati i u samoj definiciji te kolone, navođenjem kompletne definicije stranog ključa.

Pri brisanju (DELETE) vrsta iz odnosne tabele, može se primeniti jedno od sledećih pravila brisanja (tj. *efekat* u ON DELETE opciji može biti sledeći):

- NO ACTION: omogućuje brisanje vrste u odnosnoj tabeli samo ako ne postoji vrsta u zavisnoj tabeli koja zavisi od vrste koja se briše (tj. ako ne postoji vrsta u zavisnoj tabeli čija je vrednost stranog ključa jednaka vrednosti primarnog ključa vrste koja se briše). U suprotnom, sistem odbija da izvrši brisanje iz odnosne tabele i javlja grešku. Ovo je i podrazumevani efekat koji se primenjuje ukoliko se ništa ne navede.

- RESTRICT: efekat je isti kao NO ACTION, razlikuju se samo po vremenu kada se primenjuju. Naime, u slučaju da se operacija brisanja reflektuje na veći broj tabela sa različitim efektima pri brisanju, RESTRICT efekat primenjuje se *pre* svih drugih efekata, a NO ACTION – *posle* svih drugih efekata. Zato u nekim (retkim) situacijama ovi efekti mogu da dovedu do različitih rezultata. [2]

- SET NULL: omogućuje brisanje vrste u odnosnoj tabeli i postavljanje NULL vrednosti u kolone stranog ključa zavisne tabele koje to dopuštaju (nisu NOT NULL); NULL vrednosti se postavljaju u vrste zavisne tabele koje zavise (u smislu opisanom uz prethodni efekat) od vrste koja se briše. Aktivnost postavljanja NULL vrednosti ograničena je na neposredno zavisne tabele (ne prenosi se dalje). Efekat SET NULL se ne može primeniti ako je neka od kolona stranog ključa NOT NULL kolona.

- CASCADE: omogućuje brisanje naznačene vrste odnosne tabele i brisanje svih zavisnih vrsta zavisne tabele, uz poštovanje pravila brisanja njihovih zavisnih tabela.

Pri ažuriranju (UPDATE) vrsta odnosne tabele, i to na kolonama koje ulaze u primarni ključ, dopušteno pravilo je samo NO ACTION (podrazumevano) ili RESTRICT. To znači da se vrednost primarnog ključa p_1 odnosne tabele može zameniti drugom vrednošću samo ako u zavisnoj tabeli ni jedna vrsta nema na odgovarajućem stranom ključu vrednost p_1.

Uslov ograničenja je logički izraz čiji su argumenti kolone tabele, konstante i izrazi nad njima, i predstavlja ograničenje nad podacima koji se unose u tabelu. Sintaksa opcije zadavanja uslova ograničenja je

`CONSTRAINT` *ime* `CHECK (` *uslov*`)`

Uslov ograničenja koji se odnosi samo na jednu kolonu tabele može se zadati i u definiciji te kolone, istom sintaksom.

Provera uslova ograničenja kao i referencijalnog integriteta može biti, posebnim iskazom (SET CONSTRAINT), odložena (deaktivirana) i aktivirana. Deaktiviranje provere uslova ograničenja posebno je korisno kod masovnog unosa podataka u bazu.

[2] Na primer, neka tabela T3 ima dva strana ključa koji se odnose na tabele T1 i T2, i neka je pravilo pri brisanju za tabelu T2 – CASCADE. Neka je, dalje, definisan pogled (v. glavu 4 – Pogledi) V1 sledećim iskazom:

```
CREATE VIEW V1 AS SELECT * FROM T1 UNION ALL SELECT * FROM T2
```

Tada se rezultat izvršavanja iskaza

```
DELETE FROM V1
```

razlikuje u slučajevima kada je pravilo pri brisanju iz tabele T1 RESTRICT odnosno NO ACTION.

Ostali elementi iskaza za kreiranje tabele, *drugi-parametri*, odnose se na elemente fizičke reprezentacije bazne tabele (u sistemu DB2 takvi elementi su prostor tabela u koji se tabela smešta i njegove karakteristike, prostor tabela u koji će se smestiti svaki budući indeks nad tabelom, kao i prostor tabela u koji se smeštaju veliki objekti, itd; v. tačku 18.1.5 – Fizička reprezentacija podataka u sistemu DB2).

Primer 3.1 Kreirati tabele K, I, KI sa strukturom opisanom u odeljku 1.2.

```
CREATE  TABLE  K
 (K_SIF           CHAR (5)   NOT NULL,
  NASLOV          CHAR (50)  NOT NULL WITH DEFAULT,
  OBLAST          CHAR (20),
  PRIMARY KEY     (K_SIF))
CREATE  TABLE  I
 (I_SIF           CHAR (6)   NOT NULL,
  NAZIV           CHAR (20)  NOT NULL WITH DEFAULT,
  STATUS          SMALLINT,
  DRZAVA          CHAR (20)  NOT NULL WITH DEFAULT,
  PRIMARY KEY     (I_SIF))
CREATE  TABLE   KI
 (K_SIF           CHAR(5)    NOT NULL,
  I_SIF           CHAR(6)    NOT NULL,
  IZDANJE         SMALLINT   NOT NULL,
  GODINA          SMALLINT,
  TIRAZ           INTEGER,
  PRIMARY KEY     (K_SIF, I_SIF, IZDANJE),
  FOREIGN KEY KSK (K_SIF) REFERENCES K ON DELETE RESTRICT
ON UPDATE RESTRICT,
  FOREIGN KEY ISK (I_SIF) REFERENCES I ON DELETE CASCADE)
```

Efekat ovih iskaza je kreiranje novih, praznih baznih tabela koje se zovu xyz.K, xyz.I, xyz.KI, redom; xyz je ime sheme u kojoj su kreirane ove tabele, ili ime po kome je sistemu poznat korisnik koji zadaje CREATE TABLE iskaz. U okviru sheme xyz (tj. korisnik xyz) može se obraćati ovim tabelama kvalifikovanim imenom xyz.K (xyz.I, xyz.KI) ili nekvalifikovanim imenom K (I, KI), dok se iz ostalih shema (tj. od strane ostalih korisnika) mora upotrebiti kvalifikovano ime.

Pokušaj da se briše vrsta u tabeli K sa vrednošću k u koloni K_SIF biće uspešan samo kada u tabeli KI ne postoji vrsta sa istom vrednošću k u koloni K_SIF, jer strani ključ K_SIF u tabeli KI ima pravilo brisanja DELETE RESTRICT. Slično, pokušaj da se vrednost k u koloni K_SIF neke vrste promeni biće uspešan samo ako u tabeli KI ne postoji vrsta sa vrednošću k u koloni K_SIF (ON UPDATE RESTRICT).

Brisanje vrste u tabeli I sa vrednošću i atributa I_SIF proizvodi brisanje svih vrsta u tabeli KI sa vrednošću i kolone I_SIF, jer strani ključ I_SIF u tabeli KI

ima pravilo brisanja DELETE CASCADE. Ovde se brisanje završava jer nema više zavisnih tabela (tj. vrsta). Ako bi postojala tabela zavisna od tabele KI, tj. od njene kolone I_SIF, i na nju bi se primenilo pravilo brisanja definisano za njen strani ključ. Ovakvo *kaskadno* brisanje se završava i u slučaju da se dođe do zavisne tabele za koju je definisan strani ključ sa pravilom brisanja DELETE SET NULL. Za promenu vrednosti vrste tabele I na koloni I_SIF važi isto što i za tabelu K i kolonu K_SIF (podrazumevano pravilo ažuriranja NO ACTION ima isto dejstvo kao i RESTRICT).

Iskaz izmene. Postojeća bazna tabela može se u svako doba promeniti dodavanjem novih kolona, dodavanjem ili uklanjanjem primarnog, stranog ključa, ili uslova ograničenja u opštem slučaju, ili promenom fizičkih parametara tabele. Iskaz promene bazne tabele ima oblik

```
ALTER TABLE   bazna-tabela
      ADD  ime-kolone tip-podataka [NOT NULL [WITH] DEFAULT [ vrednost]]
    | ADD  def-prim-ključa
    | ADD  def-str-ključa
    | ADD  uslov-ograničenja
    | DROP  def-prim-ključa
    | DROP  def-str-ključa
    | DROP  uslov-ograničenja
    |  drugi parametri
```

pri čemu svi uvedeni identifikatori (npr. *vrednost, def-prim-ključa*, itd.), imaju isto značenje kao i u CREATE TABLE iskazu. Posebno, za novu dodatu kolonu važe iste mogućnosti kao i za definisanu kolonu u CREATE TABLE iskazu. Na primer,

```
    ALTER TABLE  KI
ADD CENA INTEGER
```

Ovaj iskaz dodaje kolonu CENA tabeli KI. Sve postojeće vrste tabele KI proširuju se šestom kolonom koja uzima vrednost NULL (ili 0 u slučaju opcije NOT NULL WITH DEFAULT). Opcija NOT NULL (bez kvalifikacije WITH DEFAULT) u ALTER iskazu nije dozvoljena.

Primarni ključ se može dodati ako tabela već nema definisan primarni ključ i ako sve kolone koje se uključuju u primarni ključ imaju svojstvo NOT NULL.

Pri uklanjanju primarnog ključa (DROP PRIMARY KEY) definicije stranih ključeva koje su se prethodno odnosile na uklonjeni primarni ključ ukidaju se automatski.

Iskazom ALTER TABLE ne može se menjati tip podataka postojećih kolona (osim dužine VARCHAR kolone), niti se mogu uklanjati postojeće kolone.

Ako jedan ALTER TABLE iskaz sadrži više radnji dodavanja, uklanjanja (kolona, primarnog, stranog ključa, itd.) ili promene tabele, prvo se izvršava radnja dodavanja kolona, a ostale radnje se izvršavaju u redosledu kojim su navedene.

Iskaz uklanjanja. Postojeća bazna tabela može se ukloniti SQL iskazom

`DROP TABLE` *bazna-tabela*

Izvršavanjem ovog iskaza, opis te bazne tabele uklanja se iz kataloga, a svi indeksi, pogledi, uslovi ograničenja uključujući primarni i strane ključeve, definisani nad tom baznom tabelom, kao i sve definicije stranih ključeva koje se odnose na tu tabelu – automatski se uklanjaju. Pri tome se pravila brisanja za strane ključeve ne primenjuju (npr. uklanjanje tabele I ne proizvodi brisanje vrsta iz tabele KI). DROP iskaz može se primentiti i na druge objekte – indekse, poglede, prostore tabela, uslove ograničenja, itd.

3.2.3 Indeksi

Indeks se može posmatrati kao uređeni skup pokazivača na vrste bazne tabele, fizički odvojen od podataka u tabeli. Svaki indeks baziran je na vrednostima podataka jedne ili više kolona tabele.

Indeks je veoma koristan mehanizam koji ubrzava pristup vrstama tabele. Kao što je u pronalaženju nekog pojma u knjizi koristan indeks, koji eliminiše potrebu za prelistavanjem knjige stranu po stranu, tako je i u pretraživanju vrsta tabele po nekom uslovu mnogo efikasnije koristiti indeks po koloni koja učestvuje u uslovu, nego pretraživati tabelu sekvencijalno, vrstu po vrstu. Tako, na primer, ako nad tabelom K postoji indeks po koloni OBLAST, pristup knjigama po uslovu da je knjiga – roman, mnogo je brži nego ako takav indeks ne postoji.

Izvršavanjem iskaza za kreiranje indeksa, RSUBP gradi odgovarajuću strukturu podataka i održava je automatski. O načinu organizovanja, izgradnje i korišćenja indeksa kao metode pristupa vrstama tabele, biće opširno reči u glavi 12. U ovoj tački biće naveden samo način na koji SQL podržava rad sa indeksima.

Iskaz kreiranja. Nad indeksima se mogu izvršavati samo SQL iskazi za definisanje, tj. iskazi za kreiranje i uklanjanje indeksa.

Iskaz kreiranja indeksa je oblika

`CREATE [UNIQUE] INDEX` *ime-indeksa* `ON` *bazna-tabela*
`(` *ime-kolone* `[redosled]` {`,` *ime-kolone* `[redosled]`}`)`
`[` *drugi-parametri*`]`

Opcija UNIQUE precizira da dve razne vrste u baznoj tabeli nad kojom se kreira indeks ne mogu uzeti istovremeno istu vrednost na indeksnoj koloni ili njihovoj kombinaciji. Redosled uz svaku kolonu – komponentu indeksa je ili ASC (rastući) ili DESC (opadajući); ASC se podrazumeva ako se redosled ne navede. Nizanje kolona sleva na desno odgovara glavnom-ka-sporednom redosledu kolona, pri čemu su vrste indeksa uređene po zadatom redosledu (ASC, DESC) vrednosti navedenih kolona. Uz izvesna ograničenja, nad svakom kolonom se može izgraditi indeks (npr. DB2 sistem ne podržava indekse nad kolonama "dugačkih" – LONG – tipova kao

što su dugački niskovni ili grafički tip promenljive dužine, veliki binarni objekti itd).

Drugi-parametri odnose se na fizičke karakteristike indeksa, npr. prostor tabela u koji se smešta indeks, ili da li je indeks grupišući ili negrupišući (v. odeljak 12).

Jedna od komponenti RSUBP-a, upravljač podacima (engl. data manager), automatski održava jednom kreirani indeks, tako da on odslikava ažuriranja bazne tabele, sve dok se indeks ne ukloni.

U iskazu

```
CREATE INDEX XKIC ON KI (CENA)
```

opcija UNIQUE se ne navodi jer veći broj izdanja može imati istu cenu.

Iskaz uklanjanja. Iskaz za uklanjanje indeksa ima oblik

```
DROP INDEX  ime-indeksa
```

Izvršavanjem ovog iskaza opis indeksa se uklanja iz kataloga. U sistemu DB2, svaki pristupni paket (v. refsec:klijserv) koji koristi takav indeks označava se nevažećim i izgrađuje se ponovo pre sledećeg izvršenja.

3.2.4 Pogledi

Pogledi su virtuelne tabele koje nisu predstavljene fizičkim podacima, već je njihova definicija (u terminima drugih tabela) zapamćena u katalogu.

Iskaz za kreiranje pogleda u SQL-u je oblika:

```
   CREATE VIEW   ime-pogleda [( kolona {, kolona})]
AS  puni upitni blok
[WITH CHECK OPTION]
```

Puni upitni blok pripada manipulativnom delu SQL-a (v. tačku 3.3.9), i zato će pogledi biti prikazani detaljnije u posebnoj glavi o pogledima. CHECK opcija se odnosi na primenu ograničenja navedenih u logičkom izrazu WHERE linije upitnog bloka, pri unošenju i izmeni podataka u pogledu. Na primer,

```
   CREATE VIEW  JUGIZD
AS SELECT I_SIF, NAZIV
   FROM   I
   WHERE  DRZAVA='Jugoslavija'
```

Kada se izvrši ovaj CREATE iskaz, podupit SELECT-FROM-WHERE se ne izvršava, već se samo definicija upita sačuva u katalogu.

Pogled JUGIZD je "prozor", i to dinamički, na tabelu I. Izmene nad pogledom JUGIZD automatski se izvršavaju nad tabelom I, a izmene nad tabelom I

automatski se odražavaju ("vide") na pogled JUGIZD (ako zadovoljavaju uslove koji definišu taj pogled).

Korisnik može ali i ne mora biti svestan da pogled nije bazna tabela, jer se nad pogledom mogu postavljati upiti (gotovo svi) kao i nad baznom tabelom.

Iskaz za uklanjanje pogleda ima oblik

```
DROP VIEW  ime-pogleda
```

Na primer

```
DROP VIEW JUGIZD
```

Definiciju pogleda nije moguće menjati.[3]

3.3 Manipulisanje podacima

SQL podržava četiri manipulativna iskaza:

- SELECT – pretraživanje
- INSERT – umetanje
- UPDATE – ažuriranje
- DELETE – brisanje

U ovom odeljku biće, navedenim redosledom, postupno izložena sintaksa i semantika ovih iskaza.

3.3.1 Pretraživanje

Osnovni oblik iskaza pretraživanja u SQL-u sastoji se od tzv. *jednostavnog upitnog bloka* oblika [4]

```
SELECT  lista-kolona
FROM    ime-tabele
WHERE   logički-izraz
```

Lista-kolona je spisak kolona iz tabele *ime-tabele*. *Logički-izraz* uključuje poređenja atributa, konstanti i izraza (odgovarajućeg tipa) u kojima atributi i konstante učestvuju, kao i ispitivanje pripadnosti vrednosti takvih izraza zadatom skupu. Pripadne relacijske operacije su =, <>, <, <=, >, >=, IN, BETWEEN, LIKE, IS

[3] SQL sistema DB2 ima iskaz ALTER VIEW ali se on odnosi samo na jednu specifičnu karakteristiku postojećih kolona pogleda.

[4] Upitni blok ovog oblika osnovna je komponenta i ostalih manipulativnih iskaza; isto važi i za tzv. *puni upitni blok* (v. 3.3.9.

NULL (NOT IN, NOT BETWEEN, NOT LIKE, IS NOT NULL); od kvantifikatora dopušten je egzistencijalni kvantifikator. Moguća struktura logičkog izraza biće prikazana primerima koji slede. Svi primeri odnose se na tabele izdavačke baze podataka opisane u odeljku 1.2 (o strukturnom delu relacionog modela).

Rezultat operacije koju definiše SELECT upitni blok je tabela koja je izvedena iz tabele navedene u FROM liniji, pri čemu su njene kolone određene listom kolona a vrste – tačnošću logičkog izraza: samo one vrste tabele iz FROM linije za koje logički izraz ima vrednost 'tačno' pojavljuju se u rezultujućoj tabeli.

Primer 3.2 Naći šifre i statuse izdavača iz Amerike.

```
SELECT I_SIF, STATUS
FROM   I
WHERE  DRZAVA='Amerika'
```

Rezultat:

I_SIF	STATUS
i2	20

Imena kolona mogu se (a nekad i moraju, kao što će biti pokazano) kvalifikovati imenima tabela iz kojih potiču, pa se prethodni primer ekvivalentno zapisuje kao

```
SELECT I.I_SIF, I.STATUS
FROM   I
WHERE  I.DRZAVA='Amerika'
```

SELECT iskaz ima mnogo opštiju strukturu i izražajne mogućnosti. One će u sledećim tačkama biti uvođene redom, od svojstava jednostavnog upitnog bloka (tačke 3.3.2 do 3.3.8), preko punog upitnog bloka (tačka 3.3.9) do opšte strukture SELECT iskaza (3.3.10).

3.3.2 Jednorelacioni upiti

Dve osnovne klase jednorelacionih upita (upita nad jednom tabelom) jesu upiti *nekvalifikovanog* pretraživanja, u kojima nema WHERE linije, pa se sve vrste tabele kandiduju za rezultat, i upiti *kvalifikovanog* pretraživanja, koji sadrže WHERE liniju i kriterijum odabira vrsta.

Primer 3.3 (zadržavanje duplikata)
Naći šifre svih izdatih knjiga.

```
SELECT K_SIF
FROM KI
```

ili, ekvivalentno:

```
SELECT ALL K_SIF
FROM KI
```

Rezultat:

K_SIF
k1
k2
k3
k4
k5
k6
k6

Rezultat izvršavanja ovog upita sadrži i duplikate (pa relacija rezultat nije skup, suprotno definiciji relacije u relacionom modelu).

Primer 3.4 (eliminacija duplikata)

Naći sve različite šifre izdatih knjiga.

```
SELECT DISTINCT K_SIF
FROM KI
```

Rezultat:

K_SIF
k1
k2
k3
k4
k5
k6

Relacija rezultat izvršavanja ovog upita jeste skup.

Primer 3.5 (pretraživanje cele tabele)

Naći sve podatke o svim izdavačima.

```
SELECT *        (ekvivalentno sa  SELECT I_SIF, NAZIV, STATUS, DRZAVA
FROM   I                          FROM   I)
```

Rezultat izvršavanja upita je kopija cele tabele I.

Primer 3.6 (skalarni izrazi u SELECT liniji)

Za svaku izdatu knjigu, naći šifru izdavača, šifru knjige, izdanje i tiraž izdanja u hiljadama primeraka (tiraž knjige u tabeli KI je u primercima).

```
SELECT K_SIF, I_SIF, IZDANJE, 'Tiraz u hiljadama=', TIRAZ/1000
FROM   KI
```

Rezultat:

K_SIF	I_SIF	IZDANJE	4	5
k1	i1	2	Tiraz u hiljadama=	10
k2	i1	2	Tiraz u hiljadama=	7
k3	i1	1	Tiraz u hiljadama=	10
k4	i1	2	Tiraz u hiljadama=	10
k5	i2	4	Tiraz u hiljadama=	5
k6	i3	1	Tiraz u hiljadama=	3
k6	i4	3	Tiraz u hiljadama=	5

Oznake "4" i "5" umesto (nepostojećih) imena kolona uvodi sâm sistem, kao redne brojeve neimenovanih kolona u relaciji rezultatu. Sistem DB2 (kao i standard SQL2) omogućuje imenovanje kolona nastalih od izraza, pa se prethodni iskaz može zapisati i na sledeći način:

```
SELECT K_SIF, I_SIF, IZDANJE, 'Tiraz u hiljadama=' AS KOMENTAR,
       TIRAZ/1000 AS TIRHILJ
FROM   KI
```

Četvrta i peta kolona rezultujuće tabele sada će imati naziv KOMENTAR, TIRHILJ, redom.

Kao što nekvalifikovani oblik SELECT iskaza omogućuje izbor kolona navedene tabele, navoďenje logičkog izraza u WHERE liniji tog iskaza omogućuje i izbor samo onih vrsta za koje je vrednost tog izraza 'tačno'. Sve opcije navedene uz nekvalifikovano pretraživanje u prethodnim primerima odnose se i na kvalifikovano pretraživanje. Osim toga, WHERE linija ovih upita može da uključi i specifične operacije, pa se razlikuju sledeće podklase upita kvalifikovanog pretraživanja.

Primer 3.7 (poreďenja i logičke operacije u logičkom izrazu)
Naći šifre izdavača iz Jugoslavije sa statusom većim od 20.

```
SELECT I_SIF
FROM I
WHERE DRZAVA='Jugoslavija' AND STATUS>20
```

Rezultat:

I_SIF
i1
i4

Umesto uobičajenih operacija poreďenja, u logičkom izrazu može se pojaviti i neka od specifičnih relacijskih operacija: IN, BETWEEN, LIKE, IS NULL (NOT IN, NOT BETWEEN, NOT LIKE, IS NOT NULL).

Primer 3.8 (BETWEEN)
Naći šifre izdavača čiji je status u intervalu od 15 do 30, kao i njihove statuse.

```
SELECT I_SIF, STATUS
FROM I
WHERE STATUS BETWEEN 15 AND 30
```

Rezultat:

I_SIF	STATUS
i1	30
i2	20
i4	30

Kao i BETWEEN, relacijska operacija IN (tj. NOT IN) je skraćeni zapis za disjunkciju (OR) niza poreďenja na jednakost (odnosno konjunkciju – AND – niza poreďenja na različitost).

Primer 3.9 (IN)
Naći šifre izdavača čiji je status 20 ili 30, kao i njihove statuse.

```
SELECT I_SIF, STATUS
FROM   I
WHERE  STATUS IN (20, 30)
```

Za dati sadržaj tabele I (odeljak 1.2) ovaj upit ima isti rezultat kao i upit iz prethodnog primera.

Relacijska operacija LIKE (odnosno NOT LIKE) omogućuje izbor vrednosti iz date kolone (obavezno tipa CHAR(n)) na osnovu sličnosti (odnosno različitosti) sa navedenim obrascem istog tipa.

Primer 3.10 (LIKE)
Naći šifre izdavača iz država koje u imenu imaju bar jedno 'u', kao i te države.

```
SELECT I_SIF, DRZAVA
FROM I
WHERE DRZAVA LIKE '%u%'
```

Rezultat:

I_SIF	DRZAVA
i1	Jugoslavija
i3	Jugoslavija
i4	Jugoslavija

Relacijska operacija IS NULL (odnosno IS NOT NULL) omogućuje testiranje nedostajuće (NULL) vrednosti zadatog atributa.

Primer 3.11 (IS NULL)
Naći sve šifre izdavača i izdanja knjige sa šifrom k1 čiji je tiraž poznat u bazi (nema nedostajuću vrednost).

```
SELECT I_SIF, IZDANJE
FROM KI
WHERE K_SIF = 'k1' AND
TIRAZ IS NOT NULL
```

Rezultat:

I_SIF	IZDANJE
i1	2

3.3.3 Upiti spajanja

Spajanje tabela predstavlja mogućnost istovremenog pretraživanja podataka iz više tabela, i može se izvesti po jednoj od operacija poređenja (=, <, <=, >, >=, <>) nad uporedivim kolonama dve tabele. Ova operacija učestvuje u izgradnji logičkog izraza u WHERE liniji, a sve tabele koje učestvuju u spajanju navode se u FROM liniji. U slučaju istih imena kolona u dve tabele, njihova pojavljivanja moraju biti kvalifikovana navođenjem imena tabela iz kojih potiču. Izdvaja se nekoliko karakterističnih tipova upita spajanja:

Primer 3.12 (spajanje dve tabele)
Naći sve informacije o izdavačima i piscima iz iste države.

```
SELECT *                    ( ekvivalentno sa:  SELECT I.*, P.*
FROM   I, P                                     FROM   I, P
WHERE  I.DRZAVA = P.DRZAVA                      WHERE  I.DRZAVA = P.DRZAVA
```

Rezultat:

I_SIF	NAZIV	ST.	DRZ.	P_SIF	IME	BR_N.	DRZ.
i1	Prosveta	30	Jugosl.	p1	B.Ćopić	2	Jugosl.
i1	Prosveta	30	Jugosl.	p3	B.Šimšić	1	Jugosl.
i1	Prosveta	30	Jugosl.	p4	D.Maksimović	2	Jugosl.
i2	Add.Wesley	20	Amer.	p5	C.J.Date	1	Amer.
i3	D.novine	10	Jugosl.	p1	B.Ćopić	2	Jugosl.
i3	D.novine	10	Jugosl.	p3	B.Šimšić	1	Jugosl.
i3	D.novine	10	Jugosl.	p4	D.Maksimović	2	Jugosl.
i4	M.srpska	30	Jugosl.	p1	B.Ćopić	2	Jugosl.
i4	M.srpska	30	Jugosl.	p3	B.Šimšić	1	Jugosl.
i4	M.srpska	30	Jugosl.	p4	D.Maksimović	2	Jugosl.

(skraćenja oznaka vrednosti su iz prostornih razloga)

Primer 3.13 Naći sve parove šifara izdavača/pisca iz iste države, pri čemu je status izdavača veći od 20.

```
SELECT I.I_SIF, P.P_SIF
FROM I, P
WHERE I.DRZAVA = P.DRZAVA
AND I.STATUS > 20
```

Rezultat:

I_SIF	P_SIF
i1	p1
i1	p3
i1	p4
i4	p1
i4	p3
i4	p4

Najznačajniji oblik spajanja je prirodno spajanje koje predstavlja spajanje po relacijskoj operaciji =, ali sa izostavljanjem jedne od dve identične kolone. U SQL-u to se postiže eksplicitnim navoďenjem svih kolona tabela koje se spajaju, osim po jedne iz svakog para identičnih kolona dobijenih pri spajanju.

Primer 3.14 (prirodno spajanje)
Naći sve informacije o izdavačima i piscima iz iste države, pri čemu se država pojavljuje samo kao jedna kolona.

```
SELECT I_SIF, NAZIV, STATUS, I.DRZAVA, P_SIF, IME, BR_NASLOVA
FROM   I, P
WHERE  I.DRZAVA = P.DRZAVA
```

Rezultat izvršavanja upita je isti kao u primeru 3.12, samo bez druge kolone DRZAVA.

Primer 3.15 (spajanje n ($n > 2$) tabela)
Naći sve parove država takve da izdavač iz prve države izdaje knjigu pisca iz druge države.

```
SELECT DISTINCT I.DRZAVA AS IDRZAVA, P.DRZAVA AS PDRZAVA
FROM   I, KI, KP, P
WHERE  I.I_SIF = KI.I_SIF AND KI.K_SIF = KP.K_SIF
                          AND KP.P_SIF = P.P_SIF
```

Rezultat:

IDRZAVA	PDRZAVA
Jugoslavija	Jugoslavija
Jugoslavija	Engleska
Amerika	Amerika

Primer 3.16 (spajanje tabele sa samom sobom)
Naći sve parove šifara izdavača iz iste države.

Upit se može razmatrati po analogiji sa primerom 3.12, tj. sa upitom "naći sve parove šifara izdavača/pisca iz iste države", koji se izražava SELECT iskazom

```
SELECT I.I_SIF, P.P_SIF
FROM   I, P
WHERE  I.DRZAVA = P.DRZAVA
```

U novom upitu se, koncepcijski, i umesto tabele P pojavljuje (još jednom) tabela I (slično, I_SIF umesto P_SIF), pa je problem koji se javlja – nejednoznačnost imena I u logičkom izrazu WHERE linije. Koncepcijsko rešenje sastoji se u sledećem: treba posmatrati dve identične kopije tabele I (RSUBP, npr. DB2 ih, naravno, ne realizuje fizički), "prvu" i "drugu". Sada se posmatraju svi parovi vrsta iz "prve"

i "druge" kopije i odabiraju oni parovi sa jednakom državom. Ove kopije potrebno je različito imenovati (što je omogućeno tzv. *korelacionim imenom tabele* u sintaksi FROM linije), pri čemu se upit svodi na upit analogan prethodnom:[5]

```
SELECT PRVI.I_SIF, DRUGI.I_SIF
FROM I PRVI, I DRUGI
WHERE PRVI.DRZAVA = DRUGI.DRZAVA
```

Rezultat:

I_SIF	I_SIF
i1	i1
i1	i3
i1	i4
i2	i2
i3	i1
i3	i3
i3	i4
i4	i1
i4	i3
i4	i4

Upotrebljeni oblik FROM linije identifikuje ime PRVI sa tabelom I, kao i ime DRUGI sa istom tabelom. Nasuprot ovom koncepcijskom rešenju, u realizaciji ovakvog upita nije neophodno imati dve kopije iste tabele. Dovoljno je obezbediti mehanizam za obraćanje dvema vrstama tabele I istovremeno. Taj mehanizam su dve n-torne promenljive (PRVI, DRUGI, u ovom slučaju), od kojih svaka, kao skup mogućih vrednosti, ima skup vrsta tabele I, tj. u svakom trenutku izvršavanja upita ima kao svoju vrednost jednu vrstu te tabele (nezavisnu od vrednosti druge n-torne promenljive).

Pošto u prethodnom rezultatu ima suvišnih parova (npr. (i1, i1) kao i jedan od parova (i1, i3), (i3, i1)), upit se može dopuniti uslovom PRVI.I_SIF < DRUGI.I_SIF, tj.

```
SELECT PRVI.I_SIF, DRUGI.I_SIF
FROM I PRVI, I DRUGI
WHERE PRVI.DRZAVA = DRUGI.DRZAVA AND
PRVI.I_SIF < DRUGI.I_SIF
```

Rezultat:

I_SIF	I_SIF
i1	i3
i1	i4
i3	i4

Dopušteno je upotrebiti n-torne promenljive (korelaciono ime tabele) i u kontekstu u kome nisu neophodne. Njihova dosledna upotreba smatra se odrazom dobrog stila. Štaviše, kadgod n-torna promenljiva nije eksplicitno dodeljena tabeli, SQL uvodi implicitnu n-tornu promenljivu sa istim imenom kao i ime tabele.

3.3.4 Podupiti

Činjenica da je rezultat izvršavanja upitnog bloka nad tabelom opet tabela omogućuje ugnježdenje SELECT upitnih blokova.

[5] Korelaciono ime tabele je nekvalifikovani identifikator; kolone u SELECT i WHERE linijama kvalifikuju se korelacionim imenom tabele.

Podupit je SELECT upitni blok umetnut u logički izraz WHERE linije drugog SELECT upitnog bloka (ili iskaza ažuriranja). Svojstvo ugnježdenja SELECT upitnih blokova je osnovno svojstvo na kome je SQL utemeljen (Structured u nazivu SQL ukazuje upravo na ovu vrstu struktuiranosti). Podupit se koristi za:

- predstavljanje pojedinačne vrednosti koja učestvuje u operaciji poređenja;
- predstavljanje skupa vrednosti čija se nepraznost ispituje primenom EXISTS (NOT EXISTS) kvantifikatora, koji se pretražuje IN (NOT IN) relacijskom operacijom, ili koji se kvantifikuje kvantifikatorima SOME, ANY ili ALL i zatim učestvuje u operaciji poređenja.

Ugnježdenje blokova može biti višestruko, ali u gotovo svim primenama na realne probleme ugnježdenje do nivoa 3 zadovoljava.

Ugnježdeni SELECT upitni blok se, po pravilu, može zameniti upitom spajanja, čije je izvršenje često efikasnije.

Primer 3.17 (relacijska operacija IN)
Naći nazive izdavača koji izdaju knjigu sa šifrom k6.

```
SELECT I.NAZIV
FROM I
WHERE I.I_SIF  IN
               (SELECT KI.I_SIF
                FROM KI
                WHERE KI.K_SIF='k6')
```

Rezultat:

NAZIV
Dečje novine
Matica srpska

Iskaz se izvršava prvo izvršavanjem podupita koji vraća skup vrednosti – šifara izdavača koji izdaju knjigu sa šifrom k6 (ovaj podupit izvršava se samo jedanput pri jednom izvršavanju celog iskaza). Zatim se pretražuje tabela I, i za svakog izdavača čija je šifra u izračunatom skupu izdaje se naziv tog izdavača. U slučaju da izostane eksplicitno kvalifikovanje imena (I_SIF umesto I.I_SIF i KI.I_SIF), primenjuje se implicitno kvalifikovanje imena (I.I_SIF za kolonu I_SIF ispred IN i KI.I_SIF za kolonu I_SIF iza IN). Ovo je saglasno pravilu da se nekvalifikovano ime kolone kvalifikuje imenom tabele (ili imenom n-torne promenljive) koja je navedena u FROM liniji koja je najdirektnije deo istog upita ili podupita. Zato je prethodni iskaz ekvivalentan sledećem iskazu (tj. ima identičan rezultat kao i taj iskaz):

```
SELECT NAZIV
FROM   I
WHERE  I_SIF IN
              (SELECT I_SIF
               FROM   KI
               WHERE  K_SIF='k6')
```

Isti upit može se formulisati i upotrebom operacije spajanja, bez podupita:

```
SELECT I.NAZIV
FROM   I, KI
WHERE  I.I_SIF = KI.I_SIF AND KI.K_SIF = 'k6'
```

Primer 3.18 (podupit i spoljašnji upit nad istom tabelom)
Naći šifre izdavača koji izdaju bar jednu knjigu koju izdaje i izdavač sa šifrom i3.

```
SELECT DISTINCT I_SIF
FROM   KI
WHERE  K_SIF IN
              (SELECT K_SIF
               FROM   KI
               WHERE  I_SIF='i3')
```

Analogno prethodnim primerima formulišu se i drugi oblici iskaza – kvalifikovanjem imena i operacijom spajanja umesto podupita.

Primer 3.19 (višestruki nivo ugnježdenja)
Naći nazive izdavača koji izdaju bar jedan roman.

```
SELECT I.NAZIV
FROM I
WHERE I.I_SIF IN
        (SELECT KI.I_SIF
         FROM KI
         WHERE KI.K_SIF IN
                (SELECT K.K_SIF
                 FROM K
                 WHERE K.OBLAST='roman'))
```

Rezultat:

NAZIV
Prosveta

Primer 3.20 (operacija poređenja)
Naći šifre izdavača iz iste države kao izdavač sa šifrom i1.

SELECT iskaz kojim se zadaje ovaj upit može da koristi implicitne ili eksplicitne n-torne promenljive, tj. sledeća dva iskaza su ekvivalentna:

```
SELECT I_SIF
FROM I
WHERE DRZAVA =
        (SELECT DRZAVA
         FROM I
         WHERE I_SIF='i1')
```

```
SELECT IX.I_SIF
FROM I IX
WHERE IX.DRZAVA =
        (SELECT IY.DRZAVA
         FROM I IY
         WHERE IY.I_SIF='i1')
```

Ovakvi podupiti čiji nekvantifikovani rezultat učestvuje u operaciji poređenja opravdani su samo kada se unapred zna da rezultat podupita sadrži najviše jedan element. Sledeća dva primera ilustruju korišćenje podupita čiji rezultati sadrže više od jednog elementa, ali im kvantifikatori SOME, ANY, ALL, omogućuju učestvovanje u operaciji poređenja (tzv. *kvantifikovano poređenje*).

Primer 3.21 (kvantifikovano poređenje – kvantifikator SOME)
Naći nazive izdavača koji izdaju knjigu sa šifrom k6

```
SELECT NAZIV
FROM   I
WHERE  I_SIF = SOME(SELECT I_SIF
                    FROM   KI
                    WHERE  K_SIF='k6')
```

Parafraza ovog iskaza je da se traže nazivi onih izdavača čija je šifra jednaka sa NEKOM (SOME) šifrom izdavača koji izdaje knjigu sa šifrom 'k6'. Isto je značenje i kvantifikatora ANY.

Primer 3.22 (kvantifikovano poređenje – kvantifikator ALL)
Naći sve informacije o izdanjima čiji je tiraž manji od svakog tiraža izdanja izdavača sa šifrom 'i1'.

```
SELECT *
FROM KI
WHERE TIRAZ <
    ALL (SELECT TIRAZ
      FROM KI
      WHERE I_SIF='i1')
```

Rezultat:

K_SIF	I_SIF	IZD.	GOD.	TIRAZ
k5	i2	4	1986	5000
k6	i3	1	1966	3000
k6	i4	3	1988	5000

3.3.5 Korelisani podupiti

Pored strukture upita u kojoj se podupit izvršava samo jedanput pri jednom izvršavanju celog upita, moguća je struktura u kojoj podupit zavisi od tabele iz FROM linije spoljašnjeg upita (tj. sadrži n-tornu promenljivu koja odgovara ovoj tabeli). Tada se podupit mora izvršiti za svaku vrstu tabele iz spoljašnjeg upita, da bi se utvrdilo da li se ta vrsta kvalifikuje za rezultat celog upita ili ne. Takav podupit zove se *korelisani podupit*.

Primer 3.23 Isti upit kao u primeru 3.17 ("naći nazive izdavača koji izdaju knjigu sa šifrom k6") može se preformulisati tako da uključi korelisani podupit na sledeći način:

```
SELECT I.NAZIV
FROM   I
WHERE  'k6' IN
           (SELECT KI.K_SIF
            FROM   KI
            WHERE  KI.I_SIF=I.I_SIF)
```

U podupitu je neophodno eksplicitno kvalifikovati ime kolone I.I_SIF da bi se razlikovalo od nekvalifikovanog imena kolone I_SIF, koja ima implicitnu kvalifikaciju KI.I_SIF.

Ovaj iskaz se izvršava na sledeći način: za svaku vrstu tabele I, i specifičnu vrednost Ix njene kolone I_SIF (šifra izdavača), izvršava se podupit

```
(SELECT K_SIF
 FROM   KI
 WHERE  I_SIF='Ix')
```

Ako pretraženi skup šifara knjiga K_SIF sadrži šifru 'k6', posmatrana vrsta tabele I se dalje obrađuje i izdaje se vrednost njene kolone NAZIV; postupak se ponavlja za sledeću vrstu tabele I.

Prethodni podupit zove se korelisani podupit jer je u neposrednoj zavisnosti od spoljašnjeg upitnog bloka.

Analogna forma upita odgovara i slučaju kada se korelisani podupit i spoljašnji upitni blok odnose na istu tabelu. U ovom slučaju se moraju uvesti eksplicitne n-torne promenljive različite od imena tabele.

Primer 3.24 Naći šifre svih knjiga koje izdaje više od jednog izdavača.

```
SELECT DISTINCT KIX.K_SIF
FROM KI KIX
WHERE KIX.K_SIF IN
            (SELECT KIY.K_SIF
             FROM KI KIY
             WHERE KIY.I_SIF <> KIX.I_SIF)
```

Rezultat:

K_SIF
k6

3.3.6 Egzistencijalni kvantifikator

U SQL-u, egzistencijalno kvantifikovani logički izraz je oblika EXISTS (SELECT ... FROM ...) (odnosno NOT EXISTS (SELECT ... FROM ...)). Ovaj izraz izračunavanjem dobija vrednost 'tačno' ako i samo ako je rezultat podupita predstavljenog konstrukcijom 'SELECT ... FROM ...' neprazan (odnosno prazan).

Primer 3.25 Naći nazive izdavača koji izdaju knjigu sa šifrom k3.

```
SELECT I.NAZIV
FROM   I
WHERE  EXISTS
       (SELECT *
        FROM   KI
        WHERE  KI.I_SIF=I.I_SIF AND KI.K_SIF='k3')
```

Ovaj iskaz nalazi nazive svih izdavača za koje postoji vrsta u tabeli izdavaštva (KI) sa istom šifrom izdavača i šifrom knjige k3. Na primer, prva vrsta tabele I ima naziv izdavača 'Prosveta' i šifru izdavača i1. U tabeli KI postoji vrsta sa šifrom izdavača i1 i šifrom knjige k3, pa je 'Prosveta' jedan (i jedini) od naziva koji se pojavljuju u rezultatu.

Egzistencijalni kvantifikator je moćan mehanizam SQL-a. Forma prethodnog upita je samo preformulacija upita izraženog bez egzistencijalnog kvantifikatora, jer se svaki upit formulisan IN relacijskom operacijom može formulisati i korišćenjem EXISTS kvantifikatora. Da obratno ne važi, sugeriše sledeći primer.

Primer 3.26 Naći nazive izdavača koji izdaju sve knjige Branka Ćopića.

Ovaj upit je već razmatran kao ilustracija operacije deljenja u relacionoj algebri (odeljak 2.1). Pogledajmo šta on novo donosi kada je u pitanju SQL.

Prirodna formulacija upita uključila bi univerzalni kvantifikator "za svaki" (FORALL), koji nije podržan u SQL-u. Univerzalni kvantifikator se može izraziti negacijom egzistencijalnog kvantifikatora, tj.

$$(\text{FORALL x})\text{p(x)} \equiv \text{NOT } (\text{EXISTS x})(\text{NOT}(\text{p(x)})),$$

gde je x – promenljiva, a p – logički izraz koji uključuje promenljivu x.

Tako se izraz "nazivi izdavača takvih da, za svaku knjigu Branka Ćopića, postoji vrsta u KI tabeli koja utvrđuje da taj izdavač izdaje tu knjigu", može preformulisati u ekvivalentni izraz "nazivi izdavača takvih da ne postoji knjiga Branka Ćopića takva da ne postoji vrsta u KI tabeli koja utvrđuje da taj izdavač izdaje tu knjigu". Da bi se pojednostavio upit, izdvojmo prvo sve knjige Branka Ćopića u virtuelnu tabelu – pogled K_BC sledećim SQL iskazom:

```
CREATE VIEW  K_BC AS
       (SELECT *
        FROM   K
        WHERE  K_SIF IN
               (SELECT K_SIF
                FROM   KP
                WHERE  P_SIF =
                       (SELECT P_SIF
                        FROM   P
                        WHERE  IME = 'Branko Ćopić')))
```

Sada se nalaženje naziva izdavača koji su izdali sve knjige Branka Ćopića može izraziti sledećim SQL iskazom:

```
SELECT I.NAZIV
FROM I
WHERE NOT EXISTS
    (SELECT *
     FROM K_BC
     WHERE NOT EXISTS
        (SELECT *
         FROM KI
         WHERE KI.I_SIF=I.I_SIF AND KI.K_SIF=K_BC.K_SIF))
```

Rezultat:

NAZIV
Prosveta

Korisna sugestija je da se u izražavanju komplikovanog upita, kakav je poslednji primer, pođe od pseudo SQL iskaza koji uključuje FORALL kvantifikator, a zatim izvrši konverzija u pravi SQL upit koji uključuje NOT EXISTS kvantifikator umesto univerzalnog kvantifikatora FORALL ([24]).

Druga korisna sugestija za formulaciju složenih upita je da se, kadgod to odgovara prirodi upita, pođe od implikacije, a da se ona zatim preformuliše u logički izraz koji je korektan u SQL-u. Tako bi se prethodni primer prirodno formulisao kao "naći naziv izdavača (npr. Ix) takvog da za svaku knjigu važi implikacija: ako je autor knjige Branko Ćopić, onda je izdavač Ix izdao tu knjigu".

Implikacija "ako p onda q" ekvivalentna je izrazu "NOT (p) OR q", pa se implikacija iz našeg primera može izraziti kao "autor knjige nije Branko Ćopić ili je izdavač Ix izdao tu knjigu". Zajedno sa prethodnim razmatranjem kvantifikatora FORALL, dobija se ekvivalencija izraza

FORALL x (IF p THEN q) $\equiv$ NOT EXISTS x (NOT NOT p AND NOT q), tj.
FORALL x (IF p THEN q) $\equiv$ NOT EXISTS x (p AND NOT q).

Zato ceo prethodni upit dobija formulaciju: "Naći naziv izdavača (npr. Ix) za kojeg važi sledeće: ne postoji knjiga čiji je autor Branko Ćopić a koju nije izdao izdavač Ix" (ovo je upravo formulacija izražena SQL-om). Opisani postupak predstavlja sistematični pristup obradi univerzalnog kvantifikatora i implikacije.

3.3.7 Agregatne funkcije

Za razliku od *skalarnih funkcija*, čiji je argument skalarna vrednost, tj. jedna vrednost iz jednog domena, *agregatne funkcije* preslikavaju skup vrednosti jedne kolone (obično izvedene tabele) u skalarnu vrednost. Pod izvedenom tabelom ovde se podrazumeva rezultat primene logičkog izraza iz WHERE linije na tabele iz FROM linije.

SQL podržava širok spektar skalarnih funkcija, od konverzionih (npr. funkcija za pretvaranje celobrojnog ili realnog tipa u decimalni tip, DECIMAL, za pretvaranje brojevnih tipova u znakovni tip, realni tip dvostruke tačnosti ili celobro-

jni tip – DIGITS, FLOAT, INTEGER, redom), preko opštih (npr. funkcija koja vraća dužinu podatka u bajtovima – LENGTH) i aritmetičkih (ABS, TAN, SQRT, TRUNC), do funkcijâ za obradu znakovnih podataka (npr. SUBSTR – vraća podnisku niske, navedene dužine, od pozicije kao početka, ili POSSTR – vraća poziciju prvog pojavljivanja jedne niske u drugoj), kao i druge funkcije vezane za specifične tipove određenog sistema (npr. TIME, YEAR, CHAR, DATE, DAY – nad podacima tipa TIME, DATE, TIMESTAMP). Poziv skalarne funkcije može se navesti kao operand u izrazu odgovarajućeg tipa.

Primer 3.27 Izdati nazive izdavača i nazive njihovih država prevedene na engleski jezik.

Iskaz kojim se realizuje ovaj upit ilustrovaće skalarnu funkciju SUBSTR i mogućnost SQL-a sistema DB2 u selektivnoj dodeli vrednosti skalarnom izrazu tabele-rezultata.

```
SELECT NAZIV,
  CASE SUBSTR(DRZAVA,1,1)
    WHEN 'J' THEN 'Yugoslavia'
    WHEN 'A' THEN 'USA'
    WHEN 'E' THEN 'UK'
  END AS DRZAVA_ENG
FROM I
```

Rezultat:

NAZIV	DRZAVA_ENG
Prosveta	Yugoslavia
Addison Wesley	USA
Decje novine	Yugoslavia
Matica Srpska	Yugoslavia

Mada je SQL i bez agregatnih funkcija dovoljan za pretraživanje svakog pojedinačnog podatka iz baze podataka (relaciono je kompletan), bez mehanizma agregatnih funkcija on ne omogućuje izražavanje praktičnih i često potrebnih upita tipa "Koliko ima izdavača?"

SQL podržava agregatne funkcije COUNT, SUM, AVG, MAX i MIN[6], sa sledećim značenjem:

COUNT	–	broj vrednosti u koloni
SUM	–	zbir vrednosti u koloni
AVG	–	srednja vrednost u koloni
MAX	–	najveća vrednost u koloni
MIN	–	najmanja vrednost u koloni.

Agregatna funkcija se poziva tako što se navede njeno ime sa argumentom – imenom kolone u zagradi. Poziv agregatne funkcije može se naći u SELECT liniji bilo glavnog bilo ugnježdenog upita. Ako se takav poziv navede, odgovarajuća SELECT linija mora se sastojati samo od takvih poziva (izuzetak su upiti koji uključuju GROUP BY liniju, v. tačku 3.3.8). Poziv agregatne funkcije ne može se naći u WHERE liniji upita.

[6]SQL sistema DB2 podržava i agregatne funkcije standardne devijacije i varijanse – STDDEV, VARIANCE.

Argument za funkcije SUM i AVG mora biti kolona brojevnih vrednosti. Argumentu može da prethodi rezervisana reč DISTINCT, koja ukazuje na eliminaciju duplikata u koloni–argumentu pre primene funkcije, ili rezervisana reč ALL (što se i podrazumeva) kada se funkcija primenjuje na sve vrednosti u koloni, uključujući i duplikate.

Agregatna funkcija COUNT mora imati DISTINCT opciju ispred argumenta, osim u svom specijalnom obliku COUNT(∗) kojim se prebrojavaju vrste (a ne vrednosti kolone) koje zadovoljavaju zadati logički uslov; ovaj oblik agregatne funkcije COUNT ne dozvoljava navođenje DISTINCT opcije i podrazumeva opciju ALL.

Za sve agregatne funkcije (osim COUNT(∗)) važi da se primenjuju samo na ne-NULL vrednosti kolone–argumenta iz onih vrsta odgovarajuće tabele koje zadovoljavaju logički uslov WHERE linije upita; agregatna funkcija COUNT(∗) primenjuje se na sve vrste koje zadovoljavaju zadati uslov.

Argument agregatne funkcije može biti i izraz u kome učestvuje kolona (npr. TIRAZ/1000), ali samo kada opcija DISTINCT nije navedena.

Pojedini sistemi primenjuju i niz drugih pravila i ograničenja za upotrebu agregatnih funkcija ([37], [24]).

Primer 3.28 (COUNT)
Naći ukupan broj izdavača koji su izdali bar po jednu knjigu.

```
SELECT COUNT(DISTINCT I_SIF)
FROM KI
```

Rezultat:

1
4

Primer 3.29 (COUNT(∗))
Naći broj izdanja knjige sa šifrom k6.

```
SELECT COUNT(*)
FROM KI
WHERE KI.K_SIF='k6'
```

Rezultat:

1
2

Primer 3.30 (SUM)
Naći ukupni tiraž knjige sa šifrom k6 (svih izdanja svih izdavača).

```
SELECT SUM(TIRAZ)
FROM KI
WHERE KI.K_SIF='k6'
```

Rezultat:

1
8000

Primer 3.31 (MAX)
Naći šifre izdavača čiji je status niži od maksimalnog statusa u I tabeli.

```
SELECT I.I_SIF
FROM I
WHERE STATUS <
            (SELECT MAX(STATUS)
             FROM I)
```

Rezultat:

I_SIF
i2
i3

Primer 3.32 (AVG)
Naći šifru, status i državu izdavača čiji je status viši ili jednak prosečnom statusu izdavača iz te države (ovo je ujedno i primer agregatne funkcije u korelisanom podupitu).

```
SELECT IX.I_SIF, IX.STATUS, IX.DRZAVA
FROM I IX
WHERE IX.STATUS >=
        (SELECT AVG(STATUS)
         FROM I IY
         WHERE IY.DRZAVA = IX.DRZAVA)
```

Rezultat:

I_SIF	STATUS	DRZAVA
i1	30	Jugoslavija
i2	20	Amerika
i4	30	Jugoslavija

3.3.8 Operator grupisanja

Na jednostavni SELECT upitni blok moguće je primeniti i operator grupisanja

```
GROUP BY ime-kolone
```

koji preureďuje (logički) tabelu iz FROM linije u particije ili grupe tako da unutar jedne grupe sve vrste imaju istu vrednost u koloni iz GROUP BY operatora. Takoďe važi i da su sve vrste tabele sa istom vrednošću kolone iz GROUP BY operatora – u jednoj grupi. Za potrebe grupisanja, sve NULL vrednosti u koloni iz GROUP BY operatora tretiraju se kao jednake.

SELECT linija se sada primenjuje na svaku grupu, a ne na svaku vrstu tabele.

Primer 3.33 Za svaku izdatu knjigu, naći šifru knjige i ukupni tiraž u kome je izdata.

```
SELECT K_SIF, SUM(TIRAZ)
FROM KI
GROUP BY K_SIF
```

Rezultat:

K_SIF	2
k1	10000
k2	7000
k3	10000
k4	10000
k5	5000
k6	8000

Svaki izraz u SELECT liniji mora biti singlton (jednočlan skup) za svaku grupu, tj. mora biti jedna od kolona iz GROUP BY operatora, konstanta ili agregatna funkcija nad svim vrednostima neke kolone unutar te grupe. Precizirajmo sada pravilo iz tačke 3.3.7: ukoliko se agregatna funkcija javlja u SELECT liniji upita u kome se ne primenjuje GROUP BY operator na istom nivou ugnježdenja, ta linija ne sme da sadrži ime kolone, konstantu ili izraz.

Ukoliko u SELECT upitnom bloku postoji WHERE linija, vrste koje ne zadovoljavaju logički izraz iz WHERE linije eliminišu se pre bilo kakvog grupisanja.

Primer 3.34 Za svaku knjigu, naći šifru knjige i maksimalni tiraž izdanja te knjige, isključujući izdanja izdavača sa šifrom i1.

```
SELECT K_SIF, MAX(TIRAZ)
FROM KI
WHERE I_SIF <> 'i1'
GROUP BY K_SIF
```

Rezultat:

K_SIF	2
k5	5000
k6	5000

GROUP BY operator može se primeniti, sa analognom semantikom, i na bilo koju kombinaciju kolona tabele iz FROM linije.

Kvalifikovano grupisanje. Kao što pretraživanje vrstâ tabele SELECT upitnim blokom može biti kvalifikovano WHERE logičkim izrazom kojim se biraju odnosno eliminišu vrste tabele iz FROM linije, tako i pretraživanje grupa dobijenih GROUP BY operatorom može biti kvalifikovano primenom logičkog izraza kojim se biraju one grupe na koje će se SELECT linija primenjivati, odnosno eliminišu ostale grupe. Taj novi logički izraz zadaje se operatorom

```
HAVING logički-izraz
```

Izrazi koji učestvuju u logičkom izrazu HAVING operatora moraju imati jedinstvenu vrednost po grupi.

Primer 3.35 Naći šifru knjige za svaku knjigu koja ima više izdanja:

```
SELECT K_SIF
FROM KI
GROUP BY K_SIF
HAVING COUNT(*) > 1
```

Rezultat:

K_SIF
k6

Operatori GROUP BY i HAVING mogu se primeniti i u podupitima.

Značajno je da se svaki upit formulisan GROUP BY i HAVING operatorom može preformulisati u oblik koji ih ne koristi. Tako se poslednji primer može izraziti i na sledeći način:

```
SELECT K_SIF
FROM K
WHERE 1 <
         (SELECT COUNT (*)
          FROM KI
          WHERE K_SIF=K.K_SIF)
```

Zbog nedostataka ova dva operatora koji su izvan okvira ovog teksta, u literaturi ([24]) preporučuju se alternativne formulacije upita (koje ne koriste GROUP BY – HAVING konstrukcije).

3.3.9 Puni upitni blok

Pored osnovnog oblika iskaza pretraživanja SQL-a sa strukturom jednostavnog upitnog bloka koja uključuje sve dosad pomenute mogućnosti, iskaz pretraživanja može imati i oblik *punog upitnog bloka*. Neformalno, puni upitni blok se dobija primenom (0 ili više) skupovnih operacija unije, preseka i/ili razlike na jednostavne ili druge pune upitne blokove[7].

Mogućnost primene skupovnih operacija na upitne blokove posledica je činjenice da je rezultat izvršavanja svakog upitnog bloka – (izvedena) relacija tj. skup – skup vrsta. Operacije unije (UNION), preseka (INTERSECT) i razlike (EXCEPT) u SQL-u se mogu primenjivati samo na unijski kompatibilne relacije (u smislu u kome je unijska kompatibilnost uvedena u tački 2.1.2). Ako su odgovajujuće kolone jednako imenovane, kolona tabele-rezultata nasleđuje to ime. U suprotnom, kolona rezultujuće tabele je neimenovana (tj. dobija sistemski generisano ime).

Obzirom da je eliminacija duplikata, kao sastavni deo ovih skupovnih operacija, (vremenski) skupa operacija, SQL nudi mogućnost da se ona ne izvrši (opcija ALL uz svaku od skupovnih operacija). Ovo je naročito pogodno kada se unapred zna da rezultat neće imati duplikata.

Primer 3.36 Naći šifre knjiga koje su romani ili ih izdaje izdavač sa šifrom i2 (ili za koje važe oba uslova).

```
SELECT K_SIF
FROM K
WHERE OBLAST = 'roman'
UNION
SELECT K_SIF
FROM KI
WHERE I_SIF = 'i2'
```

Rezultat:

K_SIF
k1
k3
k5

Primer 3.37 Naći nazive država u kojima su registrovani i izdavači i pisci.

[7] Dakle, i jednostavni upitni blok jeste puni upitni blok.

```
SELECT DRZAVA
FROM I
INTERSECT
SELECT DRZAVA
FROM P
```

Rezultat:

DRZAVA
Amerika
Jugoslavija

Ako se u ovom primeru operacija INTERSECT zada sa opcijom ALL (INTERSECT ALL), u rezultatu će se dobiti i ponovljene vrednosti:

```
SELECT DRZAVA
FROM I
INTERSECT ALL
SELECT DRZAVA
FROM P
```

Rezultat:

DRZAVA
Amerika
Jugoslavija
Jugoslavija

Primer 3.38 Izdati nazive država u kojima su registrovani pisci ali ne i izdavači.

```
SELECT DRZAVA
FROM P
EXCEPT
SELECT DRZAVA
FROM I
```

Rezultat:

DRZAVA
Engleska

Isti bi se rezultat dobio, u našem primeru, i za operaciju EXCEPT ALL, zato što je u relacijama P i I jednak broj vrsta sa istim vrednostima atributa DRZAVA (po 1 sa vrednošću 'Amerika' i po tri sa vrednošću 'Jugoslavija'). Međutim, kada bi, npr. u relaciji P bilo registrovano četiri pisca iz Jugoslavije a u relaciji I – tri izdavača iz Jugoslavije, tada bi operacija EXCEPT ALL u našem primeru proizvela, pored vrednosti 'Engleska', i vrednost 'Jugoslavija'. Ovo je u suprotnosti sa skupovnim pojmom razlike pa i sa samim relacionim modelom.

Pri višestrukoj primeni skupovnih operacija, redosled njihovog izvršavanja je sleva nadesno ("odozgo naniže"), pri čemu operacija preseka (INTERSECT) ima prioritet. Redosled izvršenja se može zagradama menjati.

Puni upitni blok, pa prema tome i skupovne operacije UNION, INTERSECT, EXCEPT, mogu se primeniti i u podupitima iskaza pretraživanja, ali i u drugim iskazima SQL-a – npr. u iskazu za kreiranje pogleda (CREATE VIEW, v. 4), ili iskazima za unošenje, brisanje i ažuriranje (v. 3.3.13, 3.3.15, 3.3.14).

3.3.10 Operator uređenja i operator WITH

Kompletni iskaz pretraživanja – SELECT iskaz, pored strukture jednostavnog i punog upitnog bloka, može imati i dodatne operatore – operator uređenja po vred-

nostima izabranih kolona rezultujuće tabele (ORDER BY) i operator kojim se pretraživanje punim upitnim blokom primenjuje nad imenovanom izvedenom tabelom – rezultatom izvršavanja nekog punog upitnog bloka (WITH operator).

Primer 3.39 (pretraživanje sa uređenjem)
Prikazati šifre i statuse izdavača u opadajućem redosledu statusa.

```
SELECT I_SIF, STATUS
FROM I
ORDER BY STATUS DESC
```

Rezultat:

I_SIF	STATUS
i1	30
i4	30
i2	20
i3	10

Uređenje se može zadati na isti način kao u CREATE INDEX iskazu. Osim opadajućeg (DESC), moguć je i rastući poredak (ASC) koji se i podrazumeva. Kolone u ORDER BY liniji (može ih biti više) mogu se identifikovati i svojim rednim brojevima (sleva nadesno u rezultujućoj tabeli) umesto imenima; ako je kolona – neimenovani skalarni izraz, ona se mora identifikovati svojim rednim brojem.

Primer 3.40 Uređenje tabele iz primera 3.6 u rastućem poretku šifara izdavača unutar rastućih tiraža u hiljadama postiže se SELECT iskazom

```
SELECT   K_SIF, I_SIF, IZDANJE, 'Tiraz u hiljadama =', TIRAZ/1000
FROM     KI
ORDER BY 5, I_SIF
```

Rezultat:

K_SIF	I_SIF	IZDANJE	4	5
k6	i3	1	Tiraz u hiljadama =	3
k5	i2	4	Tiraz u hiljadama =	5
k6	i4	3	Tiraz u hiljadama =	5
k2	i1	2	Tiraz u hiljadama =	7
k1	i1	2	Tiraz u hiljadama =	10
k3	i1	1	Tiraz u hiljadama =	10
k4	i1	2	Tiraz u hiljadama =	10

Primer 3.41 Pretraživanje nad imenovanom izvedenom tabelom
Iz tabele sa parovima (IME, NAZIV) koja sadrži imena pisaca i nazive izdavača koji izdaju knjige tih pisaca, izdati nazive svih izdavača Branka Ćopića.

```
WITH PI AS
       (SELECT DISTINCT IME, NAZIV
        FROM P, I, KP, KI
        WHERE P.P_SIF=KP.P_SIF AND KP.K_SIF=KI.K_SIF AND
```

```
                KI.I_SIF=I.I_SIF)
SELECT NAZIV
FROM PI
WHERE IME='B.Copic'
```

Rezultat:

NAZIV
Prosveta

Operatori WITH i ORDER BY mogu se primeniti u istom SELECT iskazu, da bi se dobilo uređenje po vrednostima kolona rezultata pretraživanja imenovane izvedene tabele. U prethodnom primeru, na kraju SELECT iskaza mogao bi se navesti operator ORDER BY NAZIV kako bi se dobili nazivi izdavača knjiga Branka Ćopića uređeni po abecednom redu.

3.3.11 Način izvršavanja SELECT iskaza

SELECT iskaz koji ne sadrži skupovne operacije UNION, INTERSECT, EXCEPT, može biti dosta složen, i može da uključi sledeće linije, obavezno u navedenom redosledu:

```
SELECT    ...
FROM      ...
WHERE     ...
GROUP BY  ...
HAVING    ...
ORDER BY  ...
```

Ovakav SELECT iskaz može, osim toga, da uključi i podupite. Redosled u kome se (konceptualno) izvršavaju linije ovakvog SELECT iskaza je isti kao i redosled u kome su zapisane, osim SELECT linije koja se izvršava "pretposlednja" – neposredno ispred ORDER BY operatora. Tako algoritam izvršavanja SELECT iskaza sadrži sledeće korake:

1. izračunati Dekartov proizvod tabelâ iz FROM linije;
2. iz rezultata prethodnog koraka eliminisati sve n-torke koje ne zadovoljavaju logički izraz WHERE linije;
3. rezultat prethodnog koraka grupisati po vrednostima kolona navedenih u GROUP BY liniji;
4. iz rezultata prethodnog koraka eliminisati grupe koje ne zadovoljavaju logički izraz HAVING linije;
5. za svaku grupu iz rezultata prethodnog koraka proizvesti po jednu n-torku, prema sadržaju SELECT linije;
6. uređenjem rezultata prethodnog koraka prema kolonama u ORDER BY liniji dobiti konačni rezultat SELECT iskaza.

Ukoliko se u SELECT iskazu nalazi ugnježden podupit, onda se sve (prisutne) linije izvršavaju navedenim redom u podupitu, pa zatim u spoljašnjem upitu.

Prethodni algoritam je samo konceptualan zato što daje korektan rezultat, ali bi njegovo striktno izvršavanje bilo veoma neefikasno. Zbog toga se pristupa raznim tehnikama optimizacije koje obezbeđuju istu korektnost u izvršavanju SELECT iskaza, ali uz maksimalnu efikasnost.

Ako SELECT iskaz sadrži skupovnu operaciju, prvo se izvršavaju njeni operandi (pojedinačni SELECT–FROM–WHERE–itd. blokovi), prema prethodnoj konceptualnoj metodi, ali bez eventualnih ORDER BY operatora. Zatim se na rezultate izvršavanja ovih blokova primenjuje skupovna operacija, i najzad operator ORDER BY, ako postoji (u jednom SELECT iskazu moguće je navesti najviše jedan ORDER BY operator i to na kraju iskaza).

3.3.12 Relaciona kompletnost SQL-a

Za SQL se može jednostavno dokazati da je relaciono kompletan, tako što se pokazuje da podržava sve operacije relacione algebre (za koju je pokazano da je relaciono kompletna, v. odeljak 2.2). U ovoj tački koristiće se formalniji termin "relacija" (a ne njegov sinonim "tabela", uobičajen u terminologiji SQL-a), jer je izlaganje tesno vezano za formalizam relacionog modela – relacionu algebru.

1. Skupovne operacije unije, preseka i razlike SQL podržava direktno operacijama UNION, INTERSECT, EXCEPT.

2. Skupovna operacija Dekartovog proizvoda relacija R i S realizuje se upitom spajanja

   SELECT $*$
   FROM R, S

3. Operacija projekcije relacije R na atribute $A_{i_1}, \ldots, A_{i_k}$ realizuje se upitom bez WHERE linije

   SELECT A_{i_1}, $\ldots$, A_{i_k}
   FROM R

4. Operacija restrikcije po logičkom izrazu Ψ realizuje se upitom sa WHERE linijom

   SELECT $*$
   FROM R
   WHERE Ψ

5. Operacija slobodnog Θ spajanja relacija R i S po paru atributa A_i, B_j realizuje se upitom spajanja

   SELECT $*$
   FROM R, S
   WHERE $R.A_i \,\Theta\, S.B_j$

6. Operacija prirodnog spajanja relacija $R(A_1, \ldots, A_n)$, $S(B_1, \ldots, B_m)$ po atributima A_i i B_j realizuje se upitom spajanja

   SELECT $A_1, \ldots, A_n, B_1, \ldots, B_{j-1}, B_{j+1}, \ldots, B_m$
   FROM R, S
   WHERE $A_i = B_j$

7. Operacija deljenja relacije $R(A, B)$ relacijom $S(C)$, po atributu B, ima sledeću interpretaciju: naći skup vrednosti atributa A, takvih da je svaka vrednost u relaciji R sa svakom vrednošću iz S, tj. naći skup vrednosti $a \in R[A]$ takvih da za svaku vrednost $c \in S$ postoji n-torka (par) $t \in R$,

takva da je $t = (a, c)$; primenom ekvivalencije $(\forall x)P(x) \equiv (\neg\exists x)\neg P(x)$, interpretacija je ekvivalentna sledećoj: naći skup vrednosti $a \in R[A]$, sa svojstvom da ne postoji vrednost $c \in S$ za koju ne postoji n-torka $t \in R$ takva da je $t = (a, c)$, što se u SQL-u može neposredno izraziti sledećim iskazom:

```
SELECT A
FROM R RX
WHERE   NOT EXISTS
          (SELECT *
           FROM S
           WHERE   NOT EXISTS
                     (SELECT *
                      FROM R
                      WHERE R.B = S.C AND R.A = RX.A))
```

(slično i za relaciju R sa n atributa, odnosno relaciju S sa m atributa).

Dakle, SQL je relaciono kompletan.

3.3.13 Unošenje

Nove vrste se unose u tabelu iskazom INSERT čija sintaksa ima jedan od sledeća dva oblika:

```
INSERT INTO  ime-tabele [( ime-kolone {,ime-kolone})]
VALUES ( konstanta {,konstanta})
```

ili

```
INSERT INTO  ime-tabele [( ime-kolone {,ime-kolone})]
puni upitni blok
```

Prvim oblikom INSERT iskaza u tabelu se upisuje pojedinačna vrsta sa navedenim konstantnim vrednostima u navedenim kolonama. Broj konstanti mora biti jednak broju kolona, a pridruživanje se vrši u navedenom redosledu. U kolone koje nisu navedene unose se NULL (odnosno podrazumevane) vrednosti ako kolona nije definisana kao NOT NULL kolona; pod istim uslovom vrednost NULL može se i eksplicitno navesti kao konstanta.

Drugim oblikom INSERT iskaza u tabelu se upisuje jedna ili više vrsta, koje predstavljaju rezultat izvršavanja punog upitnog bloka; pri tom i-ta kolona rezultata izvršavanja punog upitnog bloka odgovara i-toj koloni u zapisu iskaza. Ako je rezultat izvršavanja punog upitnog bloka prazan, INSERT iskazom se ne dodaje nijedna vrsta tabeli.

U oba slučaja, redosled u kome se kolone navode ne mora da odgovara redosledu kolona u tabeli u koju se unosi. Ako se imena kolona ne navedu, vrednosti se dodeljuju kolonama u redosledu definisanom tabelom. U oba slučaja, takoďe, za svaku vrstu koja se unosi mora biti obezbeďena vrednost u svakoj NOT NULL koloni koja nije podržana DEFAULT opcijom.

Primer 3.42 Dodati tabeli K knjigu sa šifrom k7, sa naslovom – 'Čarobna šuma' i sa trenutno nepoznatom oblašću.

```
INSERT INTO K (K_SIF, NASLOV)
VALUES ('k7', 'Čarobna šuma')
```

Uneta vrsta ima nedostajuću (NULL) vrednost u koloni OBLAST.

Isti efekat proizvodi i sledeći iskaz:

```
INSERT INTO K (K_SIF, NASLOV, OBLAST)
VALUES ('k7', 'Čarobna šuma', NULL)
```

Operacijom unošenja može da bude narušen neki od uslova integriteta baze podataka, pri čemu takvu operaciju sistem odbija da izvrši.

Primer 3.43 Iskaz

```
INSERT INTO KI (K_SIF, I_SIF, IZDANJE, TIRAZ)
VALUES ('k20', 'i20', 3, 10000)
```

bio bi odbijen, jer bi se njegovim izvršavanjem realizovalo unošenje vrste u tabelu izdanja KI, koja se odnosi na izdavača (i20) koji nije poznat u tabeli izdavača I (i na knjigu k20 koja nije poznata u tabeli knjiga K); time bi bio narušen referencijalni integritet (v. tačku 1.4.2).

Primer 3.44 Uneti u tabelu KI_KOPIJA vrste tabele KI koje se odnose na izdavače i1 i i2.

Tabela KI_KOPIJA može da se kreira prema obrascu tabele KI (analognim CREATE TABLE iskazom), sa istim kolonama kao i tabela KI. Izvršavanjem sledećeg iskaza, u tabelu KI_KOPIJA unosi se odgovarajući sadržaj:

```
 INSERT INTO KI_KOPIJA
SELECT K_SIF, I_SIF, IZDANJE, TIRAZ
FROM   KI
WHERE  I_SIF IN ('i1', 'i2')
```

KI_KOPIJA:

K_SIF	I_SIF	IZDANJE	GODINA	TIRAZ
k1	i1	2	NULL	10000
k2	i1	2	NULL	7000
k3	i1	1	NULL	10000
k4	i1	2	NULL	10000
k5	i2	4	NULL	5000

Tabeli se mogu dodati i vrste koje su rezultat operacije spajanja dve ili više tabela, kao što pokazuje sledeći primer:

Primer 3.45 Kreirati tabelu IP_DRZAVA sa kolonama I_SIF, P_SIF, DRZAVA i uneti u tu tabelu sve parove šifara izdavača i pisaca iz iste države, kao i odgovarajuće države.

```
    CREATE TABLE IP_DRZAVA
(I_SIF   CHAR(6) NOT NULL,
 P_SIF   CHAR(6) NOT NULL,
 DRZAVA  CHAR(20),
 PRIMARY KEY (I_SIF,P_SIF));
    INSERT INTO IP_DRZAVA
 SELECT I_SIF, P_SIF, I.DRZAVA
 FROM   I, P
 WHERE  I.DRZAVA = P.DRZAVA
```

Za razliku od upita u primeru 3.12, navedeni par CREATE, INSERT iskaza kreira trajnu (baznu) tabelu IP_DRZAVA koja se može pretražiti sledećim SELECT iskazom:

```
SELECT *
FROM IP_DRZAVA
```

Rezultat:

I_SIF	P_SIF	DRZAVA
i1	p1	Jugoslavija
i1	p3	Jugoslavija
i1	p4	Jugoslavija
i2	p5	Amerika
i3	p1	Jugoslavija
i3	p3	Jugoslavija
i3	p4	Jugoslavija
i4	p1	Jugoslavija
i4	p3	Jugoslavija
i4	p4	Jugoslavija

Drugi oblik INSERT iskaza omogućuje i da se puni upitni blok primeni na imenovanu izvedenu tabelu WITH operatorom (slično kao kod SELECT iskaza).

Primer 3.46 Kreirati tabelu IZD_BC sa jednom kolonom NAZIV (izdavača) i u nju upisati sve izdavače koji izdaju knjige Branka Ćopića.

Nazive svih izdavača knjiga Branka Ćopića dobićemo iz izvedene tabele sa parovima (IME, NAZIV) koja sadrži imena pisaca i nazive izdavača koji izdaju knjige tih pisaca.

```
  CREATE TABLE IZD_BC
(NAZIV  CHAR(20) NOT NULL)
```

```
  INSERT INTO IZD_BC
 WITH PI AS
(SELECT DISTINCT IME, NAZIV
 FROM P, I, KP, KI
 WHERE P.P_SIF=KP.P_SIF AND KP.K_SIF=KI.K_SIF
AND KI.I_SIF=I.I_SIF)
 SELECT NAZIV
 FROM PI
 WHERE IME='B.Copic'
```

Rezultat:

NAZIV
Prosveta

3.3.14 Ažuriranje

Za ažuriranje u užem smislu, tj. za izmenu postojećih vrednosti u tabeli, koristi se UPDATE iskaz. Sintaksa iskaza ima oblik:

```
   UPDATE  ime-tabele  [ ime n-torne promenljive]
  SET  ime-kolone = izraz {,ime-kolone = izraz}
[ WHERE  logički-izraz ]
```

U UPDATE iskazu navodi se ime tabele čiji se sadržaj menja. To može biti ime bazne tabele ili ime pogleda. Ime n-torne promenljive (ili korelaciono ime) upotrebljava se u slučaju da logički izraz WHERE linije uključuje upitni blok nad tabelom koja se ažurira, i u čijoj WHERE liniji figuriše tabela iz UPDATE iskaza.

U SET liniji navode se sve kolone koje se menjaju kao i izraz po kome se izračunava vrednost za svaku kolonu.

Kao argumente, izraz može da uključi konstante, druge kolone, programske promenljive (u aplikativnom SQL-u), pri čemu se vrednost kolone *ime-kolone* u pojedinoj vrsti zamenjuje izračunatom vrednošću odgovarajućeg izraza za tu istu vrstu.

Ako se WHERE linija ne navede, vrednosti navedenih kolona menjaju se u svim vrstama tabele; ako se navede, vrednosti se menjaju u svakoj vrsti tabele koja zadovoljava logički izraz WHERE linije.

Primer 3.47 Uvećati tiraže svih izdanja izdavača i3 za 10%.

```
 UPDATE KI
SET   TIRAZ = TIRAZ * 1.1
WHERE I_SIF='i3'
```

Ako pri izvršavanju UPDATE iskaza dođe do greške, izvršavanje se obustavlja. Pri tome ni jedna vrsta tabele nije promenjena, odnosno ako su promene nad nekim vrstama već izvršene, biće poništene. Do greške može doći, na primer, u slučaju da je nova vrednost suviše velika za datu kolonu, s obzirom na definiciju te kolone. Izvršavanje UPDATE iskaza može da bude odbijeno i u slučaju da se njime narušava referencijalni integritet.

Ako tabela koja se ažurira ima primarni ključ koji je strani ključ drugih tabela, UPDATE iskaz nad kolonama koje uključuju primarni ključ odnosne tabele tj. strani ključ zavisne tabele izvršava se prema navedenim pravilima ažuriranja u definiciji stranih ključeva zavisnih tabela. Operacija ažuriranja definisana UPDATE iskazom će biti uspešna samo ako su pravila ažuriranja svih zavisnih tabela zadovoljena.

Pravila ažuriranja (UPDATE RESTRICT, UPDATE NO ACTION) opisana su u tački 3.2.2. Tako, ne može se promeniti vrednost kolone – primarnog ključa, za koju postoji odgovarajuća vrednost stranog ključa u zavisnoj tabeli. Takođe, ako se menja vrednost kolone – stranog ključa, nova vrednost mora postojati kao vrednost odgovarajuće kolone tabele u kojoj je ta kolona deo primarnog ključa.

Primer 3.48 Zahtev za izvršenje iskaza

```
 UPDATE KI
SET   I_SIF = 'i6'
WHERE I_SIF = 'i4'
```

biće odbijen, jer vrednost i6 ne postoji u tabeli I u kojoj je kolona I_SIF – primarni ključ.

Primer 3.49 Uvećati tiraž svih izdanja izdavača iz Jugoslavije za 10%.

```
 UPDATE KI
 SET TIRAZ = TIRAZ * 1.1
 WHERE I_SIF IN
(SELECT I_SIF
 FROM   I
 WHERE  DRZAVA = 'Jugoslavija')
```

3.3.15 Brisanje

Za brisanje vrsta iz tabele upotrebljava se iskaz oblika

```
   DELETE FROM  ime-tabele [ ime n-torne promenljive]
[ WHERE  logički-izraz ]
```

Izvršavanjem ovog iskaza iz tabele *ime-tabele* brišu se sve vrste te tabele koje zadovoljavaju logički izraz. Vrste se brišu u celosti (nije moguće izbrisati samo neke vrednosti iz date vrste). Ako se WHERE linija ne navede, brišu se sve vrste navedene tabele. Tabela može biti i pogled, pri čemu se vrste brišu iz bazne tabele nad kojom je pogled definisan.

Primer 3.50 Izbrisati iz tabele KI sva izdanja izdavača i4.

```
 DELETE FROM KI
WHERE I_SIF='i4'
```

Izvršavanjem ovog iskaza iz tabele KI briše se jedna vrsta.

Ako za vreme izvršavanja DELETE iskaza dođe do greške, izvršavanje iskaza se prekida. Sadržaj tabele neće biti promenjen. Greška može biti, na primer, rezultat narušenog referencijalnog integriteta.

Ako tabela iz koje se briše ima primarni ključ koji je strani ključ drugih tabela, DELETE iskaz se izvršava prema navedenim pravilima brisanja u definiciji stranih ključeva zavisnih tabela. Operacija brisanja definisana DELETE iskazom će biti uspešna samo ako su pravila brisanja svih zavisnih tabela zadovoljena.

Pravila brisanja (DELETE RESTRICT, DELETE CASCADE, DELETE SET NULL) opisana su u tački 3.2.2. Na primer, ne može se izbrisati knjiga iz tabele K ako postoji izdanje te knjige u tabeli KI (jer je K_SIF u tabeli KI strani ključ sa pravilom brisanja DELETE RESTRICT, a kolona K_SIF je primarni ključ u tabeli K). S druge strane, moguće je obrisati izdavača i3 iz tabele izdavača I iskazom

```
 DELETE FROM I
WHERE I_SIF='i3'
```

Izvršavanjem ovog iskaza brišu se i sve vrste u tabeli KI koje se odnose na izdavača i3, jer strani ključ I_SIF u tabeli KI ima pravilo brisanja DELETE CASCADE. Ovde se prenošenje brisanja završava jer nema tabela koje, preko svog stranog ključa, zavise od tabele KI. Ako bi postojala tabela zavisna od tabele KI, tj. od njene kolone I_SIF, i na nju bi se primenilo pravilo brisanja definisano za njen strani ključ. Prenošenje brisanja može se završiti i u slučaju da se dođe do zavisne tabele čiji strani ključ ima pravilo brisanja DELETE SET NULL.

U logičkom izrazu WHERE linije (kao i u slučaju UPDATE iskaza) može se naći i predikat koji uključuje (ugnježdeni) puni upitni SELECT blok.

Korelisani podupit u ažuriranju u brisanju. U WHERE liniji DELETE (UPDATE) iskaza može se primeniti i korelisani podupit koji se izvršava po jedan put za svaku vrstu tabele iz DELETE (UPDATE) iskaza, da bi se odredilo da li se ta vrsta briše (ažurira) ili ne.

Primer 3.51 Izbrisati izdavače koji izdaju knjigu sa šifrom k3.

```
     DELETE FROM I
    WHERE 'k3' IN
(SELECT KI.K_SIF
 FROM   KI
 WHERE  KI.I_SIF = I.I_SIF)
```

3.4 Aplikativni SQL

Većina SQL proizvoda podržava dva vida izvršavanja SQL iskaza: interaktivni – izvršavanje samostalnih SQL iskaza preko on-line terminala, i aplikativni – izvršavanje SQL iskaza umetnutih u program na višem programskom jeziku, pri čemu se SQL iskazi mogu izvršavati naizmenično sa iskazima programskog jezika. Program na aplikativnom SQL-u prolazi kroz prekompiliranje kao prvu fazu svoje obrade. Aplikativni SQL, dalje, može da uključi dve vrste SQL iskaza:

- *statičke* SQL iskaze koji već u fazi pisanja programa imaju precizan oblik i čija se analiza (utvrđivanje tipa iskaza i objekata koji u njemu učestvuju – npr. tabela, pogleda, kolona, itd.) i priprema za izvršenje može u potpunosti obaviti u fazi prekompiliranja;
- dinamičke SQL iskaze koji se zadaju u obliku niske znakova koja se dodeljuje programskoj promenljivoj, i čija se analiza i priprema može obaviti tek u fazi izvršavanja programa kada je poznat njihov precizni oblik.

Pored ovakvog načina korišćenja baze podataka u programu na programskom jeziku, postoje i druge mogućnosti ugradnje blokova za komunikaciju sa SUBP u program. U slučaju sistema DB2, te mogućnosti su direktni pozivi DB2 funkcija (Call Level Interface – CLI) i upotreba ODBC standarda (Open Database Connectivity). ODBC standard odnosi se na standardizovanje sumeđe korisnika i međusistema (prema SUBP), nezavisno od SUBP kome će međusistem proslediti korisnički zahtev. Pri tome, nemoguće je koristiti specifičnosti pojedinog sistema (u ovom slučaju DB2), a svi upiti upućeni sistemu DB2 preko ODBC-a izvršavaju se dinamički. S druge strane, direktni pozivi DB2 funkcija (CLI-sistem) su specifični za DB2 sistem i koriste sve njegove mogućnosti (kao i aplikativni SQL), mada su maksimalno usaglašeni sa ODBC pristupom (v. **??**). Za razliku od programa na aplikativnom SQL-a koji mora da se prekompilira, kompilira, poveže sa bazom i najzad izvrši, DB2 CLI aplikacija ne mora ni da se prekompilira ni da se povezuje sa bazom; ona koristi standardni skup funkcija za izvršenje SQL iskaza u vreme izvršenja programa. Na taj način DB2 CLI aplikacija postaje "prenosivija" u odnosu na aplikaciju u aplikativnom SQL-u (bar kada su razne DB2 platforme u pitanju), ali je osnovna prednost aplikativnog SQL-a u tome što može da koristi statički SQL i što je u značajnoj meri standardizovan pa se može koristiti (u manjoj ili većoj meri) i među različitim SUBP-platformama.

U slučaju da aplikacija zahteva prednosti oba mehanizma (DB2 CLI i aplikativnog SQL-a), moguće je koristiti statički SQL unutar DB2 CLI aplikacije kreiranjem *zapamćenih procedura* napisanih na statičkom SQL-u. Zapamćena procedura

se poziva iz DB2 CLI aplikacije i izvršava se na serveru. Jednom napisanu zapamćenu proceduru može da pozove bilo koja DB2 CLI ili ODBC aplikacija. Naime, aplikacija može biti projektovana tako da se izvršava u dva dela – jedan na klijentu a drugi na serveru. Zapamćena procedura je deo koji se izvršava na serveru, a može biti napisana i korišćenjem DB2 CLI funkcija. Zapamćene procedure imaju niz prednosti, od smanjenja mrežnog saobraćaja pri prenosu podataka do koncepta učaurenja (v. A) koji povećava bezbednost podataka. O konceptu i implementaciji direktnih poziva DB2 funkcija, ODBC standarda i zapamćenih procedura u sistemu DB2 može se više detalja naći u priručniku [**?**].

SQL kakav je opisan u prethodnim odeljcima odnosi se na interaktivni vid izvršavanja – tzv. interaktivni SQL. U ovom i sledećem odeljku biće opisan aplikativni SQL i to statički i dinamički SQL, redom.

Potreba za umetanjem SQL-a u viši programski jezik javlja se zbog odsustva kontrolnih struktura u upitnom jeziku, često neophodnih pri obradi podataka iz baze podataka. Umetanjem (tzv. *ugnježdenjem*) SQL-a u viši programski jezik (tzv. matični jezik – engl. host language), kao što su C, FORTRAN, COBOL, PL/I, Pascal, Ada ili Assembler, dobija se aplikativni SQL, na kome se mogu pisati kompleksni programi (aplikacije) za najraznovrsnije obrade. Osnovni princip svih aplikativnih SQL jezika je princip dualnosti, prema kome se svaki interaktivni SQL iskaz može izraziti i u aplikativnom SQL-u, ali obratno ne važi (što i opravdava ovakva umetanja). Svaki program na aplikativnom SQL-u obrađuje se u nekoliko faza; o njima će biti reči u odeljku 18.1.4 – Priprema i izvršenje aplikativnog programa.

Izvršavanje SQL upita (interaktivnog ili aplikativnog) proizvodi uvek tabelu, koja u opštem slučaju sadrži veći broj redova. U slučaju aplikativnog SQL-a, u kome se rezultat izvršavanja upita predaje programu na dalju obradu, ta skupovna priroda rezultata SQL upita je u suprotnosti sa "slogovnom" prirodom programskih jezika (oni mogu da obrađuju samo po jedan slog, red ili podatak u jednom trenutku). Mehanizam kojim se premošćuje ova razlika između programske i upitne (slogovne i skupovne) komponente aplikativnog SQL-a zove se *kursor*. Kursor je oblik pokazivača na vrstu u rezultujućoj tabeli, koji može da prelazi preko svake vrste, ostvarujući njihovu programsku obradu jednu po jednu. Ukoliko rezultat izvršavanja upita jeste tabela sa jednom vrstom, kursor nije potreban; tada se u SELECT iskaz uvodi još jedan red (INTO linija) kojim se navodi u koje se programske promenljive (ili u koju se programsku strukturu) smeštaju komponente jedine vrste tabele – rezultata.

Neke od osnovnih karakteristika ugnježdenja SQL-a u matični jezik su:

1. Svaki umetnuti SQL iskaz počinje rezervisanim prefiksom EXEC SQL koji omogućuje prekompilator lakše razlikovanje SQL iskaza od iskaza matičnog jezika.

2. SQL iskaz može da uključi matične promenljive (promenljive matičnog jezika) tako što im dodeljuje vrednost, ili tako što njihovu vrednost upisuje u bazu. Svaka matična promenljiva (ili struktura) koja se koristi u SQL iskazu mora biti deklarisana pre korišćenja; deklarisanje se vrši u okviru posebne sekcije koja počinje deklarativnim iskazom

   ```
   EXEC SQL BEGIN DECLARE SECTION
   ```

   a završava se deklarativnim iskazom

   ```
   EXEC SQL END DECLARE SECTION
   ```

   Ovakvih sekcija može biti više u programu.

3. Obraćanju matičnoj promenljivoj u okviru SQL iskaza prethodi dvotačka, koja razlikuje tu promenljivu od imena atributa SQL-a.

   Matična promenljiva se može naći bilo gde u SQL iskazu gde može i konstanta, kao i u INTO liniji, pri čemu mora da se uskladi tip matične promenljive i odgovarajućeg atributa sa kojim se poredi, ili kojoj se vrednost atributa dodeljuje. Matična promenljiva i atribut mogu imati isto ime. Na primer,

   ```
   EXEC SQL SELECT NASLOV, OBLAST
    INTO  :KNJIGA, :OBLAST
    FROM  K
    WHERE K_SIF = :DATI_K_SIF
   ```

   Kako je atribut K_SIF primarni ključ, rezultat prethodnog upita je samo jedna n-torka, tj. tabela sa jednom vrstom čije se vrednosti atributa NASLOV, OBLAST upisuju, redom, u programske promenljive KNJIGA, OBLAST.

   U slučaju da je vrednost atributa OBLAST u n-torci iz prethodnog primera bila NULL, došlo bi do greške pri izvršenju SELECT iskaza, jer programska promenljiva OBLAST ne može da primi NULL vrednost. Zato se uz matične promenljive uvode *indikatorske promenljive* koje "amortizuju" ovu nesaglasnost tipa matične promenljive i NULL vrednosti atributa. Sada je obraćanje matičnoj promenljivoj oblika :V1:V2,[8] gde je V1 – matična promenljiva, a V2 – indikatorska promenljiva. U slučaju SELECT – INTO iskaza, ako je vrednost atributa u SELECT liniji NULL, vrednost matične promenljive V1 se neće promeniti, a indikatorska promenljiva V2 će dobiti negativnu vrednost.

   Prethodni SELECT iskaz bi, uz korišćenje indikatorske promenljive, imao oblik:

[8] Puno obraćanje je oblika :V1 INDICATOR :V2.

```
EXEC SQL SELECT NASLOV, OBLAST
 INTO  :KNJIGA, :OBLAST:O1
 FROM  K
 WHERE K_SIF = :DATI_K_SIF
```

U ovom slučaju ne dolazi do greške u izvršenju SELECT iskaza, čak i ako atribut OBLAST ima vrednost NULL. Za matičnu promenljivu KNJIGA nije potrebno uvoditi indikatorsku promenljivu jer se promenljivoj KNJIGA dodeljuje vrednost atributa NASLOV koji je definisan opcijom NOT NULL WITH DEFAULT; ovaj atribut ne uzima NULL vrednost (eventualno podrazumevanu vrednost – belinu).

4. Program koji sadrži izvršne SQL iskaze komunicira sa sistemom DB2 preko memorijskog prostora koji se zove SQL prostor za komunikaciju (engl. SQL Communication Area, SQLCA). SQLCA je struktura koja se ažurira posle izvršenja svakog SQL iskaza. U ovu strukturu upisuju se informacije o izvršenom SQL iskazu.[9]

   Najčešće korišćena promenljiva SQLCA strukture je SQLCODE. To je indikator uspešnosti izvršenja SQL iskaza, koji može da dobije sledeće vrednosti:

   – 0, ako je izvršenje uspešno (u prethodnom primeru – ako je nađena tačno jedna vrsta u rezultujućoj tabeli),
   – pozitivnu vrednost, ako se pri uspešnom izvršenju dogodilo nešto izuzetno (npr. +100 ako je rezultujuća tabela prazna),
   – negativnu vrednost, ako se zbog nastale greške iskaz nije izvršio uspešno.

   SQLCA prostor se u program uključuje iskazom

   ```
   EXEC SQL INCLUDE SQLCA
   ```

   koji mora da se navede pre prvog izvršnog SQL iskaza.

   Provera statusa izvršenja SQL iskaza, sačuvanog u promenljivoj SQLCODE, zadaje se direktivom prekompilatoru da umetne odgovarajuće IF – THEN iskaze u program. Direktiva je deklarativni iskaz oblika

   ```
   EXEC SQL WHENEVER  uslov   akcija
   ```

   uslov može biti:

   – NOT FOUND (nije nađen – drugi zapis za SQLCODE = 100),

[9] I program za interaktivno izvršenje SQL iskaza komunicira sa sistemom DB2 preko SQLCA. Tako se i pri interaktivnom izvršenju SQL iskaza, razmatranom u prethodnim odeljcima, u SQLCA upisuju informacije o izvršenju tih iskaza.

– SQLERROR (indikator greške – drugi zapis za SQLCODE < 0) ili
– SQLWARNING (indikator upozorenja – drugi zapis za SQLCODE > 0 AND SQLCODE <> 100);

akcija je

– CONTINUE – program nastavlja sa izvršenjem ili
– GOTO *obeležje* – skok na deo programa u kome se nastavlja obrada, npr. izveštavanje o uzroku nastalog uslova.

Dakle, deklarativni iskaz WHENEVER omogućuje programeru da zada način na koji će se proveravati SQLCODE posle svakog izvršnog SQL iskaza. Ovakvih iskaza može biti veći broj u programu. Svaki WHENEVER iskaz odnosi se na sve SQL iskaze (u redosledu njihovog zapisa, ne izvršenja) do navođenja sledećeg WHENEVER iskaza.

Na primer, iskaz oblika

```
EXEC SQL WHENEVER SQLERROR CONTINUE
```

obezbediće nastavak izvršenja programa i u slučaju da je pri izvršavanju SQL iskaza došlo do greške. To znači da program neće imati prinudni završetak, ali, sa druge strane, to obavezuje programera da posle svakog SQL iskaza u programu eksplicitno ispituje vrednost promenljive SQLCODE; u slučaju greške kontrolu upravljanja treba preneti na deo kojim se greška obrađuje.

Osim promenljive SQLCODE, SQLCA struktura uključuje i veći broj drugih promenljivih. Najvažnija među njima je promenljiva SQLSTATE tipa CHAR[5] koja slično promenljivoj SQLCODE sadrži kôd rezultata izvršavanja SQL iskaza (npr. prva dva karaktera '01' promenljive SQLSTATE predstavljaju grupu kodova – upozorenja i ukazuju, slično SQLCODE>0, da je došlo do pojave koja rezultuje upozorenjem). Tako, uslov NOT FOUND u direktivi WHENEVER predstavlja, u stvari, disjunkciju uslova vrednosti promenljivih SQLCODE i SQLSTATE: SQLCODE=100 OR SQLSTATE='02000'. Među ostalim zanimljivijim promenljivim SQLCA strukture je niz celih brojeva SQLERRD[6] koji se odnosi na razne vrste informacija o nastaloj grešci ili rezultatu izvršavanja SQL iskaza, kao i skup CHAR(1) promenljivih SQLWARN0 – SQLWARN10 koje sadrže informacije o uzroku upozorenja. Značenje nekih od ovih elemenata SQLCA strukture je sledeće (opis kompletne SQLCA strukture može se naći u [37]):

– SQLERRD[3] sadrži broj vrsta obrađenih (unetih, ažuriranih ili izbrisanih) izvršavanjem poslednjeg INSERT, UPDATE ili DELETE iskaza (ne računaju se vrste izbrisane kao posledica kaskadnog brisanja; v. prethodni odeljak);

- SQLERRD[5] sadrži ukupan broj izbrisanih, unetih ili ažuriranih vrsta, direktno ili indirektno, izvršavanjem odgovarajućeg iskaza;
- SQLwarn0 sadrži 'W' ako bar jedna od preostalih SQLwarn*i* promenljivih sadrži upozorenje ('W'); tako uslov SQLWARNING u direktivi WHENEVER predstavlja disjunkciju uslova (SQLCODE >0 AND SQLCODE <>100) OR SQLWARN0 = 'W';
- SQLWARN1 sadrži 'W' ako je pri dodeli promenljivoj matičnog jezika odesečena vrednost kolone tipa niske znakova (CHAR(m));
- SQLWARN2 sadrži 'W' ako je došlo do eliminacije NULL vrednosti pri primeni agregatne funkcije.

5. Ugnježdeni SQL, pored izvršnih SQL iskaza (koje sadrži i interaktivni SQL), sadrži i deklarativne SQL iskaze. Takav je, npr. iskaz EXEC SQL DECLARE CURSOR koji se ne izvršava, već pomaže prekompilatoru u izvođenju sintaksnih provera. Deklarativni SQL iskazi se pišu u delu programa predviđenom za deklaracije matičnog jezika (npr. na početku programa), dok se izvršni SQL iskazi mogu navesti na bilo kom mestu u programu gde se mogu navesti i izvršni iskazi matičnog jezika. Svaki kursor koji se koristi u programu mora se prethodno deklarisati. Tako se sledećim deklarativnim iskazom može uvesti kursor nad rezultatom navedenog upita:

```
EXEC SQL DECLARE X CURSOR FOR
  SELECT I_SIF, NAZIV, STATUS
  FROM   I
  WHERE  DRZAVA = :Y
```

Ako će se kursor koristiti samo za čitanje vrsta tabele – rezultata izvršavanaja SELECT iskaza, uz deklaraciju kursora može se navesti i opcija FOR FETCH ONLY (FOR READ ONLY). To omogućuje racionalniji pristup podacima većem broju korisnika jer sistem za upravljanje bazama podataka "zna" da će se podaci samo čitati i da ne zahtevaju ekskluzivan pristup.

Ukoliko je potrebno ažurirati neke kolone rezultata SELECT iskaza nad kojim je deklarisan kursor, kursor se deklariše kao kursor za ažuriranje nad navedenim kolonama, dodavanjem linije

`FOR UPDATE OF` *lista-kolona*

Tabela – rezultat SELECT iskaza nad kojim je deklarisan kursor ne može se ažurirati (tj. kursor se ne može deklarisati kao kursor za ažuriranje) ako SELECT iskaz uključuje neku od sledećih konstrukcija:

- DISTINCT opciju u spoljašnjoj SELECT liniji;

- skupovnu (UNION, INTERSECT, EXCEPT) operaciju osim UNION ALL;
- agregatnu funkciju u spoljašnjoj SELECT liniji;
- GROUP BY ili HAVING liniju u spoljašnjem SELECT iskazu;
- ORDER BY liniju;
- podupit nad istom tabelom;
- dve ili više tabela u FROM liniji
- tabelu-pogled (u FROM liniji) koja se ne može ažurirati;
- FOR FETCH ONLY (FOR READ ONLY) opciju.

Ako se kursor koristi samo za pretraživanje vrsta rezultata SELECT iskaza, vrste se mogu pretraživati u specifičnom redosledu navođenjem ORDER BY linije.

Pre nego što se prethodno deklarisanim kursorom može "uzeti" (FETCH) pojedina vrsta rezultujuće tabele, i zatim izvršiti obrada te vrste, potrebno je nad kursorom izvršiti operaciju *otvaranja* (OPEN). Time se upit kojim je deklarisan kursor izvršava, a pokazivač vrste priprema za pozicioniranje na prvu vrstu rezultata. Po završenoj obradi i poslednje vrste vrši se operacija *zatvaranja kursora* (CLOSE). Na primer,

```
EXEC SQL OPEN X;      /* otvaranje kursora */
WHILE (SQLCODE == 0)  /* ima još vrsta i FETCH se uspešno izvršava */
  { EXEC SQL FETCH X INTO :I_SIF, :NAZIV, :STATUS;
      /* "uzimanje" sledece vrste */
     ...
  }
EXEC SQL CLOSE X
```

Ovi iskazi se navode u proceduralnom delu programa. Otvaranjem kursora (OPEN X) SELECT iskaz iz deklaracije kursora se izvršava, a zatim se, iskazom FETCH X, pretražuje jedna po jedna vrsta, redom, iz tabele koja je dobijena izvršavanjem SELECT iskaza. Vrednosti komponenti pretražene vrste dodeljuju se programskim promenljivim I_SIF, NAZIV, STATUS. I FETCH iskaz, kao i SELECT INTO, može da uključi indikatorske promenljive gdegod je to potrebno, pa je prethodni primer FETCH iskaza bolje zapisati u obliku

```
EXEC SQL FETCH X INTO :I_SIF, :NAZIV, :STATUS:IND;
```

FETCH iskaz je jedini iskaz kojim se može pomerati kursor. Pošto, u opštem slučaju, rezultujuća tabela sadrži veći broj vrsta, pretraživanje vrste iz rezultujuće tabele obično se stavlja u petlju (WHILE, na primer, u C-u). Petlja se ponavlja sve

dok ima vrstâ u rezultujućoj tabeli. Po izlasku iz petlje kursor se zatvara (CLOSE X).

Ako je kursor X u jednom trenutku pozicioniran na neku vrstu ("tekuća" vrsta kursora X, u oznaci CURRENT OF X), ta vrsta može se ažurirati ili izbrisati, iskazima UPDATE ili DELETE – CURRENT OF X. Sintaksa tih iskaza ilustrovana je sledećim primerom:

```
EXEC SQL UPDATE I
 SET STATUS = STATUS + :POVECANJE
 WHERE CURRENT OF X
```

Jedan program može imati veći broj deklaracija (različitih) tabela i kursora. Na primer, povećanje statusa (za 10%) svim izdavačima koji su izdali knjigu sa šifrom koja se nalazi u programskoj promenljivoj "knjiga", može se izraziti programskom strukturom sledećeg oblika:

```
#include <stdio.h>
#include <stdlib.h>



EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
     char knjiga[5];
     char i[6];
     char naziv[20];
     short status;
     char drzava[20];
     short n_ind, s_ind, d_ind;
EXEC SQL END DECLARE SECTION;


void greska(char *poruka)
{
     ⋮
     printf("SQL greska, SQLCODE = %i, %s\n ", SQLCODE, poruka);
     EXEC SQL ROLLBACK;
     exit(0);
}

main()
{
     EXEC SQL CONNECT to izdavastvo;
```

```
    if (SQLCODE <> 0) greska(" pri otvaranju konekcije");
    EXEC SQL DECLARE Z CURSOR FOR
    SELECT I_SIF, NAZIV, STATUS, DRZAVA
    FROM   I
    WHERE  EXISTS
     (SELECT *
      FROM   KI
      WHERE  KI.I_SIF = I.I_SIF AND KI.K_SIF = :knjiga)
    FOR UPDATE OF STATUS

    EXEC SQL WHENEVER NOT FOUND  CONTINUE;
    EXEC SQL WHENEVER SQLERROR   CONTINUE;
    EXEC SQL WHENEVER SQLWARNING CONTINUE;

    ⋮
    EXEC SQL OPEN Z;
    if (SQLCODE <> 0) greska(" pri otvaranju kursora Z");
    EXEC SQL FETCH Z INTO :i, :naziv:n_ind, :status:s_ind, :drzava:d_ind;
    if (SQLCODE == 100)
greska(" nema izdavaca koji su izdali datu knjigu");
    if (SQLCODE <>0)
greska(" pri prvom uzimanju vrste o trazenom izdavacu");
    while(SQLCODE <> 100)
    {
printf("naziv izdavaca je %s", naziv);
EXEC SQL UPDATE I
SET STATUS = STATUS * 1.1
WHERE CURRENT OF Z;
if ((SQLCODE <> 0))
     greska(" pri azuriranju tekuceg izdavaca");
EXEC SQL FETCH Z INTO :i, :naziv:n_ind, :status:s_ind, :drzava:d_ind;
if ((SQLCODE <> 0) && (SQLCODE <> 100))
     greska(" pri uzimanju (FETCH) sledeceg izdavaca");
    }

    EXEC SQL CLOSE Z;
    if (SQLCODE <> 0) greska(" pri zatvaranju kursora Z");
    EXEC SQL COMMIT;
    if (SQLCODE <> 0) greska(" pri operaciji COMMIT");
    EXEC SQL CONNECT RESET;
    if (SQLCODE <> 0) greska(" pri zatvaranju konekcije");
}
```

Značenje iskaza CONNECT i CONNECT RESET je uspostavljanje i zatvaranje konekcije sa bazom podataka (u ovom primeru "izdavastvo") koja je na serveru,

u slučaju klijent/server RSUBP (v. 18). Pre pripreme i izvršenja ovog programa neophodno je da je takva baza kreirana.

Značenje iskaza COMMIT je da se, u slučaju uspešnog izvršenja jednog ili više SQL iskaza, promene koje ti iskazi prouzrokuju i koje su privremenog karaktera, trajno upišu u bazu. Značenje iskaza ROLLBACK je da se, u slučaju greške u nekom koraku izvršavanja jednog ili više SQL iskaza, sve privremene promene ponište. Greška u izvršenju može da nastane, na primer, zbog uslova prekoračenja. Program može zatim normalno da završi sa radom, kao u prethodnom primeru (**exit(0)**) u kome moduo **greska** obavlja i funkciju ROLLBACK-a, ili može da nastavi sa izvršenjem sledeće grupe svojih iskaza – sledeće *transakcije* (v. deo **??** – Upravljanje transakcijama). Izvršavanjem COMMIT odnosno ROLLBACK iskaza zatvaraju se i svi otvoreni kursori.

3.5 Dinamički SQL

Kao što je naglašeno u prethodnom odeljku, svaki ugnježdeni statički SQL iskaz u fazi prekompilacije sasvim je precizno određen u smislu da je poznat njegov tip (npr. SELECT, UPDATE, INSERT), i da su poznata imena tabela i kolona koje u njemu učestvuju; SQL iskaz se zamenjuje pozivom (u matičnom jeziku) odgovarajuće CLI procedure, ali se izvršavanjem programa taj poziv (pa dakle i sâm SQL iskaz) ne menja.

Za razliku od ovog, statičkog SQL-a, u *dinamičkom* SQL-u iskazi nisu poznati u vreme pisanja aplikativnog programa i preciziraju se tek u fazi izvršavanja programa. Takvi SQL iskazi, umesto da se eksplicitno navode u programu, zadaju se u obliku niske znakova koja se dodeljuje matičnoj promenljivoj. SQL iskaz se tada izgrađuje na osnovu podataka programa (npr. vrednost odgovarajuće matične promenljive može da se dobije preko terminala). Kako vrednost matične promenljive može da se menja u toku izvršavanja programa, to i SQL iskaz dodeljen toj matičnoj promenljivoj takođe može da se menja u toku izvršavanja programa.

Da bi dinamički SQL iskaz mogao da se izvrši, potrebno je pripremiti ga za izvršenje ugnježdenim (statičkim) iskazom PREPARE. Zatim se dinamički iskaz može izvršiti ugnježdenim (statičkim) iskazom EXECUTE.

Primer 3.52 Neka je V1 matična promenljiva tipa niske znakova, koja je u programu dobila vrednost – nisku znakova nekog izvršnog SQL iskaza. Tada iskaz

```
EXEC SQL PREPARE  S1 FROM :V1
```

kreira izvršni SQL iskaz čije je ime S1, od niske znakova kojom je iskaz zadat u matičnoj promenljivoj V1. Posle provere uspešnosti izvršenja PREPARE iskaza može se veći broj puta izvršiti EXECUTE iskaz oblika

```
EXEC SQL EXECUTE S1
```

Dinamički se mogu pripremiti i izvršiti samo izvršni SQL iskazi, i to najveći deo njih (ne svi). Neki od tih iskaza su: ALTER, COMMIT, CREATE, DELETE, DROP, INSERT, ROLLBACK, SELECT, SET, UPDATE. Potpuna lista ovih iskaza može se naći u [37].

Moguće je koristiti i ugnježdeni (statički) iskaz EXECUTE IMMEDIATE da bi se u jednom koraku pripremio i izvršio dinamički SQL iskaz (primenjuje se obično kada je dinamički SQL iskaz potrebno izvršiti samo jedanput).

Primer 3.53 Ako vrednost matične promenljive V1 jeste niska znakova “DELETE FROM KI WHERE K_SIF = ’k1’ ”, onda iskaz

```
EXEC SQL EXECUTE IMMEDIATE :V1
```

izvršava DELETE iskaz nad tabelom KI, pri čemu se brišu sva izdanja knjige sa šifrom k1.

Dinamički SQL iskaz ne sme da sadrži obraćanja matičnim promenljivim. Umesto toga mogu se koristiti oznake parametara. Oznaka parametra je znak pitanja (?) i navodi se na mestu na kome bi se navela matična promenljiva kada bi iskaz bio statički. U vreme izvršavanja pripremljenog SQL iskaza, oznake parametara se zamenjuju tekućim vrednostima matičnih promenljivih (ili strukture) koje se navode u EXECUTE iskazu.

Primer 3.54 Ako je vrednost matične promenljive V1 niska znakova "INSERT INTO K VALUES (?, ?, ?)", posle izvršenja iskaza

```
EXEC SQL PREPARE K_INSERT FROM :V1
```

može se izvršiti iskaz

```
EXEC SQL EXECUTE K_INSERT USING :KSIFRA, :KNASLOV, :KOBLAST
```

KSIFRA, KNASLOV i KOBLAST su matične promenljive (tipa CHAR(5), CHAR(50), CHAR(20), redom) koje odgovaraju shemi tabele K, tj. atributima K_SIF, NASLOV, OBLAST, redom.

Kada se, korišćenjem EXECUTE ili EXECUTE IMMEDIATE iskaza, izvrši neki dinamički SQL iskaz, uspešnost izvršenja tog dinamičkog SQL iskaza registruje se u promenljivoj SQLCODE prostora SQLCA, i može se ispitivati na isti način kao i u statičkom SQL-u.

Među izvršnim SQL iskazima koji ne mogu da se izvršavaju dinamički nalaze se, na primer, i sledeći iskazi:

CLOSE	FETCH
DECLARE	OPEN
EXECUTE	PREPARE
EXECUTE IMMEDIATE	WHENEVER

Dinamički pripremljeni SELECT iskaz se može izvršiti korišćenjem kursora, tako što se izvrše sledeći koraci (u slučaju fiksne liste atributa):

1. priprema iskaza
2. deklarisanje kursora nad imenom iskaza
3. otvaranje kursora
4. uzimanje vrsta iz rezultujuće tabele

5. zatvaranje kursora.

Primer 3.55 Neka program pretražuje šifre knjiga i šifre izdavača iz tabele KI, dinamičkim izvršavanjem SELECT iskaza oblika

```
SELECT K_SIF, I_SIF
FROM   KI
WHERE  ...
```

Iskaz se čita sa terminala, pri čemu korisnik zadaje WHERE liniju. Neka je iskaz dodeljen matičnoj promenljivoj tipa niske znakova, KI_NISKA. Izvršenje dinamičkog iskaza realizovaće se izvršenjem sledećih statičkih iskaza:

```
EXEC SQL PREPARE ISKAZ FROM :KI_NISKA;
   ...
EXEC SQL DECLARE C1 CURSOR FOR ISKAZ;
   ...
EXEC SQL OPEN C1;
WHILE ...
{   ...
     EXEC SQL FETCH C1 INTO :K_BR, :I_BR;
     ...
}
...
EXEC SQL CLOSE C1;
```

Dinamički SQL nudi potpunu fleksibilnost programiranja. U kompleksnom programu koji treba da izvršava različite tipove i strukture SQL iskaza, teško je (ili čak nemoguće) modelirati sve te iskaze. Tada je pogodno koristiti dinamički SQL. Takav program je svaki program za interaktivno izvršavanje SQL iskaza koji može da prihvati i izvrši gotovo svaki SQL iskaz. Cena ove fleksibilnosti je dodatna složenost programiranja, kao i smanjena brzina izvršavanja programa. Posebno je složeno dinamičko izvršavanje SQL SELECT iskaza koji ima promenljivu SELECT listu ([38]).

3.6 Autorizacija i prava korisnika baze podataka

Da bi baza podataka mogla bezbedno i efikasno da se koristi, svaki korisnik mora imati *autorizaciju* (engl. authorization) za pristup sistemu za upravljanje bazama podataka, tj. svakom korisniku mora da bude dodeljeno korisničko ime kojim se predstavlja tom sistemu.

Pored ovog, prvog nivoa sigurnosti baze, SUBP obično podržava i više nivoe bezbednosti. Na primer, različitim korisnicima ili grupama korisnika može se omogućiti da obavljaju različite opšte zadatke nad bazama podataka, kao što su

povezivanja sa bazom, kreiranje tabela ili administriranje sistema. Za korisnike se tada kaže da se razlikuju po *nivoima autorizacije*. S druge strane, različitim korisnicima ili grupama korisnika mogu se dodeliti različita *prava* (engl. privilegies) pristupa i izvršenja specifičnih operacija nad specifičnim objektima u bazi. Na primer, da bi jedan korisnik mogao da menja podatke u nekoj tabeli, on mora imati pravo (dodeljeno eksplicitno ili implicitno) za promenu podataka u toj tabeli. Upravljanje autorizacijom i pravima je jedna od funkcija SUBP.

Sistem autorizacije i prava pristupa i manipulisanja objektima predstavlja hijerarhiju. Autorizacija višeg nivoa uključuje mogućnosti autorizacije nižeg nivoa, dok neka prava uključuju skup drugih prava; na primer, pravo ažuriranja uključuje i pravo čitanja podataka. Postoji i veza između nivoa autorizacije i prava kojima raspolaže korisnik sa tim nivoom autorizacije.

Nivo autorizacije dodeljuje se korisniku eksplicitno. Pravo (ili skup prava) na određenom objektu baze podataka korisniku se može dodeliti eksplicitno ili implicitno. Za eksplicitnu dodelu koristi se SQL iskaz GRANT čija je sintaksa oblika

`GRANT` *lista-prava* `ON` *objekat* `TO` *lista-korisnika*

Na primer, jedno pravo nad bazom podataka je CONNECT (pristup bazi), nad tabelom i indeksom – ALTER (promena), DELETE (pravo brisanja slogova), SELECT (pravo postavljanja upita).

Da bi se korisniku kome se dodeljuje pravo nad jednim objektom omogućilo da to pravo prenese na trećeg korisnika ("prenosivo pravo"), ono se mora dodeliti GRANT iskazom sa opcijom WITH GRANT OPTION. Na primer, korisniku STUDENT mogu se dodeliti prava čitanja i unosa u tabelu *tabela*, sa pravom da ta prava prenese, iskazom

```
GRANT SELECT, INSERT ON  tabela TO USER STUDENT
      WITH GRANT OPTION
```

Implicitno se dodela prava grupi korisnika vrši dodelom nivoa autorizacije (npr. korisnik sa najvišim nivoom autorizacije ima sva prava nad svim objektima baza), izvršavanjem nekih operacija (npr. operacija kreiranja baze podataka obezbeđuje svim korisnicima prava povezivanja na tu bazu i kreiranja tabela u njoj), ili izvođenjem iz opštijeg prava (na primer, pravo ažuriranja objekta implicira pravo čitanja vrednosti objekta).

Prethodno dodeljeno (implicitno ili eksplicitno) pravo nad pojedinim tipovima objekata u bazi podataka može se oduzeti korisniku SQL iskazom REVOKE oblika

`REVOKE` *lista-prava* `ON` *objekat* `FROM` *lista-korisnika*.

Na primer,

```
REVOKE SELECT, INSERT  ON  tabela FROM STUDENT
```

U sistemu DB2 postoje dva nivoa autorizacije upravljanja bazama podataka i dva nivoa autorizacije sistemske kontrole (v. sliku 3.1). Autorizacija upravljanja podacima ima odgovornost za bezbednost i integritet podataka. Najviši nivo autorizacije celokupnog sistema jeste autorizacija za administriranje SUBP (SYSADM) koja obezbeđuje kontrolu nad svim resursima (alatima i objektima) upravljača bazama podataka, i uključuje sva niže nivoe autorizacije – DBADM, SYSCTRL, SYSMAINT, kao i pravo da dodeli ili oduzme DBADM autorizaciju.

Drugi nivo autorizacije upravljanja bazama podataka (DBADM) vezan je za administriranje specifične baze podataka. On uključuje prava kreiranja objekata, izdavanja SQL komandi i pristup podacima u svakoj tabeli. Ovaj nivo autorizacije uključuje pravo dodele ili oduzimanja najopštijeg prava nad specifičnim objektom (CONTROL prava) kao i pojedinačnih prava.

Autorizacija nivoa sistemske kontrole odnosi se samo na operacije koje se tiču sistemskih resursa i ne dopušta direktan pristup podacima. Najviši nivo sistemske kontrole (SYSCTRL) omogućuje prava kreiranja, ažuriranja i uklanjanja baze podataka odnosno prostora tabela. Drugi nivo sistemske kontrole – nivo održavanja sistema (SYSMAINT), uključuje prava održavanja baze podataka – ažuriranje konfiguracije baze podataka, pravljenje sigurnosnih kopija baze podataka ili prostora tabela, restauraciju i praćenje postojeće baze podataka.

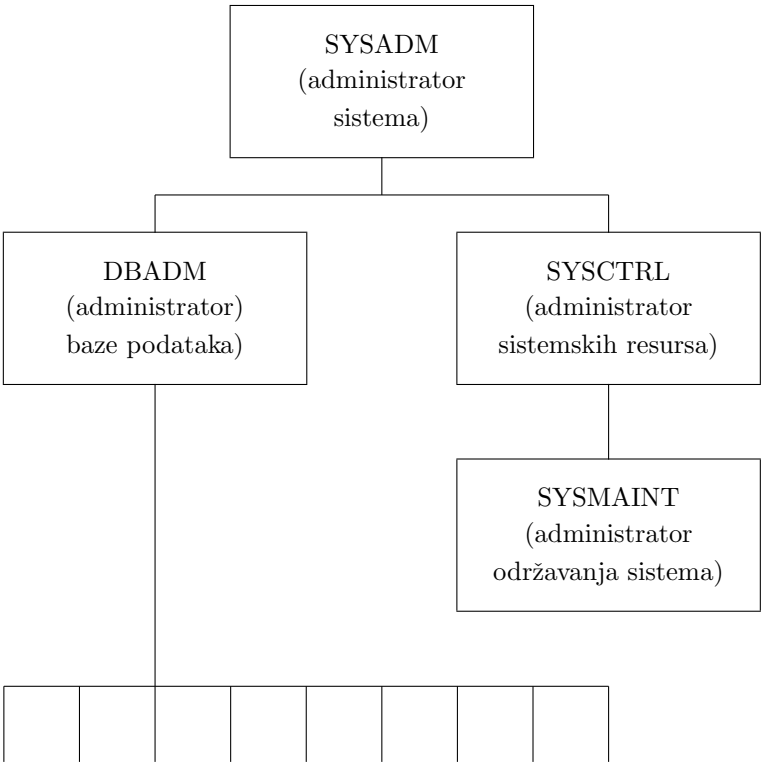
U hijerarhiji prava, najviši nivo pripada CONTROL pravu nad objektom baze podataka koje uključuje punu kontrolu nad tim objektom, kao i pravo dodele (GRANT) i oduzimanja (REVOKE) svih prava (osim CONTROL) nad tim objektom drugim korisnicima. Na primer, kreiranjem tabele odnosno pogleda stiče se automatski CONTROL pravo nad tom tabelom, kao i nad pogledom ako se poseduje CONTROL pravo nad svim tabelama i pogledima nad kojima je taj pogled definisan.

U sistemu DB2, iskazom GRANT mogu se dodeliti, kako prava, tako i nivoi autorizacije. Tako, nivo autorizacije DBADM može dodeliti samo korisnik sa SYSADM autorizacijiom. Za dodelu ostalih nivoa autorizacije potrebno je da korisnik ima neku od autorizacija administriranja SUBP. Ako je *objekat* u sintaksi GRANT iskaza – baza podataka (DATABASE), moguća prava vezana su za bazu u celini (npr. CONNECT za povezivanje na bazu, CREATETAB za kreiranje tabela u bazi, itd.), inače, ako je *objekat* – neki objekat baze podataka (npr. tabela, pogled, shema, indeks, itd.), *lista prava* koja se mogu nad tim objektom dodeliti korisnicima zavisi od vrste objekta (v. [**?**]). Dodeljeno pravo je "prenosivo" (opcijom WITH GRANT OPTION) u sistemu DB2 samo za neke vrste objekata (npr. sheme, tabele, poglede).

Lista-korisnika može biti lista pojedinačnih korisnika (USER sa navedenim imenima), grûpa korisnika (GROUP sa navedenim imenom grupe) ili svih korisnika (PUBLIC).

Na primer,

```
GRANT UPDATE ON TABLE I TO PUBLIC
```



Slika 3.1: Hijerarhija nivoa autorizacije i prava

svim korisnicima daje pravo ažuriranja tabele I u tekućoj bazi podataka. Ovaj iskaz može da izvrši korisnik koji ima bar DBADM autorizaciju, CONTROL pravo nad tabelom I, ili mu je pravo UPDATE nad tabelom I dodeljeno uz opciju WITH GRANT OPTION.

3.7 Pitanja i zadaci

1. Kako se mogu klasifikovati relacioni upitni jezici?
2. Koji su jezici prethodili SQL-u i šta ih karakteriše?
3. Navesti iskaze za manipulisanje podacima u SQL-u.
4. Definisati sledeće pojmove SQL-a:

jednostavni upitni blok	operator grupisanja
puni upitni blok	interaktivni SQL
SELECT iskaz	aplikativni SQL
podupit	statički SQL
korelisani podupit	dinamički SQL
agregatna funkcija	kursor

5. Dokazati relacionu kompletnost SQL-a.
6. Iskazima SQL-a formulisati upite iz zadatka 10 glave 2.
7. Iskazima SQL-a formulisati sledeće upite:
   - Naći ukupni broj knjiga koje izdaje izdavač sa šifrom i1.
   - Naći ukupni tiraž knjige sa šifrom k6.
   - Naći ukupni tiraž knjiga čiji je jedan od autora pisac sa šifrom p2.
   - Za svaku knjigu koju izdaje neki izdavač naći šifru knjige, šifru izdavača i odgovarajući ukupni tiraž.
   - Naći šifre knjiga koje je izdao samo izdavač sa šifrom i1.
   - Naći šifre autora knjiga koje su izdali svi izdavači.
   - Naći šifre izdavača koji izdaju knjigu sa šifrom k6 u izdanjima čiji je prosečni tiraž manji od najmanjeg tiraža izdavača i1.
8. Navesti primere iskaza unošenja, brisanja i ažuriranja koji se korektno izvršavaju, i onih čije izvršenje sistem odbija. Objasniti.
9. Iskazima SQL-a formulisati sledeće zahteve:
   - Povećati tiraž svim izdanjima izdavača 'Prosveta' za 10%.
   - Iz izdavačke baze podataka izbrisati sve podatke o knjizi koja nije izdata.
   - U tabelu I uneti podatke o izdavaču sa šifrom i10 čiji status još nije poznat.
10. Koje su osnovne karakteristike aplikativnog SQL-a?
11. Šta su kursori i koje se operacije nad kursorima mogu izvršavati?
12. Navesti moguće vrednosti promenljive SQLCODE i objasniti njihovo značenje.
13. Napisati program na aplikativnom SQL-u umetnutom u programski jezik C, koji štampa sve podatke o izdavačima u rastućem poretku njihovih naziva. Uz svakog izdavača program treba da štampa i sve podatke o knjigama koje taj izdavač izdaje, u rastućem poretku naslova knjiga.
14. Definisati sledeće pojmove:

statički SQL	oznaka parametra
dinamički SQL	autorizacija
PREPARE iskaz	nivo autorizacije
EXECUTE iskaz	pravo pristupa podacima
EXECUTE IMMEDIATE iskaz	prenosivo pravo

15. Da li su iskazi PREPARE i EXECUTE dinamički ili statički?
16. Koji se iskazi SQL-a mogu dinamički izvršavati, a koji ne mogu? Navesti primere.
17. Navesti primer izvršavanja dinamičkog SELECT iskaza pomoću kursora.
18. Koje prednosti a koje nedostatke ima dinamički SQL?
19. Koji su nivoi autorizacije u sistemu DB2?
20. Kakav je odnos nivoa autorizacije i skupa prava?
21. Opisati strukturu i semantiku GRANT i REVOKE iskaza
22. Navesti primer iskaza dodele prenosivog prava ažuriranja tabele svim korisnicima.

4

Pogledi

U ovoj glavi biće predstavljene virtuelne relacije (tabele) – pogledi, u obliku u kome je njihovo definisanje i manipulisanje njima podržano u SQL standardima i aktuelnim implementacijama SQL-a. Jedan novi pristup pogledima, koji razrešava mnoge nelogičnosti u njihovom tretmanu u SQL-u, biće prikazan u poslednjem odeljku ove glave ([27]).

4.1 Definisanje pogleda

4.1.1 Kreiranje pogleda

Pogledi su virtuelne tabele koje nemaju odgovarajuću fizičku reprezentaciju, već je njihova definicija, u terminima drugih tabela, zapamćena u sistemskom katalogu.

Iskaz za kreiranje pogleda u SQL-u ima sledeći osnovni oblik:

```
CREATE VIEW  pogled [( kolona{,kolona})]
       AS    podupit
[WITH CHECK OPTION]
```

Na primer,

```
CREATE VIEW  IJUG
       AS  SELECT I_SIF, STATUS, DRZAVA
           FROM   I
           WHERE  DRZAVA = 'Jugoslavija'
```

Kada se izvrši ovaj CREATE iskaz, podupit SELECT... se ne izvršava, već se samo struktura upita sačuva u katalogu.

Pogled IJUG je "prozor", i to dinamički, na tabelu I. Izmene nad pogledom IJUG automatski se odražavaju na baznu tabelu I, a izmene nad baznom tabelom I automatski se projektuju na pogled IJUG, pod uslovom da se odnose na deo tabele I

nad kojim je pogled IJUG definisan, tj. pod uslovom da se izmene odnose na šifru, status ili državu jugoslovenskog izdavača. Svaki upit pretraživanja ili ažuriranja nad pogledom IJUG sistem konvertuje u ekvivalentni upit nad baznom tabelom I. Na primer, iskaz

```
SELECT *
FROM   IJUG
WHERE  STATUS > 20
```

SQL kompilator konvertuje u ekvivalentni iskaz nad tabelom I:

```
SELECT I_SIF, STATUS, DRZAVA
FROM   I
WHERE  STATUS > 20 AND DRZAVA = 'Jugoslavija'
```

Konverzija se vrši združivanjem SELECT iskaza koji zadaje korisnik i SELECT iskaza zapamćenog u katalogu pri definisanju pogleda. Iz kataloga se zaključuje da FROM IJUG zapravo znači FROM I, da se svaki izbor iz IJUG mora dodatno kvalifikovati poređenjem DRZAVA = 'Jugoslavija' u logičkom izrazu WHERE linije, a da SELECT $*$ znači SELECT I_SIF, STATUS, DRZAVA.

Slično bi se izvršio i iskaz unošenja, brisanja ili ažuriranja nad tabelom – pogledom IJUG.

Podupit u CREATE VIEW iskazu je puni upitni blok (v. 3.3.9) a ne SELECT iskaz u najopštijem obliku, pa ne može, na primer, da uključi operaciju uređenja ORDER BY (što važi za podupite uopšte).

CHECK opcija označava da se operacije UPDATE i INSERT nad pogledom izvršavaju samo ako vrsta koja se menja odnosno unosi zadovoljava WHERE logički uslov podupita.

Mada su pogledi virtuelne tabele nad kojima je moguće vršiti gotovo sve operacije kao i nad baznom tabelom, neka ograničenja u radu sa tabelama – pogledima ipak postoje. To su, na primer, sledeća ograničenja:

- Operacija INSERT nad tabelom IJUG uvodi nedostajuću vrednost na koloni NAZIV bazne tabele I ako ona nije definisana kao NOT NULL, odnosno belinu ako je kolona definisana kao NOT NULL WITH DEFAULT. Ako je kolona NAZIV definisana kao NOT NULL kolona, operacija INSERT nije dopuštena.

- Kako se ažuriranja nad pogledom izvršavaju kao ažuriranja nad baznom tabelom, nije dopušteno pogledu dodavati vrstu sa vrednošću primarnog ključa koja je već prisutna u baznoj tabeli, čak i kada nije "vidljiva" u pogledu; na primer, šifra izdavača i2 nije vidljiva kroz pogled IJUG, ali se vrsta (i2, 30, Jugoslavija) ipak ne može dodati INSERT iskazom pogledu IJUG kao ni baznoj tabeli I, a ne može se ni UPDATE iskazom vrednost kolone I_SIF izmeniti na i2. Takva operacija se odbija.

- Operacija

```
UPDATE IJUG
SET   DRZAVA = 'Srbija'
WHERE I_SIF='i1'
```

izbacila bi iz pogleda ažuriranu vrstu, a operacija

```
INSERT INTO IJUG(I_SIF, STATUS, DRZAVA)
VALUES ('i8', 10, 'Amerika')
```

unela bi, kroz pogled, vrstu u baznu tabelu I. Uneta vrsta ne bi ni bila uključena u pogled IJUG.

Poslednje dve operacije ne bi mogle da se izvrše nad pogledom čija definicija uključuje CHECK opciju (npr. ako definicija pogleda IJUG uključuje WITH CHECK OPTION). CHECK opcija se može zadati samo ako je pogled takav da se može ažurirati (v. 4.3). Ako je UPDATE operacija dozvoljena samo na nekim kolonama, onda se i CHECK opcija primenjuje samo na te kolone.

Sistem DB2 podržava i precizniji ali i opštiji oblik definisanja pogleda. Tako, CHECK opcija može da ima dva pojavljivanja: LOCAL i CASCADED (podrazumevano).

Ako je pogled V definisan sa opcijom WITH CASCADED CHECK OPTION, onda on nasleđuje uslove kojima je definisan svaki pogled od koga zavisi (koji se pojavljuje, neposredno ili posredno, u FROM liniji pogleda V). Osim toga, i pogledi definisani nad V nasleđuju sve te uslove. Dakle, nad uslovima kojima je definisan pogled V i svi pogledi od kojih V zavisi, vrši se operacija konjunkcije (AND) i njen rezultat predstavlja uslov koji se primenjuje na unošenje i ažuriranje pogleda V i svakog pogleda koji je definisan nad V.

Ako je pogled V definisan sa opcijom WITH LOCAL CHECK OPTION, onda se na unošenje i ažuriranje pogleda V primenjuje samo uslov kojim je pogled V definisan. Isto ovaži i za poglede definisane nad pogledom V.

Na primer, neka je definisan sledeći niz pogleda (P označava pojavljivanje CHECK opcije i može biti LOCAL ili CASCADED):

V1 je definisan nad baznom tabelom T
V2 je definisan nad pogledom V1 WITH P CHECK OPTION
V3 je definisan nad pogledom V2
V4 je definisan nad pogledom V3 WITH P CHECK OPTION
V5 je definisan nad pogledom V4

Tada pogledi V1–V5, u zavisnosti od opcije LOCAL/CASCADED, pri unošenju ili ažuriranju podležu proveri uslova sledećih pogleda:

	P LOCAL	P CASCADED
V1	–	–
V2	V2	V2,V1
V3	V2	V2,V1
V4	V2, V4	V4,V3, V2,V1
V5	V2, V4	V4,V3, V2,V1

Dalje, kao što SELECT iskaz može da realizuje pretraživanje punim upitnim blokom nad imenovanom izvedenom tabelom – rezultatom izvršavanja nekog upitnog bloka (WITH operatorom), sličnu mogućnost ima i definicija pogleda: podupit kojim se pogled definiše može se primeniti na izvedenu imenovanu tabelu. Tada je sintaksa iskaza za definisanje pogleda oblika

```
CREATE VIEW  pogled [( kolona{,kolona})]
       AS  WITH izvedena_tabela   podupit
[WITH CHECK OPTION]
```

Na primer, pogled kojim se iz tabele I izdvajaju jugoslovenski izdavači čiji je status>20,može se definisati i sledećim iskazom:

```
CREATE VIEW ijugw
       AS  WITH ijug1 AS (SELECT I_SIF, STATUS, DRZAVA
                          FROM I
                          WHERE DRZAVA = 'Jugoslavija')
           SELECT *
           FROM ijug1
           WHERE STATUS>20
WITH CHECK OPTION
```

4.1.2 Eksplicitno imenovanje kolona pogleda

U iskazu kojim se kreira pogled IJUG nisu eksplicitno imenovane kolone tog pogleda. Imena kolona nasleđena su od rezultata podupita kojim je pogled definisan (I_SIF, STATUS, DRZAVA). U slučaju pogleda IJUG moguće je i eksplicitno imenovanje kolona pogleda

```
CREATE VIEW IJUG (I_SIF, STATUS, DRZAVA)
```

pri čemu imena kolona pogleda mogu biti različita od onih u rezultatu podupita, ali mora ih biti isti broj.

Postoje slučajevi u kojima pogled ne može da nasledi imena kolonâ od podupita koji ga definiše, pa je eksplicitno imenovanje kolonâ pogleda neophodno. To se dešava kada je neka kolona pogleda izvedena, funkcijom ili izrazom, iz kolone tabele nad kojom je pogled definisan (pa ne postoji ime koje bi nasledila), ili ako dve ili više kolona pogleda treba da imaju isto ime. Ovi slučajevi su ilustrovani sledećim primerima.

Primer 4.1 Kreirati pogled koji prikazuje šifre izdatih knjiga i njihove ukupne tiraže.

```
CREATE VIEW  KNJIGATIR (K_SIF, UK_TIRAZ)
       AS    SELECT K_SIF, SUM(TIRAZ)
             FROM   KI
             GROUP BY K_SIF
```

Primer 4.2 Kreirati pogled koji prikazuje parove država takvih da izdavač iz prve države izdaje knjigu pisca iz druge države (primer ujedno ilustruje i pogled definisan nad većim brojem baznih tabela).

```
CREATE VIEW  PARDRZAVA (IDRZAVA, PDRZAVA)
   AS SELECT DISTINCT I.DRZAVA, P.DRZAVA
      FROM  I, KI, KP, P
      WHERE I.I_SIF=KI.I_SIF AND KI.K_SIF=KP.K_SIF AND KP.P_SIF=P.P_SIF
```

4.1.3 Kreiranje pogleda nad drugim pogledima

Kako SELECT iskaz može da se izvršava nad pogledom, moguće je i da SELECT iskaz kojim se pogled definiše, u svojoj FROM liniji ima neki prethodno definisani pogled. Na primer,

```
CREATE VIEW DOBRIIJUG
       AS   SELECT I_SIF, STATUS
            FROM  IJUG
            WHERE STATUS > 20
```

Opcija provere (CHECK) nad pogledom koji je definisan nad drugim pogledom realizuje se, u zavisnosti od pojavljivanja (LOCAL, CASCADED) prema pravilima iz prethodne tačke.

4.1.4 Uklanjanje pogleda

Definisani pogled može se ukloniti SQL iskazom

```
DROP VIEW  ime-pogleda
```

Definicija pogleda se uklanja iz kataloga, i svi pogledi definisani nad tim pogledom postaju neoperativni (oni još uvek postoje tako da se njihovo ime ne može koristiti za druge objekte dok se eksplicitno, DROP iskazom, i oni ne uklone, ali se ne mogu koristiti dok se ponovo ne kreiraju definicijom identičnom inicijalnoj). Ako se ukloni bazna tabela, svi pogledi definisani nad njom ili nad pogledima definisanim nad tom tabelom, takođe postaju neoperativni.

Iskaz ALTER VIEW omogućuje izmenu vrlo spcifičnih svojstava kolona pogleda, koji su izvan dometa našeg interesovanja. Zato ćemo smatrati da se definicija jednom kreiranog pogleda ne može menjati.

4.2 Pretraživanje pogleda

Konverzija upita pretraživanja pogleda u upite pretraživanja baznih tabela može se sprovesti kao što je opisano ranije (tačka 4.1.1), pri čemu je taj proces obično jednostavan i pravolinijski. Problemi mogu da nastupe u nekim slučajevima, kao

kada se, na primer, kolona pogleda koja je izvedena primenom agregatne funkcije pretražuje kao konvencionalna kolona.

Primer 4.3 Neka se nad prethodno definisanim pogledom KNJIGATIR (primer 4.1) izvrši SELECT iskaz pretraživanja

```
SELECT *
FROM   KNJIGATIR
WHERE  UK_TIRAZ > 7000
```

Primenom mehanizma konverzije dobio bi se SELECT iskaz koji ima sledeću nekorektnu formu u SQL-u:

```
SELECT K_SIF, SUM(TIRAZ)
FROM   KI
WHERE  SUM(TIRAZ) > 7000
GROUP BY K_SIF
```

Ta forma je nekorektna jer se u WHERE liniji ne sme naći agregatna funkcija (SUM). Odgovarajući korektni iskaz imao bi oblik

```
SELECT K_SIF, SUM(TIRAZ)
FROM   KI
GROUP BY K_SIF
HAVING SUM(TIRAZ) > 7000
```

ali neki SQL kompilatori "ne umeju" da naprave takvu konverziju.

U sistemu DB2 moguće je pretraživati sve poglede na isti način kao i bazne tabele.

4.3 Ažuriranje, unošenje i brisanje nad pogledima

Ažuriranje pogleda u ovom odeljku razmatraće se u najširem smislu, koji uključuje i unošenje, i brisanje, i izmenu (tj. ažuriranje u užem smislu).

Mada se pogledi, onako kako su ovde uvedeni, mogu (gotovo) svi pretražiti, samo njihov manji deo može se ažurirati SQL-om.

Postoje pogledi za koje se značenje ažuriranja može precizno opisati (kaže se da se ti pogledi teorijski mogu ažurirati), a postoje i pogledi za koje takav opis ne postoji (oni se ni teorijski ne mogu ažurirati). Primer za prvu grupu pogleda je pogled IJUG: tu je jasno šta znači uneti vrstu (i5, 40, Italija) (u tabelu I unosi se vrsta (i5, ' ', 40, Italija)), brisanje vrste (i1, 30, Jugoslavija) izvršava se kao brisanje iz tabele I vrste (i1, Prosveta, 30, Jugoslavija), a izmena države izdavača

i1 sa 'Jugoslavija' na 'Srbija' izvršava se kao izmena na istoj koloni i vrsti u baznoj tabeli.

Nemogućnost teorijskog ažuriranja nekih pogleda proističe iz strukture tih pogleda, a ne iz neprilagođenosti jezika ili sistema da to ažuriranje realizuju. Na primer, u pogledu PARDRZAVA nije jasno šta bi uopšte značilo izbrisati vrstu (Jugoslavija, Jugoslavija) koja pripada tom pogledu, tj. ne postoji izmena baznih tabela I, P koja ima tačno efekat brisanja para (Jugoslavija, Jugoslavija) i ničeg drugog. Slično važi i za ažuriranje u užem smislu i unošenje u ovaj pogled.

Upitni jezik SQL omogućuje ažuriranje nekog podskupa skupa pogleda koji se teorijski mogu ažurirati. Taj podskup se opisuje na sledeći način.

Pogled koji se definiše nad jednom baznom tabelom, eliminacijom nekih vrsta i nekih kolona te tabele, naziva se pogledom koji je podskup vrsta i kolona. SQL omogućuje ažuriranje samo pogleda koji su podskupovi vrsta i kolona. Ranije kreirani pogled IJUG upravo je takvog tipa pa je njegovo ažuriranje u SQL-u moguće. Međutim, SQL nameće i specifična ograničenja da bi ovakvi pogledi mogli da se ažuriraju:

(a) Ako je kolona pogleda izvedena od izraza u kome se primenjuje skalarna operacija, skalarna funkcija ili konstanta, onda INSERT operacije nad pogledom nisu dozvoljene, UPDATE operacije su dozvoljene samo na preostalim kolonama, a operacije brisanja su dozvoljene. Na primer, nad pogledom

```
CREATE VIEW T_U_HILJ (K_SIF, I_SIF, IZDANJE, H_TIRAZ)
       AS   SELECT K_SIF, I_SIF, IZDANJE, TIRAZ/1000
            FROM   KI
```

INSERT operacija nije dozvoljena, UPDATE operacija nije dozvoljena na koloni H_TIRAZ (kompilator ne poseduje znanje o tome kako bi se ta operacija odrazila na baznu kolonu TIRAZ), a DELETE operacija je dopuštena i izvršava se brisanjem odgovarajuće vrste iz bazne tabele.

(b) Ako je kolona pogleda izvedena od agregatne funkcije, onda taj pogled ne može da se ažurira; na primer, nad pogledom

```
CREATE VIEW UT (UK_TIRAZ)
       AS   SELECT SUM(TIRAZ)
            FROM   KI
```

očigledno nema smisla ni jedna od operacija INSERT, UPDATE, DELETE.

(c) Ako definicija pogleda uključuje GROUP BY ili HAVING operatore, pogled se ne može ažurirati. Na primer, na pogled KNJIGATIR (primer 4.1) ne može se primeniti INSERT operacija, niti UPDATE operacija na koloni UK_TIRAZ. DELETE operacije i UPDATE operacija na koloni K_SIF teorijski se mogu izvršiti tako da izbrišu ili izmene sve odgovarajuće vrste u baznoj tabeli KI, ali SQL sistema DB2 ni te operacije ne dopušta.

(d) Ako definicija pogleda uključuje DISTINCT opciju, pogled se ne može ažurirati. Ovo je posledica pretpostavke da odabrana polja, čim su kvalifikovana DISTINCT opcijom, ne određuju jednoznačno vrstu tabele, tj. ne uključuju njen primarni ključ. U takav pogled ne mogu se unositi vrste jer bi to dovelo do unošenja NULL vrednosti na kolonama bazne tabele koje ulaze u primarni ključ. Na primer, u pogled

```
CREATE VIEW KBIB
       AS   SELECT DISTINCT K_SIF, I_SIF
            FROM   KI
```

ne mogu se unositi vrste (INSERT operacija nad tabelom KI podrazumeva unošenje vrednosti i za kolonu IZDANJE koja je definisana kao NOT NULL kolona). Za UPDATE i DELETE operacije važe ista razmatranja kao u slučaju (c).

U slučaju pogleda

```
CREATE VIEW KO
       AS   SELECT DISTINCT K_SIF, OBLAST
            FROM K
```

sistem DB2 (SQL) nije "svestan" činjenice da se u ovaj pogled može, teorijski, i unositi, i menjati, i brisati, jer je pogled ekvivalentan (u smislu identičnosti sadržaja) pogledu definisanom bez DISTINCT opcije (K_SIF je primarni ključ); te operacije nisu dozvoljene nad ovim pogledom zbog njegove sintakse. Sistem DB2 (tj. kompilator njegovog SQL jezika) nema mogućnost analize semantike pogleda.

(e) Ako podupit kojim je pogled definisan sadrži neku od skupovnih operacija (osim UNION ALL), pogled se ne može ažurirati. U slučaju UNION ALL operacije, u pogled se ne može vršiti unošenje; da bi operacije brisanja i ažuriranja u užem smislu mogle da se izvrše, postoje dalja ograničenja: operandi UNION ALL operacije ne smeju biti definisani nad istom baznom tabelom, a odgovarajuće kolone tih operanada moraju imati identične tipove. Na primer, nad pogledom

```
CREATE VIEW I_20_JUG
       AS   SELECT I_SIF, NAZIV, STATUS, DRZAVA
            FROM I
            WHERE STATUS < 20
               UNION ALL
            SELECT I_SIF, NAZIV, STATUS, DRZAVA
            FROM I
            WHERE DRZAVA = 'Jugoslavija'
```

INSERT operacija nije dozvoljena (jer definicija pogleda uključuje UNION ALL operaciju), ali nisu dozvoljene ni UPDATE i DELETE operacije (jer su operandi UNION ALL operacije definisani nad istom baznom tabelom).

(f) Ako FROM linija u definiciji pogleda uključuje (eksplicitno ili implicitno) veći broj n-tornih promenljivih, pogled se ne može ažurirati. Na primer, pogled

```
CREATE VIEW K6_IZDAVACI
       AS   SELECT I.I_SIF, NAZIV, STATUS, DRZAVA
            FROM   I, KI
            WHERE  I.I_SIF=KI.I_SIF AND KI.K_SIF='k6'
```

ima rezultat koji je podskup vrsta i kolona (bazne tabele I), ali SQL kompilator to ne "prepoznaje" jer se u FROM liniji SELECT iskaza nalaze dve tabele. Ovaj pogled zato ne može da se ažurira u SQL-u, mada ga je teorijski moguće ažurirati. Međutim, kada se ovaj pogled predefiniše na sledeći način, dobija se ekvivalentan pogled koji se može ažirirati i u SQL-u:

```
CREATE VIEW K6_IZDAVACI
             AS   SELECT I_SIF, NAZIV, STATUS, DRZAVA
                  FROM I
                  WHERE I_SIF IN (SELECT I_SIF
                                  FROM KI
                                  WHERE K_SIF = 'k6')
```

U ovom pogledu nema višestrukih n-tornih promenljivih u FROM liniji pa se "prepoznaje" da je pogled podskup vrsta i kolona.

4.4 Prednosti pogleda

Mehanizam pogleda, kao realizacija koncepta podmodela, omogućava značajna olakšanja u manipulisanju podacima relacione baze podataka. Zbog toga se podrška mehanizmu pogleda, mada ne spada u minimalni skup svojstava koja određuju RSUBP, smatra važnim konceptom koji podržavaju svi ozbiljniji prototipski i komercijalni relacioni sistemi za upravljanje bazama podataka.

Neke od prednosti mehanizma pogleda su sledeće:

- izvesna količina logičke nezavisnosti podataka; ako se baza podataka preprojektuje, pri čemu su iste informacije organizovane u nove tabele, programi ne moraju da se menjaju jer mehanizam pogleda može da se upotrebi da "simulira" postojanje starih tabela;
- iste podatke razni korisnici mogu videti na razne načine; ovaj aspekt je naročito važan ako postoji veći broj raznorodnih grupa korisnika baze podataka;

- pojednostavljena korisnička predstava o bazi i manipulisanje njom; na primer, pretraživanje tabele PARDRZAVA je znatno jednostavnije od pretraživanja istih informacija iz pripadnih baznih tabela;
- skrivanje podataka kao način kontrole ovlašćenja; na primer, ne može se prići izdavaču i2 preko pogleda IJUG. Ako je skrivanje informacije o izdavaču i2 značajno za neke grupe korisnika, najjednostavnije je to skrivanje ostvariti mehanizmom pogleda. Ubedljiv primer ovog aspekta pogleda je grupa korisnika baze podataka o građanima, koja se bavi statističkim obradama. Ovakvoj grupi se, kroz odgovarajući pogled, ne dopušta uvid u specifične podatke o određenom licu.

4.5 Novi pristup pogledima

U ovom odeljku biće izložen jedan novi pristup pogledima ([27]), koji na formalan i dosledan način tretira poglede kao imenovane izraze relacione algebre ili relacionog računa. Iz ovog pristupa formalno logički sledi niz svojstava pogleda koja se odnose na mogućnost pretraživanja i ažuriranja pogleda, eliminišući sasvim potrebu za *ad hoc* pravilima u ovim domenima; takva pravila su inače karakteristična za sve standarde SQL-a, kao i za postojeće implementacije (v. prethodne odeljke ove glave). Ovaj pristup omogućuje i ažuriranje znatno šire klase pogleda (npr. pogleda definisanih nad većim brojem tabela) nego što je to slučaj sa prethodno izloženim, klasičnim pristupom pogledima. Takođe, kako se relacioni izraz koji definiše pogled zadaje relacionom algebrom ili relacionim računom, pojmovi kao što su operatori grupisanja GROUP BY i HAVING ili opcija DISTINCT, karakteristični za specifični (i u znatnoj meri manjkavi) upitni jezik SQL, ne učestvuju više u razmatranju mogućnosti pretraživanja i ažuriranja pogleda.

Ovaj odeljak se odnosi na poglede kao koncept koji, u obliku u kome se izlaže, nema podršku SQL-a. Stoga će i primeri i pravila biti izraženi u terminima relacione algebre kao formalizma koji je u direktnoj vezi (za razliku od SQL-a) sa relacionim modelom.

Iskaz kreiranja pogleda uključuje ime pogleda i relacioni izraz. Imena kolona pogleda nasleđuju se od rezultata relacionog izraza, s tim što u relacionom izrazu može da učestvuje (više puta) i operacija proširene relacione algebre za preimenovanje atributa, RENAME. Na primer (analogon primeru 4.2),

```
CREATE VIEW PARDRZAVA
     ((I RENAME DRZAVA AS IDRZAVA) * KI * KP *
      (P RENAME DRZAVA AS PDRZAVA) ) [IDRZAVA, PDRZAVA]
```

Pogledi se pretražuju ili konverzijom upita nad pogledom u upit nad baznom tabelom, čija je restrikcija pojačana uslovima koji definišu sâm pogled, ili materijalizacijom pogleda (kao privremene bazne tabele) i izvršavanjem upita nad materijalizovanim pogledom.

Mehanizam ažuriranja pogleda mora da zadovolji sledeće osnovne principe.

- Mogućnost ažuriranja pogleda mora biti bazirana na semantici pogleda a ne na njegovoj sintaksi, tj. ne sme se dogoditi da je pogled definisan na jedan način moguće ažurirati, a da je isti pogled, definisan na drugi način, nemoguće ažurirati (v. pogled K6_IZDAVACI, odeljak 4.3).

- Mogućnost ažuriranja pogleda definisanog binarnom operacijom mora biti simetrična (npr. brisanje iz pogleda koji je definisan operacijom preseka dve relacije mora poštovati princip brisanja iz obe relacije).

- Operacije unošenja, brisanja i ažuriranja pogleda moraju se izvršavati kao odgovarajuće operacije (unošenja, brisanja, ažuriranja, redom) nad tabelama nad kojima je pogled definisan.

- Pravila ažuriranja pogleda moraju biti rekurzivno primenljiva, jer pogled može biti definisan nad drugim pogledom.

- Ako se pogled može ažurirati, onda to treba da važi za sve operacije ažuriranja u širem smislu – unošenje, brisanje i ažuriranje u užem smislu.

- Za svaki pogled podrazumeva se CHECK opcija, tj. pogled striktno ograničava skup n-torki kojima se može pristupiti, na samo one n-torke koje zadovoljavaju definiciju pogleda.

Pravila za ažuriranje pogleda sada se zadaju u zavisnosti od operacije relacione algebre kojom je definisan pogled, i to za svaku operaciju posebno. Pravila za ažuriranje pogleda koji je definisan izrazom koji uključuje veći broj operacija relacione algebre dobijaju se kombinovanjem pravila za pojedinačne operacije. Ovde će biti razmotreno nekoliko karakterističnih operacija.

4.5.1 Unija, presek, razlika

U ovoj tački razmatraćemo samo uniju, ali odgovarajuća razmatranja važe i za presek i razliku.

Unošenje n-torke u pogled oblika $R \cup S$, gde su R i S kompatibilni relacioni izrazi, izvršava se kao unošenje u rezultat onog relacionog izraza čije restrikcije n-torka zadovoljava. Na primer, u pogled definisan unijom

$$(\text{I[STATUS} >= 20]) \cup (\text{I[DRZAVA} = \text{'Jugoslavija'}]),$$

može se uneti četvorka (i5, 'Mladost', 10, 'Jugoslavija'), jer zadovoljava restrikciju drugog relacionog izraza. Četvorka se u pogled unosi unošenjem u baznu relaciju I.

Iz unije se n-torka briše tako što se izbriše iz svakog operanda unije čiju restrikciju zadovoljava.

Izmena n-torke je dopuštena samo ako i ta n-torka i njena izmenjena vrednost zadovoljavaju uslov date unije, tj. restrikciju jednog ili drugog (ne obavezno istog) njenog operanda.

4.5.2 Restrikcija

Unošenje (brisanje, ažuriranje) u pogled oblika R [*logički izraz*] može se izvršiti samo ako n-torka koja se unosi (briše, ažurira, kao i ažurirana vrednost) zadovoljava strukturu (i eventualne uslove) relacionog izraza R i logičkog izraza koji određuje restrikciju. Unošenje (brisanje, ažuriranje) se izvršava kao unošenje u rezultat relacionog izraza R (brisanje iz R, ažuriranje u R). Na primer, u pogled definisan relacionim izrazom

I[DRZAVA = 'Jugoslavija'],

može se uneti četvorka (i5, 'Mladost', 10, 'Jugoslavija'), jer zadovoljava i strukturu tabele I i logički izraz koji određuje restrikciju. Četvorka se unosi u baznu tabelu I, i istovremeno se "vidi" kroz pogled. S druge strane, unošenje koje narušava uslove integriteta bazne tabele (npr. jedinstvenost primarnog ključa) neće se izvršiti.

4.5.3 Projekcija

Pogled koji je definisan projekcijom relacije R na neki podskup X njenih atributa može se ažurirati ako skup atributa X sadrži primarni ključ relacije R. U tom slučaju unošenje u pogled vrši se popunjavanjem ostalih atributa relacije R nedostajućim vrednostima, pri čemu se poštuju pravila integriteta, npr. jedinstvenost primarnog ključa. Pri brisanju iz pogleda briše se cela n-torka relacije R, dok se pri izmeni menjaju vrednosti navedenih atributa projekcije – pogleda, ukoliko se time ne narušavaju uslovi integriteta. Na primer, u pogled definisan relacionim izrazom

I[I_SIF, STATUS],

može se uneti par (i5, 20) (kao četvorka (i5, NULL, 20, NULL) u tabelu I), ali se ne može uneti par (i1, 10), jer je vrednost i1 primarnog ključa tabele I već prisutna u tabeli I. Iz ovog pogleda može se izbrisati par (i1, 30) i to brisanjem četvorke (i1, 'Prosveta', 30, 'Jugoslavija') iz tabele I. Izmena para (i1, 30) u par (i1, 20) realizuje se kao izmena četvorke (i1, 'Prosveta', 30, 'Jugoslavija') tabele I u četvorku (i1, 'Prosveta', 20, 'Jugoslavija'); međutim, izmena para (i1, 30) u par (i2, 30) nije dopuštena, jer narušava jedinstvenost primarnog ključa I_SIF.

Ažuriranje pogleda koji je definisan projekcijom na atribute koji ne uključuju primarni ključ relacije predstavlja problem. Unošenje u takav pogled nije dopušteno, jer bi podrazumevalo unošenje nedostajućih vrednosti na atributima iz primarnog ključa, koji nisu uključeni u atribute projekcije. Brisanje iz takvog pogleda bi se realizovalo kao brisanje svih n-torki cele relacije koje zadovoljavaju logički izraz kojim je definisano brisanje. Ažuriranje n-torke ovakvog pogleda se odnosi, opet, na ažuriranje svih n-torki odgovarajuće relacije koje zadovoljavaju logički izraz kojim je definisano ažuriranje. Primer ovakvog pogleda je pogled PARDRZAVA (iz primera 4.2), definisan relacionim izrazom

((I RENAME DRZAVA AS IDRZAVA) $*$ KI $*$ KP $*$
(P RENAME DRZAVA AS PDRZAVA)) [IDRZAVA, PDRZAVA].

Ovaj relacioni izraz uključuje projekciju relacije, koja je rezultat prirodnog spajanja četiri relacije, na atribute IDRZAVA, PDRZAVA. Ovi atributi ne uključuju primarni ključ relacije koja se projektuje, pa u pogled definisan ovom projekcijom, prema prethodnom, nije dopušteno unošenje.

4.5.4 Prirodno spajanje

Novi pristup pogledima koji se u ovom odeljku izlaže ima za rezultat mogućnost ažuriranja svih pogleda koji su definisani operacijom prirodnog spajanja. Ažuriranje takvih pogleda može da ima izvesne bočne efekte, ali ti bočni efekti obezbeđuju integritet ažuriranog pogleda, tj. relacijâ nad kojima je pogled definisan.

Pretpostavimo, na primer, da tabela KI nema strani ključ I_SIF, i da je u tabeli KI prisutna petorka (i5, k5, 1, 1990, 10000) koja se odnosi na izdavača sa šifrom i5 o kome ne postoji informacija u tabeli I. Neka je pogled definisan relacionim izrazom

$$\mathrm{I} * \mathrm{KI}$$

i neka se u pogled unosi osmorka (i5, 'Mladost', 20, 'Jugoslavija', k6, 2, 1995, 5000). Ovo unošenje se realizuje kao unošenje četvorke (i5, 'Mladost', 20, 'Jugoslavija') u tabelu I, i petorke (i5, k6, 2, 1995, 5000) u tabelu KI. Međutim, unošenje u tabelu I proizvodi, kao bočni efekat, unošenje još jedne osmorke u pogled – to je osmorka (i5, 'Mladost', 20, 'Jugoslavija', k5, 1, 1990, 10000).

Uopšte, unošenje n-torke u u pogled definisan operacijom prirodnog spajanja dveju relacija, $R * S$, realizuje se kao unošenje u relaciju R dela n-torke u koji se odnosi na atribute iz R i, ako to nije već postignuto bočnim efektom, kao unošenje u relaciju S dela n-torke u koji se odnosi na atribute iz S. Slično se izvršavaju operacija brisanja i operacija izmene n-torki. Nijedna od ovih operacija ne sme da naruši uslove integriteta relacija R, S.

Detaljnije razmatranje izloženog pristupa pogledima, sa specifičnostima efekata pojedinih operacija ažuriranja na pojedine relacione izraze koji definišu poglede, može se naći u [27].

4.6 Pitanja i zadaci

1. Šta je pogled?
2. Objasniti iskaz za definisanje pogleda u SQL-u.
3. Kako se izvršava iskaz pretraživanja pogleda u SQL-u?
4. Kada je neophodno eksplicitno imenovati kolone pogleda?
5. Koji se pogledi mogu ažurirati u SQL-u?
6. Koje su prednosti pogleda?

7. Opisati novi, sistematični prilaz pogledima i razmotriti njegove prednosti.

8. Kreirati pogled nad izdavačkom bazom podataka koji se sastoji od svih izdavača koji su izdali neki roman Branka Ćopića.

9. Kreirati pogled nad izdavačkom bazom podataka koji se sastoji od svih izdavača, knjiga koje su izdali i autora tih knjiga.

10. Neka je data sledeća definicija pogleda:

```
CREATE VIEW V_TIRAZ
       AS   SELECT *
            FROM   KI
            WHERE  TIRAZ > 5000
```

Formulisati SQL iskaze nad baznim tabelama kojima se izvršavaju sledeći iskazi pretraživanja i ažuriranja pogleda:

- ```
  SELECT *
  FROM   V_TIRAZ
  WHERE  K_SIF > 'k3'
  ```
- ```
  UPDATE V_TIRAZ
  SET   TIRAZ = TIRAZ * 1.1
  WHERE GODINA > 1990
  ```
- ```
  DELETE FROM V_TIRAZ
  WHERE  GODINA < 1960
  ```
- ```
  INSERT INTO V_TIRAZ
  VALUES ('k1', 'i1', 1, 1960, 5000)
  ```

5

Standardi SQL-a

Cilj standardizovanja jezika baza podataka je da obezbedi prenosivost definicija baza podataka i aplikativnih programa medju implementacijama koje su saglasne sa standardom. Za razmenu definicija baza podataka i pogleda specifičnih za aplikaciju može se koristiti jezik za definisanje podataka. Jezik za manipulisanje bazama podataka obezbedjuje operacije nad podacima koje omogućuju razmenu kompletnih aplikacionih programa.

Od 1986. godine, kada je prvi put standardizovan (ANSI – American National Standard Institute, američki standard i ISO – International Standard Organization, međunarodni standard), jezik baza podataka SQL je u neprestanom razvoju od strane većeg broja organizacija za standardizaciju (ANSI, ISO, FIPS – Federal Information Processing Standards for the US federal government, IEC – International Electrotechnical Commision, itd). Najnovija puna verzija standarda publikovana je 1992. godine, ANSI X3.135-1992 i ISO/IEC 9075:1992. Puni naziv ovog standarda je "Međunarodni standardni jezik baza podataka (1992)" ("International Standard Database Language SQL (1992)" [**?**]), mada se sreće i pod nazivom SQL2, SQL-92.

SQL2 značajno proširuje standarde iz 1986. i 1989. godine, uključenjem jezika za manipulisanje shemom, tabelâ za informacije o shemama, novih olakšica za dinamičko kreiranje SQL iskaza, novih tipova podataka i domena. Ostale novine SQL2 standarda uključuju spoljašnje spajanje, kaskadno ažuriranje i brisanje kod referencijalnog integriteta, skupovnu algebru nad tabelama, nivoe konzistentnosti transakcija, skrolovane kursore, odloženu proveru uslova ograničenja, i značajno proširenje dijagnostike, tj. izveštavanje o izuzecima i greškama. SQL2 eliminiše veliki broj ograničenja kako bi jezik postao što fleksibilniji i neprotivurečniji ([**?**], [**?**]). Detaljniji opis rešenja ovog standarda izložen je u sledećoj tački.

Dalja proširenja i poboljšanja SQL standarda su u razvoju, i odnose se na promociju SQL-a u izračunljivo kompletan jezik za definisanje i upravljanje perzistentnim, kompleksnim objektima. Za 1995. godinu planirano je bilo objavljivanje novog standarda SQL-a, SQL3, kompatibilnog sa SQL2, ali obogaćenog objektnim aspektom podataka (o objektnom modelu podataka biće reči u Dodatku).

SQL3 uključuje korisnički definisane i apstraktne tipove podataka (ATP, koji se grade po uzoru na C++ klase), sa hijerarhijama generalizacije i specijalizacije i višestrukim nasleđivanjem, objekte kao njihove instance, metode, polimorfizam i učaurenje (enkapsulaciju). SQL3 je izračunljivo kompletan jezik (ima kontrolne strukture i ne zahteva ugnježdenja u programske jezike opšte namene).

Razna svojstva uključena u predlog SQL3 standarda su u različitim fazama razvoja, i imaju različite statuse. Godine 1993, ANSI i ISO komiteti za razvoj odlučili su da razlože budući razvoj SQL-a u višedelni standard. Ti delovi su:

I Opis strukture dokumenta.

II Jezgro specifikacije, uključujući ATP i svojstva objektnog SQL-a.

III SQL/CLI (Call Level Interface): verzija zasnovana na elementima standarda SQL2 bila je publikovana 1995. godine kao ISO/IEC 9075-3:1995. Dalji rad na ovoj komponenti, koji treba da obezbedi podršku novim svojstvima u ostalim delovima SQL-a, u razvoju je.

IV SQL/PSM: specifikacija zapamćenih procedura (engl. Persistent Stored Modules). Ova komponenta uključuje sve aspekte izračunljive kompletnosti jezika – promenljive, iskaze dodele, grananja, iteracije, strukturu bloka, poziv procedura i funkcija, obradu greške; posebno je značajna za klijent/server okruženje. Jedna verzija ove komponente standardizovana je 1996. godine (9075-4:1996).

V SQL/veznici: veznici za dinamički SQL i ugnježdeni SQL preuzeti su iz SQL2; godine 1998. publikovan je standard veznika za objektne jezike (C++ i Java) – SQL/OLB, ANSI X3.135.10–1998.

VI SQL/XA: SQL specijalizacija popularnog XA interfejsa koji je razvio X/Open, između globalnog upravljača transakcijama i SQL upravljača resursima. Standardizuje pozive funkcija prema standardu distribuirane obrade transakcija (Remote Database Access – Part3:SQL, ISO/IEC 9579-3:1996, Part2:SQL, ISO/IEC 9579-2:1998).

VII SQL za vremenske serije: SQL podprojekat za razvoj i poboljšanje mehanizama za upravljanje vremenskim serijama.

Godine 1993. odobren je novi ISO/IEC projekat za razvoj SQL biblioteke klasa za multimedijalne aplikacije, tzv. SQL Multimedia (SQL/MM). Ideja je da se specifikuju paketi definicije SQL apstraktnih tipova podataka (ATP) korišćenjem svojstava SQL3, i da se stanardizuju posebne biblioteke klasa za nauku i tehniku, obradu dokumenata i puno pretraživanje tekstova, kao i metode za upravljanje multimedijalnim objektima kao što su slika, zvuk, animacija, muzika, video.

5.1 Standardni jezik baza podataka SQL2

Puno ime ovog standarda je "Međunarodni standardni jezik baza podataka SQL (1992)", a neformalno se zove i SQL2 ili SQL-92. Izlaganje o ovom standardu organizovano je u tačke koje se odnose na definisanje podataka, manipulisanje podacima i na aplikativni SQL. Iako taj standard važi za standard relacionog upitnog jezika, nigde u njegovoj obimnoj specifikaciji ne pominje se termin "relacioni". On se definiše, jednostavno, kao standardni jezik baza podataka.

Standard SQL2 predviđa tri nivoa saglasnosti raznih varijanti SQL jezika sa rešenjima iz standarda – *puni* (full), *srednji* (intermediate) i *početni* (entry) SQL.

Početni SQL uključuje iskaze za definisanje sheme, jezik manipulisanja podacima, referencijalni integritet, proveru uslova integriteta, ugnježđenje SQL-a u sedam različitih programskih jezika, direktno izvršenje iskaza manipulisanja podacima, SQLSTATE parametar, (pre)imenovanje kolona u SELECT listi, opciju pri kreiranju pogleda WITH CHECK OPTION.

Srednji SQL uključuje bitno nova svojstva kao što su iskazi za promenu sheme, dinamički SQL i nivoi izolovanosti SQL transakcija, podršku kaskadnom brisanju, unijsko spajanje, operacije nad niskama karaktera, operacije preseka i razlike, jednostavne domene, CASE izraz, transformaciju među tipovima podataka, kompletniju dijagnostiku i analizu grešaka, višestruki izbor skupova karaktera, intervalne i pojednostavljene tipove datuma/vremena i niske karaktera promenljive dužine.

Puni SQL uključuje još i imenovanje i odloženu proveru uslova integriteta, proširenu podršku tipovima datuma/vremena, ažuriranje i brisanje iz tabele nad kojom je zadat i logički izraz WHERE linije, kaskadno ažuriranje, skrolovane kursore, niske bitova kao tipove podataka, privremene tabele, dodatne opcije uslova integriteta i opšte uslove ograničenja.

Svojstva SQL2 izložena u sledećim tačkama uglavnom se odnose na puni SQL.

Standard SQL2 sadrži niz poboljšanja u odnosu na prethodni SQL standard. Uprkos tome, on još uvek ima neke nedostatke. Osnovni nedostaci i ovog standarda leže u neadekvatnoj podršci nekim bitnim svojstvima relacionog modela. Neki od tih nedostataka su slaba podrška domenima, nedosledno operisanje pogledima, odsustvo podrške operaciji deljenja, mogućnost dupliranja vrsta, podupiti.

5.1.1 Definisanje podataka

Domen. Novost u odnosu na prethodni standard SQL-a u pogledu definisanja podataka jeste izvesna (prilično slaba) podrška domenima, kao i podrška uslovima integriteta domena.

Iskazi za kreiranje i uklanjanje domena su oblika

```
CREATE DOMAIN  naziv-domena [AS]  tip-podataka
     [  podrazumevana-definicija ]
     [  lista-definicija-ograničenja-domena ]
```

```
DROP DOMAIN  naziv-domena  opcija
```

Komponenta *tip-podataka* je skalarni tip podataka iz proširenog skupa koji uključuje, pored ranijih, i sledeće tipove:

- BIT [VARYING] (n)
- INTERVAL
- DATE
- TIME
- TIMESTAMP.

Podrazumevana (DEFAULT) *definicija* precizira podrazumevanu vrednost tog domena i oblika je

```
DEFAULT  podrazumevana-vrednost
```

Na primer,

```
DEFAULT NULL
```

Definicija-ograničenja-domena je uslov integriteta koji se odnosi na svaki atribut definisan nad tim domenom; ograničenje domena predstavlja, u suštini, skup dopuštenih vrednosti domena i zadaje se logičkim izrazom proizvoljne kompleksnosti. Na primer, iskaz

```
CREATE DOMAIN K_SIF AS CHAR(5) DEFAULT '?'
       CONSTRAINT BROJ_KNJIGE
                  CHECK (VALUE >= 'k1' AND VALUE <= 'k5001')
```

kreira domen koji se zove K_SIF i za koji se podrazumeva vrednost '?' (ako se vrednost ne navede); nad tim domenom je definisano jedno ograničenje (CONSTRAINT) koje se zove BROJ_KNJIGE i koje precizira oblik dopuštenih vrednosti.

Opcija pri uklanjanju domena može biti RESTRICT ili CASCADE, pri čemu prva onemogućuje uklanjanje domena ako postoje atributi definisani nad njim, a druga prenosi uklanjanje domena tako što atributi koji su nad njim bili definisani postaju definisani nad tipom podataka nad kojim je bio definisan domen koji se uklanja. Standard predviđa i iskaze za izmenu (ALTER) domena i njegove definicije, uvođenje i uklanjanje podrazumevane vrednosti (SET, DROP DEFAULT), dodavanje i uklanjanje ograničenja domena (ADD, DROP CONSTRAINT).

Osim ovih mogućnosti, definicija domena ne podržava pravu semantiku domena (npr. ne proverava se domenska kompatibilnost kod poređenja, korisnik ne može zadavati operacije nad novim domenima, nema podrške podtipovima i nadtipovima, itd).

Bazna tabela. Definicija bazne tabele (CREATE TABLE) je proširena mogućnošću navoďenja domena za atribute, kao i definicijom ograničenja bazne tabele koja, osim primarnog i stranog ključa, može da uključi i kandidate ključeva (oblika UNIQUE (*lista-atributa*), koji ne obavezuju na NOT NULL atribute) i provere ograničenja tabele (oblika CHECK(*logički-izraz*)). Definicija stranog ključa sada može da sadrži, osim akcije pri brisanju, i akciju pri ažuriranju koja može biti NO ACTION (slično prethodnoj RESTRICT), CASCADE, SET DEFAULT ili SET NULL.

Na primer, bazna tabela KI može biti definisana u SQL2 sledećim CREATE TABLE iskazom:

```
CREATE TABLE KI
   (K_SIF   K_SIF   NOT NULL,
    I_SIF   I_SIF   NOT NULL,
    IZDANJE IZDANJE NOT NULL,
    GODINA  GODINA  NULL,
    TIRAZ   TIRAZ   0,
    PRIMARY KEY (K_SIF, I_SIF, IZDANJE),
    FOREIGN KEY (K_SIF) REFERENCES K
                        ON DELETE NO ACTION
                        ON UPDATE NO ACTION,
    FOREIGN KEY (I_SIF) REFERENCES I
                        ON DELETE CASCADE
                        ON UPDATE CASCADE,
    CHECK (IZDANJE>0 AND IZDANJE<50 ))
```

Atribut K_SIF definisan je nad domenom K_SIF (dopuštena su ista imena atributa i domena), a slično važi i za ostale atribute; 'NULL' odnosno '0' uz definiciju atributa GODINA odnosno TIRAZ, označava podrazumevanu vrednost (u slučaju da se vrednost atributa izostavi).

Standard predviďa mogućnost definisanja *perzistentnih* (trajnih) i *privremenih* baznih tabela. Perzistentne bazne tabele su bazne tabele u uobičajenom značenju. Privremene (TEMPORARY) bazne tabele vezane su za jednu *konekciju* (priključenje) SQL-klijenta na SQL-server (u klijent-server okruženju, v. 18) i materijalizuju se tek pri izvršenju funkcija koje se u toku te konekcije pozivaju iz programa na matičnom jeziku (v. tačke 5.1.2 Manipulisanje podacima i 5.1.3 Aplikativni SQL). Na početku konekcije privremene tabele su prazne, dok se na kraju konekcije njihov sadržaj opet briše. Na privremene tabele se, za razliku od trajnih baznih tabela, ne primenjuje puna funkcionalnost RSUBP – kontrola konkurentnosti, oporavak, registrovanje u katalog, itd.

Izmena bazne tabele (ALTER TABLE) sada se može odnositi kako na dodavanje tako i na uklanjanje atributa, ključeva, podrazumevanih vrednosti i uslova ograničenja tabele.

Uklanjanje bazne tabele (DROP TABLE) može da uključi i opciju (RESTRICT

ili CASCADE) koja će zabraniti uklanjanje, odnosno preneti dejstvo uklanjanja ako postoji pogled ili uslov ograničenja definisan nad tom tabelom.

Opšti uslov ograničenja. Opšti oblik iskaza za kreiranje i uklanjanje uslova ograničenja je

```
CREATE ASSERTION  ime-pravila CHECK( uslov)

DROP ASSERTION  ime-pravila
```

Uslov je logički izraz proizvoljne složenosti, koji može da uključi (slično formuli relacionog računa) operacije poređenja, IN, BETWEEN, LIKE operacije, egzistencijalno kvantifikovane relacione izraze i logičke operacije AND, NOT, OR, kao i neke nove predikate (kvantifikovana poređenja i testove jedinstvenosti – UNIQUE, podudaranja – MATCH, preklapanja – OVERLAPS). Logičke operacije su trovalentne (v. 2.1.3, Proširena relaciona algebra), pa i logička vrednost uslova pripada skupu {true, false, unknown}.

Na primer, uslov ograničenja baze podataka o izdavaštvu koji je formulisan kao "ni jedan izdavač sa statusom manjim od 20 ne može izdavati ni jednu knjigu u tiražu većem od 5000" može se zadati sledećim iskazom:

```
CREATE ASSERTION KI1 CHECK
   (NOT EXISTS ( SELECT * FROM I, KI
                 WHERE  I.STATUS < 20 AND
                 I.I_SIF = KI.I_SIF AND KI.TIRAZ > 5000))
```

Uslov ograničenja koji se zadaje ovim iskazom može da se odnosi na celu bazu podataka (dakle, na veći broj tabela), ali može da se odnosi i na pojedinačnu tabelu, kao i na specifični atribut pojedinačne tabele. Pošto se poslednja dva slučaja mogu izraziti i na drugi način (CREATE TABLE odnosno CREATE DOMAIN iskazom), iskaz CREATE ASSERTION predstavlja izvesnu meru redundantnosti razmatranog standarda.

Shema. Prema ovom standardu, svi objekti baze podataka (domeni, tabele, pogledi, kolone, itd.) definišu se kao elementi odgovarajućih *SQL shema* (ili samo *shema*). Semantika pojma shema odgovara istom konceptu u SQL DB2 (v. 3.2.1), i razlikuje se od pojma relacione sheme dok je slična pojmu sheme relacione baze podataka, kako su definisani relacionim modelom (v. 1.2). Na ovo značenje pojma shema ukazuje i sintaksa definicije sheme u SQL2 koja od važnijih komponenti uključuje sledeće:

```
CREATE SCHEMA  ime-sheme
[  definicija-domena |  definicija-tabele |
   definicija-pogleda |  definicija-uslova-ograničenja
```

```
    {, definicija-domena}   {, definicija-tabele}
    {, definicija-pogleda}  {, definicija-uslova-ograničenja}
]
```

Shema se može i ukloniti iskazom oblika

```
DROP SCHEMA  ime-sheme
```

pri čemu se može navesti i opcija (RESTRICT ili CASCADE) koja će zabraniti uklanjanje sheme ako ona sadrži objekte baze podataka, odnosno implicitno izvršiti odgovarajuće DROP iskaze na objektima sadržanim u shemi.

Ime svake tabele koja je komponenta kreirane sheme kvalifikuje se (eksplicitno ili implicitno) imenom sheme.

5.1.2 Manipulisanje podacima

Proširenje standarda SQL2 u sferi manipulisanja podacima ogleda se u sledećim svojstvima:

- SQL2 podržava nekoliko novih skalarnih operacija i funkcija (u odnosu na prethodni standard): operaciju dopisivanja (||), funkcije UPPER, LOWER, SUBSTRING, LENGTH (za niske karaktera i bitova), operacije sabiranja (+), oduzimanja (−), množenja (∗) i deljenja (/) nad podacima tipa DATETIME odnosno INTERVAL, funkcije za konverziju tipova, operaciju CASE za navoďenje uslovne vrednosti, itd.

- SQL2 podržava relativno kompletnu dijagnostiku (informacije o izvršenju iskaza i eventualnim greškama). Manipulativni iskazi proizvode, osim vrednosti kôda u celobrojnoj promenljivoj SQLCODE, i vrednost znakovnog tipa u promenljivoj SQLSTATE, kao i informacije o uslovima izvršenja u "prostoru dijagnostike" (engl. diagnostics area), koje se mogu dobiti operacijom GET DIAGNOSTICS.

- U FROM liniji SELECT iskaza može da se pojavi i lista izvedenih tabela, tj. lista tabela – rezultata punih SELECT blokova; na primer, iskaz

```
SELECT P_SIF, I_SIF
FROM P, (SELECT *
         FROM I
         WHERE I_STATUS > 10) AS DOBRIIZD
WHERE P.DRZAVA = DOBRIIZD.DRZAVA
```

  izdaje parove šifara pisca i izdavača iz iste države, pri čemu se uzimaju u obzir samo "dobri" izdavači sa statusom većim od 10.

- SQL2 podržava dodelu imena izvedenoj koloni i izvedenoj tabeli, u SELECT liniji odnosno u FROM liniji; na primer, iskaz

```
SELECT K_SIF, I_SIF, 'Tiraz u hilj.=' AS KOM, TIRAZ/1000 AS TuH
FROM KI
     ⋮
```

proizvodi rezultat–tabelu sa četiri kolone; prve dve su imenovane imenima kolona tabele KI – K_SIF i I_SIF, treća je imenovana KOM i ima konstantni sadržaj, dok je naziv četvrte kolone TuH i ona sadrži tiraže u hiljadama primeraka (v. i prethodni primer).

- SQL2 podržava novu, eksplicitnu operaciju spajanja, koja uključuje sve vrste spajanja – Dekartov proizvod, prirodno, slobodno, unutrašnje, spoljašnje i unijsko spajanje. Sintaksa ove operacije je

  tabela-1 `CROSS JOIN |`
  `[NATURAL] [INNER | LEFT | RIGHT | FULL | UNION] JOIN`
  tabela-2
  `[ON` *logički-izraz*`]`

  CROSS JOIN označava Dekartov proizvod, dok se unijsko spajanje (UNION JOIN) odnosi na spoljašnju uniju (v. tačku 2.1.3 – Proširena relaciona algebra).

  Opcija INNER odnosi se na unutrašnje (obično) spajanje, kako prirodno tako i slobodno. Opcije LEFT, RIGHT i FULL odnose se na levo, desno i puno spoljašnje spajanje. Prirodno spajanje (NATURAL) vrši se po jednakim vrednostima jednako imenovanih atributa navedenih tabela, i pri tom se ON opcija ne sme navesti, dok je u slučaju slobodnog (Θ)-spajanja (NATURAL se izostavlja) ON opcija obavezna i definiše uslov spajanja. Na primer,

  ```
  SELECT *
  FROM   (I NATURAL JOIN KI)
  ```

  Tabele *tabela-1* i *tabela-2* mogu biti i izvedene tabele u smislu iz prethodnih paragrafa.

- Sintaksa SQL2 omogućuje zapis TABLE T ekvivalentan SELECT bloku (SELECT $*$ FROM T).

- SQL2 podržava, pored operacije UNION, i eksplicitne operacije preseka i razlike, INTERSECT i EXCEPT. Operacija preseka je višeg prioriteta od operacija unije i razlike. Tabele – argumenti ovih operacija moraju biti izvedene, tj. zadate SELECT blokovima ili izrazima oblika TABLE *tabela*. Moguće je eksplicirati redosled odgovarajućih kolona u tabelama – operandima.

- U SQL2 moguće je koristiti ugnježdene pune SELECT upitne blokove za predstavljanje skalarnih vrednosti; to je učinjeno, na primer, u SELECT liniji sledećeg iskaza:

```
SELECT I.I_SIF, (SELECT SUM(KI.TIRAZ)
                 FROM KI
                 WHERE KI.I_SIF = I.I_SIF ) AS UK_TIRAZ
FROM I
```

- Logički izraz, gdegod se pojavljuje u SQL-u, može da uključi i binarnu operaciju poklapanja MATCH UNIQUE, koja dobija vrednost 'tačno' ako se njen prvi operand (n-torka) poklapa sa tačno jednom n-torkom tabele koja je njen drugi operand. U slučaju da su prisutne i nedostajuće vrednosti, MATCH operacija ima opcije FULL (puno) i PARTIAL (delimično).

- Logički izraz uključuje i (dvovalentni) pozitivni i negativni test nedostajuće vrednosti oblika IS NULL i IS NOT NULL, kao i test vrednosti logičkog izraza (p) oblika

  (p) IS [NOT] TRUE ((p) je (nije) tačno)
  (p) IS [NOT] FALSE ((p) je (nije) netačno)
  (p) IS [NOT] UNKNOWN ((p) je (nije) nepoznato).

- Argument agregatne funkcije koja uključuje opciju DISTINCT može biti proizvoljni izraz, a ne obavezno ime kolone.

- Provera uslova ograničenja može biti odložena, tj. suspendovana i zatim ponovo aktivirana; na primer,

```
SET CONSTRAINTS KI1 DEFERRED
...
SET CONSTRAINTS KI1 IMMEDIATE
```

- SQL2 standardizuje priključenje – *konekciju* klijenta na server tj. na bazu koja je na serveru. Tako se SQL konekcija definiše kao asocijacija SQL klijenta i SQL servera (v. 18) a može se ostvariti eksplicitno, iskazom konekcije koji izvršava klijent i koji identifikuje željeni SQL server. Iskaz je oblika

```
CONNECT TO  ime-SQL-servera [ USER  ime-korisnika]
          | DEFAULT
```

  Ovim iskazom konekciji se može dodeliti i ime. Jedan klijent može da ostvari više konekcija sa raznim serverima (uključujući i podrazumevani – DEFAULT server), pri čemu je u određenom trenutku tekuća ona konekcija čiji je iskaz konekcije poslednji izvršen.

  Imenovana (ili podrazumevana) konekcija može da postane tekuća i izvršavanjem iskaza

```
SET CONNECTION  ime-konekcije | DEFAULT
```

Konekcija se može ostvariti i implicitno, pozivom (sa klijenta) procedure na serveru, pri čemu se, ako ne postoji nijedna konekcija, ostvaruje konekcija sa podrazumevanim serverom.

SQL konekcija se završava eksplicitno, iskazom

```
DISCONNECT   ime-konekcije | DEFAULT | CURRENT | ALL
```

pri čemu se zatvara imenovana, podrazumevana (DEFAULT), tekuća (CURRENT) konekcija, odnosno sve postojeće konekcije klijenta.

Konekcija se može zatvoriti i implicitno, po izvršenju poslednje procedure pozvane iz aplikativnog programa ili poslednjeg interaktivnog SQL iskaza.

5.1.3 Aplikativni SQL

Neke od značajnijih mogućnosti koje pruža standard SQL2 u domenu aplikativnog SQL-a su sledeće:

- SQL2 obezbeđuje sumeđu sa programskim jezicima PL/I, Pascal, Ada, FORTRAN, C, COBOL i MUMPS.
- Bogatiji skup operacija nad kursorima; tako, pored ostalih, tu su operacije uzimanja prethodne kao i naredne, prve kao i poslednje vrednosti kursora (FETCH PRIOR, FETCH NEXT, FETCH FIRST, FETCH LAST), pri čemu se podrazumeva naredna vrednost (NEXT) ako se ništa ne naglasi, a za svaki drugi izbor (PRIOR, FIRST, LAST) kursor mora biti deklarisan opcijom SCROLL:

  ```
  DECLARE X SCROLL CURSOR
  FOR SELECT ...
  ```

- SQL2 eksplicira koncept transakcije uvođenjem iskaza za početak transakcije (SET TRANSACTION) kojim se zadaju nivoi izolovanosti transakcije (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE) i/ili način pristupa podacime (status transakcije) – READ ONLY, READ WRITE.

  Pored iskaza za početak transakcije SET TRANSACTION, eksplicitni su i iskazi uspešnog odnosno neuspešnog završetka transakcije (COMMIT, ROLLBACK, v. odeljak 14.1 – Transakcija i integritet).

  Nivo SERIALIZABLE je najjači stepen izolovanosti i garantuje linearizovanost izvršenja; to je i podrazumevani nivo izolovanosti.

  Najslabiji nivo izolovanosti transakcije je READ UNCOMMITTED koji dopušta pojavu svih problema konkurentnosti osim izgubljenog ažuriranja (v. odeljak 14.3 – Problemi konkurentnosti). Ti problemi se manifestuju

kao fenomen čitanja nepostojećih podataka[1] (engl. dirty read, v. "zavisnost od poništenog ažuriranja", odeljak 14.3), i fenomeni neponovljivog čitanja odnosno fantomskih redova. [2]

Nivo READ COMMITTED dopušta drugi i treći od navedenih fenomena, a REPEATABLE READ samo treći. Dakle, po stepenu izolovanosti prema standardu nivoi su: SERIALIZABLE, REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED. Ovim nivoima izolovanosti odgovaraju u sistemu DB2 redom sledeći nivoi: REPEATABLE READ, READ STABILITY, CURSOR STABILITY, UNOMMITTED READ.

- Pored ugnježdenja SQL iskaza u matični jezik, SQL2 uvodi i koncept SQL procedure na serveru koja se poziva (CALL iskazom) iz programa na matičnom jeziku. Deklaracija procedure uključuje, pored parametara, jedinstveni SQL iskaz, i mora biti u skladu sa pravilima matičnog jezika iz kog se poziva.

- Kao i IBM proizvodi DB2, i SQL2 podržava dinamički SQL, izuzetno važno sredstvo za konstrukciju složenih aplikacija. Na primer, tipična on-line aplikacija (aplikacija koja podržava pristup bazi podataka preko on-line terminala) mora da uključi sledeći niz koraka:

  1. prihvatanje komande sa terminala;
  2. analiza komande;
  3. izdavanje odgovarajućeg SQL iskaza bazi podataka;
  4. vraćanje poruke i rezultata na terminal.

  Umesto da se u programu predviđaju svi mogući oblici iskaza koji se unosi sa terminala u vreme izvršavanja programa, pogodno je konstruisati SQL iskaz dinamički (iskazom PREPARE), a zatim ga kompilirati i izvršiti takođe dinamički (iskazom EXECUTE, v. odeljak 3.5 Dinamički SQL). SQL2 uključuje veći broj iskaza za podršku dinamičkom SQL-u, npr. PREPARE za pripremu iskaza za izvršenje, DEALLOCATE PREPARE za dealokaciju već pripremljenog SQL iskaza, EXECUTE za pridruživanje parametara i izvršenje pripremljenog iskaza, EXECUTE IMMEDIATE za jednokratnu pripremu i izvršenje, DESCRIBE za opis ulaznih parametara i rezultujućih kolona pripremljenog iskaza, DECLARE ili ALLOCATE CURSOR za definisanje dinamičkog kursora nad pripremljenim iskazom, OPEN za otvaranje i pridruživanje parametara dinamičkom kursoru, ALLOCATE / DEALLOCATE / GET / SET DESCRIPTOR za manipulisanje prostorom za ulazne

[1] Do čitanja nepostojećih podataka dolazi kada transakcija T_1 ažurira podatak koji zatim, pre okončanja transakcije T_1, druga transakcija T_2 čita, a onda transakcija T_1 poništi svoje ažuriranje.

[2] Do neponovljivog čitanja dolazi kada transakcija T_1 pročita podatak koji zatim, pre okončanja transakcije T_1, druga transakcija T_2 promeni ili izbriše, a onda transakcija T_1 ponovo pokuša da pročita isti podatak; do fantomskih redova dolazi kada transakcija T_1 pročita skup podataka koji zadovoljavaju zadati uslov, zatim, pre okončanja transakcije T_1, druga transakcija T_2 promeni taj skup, a onda transakcija T_1 ponovo pročita podatke po istom uslovu, pri čemu nailazi na promenjen skup.

parametre pripremljenog iskaza i za rezultujuće vrednosti dinamičkog FETCH ili SELECT iskaza.

SQL2 predviđa mogućnost dinamičkog izvršenja određenih grupa SQL iskaza, kao što su:

- manipulativni iskazi (SELECT, DELETE, INSERT, UPDATE),
- iskazi definisanja (CREATE, ALTER, DROP),
- iskazi transakcija, itd.

5.2 Pitanja i zadaci

1. Navesti karakteristike nacrta standarda SQL3.
2. Šta novo donosi standard SQL2 u oblasti domena?
3. Objasniti rešenja standarda SQL2 u oblasti integriteta.
4. Koje su nove mogućnosti manipulisanja podacima uključene u SQL2?
5. Konstruisati u SQL2 primer tabelâ i upita u kome se koristi spoljašnje spajanje, i opisati rezultat izvršavanja tog upita.
6. Uporediti rešenja SQL2 vezana za aplikativni SQL sa odgovarajućim rešenjima dijalekta DB2 SQL.

6

Upitni jezik QUEL

QUEL (QUEry Language) je upitni jezik RSUBP INGRES (INteractive Graphics and Retrieval System), razvijenog na Kalifornijskom univerzitetu u Berkliju, SAD ([23], [59]). QUEL je zasnovan na relacionom računu n-torki, i ima najviše zajedničkih svojstava sa jezikom DL/ALPHA (Data Language ALPHA), koji je konstruisao Codd 1971. godine kao jednu implementaciju relacionog računa ([15]).

Pored upitnog jezika QUEL, koji je primarni (i originalni) upitni jezik sistema INGRES, komercijalni INGRES (razvijen u kompaniji RTI, Kalifornija) podržava i relacioni upitni jezik SQL. Bez obzira na značaj koji je dobio SQL kao standard, kao i na razloge za usvajanje SQL standarda, dobri poznavaoci oba upitna jezika smatraju QUEL tehnički superiornim u odnosu na SQL, jer je QUEL lakše i efikasnije implementirati, učiti i koristiti.

6.1 Interaktivni QUEL

6.1.1 Definisanje podataka

Iskazi definisanja podataka u QUEL-u uključuju iskaze za kreiranje i uklanjanje baze podataka, i za kreiranje i uklanjanje tabele, tj. redom,

```
CREATEDB    ime-baze-podataka
DESTROYDB   ime-baze-podataka
CREATE      ime-bazne-tabele ( def-kolone {, def-kolone})
DESTROY     ime-bazne-tabele
```

Definicija kolone, *def-kolone*, ima oblik

```
ime-kolone = tip-kolone [ definisanost],
```

a *definisanost* se odnosi na dopuštanje nedostajućih vrednosti (WITH NULL), zabranu nedostajućih vrednosti (NOT NULL), odnosno njihovu zamenu specifičnim

vrednostima odgovarajućeg tipa (NOT NULL WITH DEFAULT, što se i podrazumeva ako se ništa ne navede).

Na primer, iskaz

```
CREATE I (I_SIF  =  C6,
          NAZIV  = C20,
          STATUS = I2,
          DRZAVA = C20)
```

kreira tabelu I sa tri kolone tipa niske znakova (I_SIF, NAZIV, DRZAVA) i sa jednom kolonom tipa kratkog celog broja (STATUS).

6.1.2 Manipulisanje podacima

U svom manipulativnom delu upitni jezik QUEL ima iskaz deklarisanja n-torne promenljive oblika

`RANGE OF` *lista-promenljivih* `IS` *ime-tabele*

Na primer, RANGE OF IX IS I.

Svaka n-torna promenljiva iz liste promenljivih uzima za svoje vrednosti vrste tabele *ime-tabele*.

Iskazi pomoću kojih QUEL komunicira sa bazom jesu iskazi pretraživanja (RETRIEVE), unošenja (APPEND), brisanja (DELETE) i zamene (REPLACE). Opšti oblik ovih iskaza je

iskaz `[[INTO | TO]` *ime-rezultata*`]` `(` *ciljna-lista*`)`
`[WHERE` *logički-izraz*`]`

gde je *iskaz* – RETRIEVE [UNIQUE], APPEND, DELETE ili REPLACE.

Za RETRIEVE i APPEND iskaze, *ime-rezultata* je ime tabele kojoj se dodeljuju pretražene n-torke, odnosno u koju se unose nove n-torke; u iskazu RETRIEVE tada se navodi i rezervisana reč INTO, a u iskazu APPEND – rezervisana reč TO. Za REPLACE i DELETE, *ime-rezultata* je ime n-torne promenljive koja, kroz *logički-izraz*, identifikuje n-torke koje će se modifikovati ili brisati (INTO/TO se ne navodi). Ako se u iskazu RETRIEVE *ime-rezultata* izostavi, pretražene n-torke se izdaju na ekran (štampač). Opcija UNIQUE uz iskaz RETRIEVE ima isti efekat kao opcija DISTINCT u SQL-u.

Ciljna-lista je sastavljena od elemenata dodele oblika

kolona-rezultata `=` QUEL *izraz*

Ovde *kolona-rezultata* jeste kolona rezultujuće tabele, koja treba da dobije vrednost odgovarajućeg QUEL izraza. Ukoliko QUEL izraz jednoznačno imenuje kolonu rezultujuće tabele, *kolona-rezultata* i znak jednakosti se ne navode. U DELETE iskazu nema ciljne liste.

QUEL izraz sastavljen je od "indeksiranih" promenljivih (promenljivih oblika $t.A$, gde je t – n-torna promenljiva, a A – atribut pripadne relacije), konstanti i operacija. Vrednost izraza je tipa kolone-rezultata.

Primer 6.1 Za svaku izdatu knjigu, naći šifru knjige, šifru izdavača i tiraž u hiljadama primeraka.

```
RETRIEVE (KI.K_SIF, KI.I_SIF, OBJASNJ. = 'Tiraz u hiljadama primeraka',
          TuH = KI.TIRAZ/1000)
```

Rezultat:

K_SIF	I_SIF	OBJASNJ.	TuH
k1	i1	Tiraz u hiljadama primeraka	10
k2	i1	Tiraz u hiljadama primeraka	7
k3	i1	Tiraz u hiljadama primeraka	10
k4	i1	Tiraz u hiljadama primeraka	10
k5	i2	Tiraz u hiljadama primeraka	5
k6	i3	Tiraz u hiljadama primeraka	3
k6	i4	Tiraz u hiljadama primeraka	5

Primer 6.2 Naći parove šifara izdavača iz iste države.

```
RANGE OF PRVI  IS I
RANGE OF DRUGI IS I
RETRIEVE (PRVI.I_SIF, DRUGI.I_SIF)
  WHERE PRVI.DRZAVA = DRUGI.DRZAVA AND PRVI.I_SIF < DRUGI.I_SIF
```

Za svaku n-tornu promenljivu koja se javlja u WHERE liniji a ne pojavljuje se u ciljnoj listi, podrazumeva se da je egzistencijalno kvantifikovana. Agregatna funkcija ANY, koja vraća vrednost 0 ako je njen argument prazan skup, a 1 u suprotnom, može takoďe da se upotrebi za modeliranje egzistencijalnog kvantifikatora (ANY(...) = 0 je analogon negacije SQL kvantifikatora NOT EXISTS, v. primer 6.8).

Primer 6.3 Izmeniti status izdavača sa šifrom i1 na 20.

```
REPLACE I (STATUS = 20)
WHERE   I.I_SIF = 'i1'
```

Primer 6.4 Izbrisati izdavača sa šifrom i4.

```
DELETE I WHERE I.I_SIF = 'i4'
```

Primer 6.5 Dodati tabeli K_BC (koja ima iste atribute kao i tabela K) sve vrste tabele K čija je šifra K_SIF jednaka k1 ili k3.

```
APPEND TO K_BC (K_SIF = K.K_SIF, NASLOV = K.NASLOV,
                OBLAST = K.OBLAST)
          WHERE K.K_SIF = 'k1' OR K.K_SIF = 'k3'
```

6.1.3 Agregatne operacije i funkcije

QUEL podržava agregatne funkcije (u terminologiji QUEL-a – *agregatne operacije*) COUNT, AVG, MAX, MIN, SUM, ANY, COUNTU, AVGU i SUMU (u poslednje tri operacije "U" označava jedinstvenost, engl. "unique", što realizuje efekat opcije DISTINCT u SQL-u). Za razliku od SQL-a, u QUEL-u je dopuštena upotreba agregatnih operacija i u WHERE liniji (što čini upotrebu HAVING linije nepotrebnom). Argument agregatne operacije može biti kvalifikovan WHERE linijom ili BY opcijom, pri čemu agregatna operacija u QUEL-u postaje *agregatna funkcija*.

Primer 6.6 Naći prosečni status jugoslovenskih izdavača.

```
RETRIEVE (Z = AVG(I.STATUS WHERE I.DRZAVA = 'Jugoslavija'))
```

Primer 6.7 Naći šifre izdavača koji su izdali više od 10 izdanja.

```
RETRIEVE (KI.I_SIF)
          WHERE COUNT(KI.K_SIF BY KI.I_SIF) > 10
```

Sve n-torne promenljive koje se navode unutar agregatne funkcije su lokalne za agregatnu funkciju, osim promenljivih u BY opciji koja se ponaša slično GROUP BY liniji u SQL-u. Zato se u BY opciji navode n-torne promenljive koje imaju značenje izvan agregatne funkcije, a koje mogu da učestvuju i u logičkom izrazu WHERE linije agregatne funkcije. Sledeći primer to dobro ilustruje.

Primer 6.8 Naći nazive izdavača koji su izdali sve knjige Branka Ćopića.

Ovaj upit je razmatran u odeljku 2.1 kao ilustracija operacije deljenja, i u primeru 3.26 glave 3. Pretpostavimo da je tabela K_BC već kreirana i da se u njoj nalaze sve (i samo) knjige Branka Ćopića. Tada se upit pretraživanja u QUEL-u izražava primenom agregatne funkcije ANY na sledeći način:

```
RETRIEVE (I.NAZIV)
WHERE ANY (K_BC.K_SIF BY I.I_SIF
      WHERE ANY (KI.K_SIF BY K_BC.K_SIF, I.I_SIF
            WHERE KI.I_SIF = I.I_SIF  AND
                  KI.K_SIF = K_BC.K_SIF  ) = 0 ) = 0
```

Argument agregatne funkcije može biti i druga agregatna funkcija.

Primer 6.9 Naći prosečni ukupni tiraž izdavača.

```
RETRIEVE (Z = AVG(SUM(TIRAZ BY KI.I_SIF)))
```

Ovo je primer upita koji se ne može izraziti jednim SQL iskazom. Mogu se konstruisati i razne druge vrste takvih primera.

6.1.4 Relaciona kompletnost QUEL-a

Upitni jezik QUEL je relaciono kompletan, što se jednostavno dokazuje sledećim razmatranjem.

1. Skupovna operacija unije relacija $R(A_1, \ldots, A_n)$, $S(B_1, \ldots, B_m)$, za $m = n$, može se izraziti sledećim QUEL iskazima (pretpostavlja se da je već kreirana tabela T sa n atributa $C_1, C_2, \ldots, C_n$):

   RANGE OF r IS R
   RANGE OF s IS S
   APPEND TO T $(C_1 = r.A_1, \ldots, C_n = r.A_n)$
   APPEND TO T $(C_1 = s.B_1, \ldots, C_n = s.B_n)$

   (slično za operacije preseka i razlike).

2. Skupovna operacija Dekartovog proizvoda relacija R, S može se izraziti sledećim QUEL iskazima:

   RANGE OF r IS R
   RANGE OF s IS S
   APPEND TO T $(C_1 = r.A_1, \ldots, C_n = r.A_n,$
   $C_{n+1} = s.B_1, \ldots, C_{n+m} = s.B_m)$

3. Operacija restrikcije relacije R po logičkom uslovu Ψ može se izraziti QUEL iskazom:

   RANGE OF r IS R
   APPEND TO T $(C_1 = r.A_1, \ldots, C_n = r.A_n)$
   WHERE Ψ

   (Ψ je logički izraz u QUEL sintaksi).

4. Operacija projekcije relacije R na atribute $A_{i_1}, \ldots, A_{i_k}$ može se izraziti QUEL iskazom:

   RANGE OF r IS R
   APPEND TO T $(C_1 = r.A_{i_1}, \ldots, C_k = r.A_{i_k})$

5. Operacija slobodnog Θ-spajanja relacija R, S po paru atributa A_i, B_j može se izraziti QUEL iskazom:

   RANGE OF r IS R
   RANGE OF s IS S
   APPEND TO T $(C_1 = r.A_1, \ldots, C_n = r.A_n,$
   $C_{n+1} = s.B_1, \ldots, C_{n+m} = s.B_m)$
   WHERE $r.A_i \Theta s.B_j$

6. Operacija prirodnog spajanja relacija R, S po paru atributa A_i, B_j može se izraziti QUEL iskazom:

   RANGE OF r IS R
   RANGE OF s IS S
   APPEND TO T $(C_1 = r.A_1, \ldots, C_n = r.A_n, C_{n+1} = s.B_1,$
   $\ldots, C_{n+j-1} = B_{j-1}, C_{n+j} = B_{j+1}, \ldots, C_{n+m-1} = s.B_m)$
   WHERE $r.A_i = s.B_j$

7. Operacija deljenja relacije $R(A_1, A_2)$ po atributu A_2, relacijom $S(B_1)$, može se izraziti QUEL iskazom:

   RANGE OF r_1 IS R
   RANGE OF r_2 IS R
   RANGE OF s IS S
   APPEND TO T $(C_1 = r_1.A_1)$
   WHERE ANY($s.B_1$ BY $r_1.A_1$
   WHERE ANY($r_2.A_1$ BY $s.B_1, r_1.A_1$
   WHERE $r_2.A_1 = r_1.A_1$ AND $r_2.A_2 = s.B_1$) $= 0$) $= 0$

6.1.5 Iskazi DEFINE i MODIFY

Definisanje pogleda u QUEL-u je veoma slično definisanju pogleda u SQL-u. Iskaz definisanja pogleda ima oblik

`DEFINE VIEW` *ime-pogleda* `(` *ciljna-lista*`)` `WHERE` *logički-izraz*

Nad pogledom se mogu izvršavati operacije analogne operacijama nad pogledima u SQL-u. S obzirom na znatno sistematičniju sintaksu QUEL-a, u QUEL-u se mogu izvršavati neke operacije pretraživanja pogleda koje ne mogu u SQL-u. U slučaju ažuriranja pogleda, ovi jezici ponašaju se slično.

Za definisanje fizičke reprezentacije relacije i metoda pristupa, QUEL uključuje iskaz izmene fizičke reprezentacije relacije

`MODIFY` *ime-relacije* `TO` *struktura* `[UNIQUE] ON` *atr_1, ... , atr_k*

gde *struktura* može biti HEAP (gomila, sekvencijalna neuređena reprezentacija), ISAM (sortirana, indeks-sekvencijalna reprezentacija), HASH (direktna, heš reprezentacija), BTREE (dinamička indeksna reprezentacija), HEAPSORT (sekvencijalna uređena reprezentacija), kao i odgovarajuće komprimovane reprezentacije (CHEAP, CISAM, CHASH, CBTREE, CHEAPSORT) (o fizičkoj organizaciji podataka uopšte biće više reči u delu **??**). Na primer,

```
MODIFY I TO BTREE UNIQUE ON I_SIF
```

QUEL poseduje i iskaz za definisanje sekundarnih indeksa koji se odnose na indeksnu organizaciju sa više negrupišućih indeksa (v. glavu 12),

`INDEX ON` *ime-relacije* `IS` *ime-indeksa* *(atr_1, . . . , atr_k)*

Na primer,

```
INDEX ON I IS INDNAZIV (NAZIV).
```

Integritetni deo relacionog modela u QUEL-u se ostvaruje iskazom

`DEFINE INTEGRITY ON` *n-torna-promenljiva* `IS` *logički-izraz*

Ovim iskazom mogu se zadati samo uslovi integriteta koji uključuju logički izraz tipa restrikcije, tj. logički izraz može da uključi samo n-tornu promenljivu iz ovog iskaza. Na primer, DEFINE INTEGRITY ON I IS I.STATUS > 0. Postupkom *modifikacije upita* svaka izmena nad tabelom I dopunjuje se uslovom integriteta definisanim nad tom tabelom.

Integritet entiteta je u QUEL-u podržan CREATE iskazom, dok referencijalni integritet nije podržan (osim posredno, kroz neke od komercijalnih INGRES sumeđa).

6.2 Aplikativni QUEL

Kao i SQL, i QUEL ima svoju aplikativnu varijantu. To je EQUEL (Embedded QUEL), QUEL ugnježden u programski jezik C (kao i u jezike Ada, BASIC, COBOL, FORTRAN, PL/I, Pascal).

Osnovni princip EQUEL-a je princip dualnosti, tj. princip da se svaki QUEL iskaz koji se može izvršiti interaktivno, može upotrebiti i u aplikativnom (EQUEL) programu; obrat ne važi, tj. postoji niz iskaza koji se mogu upotrebiti u aplikativnom, ali ne i u interaktivnom radu.

Osnovne karakteristike EQUEL jezika mogu se sumirati na sledeći način:

- EQUEL program mora da prođe obradu kroz prekompilator, koji izvršne QUEL iskaze zamenjuje pozivima (u matičnom jeziku) izvršnog EQUEL sistema. Zato sve linije aplikativnog programa koje treba da obradi prekompilator počinju znakom ## u prve dve zauzete kolone; to su linije koje sadrže QUEL iskaze, ali i deklaracije matičnog jezika (npr. C) kojima se deklarišu programske promenljive koje će se navesti u nekom QUEL iskazu.
- Izvršni QUEL iskazi u EQUEL programu mogu se pojaviti na svakom mestu gde mogu i izvršni iskazi matičnog jezika.
- Programske promenljive mogu se koristiti u ciljnoj listi RETRIEVE iskaza, kao imena n-tornih promenljivih, kao imena tabela i kolona, unutar QUEL izraza i logičkih izraza.
- EQUEL program može pristupiti samo jednoj bazi podataka u jednom trenutku; za pristup bazi, odnosno za kraj rada sa bazom, koriste se EQUEL iskazi “otvaranja” odnosno “zatvaranja” baze podataka:

```
## INGRES  ime-baze-podataka
## EXIT
```

- Programske promenljive i kolone tabela mogu imati isto ime, ali onda imenu kolone prethodi znak #. Na primer,

```
## RETRIEVE (STAT = I.STATUS, DRZAVA = I.#DRZAVA)
##           WHERE I.I_SIF = DATA_SIFRA
```

  (STAT, DRZAVA i DATA_SIFRA su programske promenljive prethodno deklarisane u skladu sa tipom kolona STATUS, DRZAVA i I_SIF).

Od svih QUEL iskaza, jedini koji se u aplikativnoj varijanti koncepcijski razlikuje od interaktivne jeste iskaz RETRIEVE. Problem predstavlja razlika između skupovno orijentisanog QUEL-a (objekat pretraživanja i rezultat pretraživanja je tabela, dakle skup vrsta) i slogovno orijentisanog programskog jezika. Jedan od načina za rešavanje ovog problema su, kao i u SQL-u, kursori. Drugi način je RETRIEVE *petlja*. Tako se RETRIEVE iskaz bez navođenja rezultujuće tabele, korišćenjem RETRIEVE petlje može predstaviti kao:

```
## RETRIEVE ( ciljna-lista)
##          [WHERE  logički-izraz]
##{
            programski-blok
##}
```

Ovaj iskaz izvršava se tako što se *programski-blok* izvršava po jedanput za svaku vrstu za koju je *logički-izraz* tačan.

Primer 6.10 Uneti u tabelu K vrstu sa vrednostima atributa koje su u programskim promenljivim K_SIF, NASLOV i OBL.

```
## APPEND TO K (#K_SIF = K_SIF, #NASLOV = NASLOV, OBLAST = OBL)
```

Primer 6.11 Izbrisati sva izdanja izdavača čija je država data u programskoj promenljivoj DRZAVA.

```
## DELETE KI
## WHERE KI.I_SIF = I.I_SIF AND I.#DRZAVA = DRZAVA
```

Primer 6.12 Uvećati status svih jugoslovenskih izdavača za vrednost programske promenljive DOD.

```
## REPLACE I (STATUS = I.STATUS + DOD)
##           WHERE I.DRZAVA = 'Jugoslavija'
```

Primer 6.13 Program na EQUEL-u za čitanje (sa ulaza) naziva izdavača i štampanje njegovog statusa može imati sledeći oblik:

```
main()
{
## char NAZIVIZD[20];
## int ST;
while (READ( NAZIVIZD ) )
       {
##            RANGE OF IX IS I
##            RETRIEVE (ST = IX.STATUS)
##            WHERE IX.NAZIV = NAZIVIZD
##{
               printf("Status izdavaca %s", NAZIVIZD, "je %i", ST);
##}
       }
}
```

Za svaku n-torku relacije I za koju je logički izraz tačan, C promenljivoj ST dodeljuje se vrednost atributa STATUS te n-torke, i izvršava se iskaz štampanja printf.

Detaljnije informacije o EQUEL-u mogu se naći u [23], [51].

6.3 Pitanja i zadaci

1. Napisati QUEL iskaze za kreiranje tabela K, P, KI, KP iz odeljka 1.2.
2. Neka je dat upit: Naći sve podatke o izdavačima i broju njihovih izdanja. Da li je sledeći QUEL iskaz korektan zapis ovog upita?

   ```
   RETRIEVE (I.ALL, X = COUNT( KI.IZDANJE BY I.I_SIF ) )
   ```

   Obrazložiti.
3. QUEL iskazima formulisati upite iz zadatka 10 glave 2.
4. QUEL iskazima formulisati upite iz zadatka 7 glave 3.
5. Kojim se QUEL iskazom ostvaruje integritet?
6. Napisati QUEL iskaz kojim se ostvaruju sledeće radnje:
   - Promeniti oblast svim romanima u beletristiku.
   - Izbrisati sve podatke o izdavačima koji nisu izdali nijednu knjigu.
   - Uvećati tiraž svim izdanjima čiji je tekući tiraž manji od najmanjeg tiraža izdavača i3.

- Uneti novog izdavača sa šifrom k5, nazivom Nolit, statusom 30 i državom Jugoslavija.
- Konstruisati novu tabelu sa imenom JK koja sadrži podatke o knjigama čiji je bar jedan autor jugoslovenski državljanin ili ih je izdao bar jedan jugoslovenski izdavač.
- Izbrisati sve izdavače iz Amerike i njihova izdanja.

7. Koje su osnovne karakteristike EQUEL jezika?

8. Koristeći izdavačku bazu podataka napisati EQUEL program koji štampa sve šifre pisaca. Za šifrom pisca treba da slede svi podaci o izdavačima koji izdaju knjige tog pisca.

7

Optimizacija upita

7.1 Katalog

U ovom odeljku, na početku izlaganja o optimizaciji upita, biće reči o sistemskom katalogu kao sistemskoj bazi podataka o bazama podataka. Potreba za ovim izlaganjem postoji zbog važnosti uloge koju katalog ima u procesu optimizacije upita.

Katalog, kao sistemska baza podataka (ili njen deo), sadrži informacije o raznim objektima u sistemu kao što su bazne tabele, pogledi, indeksi, baze podataka, planovi izvršavanja upita (planovi aplikacije, tj. pristupni paketi, v. glavu 18), ovlašćenja za pristup objektima, itd. Katalog se razlikuje od sistema do sistema (INGRES, SYSTEM R, DB2, ORACLE), jer uključuje informacije vezane za sistem. Karakteristično za katalog je da obično uključuje tabelu koja sadrži po jednu n-torku za svaku tabelu u sistemu, uključujući i sistemske tabele, zatim tabelu svih atributa svih tabela u sistemu, tabelu svih indeksa svih tabela u sistemu, kao i tabele pogleda, baza podataka, uslova integriteta, stranih ključeva, autorizacije, optimizovanih upita, sinonima, itd. Ove tabele sadrže informacije o imenu objekta (npr. tabele, atributa, indeksa, baze, pogleda), vlasniku, broju atributa (tabele), tabeli za koju je (atribut, indeks) vezan, tipu atributa, itd.

S obzirom da je sistemski katalog relacionog sistema – relaciona baza podataka, ona se može pretraživati istim upitnim jezikom kao i korisničke baze podataka (npr. SQL, QUEL). Međutim, tabele kataloga ne mogu se ažurirati iskazima ažuriranja upitnog jezika, jer to može da bude vrlo opasno po podatke u bazi i pristup podacima. Tako, ako bi se iz tabele o atributima u katalogu izbrisala n-torka koja se odnosi na neki atribut u nekoj baznoj tabeli, onda bi taj atribut postao nepoznat sistemu, i ne bi mu se moglo više pristupiti, iako bi njegove vrednosti fizički postojale u bazi. Ažuriranje sistemskih tabela u katalogu vrši se "sistemski" – automatski, pri izvršavanju iskaza kreiranja i uklanjanja baznih tabela, pogleda i indeksa, odnosno pri izvršavanju iskaza modifikovanja strukture baznih tabela i indeksa.

Katalog sistema DB2 sadrži više desetina sistemskih baznih tabela i pogleda

u sistemskoj bazi podataka ([24]). Navedimo neke od tih pogleda sa njihovom delimičnom strukturom.

- `SYSSTAT.TABLES (TABSCHEMA, TABNAME, DEFINER, TYPE, TBSPACE, COLCOUNT, CARD, NPAGES, PARENTS, CHILDREN, ... )`

  Za svaku tabelu u sistemu, baznu ili pogled, ova tabela sadrži n-torku u kojoj registruje niz svojstava (više od 30) te tabele: ime tabele (TABNAME), ime sheme (TABSCHEMA), identifikator vlasnika tabele (DEFINER), tip tabele (TYPE – bazna tabela, pogled, drugo ime – alias), ime prostora tabela kome pripada tabela (TBSPACE), broj kolona (COLCOUNT), ukupan broj n-torki tabele (CARD), ukupan broj stranica na kojima se pojavljuju n-torke tabele (NPAGES), itd.

- `SYSSTAT.COLUMNS (TABSCHEMA, TABNAME, COLNAME, COLCARD, ... )`

  Za svaku kolonu svake tabele u sistemu ova tabela sadrži n-torku u kojoj registruje niz svojstava te kolone: ime kolone (COLNAME), ime tabele kojoj kolona pripada (TABNAME), ime sheme kojoj tabela pripada(TABSCHEMA), broj različitih vrednosti u koloni (COLCARD), itd.

- `SYSSTAT.INDEXES (INDSCHEMA, INDNAME, FIRSTKEYCARD, COLCOUNT, FULLKEYCARD, NLEAF, NLEVELS, ... )`

  Za svaki indeks u sistemu ova tabela sadrži n-torku u kojoj registruje ime indeksa (INDNAME), shema indeksa (INDSCHEMA) broj kolona u indeksu (COLCOUNT), broj različitih vrednosti prve navedene kolone indeksa (FIRSTKEYCARD), broj različitih vrednosti indeksa (po svim njegovim kolonama, FULLKEYCARD), broj listova u strukturi indeksa (NLEAF), broj nivoa indeksnog drveta (NLEVELS), i niz drugih svojstava.

Osim ovih tabela čiji je značaj prepoznatljiv, DB2 katalog sadrži i grupe tabela sa informacijama o strukturama (bazama podataka u sistemu, prostorima tabela, izvornim definicijama pogleda, planovima izvršavanja – pristupnim paketima), o autorizaciji (pravima pristupa kolonama, bazama podataka, planovima izvršavanja, sistemskim funkcijama, fizičkim strukturama), kao i tabele o oporavku, integritetu (primarnim, stranim ključevima), itd.

Pretraživanje tabela sistemskog kataloga može se vršiti SQL iskazima. Na primer, iskaz

```
SELECT TABNAME, TABSCHEMA
FROM   SYSSTAT.COLUMNS
WHERE  COLNAME = I_SIF
```

daje imena svih tabela (u ovom slučaju I, KI) i shema u kojima su kreirane, koje imaju kolonu (atribut) I_SIF. SYSSTAT je ime sheme u kojoj se nalaze ovi kataloški pogledi.

7.2 Sistematski pristup optimizaciji

Optimizacija upita je komponenta procesora upitnog jezika koja je neophodna, bar kod velikih sistema i za velike baze podataka, da bi sistem uopšte mogao da zadovolji traženu efikasnost. Posebnu pogodnost kod optimizacije upita pružaju relacioni sistemi, s obzirom na dovoljno apstraktni nivo relacionih izraza kojima se upiti zadaju.

Kod nerelacionih sistema optimizaciju sprovodi korisnik, a ne sistem, jer nerelacioni modeli ne ostvaruju dovoljnu meru razdvajanja logičkog od fizičkog nivoa podataka i pristupa podacima; tako već u formulaciji upita mora biti sadržana i informacija o pristupnom putu i načinu izvršavanja upita. Automatska optimizacija, osim što je kod relacionih sistema moguća a kod nerelacionih nije, daje značajne prednosti u kvalitetu, jer čovek često nije u mogućnosti da postigne kvalitet optimizacije koji postiže sistem. Ova prednost sistema nad čovekom je posledica činjenice da sistem "ima uvid" u vrednosti podataka (a ne samo u njihovu strukturu) koji čovek – korisnik nema, kao i, s obzirom na brzinu rada sistema, posledica mogućnosti proizvođenja većeg broja alternativnih načina izvršavanja upita i procene njihove efikasnosti.

Cilj optimizacije kod upitnih jezika je izbor strategije za izračunavanje datog relacionog izraza, koja bi, kao i u slučaju programskih jezika, omogućila izvesna poboljšanja u odnosu na neoptimizovanu varijantu, a ne izbor optimalne strategije u smislu teorije upravljanja i optimizacije.

Dve osnovne vrste optimizacije koje se sreću u procesorima upitnih jezika jesu strategija algebarske transformacije upita i strategija procene cene. Prva ne zavisi od specifičnih podataka u bazi, dok druga zavisi. Tako bi, na primer, jedno pravilo algebarske transformacije, koje je skoro uvek dobro primeniti, bilo: primeniti restrikciju što je moguće ranije u izvršenju upita. Strategije procene cene koriste, na primer, informaciju (iz sistemskog kataloga) o postojanju indeksa nad tabelom iz upita.

Primer 7.1 Posmatrajmo upit kojim se traže nazivi izdavača koji izdaju knjigu sa šifrom k2; takav upit se u SQL-u može izraziti sledećim SELECT blokom:

```
SELECT DISTINCT I.NAZIV
FROM   I, KI
WHERE  I.I_SIF = KI.I_SIF AND KI.K_SIF = 'k2'
```

Neka baza podataka sadrži 100 izdavača (vrsta u tabeli I) i 10000 izdanja (vrsta u tabeli KI), od kojih se 20 odnosi na izdanja knjige sa šifrom k2. Jedan način izvršavanja ovog upita, bez primene optimizacije, mogao bi se sastojati u sledećem nizu radnji.

1. Izračunati Dekartov proizvod tabela I i KI; pri tom se čita 10100 vrsta sa diska i konstruiše se privremena tabela sa 1000000 vrsta, koja, zbog veličine, mora takođe da se upiše na disk.

2. Izvršiti restrikciju dobijene privremene tabele po WHERE logičkom uslovu, tj. po logičkom izrazu I.I_SIF = KI.I_SIF AND KI.K_SIF = 'k2'; ona uključuje sekvencijalno čitanje 1000000 vrsta sa diska i konstrukciju privremene tabele sa samo 20 vrsta, koja se može zadržati u unutrašnjoj memoriji.

3. Projektovati privremenu tabelu dobijenu u koraku 2 na atribut NAZIV, čime se dobija najviše 20 podataka.

Jasno je da je ovaj postupak, mada neposredno diktiran sintaksom upita, najmanje efikasan. Znatno efikasnije izvršavanje upita dao bi sledeći postupak.

1'. Izvršiti restrikciju tabele KI po uslovu KI.K_SIF = 'k2', čime se sekvencijalno čita 10000 vrsta i proizvodi privremena tabela sa 20 vrsta koja se može zadržati u unutrašnjoj memoriji.

2'. Spojiti privremenu tabelu dobijenu u koraku 1' sa tabelom I po jednakim vrednostima atributa I_SIF; time se čita 100 vrsta tabele I sa diska i proizvodi privremena tabela sa 20 vrsta koja ostaje u unutrašnjoj memoriji.

3'. Projektovati privremenu tabelu dobijenu u koraku 2' na atribut NAZIV.

Ako se broj ulazno/izlaznih operacija nad diskom (broj čitanja stranica sa diska odnosno upisa stranica na disk) uzme kao mera efikasnosti (što je realno s obzirom da te operacije oduzimaju najveći deo vremena), onda je drugi postupak 200 puta efikasniji od prvog. Ako je još tabela KI indeksirana (ili heširana, v. glavu 13) po atributu K_SIF, broj čitanja u koraku 1' je 20 a ne 10000. Indeks ili heš algoritam mogli bi da pomognu i u koraku 2', tako što bi se broj čitanja sa diska smanjio sa 100 na najviše 20.

Prethodni primer je izuzetno jednostavan, pa se zato odmah uočavaju adekvatni elementi optimizacije. U opštem slučaju, međutim, potreban je sistematski pristup optimizaciji, koji uključuje sledeće faze.

I Prevesti upit u internu reprezentaciju koja bi ga učinila neosetljivim na detalje specifične sintakse.

II Prevesti internu reprezentaciju upita u kanoničku formu, kojom bi se anulirale razlike između semantički ekvivalentnih (v. definiciju 7.2) a sintaksno različitih formulacija upita.

III Izabrati kandidate za procedure niskog nivoa kojima se mogu realizovati pojedine operacije sadržane u upitu.

IV Generisati planove izvršavanja kompletnih upita i izabrati najjeftiniji.

Opišimo nešto detaljnije navedene faze.

I Prevođenje upita u internu reprezentaciju podrazumeva njegovo "čišćenje" od specifične sintakse i predstavljanje nekim formalizmom interne reprezentacije. Taj formalizam mora biti dovoljno bogat da omogući predstavljanje svih upita upitnog jezika i što je moguće neutralniji da ne bi uticao na kasnije odluke optimizatora. Najčešći formalizmi interne reprezentacije jesu neki oblik *drveta apstraktne sintakse* (DAS) i odabrana sintaksa relacione algebre. Tako se polazni upit sa početka ovog odeljka može predstaviti DAS-om na slici 7.1 ili izrazom relacione algebre

$$((\text{I} * \text{KI})[\text{K\_SIF} = \text{'k2'}])[\text{NAZIV}].$$

Slika 7.1: Drvo apstraktne sintakse (DAS) upita

II Većina upitnih jezika omogućuje da se proizvoljni upit predstavi na mnogo (manje ili više) različitih načina. Tako, trivijalna razlika se sastoji u promeni strana operanada komutativnih operacija (npr. jednakosti, konjunkcije ili disjunkcije). Netrivijalne razlike bile bi u promenjenom redosledu operacija. Na primer, prethodni upit ((I∗ KI) [K_SIF='k2']) [NAZIV] mogao bi se postaviti i relacionim izrazom (I∗ (KI [K_SIF = 'k2'])) [NAZIV]. Zbog tih sintaksnih razlika u postavljanju semantički ekvivalentnih upita, interna reprezentacija se prevodi u *kanoničku formu*.

Definicija 7.1 (kanonička forma) Neka je dat skup Q objekata i relacija ekvivalencije među objektima. Tada se podskup $C \subseteq Q$ zove *skup kanoničkih formi* skupa Q pri izabranoj relaciji ekvivalencije ako je svaki objekat $q \in Q$ ekvivalentan nekom (tačno jednom) objektu $c \in C$. Objekat c zove se *kanonička forma* objekta q.

U kontekstu optimizacije upita, skup Q je skup upita, a relacija ekvivalencije među objektima je relacija ekvivalencije među upitima. Smisao definicije 7.1 je da omogući da se proučavanjem nekog podskupa C kanoničkih formi, umesto proučavanjem skupa Q svih upita, dokažu bitna svojstva celog skupa Q.

Relaciju ekvivalencije među upitima prirodno je definisati kao semantičku ekvivalenciju upita.

Definicija 7.2 (semantička ekvivalentnost) Upiti q_1 i q_2 su semantički ekvivalentni ako za svaki izbor sadržaja tabela nad kojima su definisani, oba proizvode isti rezultat.

Relaciju semantičke ekvivalencije upita potrebno je sintaksno opisati, što je veoma složen zadatak. Način da se to učini je definisanje pravila transformacije upita koja definišu drugu relaciju ekvivalencije među upitima, *relaciju sintaksne ekvivalencije*. Pri tome se za *sintaksno ekvivalentne* proglašavaju svi oni upiti koji

se tim pravilima mogu transformisati u isti oblik; taj transformisani oblik je njihova kanonička forma. Optimalno bi bilo da se klase semantičke i sintaksne ekvivalencije poklapaju. Obavezno je pak da svi sintaksno ekvivalentni upiti budu i semantički ekvivalentni, tj. da svaka klasa sintaksne ekvivalencije bude podskup neke klase semantičke ekvivalencije, za šta je odgovorna *semantička valjanost* pravila transformacije. Naime, ni jednim pravilom transformacije upit ne sme da promeni svoju klasu semantičke ekvivalencije. Od *kompletnosti* skupa pravila transformacije zavisi koliko će klasa sintaksne ekvivalencije biti sadržano u jednoj klasi semantičke ekvivalencije, a praktični cilj je da taj broj bude što manji.

Ako je skup objekata Q – skup svih mogućih upita interne reprezentacije, skup pravila transformacije upita u kanoničku formu mogao bi da uključi sledeća pravila:

- logičke izraze restrikcije prevesti u konjunktivnu normalnu formu, tj. u konjunkciju prostih poređenja i/ili disjunkcijâ prostih poređenja. Na primer, izraz

  p OR (q AND r) treba zameniti izrazom

  (p OR q) AND (p OR r);

- operacijama restrikcije dati viši prioritet (zagradama) u odnosu na operacije spajanja, tj. izraz oblika

  $(R * S)$[restrikcija-nad-S] treba zameniti izrazom

  (R*(S[restrikcija-nad-S])), a izraz

  $(R * S)$[restrikcija-nad-R AND restrikcija-nad-S] treba zameniti izrazom

  (R[restrikcija-nad-R])*(S[restrikcija-nad-S]);

- niz restrikcija kombinovati u jednu restrikciju, tj. izraz oblika

  (R[restrikcija-1])[restrikcija-2] treba zameniti izrazom

  R[restrikcija-1 AND restrikcija-2];

- u nizu projekcija čije su liste atributa, zdesna ulevo, u relaciji inkluzije, eliminisati sve projekcije osim poslednje, tj. izraz oblika

  (R[lista-atributa-1])[lista-atributa-2], ako je lista-atributa-2 $\subseteq$ lista-atributa-1

  zameniti izrazom

  R[lista-atributa-2];

- restrikciju projekcije zameniti projekcijom restrikcije, tj. izraz oblika

  (R[lista-atributa-1])[restrikcija-1] zameniti izrazom

  (R[restrikcija-1])[lista-atributa-1].

Lista pravila nije kompletna, i različiti sistemi je dopunjuju pravilima za koja se teorijskim i praktičnim istraživanjima pokazuje da su korisna. Tako je jedno, uvek korisno pravilo, na primer, sledeće:

– ako je $S.A$ strani ključ koji se odnosi na primarni ključ $R.A$, onda se izraz $(S * R)[A]$ može zameniti izrazom $S[A]$; na primer, izraz (KI $*$ K)[K_SIF] može da se zameni izrazom KI[K_SIF].

Sistem INGRES primenjuje i pravilo koje proširenjem restrikcije smanjuje obim relacije međurezultata, što je ilustrovano sledećim primerima:

– izraz A.F1 > B.F2 AND B.F2 = 3 može se zameniti izrazom A.F1 > B.F2 AND B.F2 = 3 AND A.F1 > 3;

– izraz I.DRZAVA > P.DRZAVA AND P.DRZAVA > 'Amerika',
može se zameniti izrazom
I.DRZAVA > P.DRZAVA AND P.DRZAVA > 'Amerika' AND I.DRZAVA > 'Amerika'.

III Pošto je upit transformisan u svoju kanoničku formu (bez obzira na to kojim pravilima pojedini sistem realizovao tu transformaciju), sledeći korak treba da odgovori na pitanje kako izračunati transformisani upit. Na ovom nivou razmatraju se pristupni putevi kao što su indeksi ili heš algoritmi, distribucija vrednosti podataka, fizičko grupisanje slogova, itd. Upitni izraz u kanoničkoj formi posmatra se kao niz operacija niskog nivoa – spajanja, restrikcije, itd. Za svaku od tih operacija optimizator ima na raspolaganju skup unapred definisanih procedura za njihovu implementaciju, u zavisnosti od svojstava relacija – argumenata operacije. Na primer, optimizator ima na raspolaganju skup procedura za izvršavanje restrikcije: jednu za '=' restrikciju na jedinstvenom atributu, drugu za indeksirani atribut restrikcije, treću za grupišući atribut restrikcije, itd. Uz svaku od procedura postoji i informacija o njenoj "ceni", tj. o prostoru i vremenu potrebnom za njeno izvršenje.

Upotrebom informacija iz sistemskog kataloga o tekućem stanju baze podataka (na primer, o kardinalnosti relacija i pristupnim putevima), optimizator bira jednu ili više procedura – kandidata za svaku od operacija u upitnom izrazu. Ovaj proces se nekada zove *izbor pristupnog puta*.

IV Poslednja faza u procesu optimizacije je konstrukcija skupa upitnih planova (planova izvršavanja upita), sa izborom najboljeg (na primer, najjeftinijeg). Svaki upitni plan je kombinacija procedurâ od kojih svaka odgovara po jednoj operaciji u upitu. Broj mogućih planova je obično suviše veliki, pa je poželjno korišćenje neke heuristike koja ograničava broj planova koji se generišu. Izbor se vrši na osnovu cene plana, a formula cene je najčešće broj operacija ulaza/izlaza sa diska, mada se nekad u formulu cene uključuje i korišćenje centralnog procesora. Treba napomenuti da je procena cene veoma težak zadatak jer svi upiti (osim najjednostavnijih) generišu tabele – međurezultate koje treba upisati na disk (i čitati sa diska), a čiji je obim teško proceniti jer je neposredno zavisan od vrednosti registrovanih u bazi. U sledećim odeljcima ove glave biće izložene metode optimizacije koje se primenjuju u dva sistema: prvom kompletnom prototipskom relacionom sistemu, SYSTEM R ([56]), i najpoznatijem akademskom relacionom sistemu INGRES ([70]).

7.3 Optimizacija u SYSTEM R

SYSTEM R je prvi kompletni prototipski RSUBP. To je kompilatorski sistem, što znači da se jednom optimizovani upit može izvršiti veliki broj puta, sve dok se ne promene parametri na osnovu kojih je optimizovani upitni plan sačinjen. Ova mogućnost značajno opravdava optimizaciju i njenu cenu. Metoda optimizacije sistema SYSTEM R sa manjim izmenama primenjena je i u svim komercijalnim proizvodima razvijenim iz njega, kao što su DB2 i SQL/DS. Zato će u ovom odeljku biti opisan postupak optimizacije u SYSTEM R.

U prvom koraku optimizator sistema odlučuje o redosledu izvršenja SELECT blokova ugnježdenih jedan u drugi; u SYSTEM R izabran je redosled od unutrašnjeg, redom, ka spoljašnjim blokovima. Zatim se bira najjeftiniji upitni plan za svaki od blokova. Prvi korak služi kao heuristika koja redukuje broj mogućih planova, i čiji nedostaci su razmatrani u [40]. Oni se pre svega odnose na činjenicu da svaki SQL upit koji sadrži ugnježdene blokove može da se transformiše u upit spajanja bez ugnježdenih blokova, pa takva transformacija daje bolje rezultate u optimizaciji celog upita.

Za dati upitni blok, postoje dva slučaja:

a) Ako je upit jednorelacioni (ne uključuje spajanje), za konstrukciju upitnog plana optimizator koristi statističke informacije iz sistemskog kataloga, formule za procenu veličine međurezultata i cene operacija niskog nivoa. Sistem izračunava, na osnovu informacija dobijenih iz kataloga, cenu svakog od konačno mnogo načina za izvršavanje upita, i odabira najpovoljniji.

   Statističke informacije u sistemskom katalogu uključuju:

   – broj n-torki u svakoj relaciji

   – broj stranica koje zauzima svaka relacija

   – procenat stranica koje svaka relacija zauzima u odnosu na ukupan broj relevantnih stranica u bazi podataka

   – broj različitih vrednosti za svaki indeks

   – broj stranica koje zauzima svaki indeks.

   Ovi podaci su statički i ažuriraju se posebnim uslužnim programom koji je potrebno izvršiti posle svakog značajnijeg ažuriranja baze. U SYSTEM R poziv tog programa je UPDATE STATISTICS. U sistemu DB2 poziv odgovarajućeg programa je RUNSTATS u kome se, kao parametar, navodi objekat (tabela, indeks, prostor tabela, itd.) čije će se statističke ocene izračunati i upisati u sistemski katalog.

   Podaci iz sistemskog kataloga imaju presudnu ulogu u izgradnji plana upita. Neka je potrebno izvršiti jednorelacioni upit za koji je iz sistemskog kataloga dobijen broj n-torki (B) i broj stranica koje zauzima relacija (S). Upit se izvršava tako što se primenjuje jedan po jedan uslov restrikcije (koja je oblika

konjunktivne normalne forme), pri čemu se u svakom koraku primenjuje, na prethodno dobijeni međurezultat, najjeftiniji mogući od sledećih postupaka ([66]):

- Ako je uslov – poređenje oblika $A = c$, gde je A atribut a c konstanta, primeniti grupišući indeks nad A (ako postoji) da bi se dobile n-torke koje zadovoljavaju uslov. Ako je I broj različitih vrednosti indeksa, onda je cena ovog koraka (broj obraćanja disku) S/I.
- Ako je uslov – poređenje oblika $A\Theta c$, gde je $\Theta \in \{<, \leq, >, \geq\}$, primeniti grupišući indeks nad A (ako postoji) da bi se dobio odgovarajući podskup relacije. Cena ovog koraka je $S/2$ jer se pretraži približno polovina relacije. (Poređenje $\neq$ nije uključeno, jer se ne očekuje značajna selektivnost; gotovo sve n-torke moraju biti pretražene).
- Ako je uslov – poređenje oblika $A = c$, primeniti negrupišući indeks nad A (ako postoji) da bi se dobile n-torke koje zadovoljavaju uslov. Cena koraka je B/I jer je, s obzirom da je indeks negrupišući, položaj n-torki nezavisan.
- Ako nema indeksa, relacija se može čitati redom, primenjujući sve uslove restrikcije, što ima cenu proporcionalnu sa S.
- Ako je uslov – poređenje oblika $A\Theta c$, gde je $\Theta \in \{<, \leq, >, \geq\}$, primeniti negrupišući indeks nad A (ako postoji). Cena ovog koraka je $B/2$.

b) Ako upit uključuje dve ili više relacija koje treba da se spoje, optimizator prvo odabira pristupni put za svaku od relacija na način opisan pod a), a zatim odlučuje o redosledu izvođenja spajanja. Ove dve radnje su zavisne, jer se strategija za pristup pojedinačnoj relaciji bira uzimajući u obzir i karakteristike rezultata koje su od značaja za operaciju spajanja (npr. sortiranost neke od relacija rezultata koje učestvuju u spajanju). Niz spajanja odvija se strogo sekvencijalno, tj. izraz $A * B * C * D$ izvršava se kao izraz $(((A * B) * C) * D)$, što takođe predstavlja jednu od heuristika kojima se smanjuje broj mogućih planova upita.

Za izvršenje specifične operacije spajanja, npr. $A * B$, SYSTEM R koristi jednu od dve metode: metodu ugnježdenih petlji ili metodu sortiranja/spajanja. Izbor metode se vrši na osnovu formula cene.

- Metoda ugnježdenih petlji može se opisati sledećim algoritmom:
  BEGIN
    FOR svaku n-torku μ relacije A DO
      FOR svaku n-torku ν relacije B DO
        IF n-torke μ i ν imaju jednake vrednosti na atributima spajanja
        THEN konstruisati spojenu (μ, ν)-torku
  END

  Razna poboljšanja metode postižu se korišćenjem indeksa (grupišućeg ili negrupišućeg) po atributima spajanja.

– Metoda sortiranja/spajanja sastoji se od sortiranja relacija A, B po vrednostima atributa spajanja, i spajanja sortiranih relacija po jednakosti atributa. Pristup torkama relacija A i B u fizičkom redosledu sada je najefikasniji, a sam algoritam se može realizovati tako da se spajanje dveju relacija obavi jednim čitanjem svake od njih (v. odeljak 11.3).

U slučaju višestrukog spajanja, npr. $(A * B) * C$, obe metode dopuštaju da se svaka proizvedena n-torka relacije $A * B$ preda procesu koji je spaja sa n-torkama relacije C, bez proizvođenja cele relacije $A * B$.

Detaljne formule cene optimizacije višerelacionih upita u SYSTEM R mogu se naći u [66]. Prema autorima sistema ([56]), cena optimizacije kompleksnih upita u sistemu SYSTEM R je samo nekoliko hiljada bajtova memorije i nekoliko desetih delova sekunde na IBM/370 sistemu, što je zanemarljivo u slučaju da se jednom optimizovani upit izvrši veći broj puta.

Sistem DB2 koristi i dodatne tehnike optimizacije u odnosu na svoj prototip. Tako, na primer, upit koji se odnosi na broj n-torki u tabeli I koje zadovoljavaju uslov STATUS = 20,

```
SELECT COUNT(*)
FROM   I
WHERE  STATUS = 20
```

DB2 izvršava bez pristupa samoj tabeli, brojanjem elemenata podataka indeksa atributa STATUS (za vrednost 20) ako takav indeks postoji.

7.4 Optimizacija u INGRES-u

Relacioni sistem INGRES primenjuje zanimljiv i efikasan algoritam obrade upita koji se zove *dekompozicija upita* ([70]). Algoritam se sastoji od razlaganja upita nad više n-tornih promenljivih u niz podupita nad pojedinačnim n-tornim promenljivim, primenom postupaka *razdvajanja* i *supstitucije n-torki*.

Razdvajanje je proces uklanjanja komponente upita koja ima tačno jednu zajedničku promenljivu sa ostatkom upita. U tekstu koji sledi prikazan je postupak razdvajanja na primeru upita "naći nazive izdavača iz Jugoslavije koji su izdali bar jednu knjigu poezije u tiražu ne manjem od 5000". Upitnim jezikom QUEL taj upit bi se izrazio na sledeći način:

```
    RANGE OF I  IS I
    RANGE OF K  IS K
    RANGE OF KI IS KI
(1) RETRIEVE (I.NAZIV) WHERE I.DRZAVA = 'Jugoslavija'  AND
              I.I_SIF  = KI.I_SIF AND KI.TIRAZ  >= 5000 AND
              KI.K_SIF = K.K_SIF  AND K.OBLAST  = 'poezija'
```

U upitu su prisutne tri n-torne promenljive I, K i KI. Svaka jednopromenljiva komponenta upita (deo logičkog izraza upita koji se odnosi samo na jednu n-tornu promenljivu) zadovoljava uslov razdvajanja, pa se može ukloniti iz upita izdvajanjem u posebni upit. Primenjeno na promenljivu K, na primer, razdvajanje proizvodi sledeća dva upita, ekvivalentna polaznom upitu (1):

```
(2) RETRIEVE INTO K'(K.K_SIF) WHERE K.OBLAST = 'poezija'

    RETRIEVE (I.NAZIV) WHERE  I.DRZAVA = 'Jugoslavija' AND
(3)           I.I_SIF  = KI.I_SIF AND KI.TIRAZ  >= 5000 AND
              KI.K_SIF = K'.K_SIF
```

Slično se iz upita (3), razdvajanjem po promenljivoj KI, dobija sledeći par upita ekvivalentan upitu (3):

```
(4) RETRIEVE INTO KI'(KI.I_SIF, KI.K_SIF) WHERE KI.TIRAZ  >= 5000

(5) RETRIEVE (I.NAZIV) WHERE  I.DRZAVA = 'Jugoslavija' AND
              I.I_SIF = KI'.I_SIF AND KI'.K_SIF = K'.K_SIF
```

Na kraju, iz upita (5), razdvajanjem po promenljivoj I dobija se sledeći par upita ekvivalentan upitu (5):

```
(6) RETRIEVE INTO I'(I.I_SIF, I.NAZIV) WHERE I.DRZAVA='Jugoslavija'

(7) RETRIEVE (I'.NAZIV) WHERE I'.I_SIF=KI'.I_SIF AND KI'.K_SIF=K'.K_SIF
```

Pošto nema više jednopromenljivih komponenti u dobijenom upitu (7), postupak se nastavlja razdvajanjem po dvopromenljivom upitu koji uključuje promenljive KI' i K'. Tako je upit (7) ekvivalentan sledećem paru upita:

```
(8)  RETRIEVE INTO KI''(KI'.I_SIF) WHERE KI'.K_SIF = K'.K_SIF

(9)  RETRIEVE (I'.NAZIV) WHERE I'.I_SIF = KI''.I_SIF
```

Dakle, polazni upit (1) dekomponovan je u tri jednopromenljiva upita – (2), (4) i (6), i dva dvopromenljiva upita – (8) i (9). Jednopromenljivi upiti (2), (4) i (6) mogu se obraditi u bilo kom redosledu, a takođe i paralelno. Upiti (8) i (9) ne mogu se dalje dekomponovati razdvajanjem, pa se do njihove dekompozicije na jednopromenljive upite dolazi drugim postupkom – supstitucijom n-torki.

Supstitucija n-torki je postupak zamene jedne promenljive u upitu, redom, jednom po jednom n-torkom relacije koja odgovara toj promenljivoj. Taj postupak se u razmatranom primeru primenjuje na sledeći način.

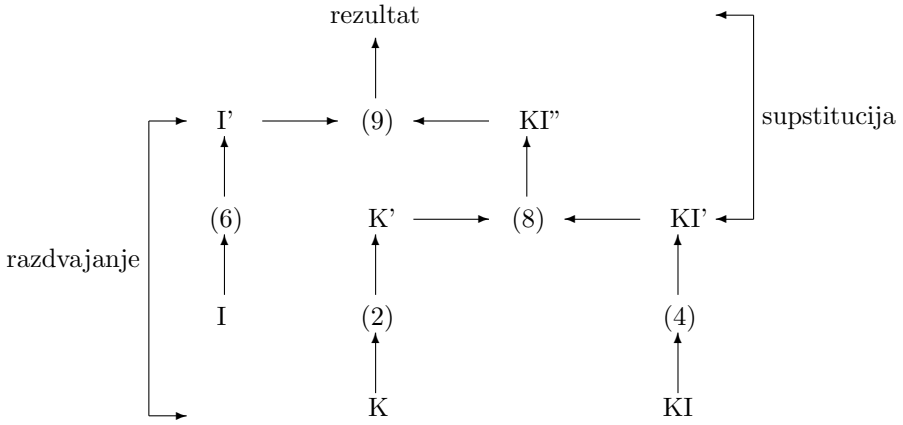
Ako, na primer, skup vrednosti atributa K_SIF u relaciji K' jeste {k2, k6}, onda se dvopromenljivi upit (8) zamenjuje jednopromenljivim upitom u kome obe vrednosti iz tog skupa zamenjuju, redom, komponentu K'.K_SIF. Tako se upit (8) zamenjuje upitom

```
RETRIEVE INTO KI''(KI'.I_SIF) WHERE KI'.K_SIF='k2' OR KI'.K_SIF='k6'
```

Slično, kako skup vrednosti atributa I_SIF u relaciji KI" jeste {i1, i4}, dvopromenljivi upit (9) zamenjuje se jednopromenljivim u kome svaka od dve vrednosti iz tog skupa zamenjuje, redom, komponentu KI".I_SIF. Tako se upit (9) zamenjuje upitom

```
RETRIEVE (I'.NAZIV) WHERE I'.I_SIF = 'i1'  OR I'.I_SIF = 'i4'
```

Slika 7.2 predstavlja drvo dekompozicije polaznog upita (1). Grana čiji je ulazni čvor upit oblika (i) ima, kao svoj izlazni čvor, relaciju koja učestvuje u izgradnji tog upita. Grana čiji je izlazni čvor upit oblika (i) ima, kao svoj ulazni čvor, relaciju – rezultat tog upita. Čvor koji ima jednu ulaznu granu predstavlja jednopromenljivi upit (takvi su čvorovi (2), (4) i (6)), dobijen razdvajanjem po jednoj n-tornoj promenljivoj. Čvor koji ima više od jedne ulazne grane predstavlja višepromenljivi upit (takvi su čvorovi (8) i (9)), dobijen razdvajanjem po dve ili više n-tornih promenljivih; ovakav upit izvršava se supstitucijom n-torki.



Slika 7.2: Drvo dekompozicije upita

Prilikom primene postupka supstitucije n-torki, optimizator bira relacije na koje će primeniti supstituciju. INGRES takvu odluku donosi na osnovu kardinalnosti relacija, pristupnih puteva (npr. indeksa), i uzimajući u obzir koja se relacija (tj. n-torna promenljiva) nalazi u ciljnoj listi.

Postupak supstitucije n-torki je manje efikasan od postupka razdvajanja, i sprovodi se samo kad razdvajanje više nije moguće.

Optimizacija jednorelacionog upita u sistemu INGRES svodi se na statistike sadržane u katalogu, i slična je optimizaciji u sistemu SYSTEM R.

7.5 Pitanja i zadaci

1. Definisati sledeće pojmove:

katalog	drvo apstraktne sintakse
algebarska transformacija upita	dekompozicija upita
strategija procene cene	metoda sortiranja/spajanja
kanonička forma	metoda ugnježdenih petlji

2. Opisati strukturu važnijih relacija sistemskog kataloga.
3. Nabrojati i opisati faze sistematske optimizacije upita.
4. Opisati optimizaciju upita u SYSTEM R i u sistemu INGRES.

Deo III

Logičko projektovanje

Pod logičkim projektovanjem relacione baze podataka podrazumeva se projektovanje sheme relacione baze podataka, tj. grupisanje pojedinih atributa u posebne relacije, ili izdvajanje uočenih entiteta i odnosa među entitetima u posebne relacije, a sve to na osnovu analize informacionih zahteva i informacionog okruženja. Značaj dobrog logičkog projektovanja relacione baze podataka leži u činjenici da je takvu bazu moguće najefikasnije održavati, i da je manipulisanje podacima u njoj najjednostavnije, najefikasnije i najpouzdanije.

Različite metodologije logičkog projektovanja dolaze do sheme relacione baze različitim postupcima, pri čemu se već u opisu značenja "logičkog projektovanja" uočavaju dva globalno različita postupka: jedan se zasniva na grupisanju atributa – svojstava informacionog okruženja – u posebne relacije, a drugi na izdvajanju entiteta i odnosa u posebne relacije. Prvi je "sintaksno" orijentisan i vođen vezama među atributima izraženim kroz koncepte zavisnosti (funkcionalnih, višeznačnih, zavisnosti spajanja), a drugi je "semantički" orijentisan i vođen je vezama među entitetima informacionog okruženja. Metoda *normalnih formi* dosledno i kompletno realizuje prvi postupak logičkog projektovanja, dok razne metode *semantičkog modeliranja* realizuju drugi postupak.

Deo **III** sastoji se od tri glave:

- U glavi 8, **Teorija zavisnosti**, razmatra se teorijska osnova metode normalnih formi – teorija zavisnosti.
- Glava 9, **Normalne forme**, uvodi pojmove normalnih formi višeg reda, i prikazuje algoritme dekompozicije relacija u skupove normalizovanih relacija, odnosno algoritam sinteze normalizovanih relacija.
- U glavi 10, **Semantičko modeliranje**, izlažu se metode semantičkog modeliranja zasnovane na proširenom relacionom modelu, proširenom modelu entiteta i odnosa i modeliranju apstrakcija.

Značajno je napomenuti da svaka metoda koja se dobro i dosledno sprovede u procesu logičkog projektovanja, proizvodi kao rezultat dobro logički projektovanu bazu; štaviše, rezultat takve primene različitih metoda najčešće je gotovo identičan.

8

Teorija zavisnosti

8.1 Funkcionalne zavisnosti i dekompozicija

Kao što je već naglašeno u kratkom opisu integritetnog dela relacionog modela (odeljak 1.4), funkcionalne zavisnosti (skraćeno FZ) jesu najvažniji oblik specifičnih uslova integriteta relacije.

S druge strane, značaj funkcionalnih zavisnosti leži u činjenici da one predstavljaju osnovni koncept na kome se zasniva jedna metoda logičkog projektovanja baze podataka – metoda normalnih formi. Naime, postojanje određene funkcionalne zavisnosti u relaciji dovoljan je uslov za zamenu te relacije, bez gubljenja informacije, njenim dvema projekcijama, određenim tom funkcionalnom zavisnošću. "Bez gubljenja informacije" znači da se početna relacija može restaurisati primenom operacije prirodnog spajanja na dobijene projekcije.

Na primer, neka u relaciji INSD = I [NAZIV, STATUS, DRZAVA] važi funkcionalna zavisnost NAZIV → STATUS, DRZAVA, i neka je sadržaj relacije I isti kao u odeljku 1.2. Tada je sadržaj relacije INSD sledeći:

INSD

NAZIV	STATUS	DRZAVA
Prosveta	30	Jugoslavija
Addison Wesley Publ. Comp.	20	Amerika
Dečje novine	10	Jugoslavija
Matica srpska	30	Jugoslavija

Projektovanjem relacije INSD na parove atributa (NAZIV, STATUS), odnosno (NAZIV, DRZAVA), dobijaju se sledeće relacije:

INSD[NAZIV, STATUS]

NAZIV	STATUS
Prosveta	30
Addison Wesley Publ. Comp.	20
Dečje novine	10
Matica srpska	30

INSD[NAZIV, DRZAVA]

NAZIV	DRZAVA
Prosveta	Jugoslavija
Addison Wesley Publ. Comp.	Amerika
Dečje novine	Jugoslavija
Matica srpska	Jugoslavija

Njihovim spajanjem dobija se polazna relacija INSD, tj. važi

$$\text{INSD} = \text{INSD}[\text{ NAZIV, STATUS }] * \text{INSD}[\text{ NAZIV, DRZAVA }].$$

U slučaju da u prethodnoj relaciji INSD atributi STATUS, DRZAVA ne zavise funkcionalno od atributa NAZIV (što znači da izdavači sa istim nazivom mogu biti registrovani u raznim državama ili imati različite statuse), projektovanjem relacije INSD na iste parove atributa može doći do gubljenja informacije. Tako, ako se u relaciji I nalazi i četvorka (i5, Prosveta, 10, Srbija), onda se relaciji INSD dodaje trojka (Prosveta, 10, Srbija), pa se projektovanjem relacije INSD na parove atributa (NAZIV, STATUS) i (NAZIV, DRZAVA) dobijaju relacije

INSD[NAZIV, STATUS]

NAZIV	STATUS
Prosveta	30
Addison Wesley Publ. Comp.	20
Dečje novine	10
Matica srpska	30
Prosveta	10

INSD[NAZIV, DRZAVA]

NAZIV	DRZAVA
Prosveta	Jugoslavija
Addison Wesley Publ. Comp.	Amerika
Dečje novine	Jugoslavija
Matica srpska	Jugoslavija
Prosveta	Srbija

Njihovim spajanjem dobija se sledeća relacija:

NAZIV	STATUS	DRZAVA
Prosveta	30	Jugoslavija
Prosveta	30	Srbija
Addison Wesley Publ. Comp.	20	Amerika
Dečje novine	10	Jugoslavija
Matica srpska	30	Jugoslavija
Prosveta	10	Jugoslavija
Prosveta	10	Srbija

Dakle, u ovom slučaju INSD[NAZIV, STATUS] $*$ INSD[NAZIV, DRZAVA] $\neq$ INSD.

Prethodno neformalno tvrđenje može se formalizovati i dokazati na sledeći način.

Teorema 8.1 *Neka je Atr(R) skup atributa relacije R. Ako u relaciji R za dva neprazna podskupa $X, Y \subseteq Atr(R)$ važi FZ $X \rightarrow Y$, tada važi jednakost $R = R[XY] * R[XZ]$[1], gde je $Z = Atr(R) \setminus XY$.*

Dokaz: Razlikujemo tri slučaja:

1. Neka je $Y \subseteq X$ (*trivijalna* FZ – vidi sledeći odeljak). Tada tvrđenje važi jer je projekcija $R[XZ]$ jednaka celoj relaciji R.

2. Neka su skupovi X i Y disjunktni. Dokažimo jednakost $R = R[XY] * R[XZ]$.

   Inkluzija $R \subseteq R[XY] * R[XZ]$ je trivijalna i važi u svakoj relaciji, jer iz definicije operacija projekcije i prirodnog spajanja sledi da za svaku n-torku $xyz \in R$ važi da je $xy \in R[XY]$ i $xz \in R[XZ]$, pa je $xyz \in R[XY] * R[XZ]$.

   Dokažimo obrnutu inkluziju, tj. da za svaku n-torku $r \in R[XY] * R[XZ]$ važi da je $r \in R$. Neka je $r = xyz$, gde je $x = r[X]$, $y = r[Y]$, $z = r[Z]$. Iz $xyz \in R[XY] * R[XZ]$ sledi da postoje n-torke $t_1, t_2 \in R$ takve da je $t_1 = xyz'$ i $t_2 = xy'z$, za neke y', z'. Iz FZ $X \rightarrow Y$ sledi da za dve n-torke $xyz' \in R$ i $xy'z \in R$ važi da je $y = y'$. Zato je $t_2 = xyz$, tj. $r(\equiv t_2) \in R$. Dakle, $R[XY] * R[XZ] \subseteq R$, što sa prethodnom (trivijalnom) inkluzijom daje traženu jednakost.

3. Neka $Y \not\subseteq X$ i neka skupovi X i Y imaju neprazan presek. Tada se skup Y može predstaviti kao $Y = Y'Y''$, gde je $Y' = X \cap Y$ i $Y'' \neq \emptyset$. Tada važi i FZ $X \rightarrow Y''$ (po aksiomi F4, v. sledeći odeljak), pri čemu su skupovi X i Y'' disjunktni. Na osnovu slučaja 2) sledi jednakost $R = R[XY''] * R[XZ]$ ($Z = Atr(R) \setminus XY''$), a zbog $XY'' \equiv XY$ i jednakost $R = R[XY] * R[XZ]$.

Dakle, tvrđenje teoreme važi u sva tri slučaja.

Ako u relaciji R važi funkcionalna zavisnost $X \rightarrow Y$, onda se kaže da relacija R *zadovoljava* tu funkcionalnu zavisnost.

Relacija R je dinamičkog sadržaja, što znači da njen sadržaj u raznim momentima može biti različit. *Stanje* r relacije R je njen sadržaj u nekom momentu. Kazaćemo da stanje r relacije R *zadovoljava* FZ $X \rightarrow Y$ ako ta funkcionalna zavisnost važi u relaciji R u momemtu koji definiše njeno stanje r. Sada se za relaciju R može reći da zadovoljava FZ $X \rightarrow Y$ ako svako njeno stanje r zadovoljava tu FZ.

Važenje funkcionalne zavisnosti $X \rightarrow Y$ u stanju r relacije R može se ispitati sledećim algoritmom:

[1] Ovde se, kao i u daljem tekstu, pod XY podrazumeva unija skupova atributa X i Y, bez ponavljanja eventualnih zajedničkih atributa.

Algoritam ZADOVOLJAVA
Ulaz: stanje r relacije R i FZ $X \to Y$;
Izlaz: TAČNO ako r zadovoljava FZ $X \to Y$, NETAČNO u suprotnom;
ZADOVOLJAVA(r, $X \to Y$)
BEGIN
    pregrupisati stanje r relacije R po vrstama tako da vrste sa jednakim X-vrednostima budu zajedno;
    ako svaki skup vrsta sa jednakim X-vrednostima ima takoďe i jednake Y-vrednosti, onda na izlazu izdati TAČNO, a u suprotnom NETAČNO
END.

8.2 Aksiome izvoďenja

Za svako stanje r relacije R postoji skup funkcionalnih zavisnosti koje to stanje zadovoljava. Problem sa održavanjem baze podataka nastaje u slučaju da jedno stanje r relacije R zadovoljava jedan skup funkcionalnih zavisnosti, a da drugo stanje r' iste relacije ne zadovoljava taj skup funkcionalnih zavisnosti. Zato je potrebno precizirati skup funkcionalnih zavisnosti F koje mora da zadovolji svako stanje r relacije R.

Skup funkcionalnih zavisnosti koje zadovoljava stanje r relacije R je konačan, jer je skup podskupova skupa $Atr(R)$ konačan. Zato se za svaki skup funkcionalnih zavisnosti F, primenom algoritma ZADOVOLJAVA, može utvrditi da li stanje r zadovoljava funkcionalne zavisnosti iz F ili ne. Međutim, ovaj je postupak nedopustivo dug. Zbog toga je poželjno imati mehanizam koji obezbeďuje sledeće: ako je poznato da stanje r relacije R zadovoljava neke funkcionalne zavisnosti F_1 iz F, tada to stanje nužno zadovoljava i ostale funkcionalne zavisnosti iz F (one se mogu *izvesti* iz F_1).

Za skup funkcionalnih zavisnosti F kaže se da *povlači* funkcionalnu zavisnost $X \to Y$ (u oznaci $F \models X \to Y$, sa interpretacijom da je FZ $X \to Y$ logička posledica od F) ako svako stanje r relacije R koje zadovoljava funkcionalne zavisnosti iz F, zadovoljava i funkcionalnu zavisnost $X \to Y$.

Aksiome izvoďenja su pravila koja utvrďuju važenje nekih funkcionalnih zavisnosti u relaciji R, na osnovu važenja nekih drugih funkcionalnih zavisnosti u R. Mada nazvana aksiomama, ova pravila su tvrďenja koja se mogu dokazati polazeći od definicija FZ i operacija relacione algebre. Među aksiomama izvoďenja karakteristične su sledeće, poznate kao Armstrongove aksiome ([3]). Tvrďenja se navode sa okosnicom dokaza, ali ne u izvornom obliku (kako ih je formulisao Armstrong), već u pogodnijem, prema [48]:

F1. *Refleksivnost*. Funkcionalna zavisnost $X \to X$ uvek važi u relaciji R, jer skup $R[X = x][X]$ sadrži najviše jednu n-torku (v. definiciju FZ, odeljak 1.4).

F2. *Proširenje*. Ako u relaciji R važi FZ $X \to Y$ i ako je $Z \subseteq Atr(R)$, onda važi i FZ $XZ \to Y$.

Ako R zadovoljava FZ $X \to Y$, onda skup $R[X = x][Y]$ sadrži najviše jednu n-torku za svako x. Tada je $R[XZ = xz] \subseteq R[X = x]$, pa je i $R[XZ = xz][Y] \subseteq R[X = x][Y]$. Zato i skup $R[XZ = xz][Y]$ sadrži najviše jednu n-torku, tj. relacija R zadovoljava funkcionalnu zavisnost $XZ \to Y$.

Primer 8.1 Jedno stanje r relacije $R(A, B, C, D)$ može biti

r

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_1	d_1
a_1	b_1	c_1	d_2
a_3	b_3	c_2	d_3

i ono zadovoljava funkcionalnu zavisnost $A \to B$, a zbog F2 zadovoljava i sledeće FZ: $AB \to B, AC \to B, AD \to B, ABC \to B, ABD \to B, ACD \to B, ABCD \to B$.

F3. *Aditivnost.* Ako u relaciji R važe FZ $X \to Y$ i $X \to Z$, onda u njoj važi i FZ $X \to YZ$.

Naime, ako ne bi važila FZ $X \to YZ$, onda bi skup $R[X = x][YZ]$ sadržao više od jedne n-torke, pa bi bar jedan od skupova $R[X = x][Y]$, $R[X = x][Z]$ imao više od jedne n-torke, što po pretpostavci nije slučaj. Dakle, važi i FZ $X \to YZ$ u relaciji R. Ova aksioma dopušta objedinjenje dve FZ s jednakim levim stranama. U prethodnom primeru, stanje r relacije R zadovoljava FZ $A \to B$ i $A \to C$, pa prema aksiomi F3 sledi i važenje FZ $A \to BC$.

F4. *Projektivnost.* Ako važi FZ $X \to YZ$ u relaciji R, onda u njoj važe i FZ $X \to Y$ i $X \to Z$.

Naime, zbog važenja FZ $X \to YZ$, skup $R[X = x][YZ]$ sadrži najviše jednu n-torku; zato i njegove projekcije na Y i Z, tj. skupovi $R[X = x][YZ][Y](\equiv R[X = x][Y])$ i $R[X = x][YZ][Z](\equiv R[X = x][Z])$ imaju najviše po jednu n-torku, pa važe FZ $X \to Y$ i $X \to Z$. Tako u prethodnom primeru važi FZ $A \to BC$, pa prema aksiomi F4, važe FZ $A \to B$ i $A \to C$. U izvesnom smislu aksioma projektivnosti je inverzna aksiomi aditivnosti.

F5. *Tranzitivnost.* Ako važe FZ $X \to Y$ i $Y \to Z$ u relaciji R, onda u njoj važi i FZ $X \to Z$.

Naime, prema samoj definiciji FZ, za svake dve n-torke $t_1, t_2 \in R$ važe implikacije

$$\begin{aligned} t_1[X] = t_2[X] &\Rightarrow t_1[Y] = t_2[Y] \quad (\text{zbog FZ } X \to Y) \\ t_1[Y] = t_2[Y] &\Rightarrow t_1[Z] = t_2[Z] \quad (\text{zbog FZ } Y \to Z), \end{aligned}$$

pa zbog tranzitivnosti relacije jednakosti sledi i

$$t_1[X] = t_2[X] \Rightarrow t_1[Z] = t_2[Z].$$

Aksioma tranzitivnosti je jedna od dve najmoćnije aksiome izvođenja (druga je F6 – pseudotranzitivnost).

Primer 8.2 Sledeće stanje r relacije $R(A, B, C, D)$

r

A	B	C	D
a_1	b_1	c_2	d_1
a_2	b_2	c_1	d_2
a_3	b_1	c_2	d_1
a_4	b_1	c_2	d_3

zadovoljava FZ $A \rightarrow B$ i $B \rightarrow C$, pa prema aksiomi F5, i FZ $A \rightarrow C$.

F6. *Pseudotranzitivnost.* Ako važe FZ $X \rightarrow Y$ i $YZ \rightarrow W$ u relaciji R, onda u njoj važi i FZ $XZ \rightarrow W$. Ovo tvrđenje može se izvesti iz prethodnih aksioma kroz sledeće korake:

1) $Z \rightarrow Z$ (prema F1)
2) $XZ \rightarrow Z$ (prema F2)
3) $X \rightarrow Y$ (pretpostavka)
4) $XZ \rightarrow Y$ (prema F2)
5) $XZ \rightarrow YZ$ (prema F3, iz 2) i 4))
6) $YZ \rightarrow W$ (pretpostavka)
7) $XZ \rightarrow W$ (prema F5, iz 5) i 6)).

Kao posledice aksioma izvođenja važe i sledeća tvrđenja:

1. Za svaki par podskupova X, Y skupa atributa relacije R, za koji važi $Y \subseteq X$, važi FZ $X \rightarrow Y$. Ovo je *trivijalna* FZ, a njeno važenje se dokazuje primenom aksioma refleksivnosti i proširenja.

2. Ako u relaciji R važe FZ $X \rightarrow Y$ i $XY \rightarrow Z$, onda u relaciji R važi i FZ $X \rightarrow Z$. Ova posledica se dokazuje primenom aksiome pseudotranzitivnosti.

Aksiome F1 – F6 navedenog sistema aksioma izvođenja nisu *nezavisne*, što znači da su neke od njih posledica preostalih. Na primer, aksioma F5 je specijalni slučaj aksiome F6, za $Z = \emptyset$. Takođe, aksioma F6 sledi iz aksioma F1, F2, F3 i F5, što je upotrebljeno u njenom dokazu. S druge strane, može se dokazati da je sistem aksioma F1 – F6 *potpun* u smislu da se svaka FZ koja je logička posledica skupa funkcionalnih zavisnosti F, može dobiti primenom aksioma F1 – F6 na skup F ([48], [65]).

Može se takođe dokazati da jedan podskup skupa aksioma F1 – F6, koji je istovremeno i potpun i nezavisan, jeste skup {F1, F2, F6}.

Jedna od pretpostavki relacionog modela je pretpostavka o jedinstvenosti funkcionalnih zavisnosti. Ne mogu, naime, postojati dve FZ $f : X \rightarrow Y$ i $g : X \rightarrow Y$, sa različitom semantikom. Na primer, nije dopušteno postojanje FZ f: RADNIK $\rightarrow$ SEKTOR sa značenjem da jedan radnik radi tačno u jednom sektoru i g: RADNIK $\rightarrow$ SEKTOR sa značenjem da jedan radnik učestvuje u upravnom odboru tačno jednog sektora.

Pojava nejedinstvenih funkcionalnih zavisnosti nije dopuštena ni kao posledica algebre funkcionalnih zavisnosti o kojoj je upravo bilo reči. Tako, na primer ([7]), FZ f: SEKTOR $\rightarrow$ DIREKTOR i g:{DIREKTOR, SPRAT} $\rightarrow$BROJ_RADNIKA mogu imati značenje da sektor jednoznačno određuje direktora, i da direktor i sprat jednoznačno određuju broj radnika jednog direktora na jednom spratu. Primenom pseudotranzitivnosti može se dobiti FZ {SEKTOR, SPRAT} $\rightarrow$ BROJ_RADNIKA, koja određuje broj radnika *direktora* specifičnog sektora na specifičnom spratu (a ne broj radnika specifičnog *sektora* na specifičnom spratu). Ako direktor može da upravlja većim brojem sektora, onda dobijena FZ nije semantički ista kao sintaksno identična FZ koja bi označavala broj radnika specifičnog *sektora* na specifičnom spratu.

Problem nejedinstvenosti funkcionalne zavisnosti obično se rešava preimenovanjem atributa. U prvom primeru, atribut RADNIK iz FZ g može se preimenovati u atribut RADNIK_UPRAVA, jer se semantika atributa RADNIK u FZ f, g bitno razlikuje. U drugom primeru atribut BROJ_RADNIKA iz FZ g trebalo bi preimenovati u BROJ_RADNIKA_DIREKTORA, pa bi se primenom pseudotranzitivnosti dobila FZ {SEKTOR, SPRAT} $\rightarrow$ BROJ_RADNIKA_DIREKTORA koja bi sada bila i sintaksno različita od FZ {SEKTOR, SPRAT} $\rightarrow$ BROJ_RADNIKA.

8.3 Zatvorenje skupa funkcionalnih zavisnosti

Ako je zadat skup F funkcionalnih zavisnosti jedne relacije, niz pitanja koja se postavljaju u vezi sa njim uključuje:

1. Kako konstruisati skup svih funkcionalnih zavisnosti koje se mogu izvesti iz F po Armstrongovim aksiomama?
2. Da li je zadata FZ logička posledica skupa F?
3. Kako konstruisati najmanji (po broju FZ) skup FZ koji generiše sve FZ kao i F?

Prva dva pitanja odnose se na konstrukciju *zatvorenja* skupa FZ, a treće – na konstrukciju *pokrivanja* skupa FZ. Značaj ovih konstrukcija je u povećanju efikasnosti održavanja integriteta baze podataka, kao i u logičkom projektovanju baza podataka.

Pored značaja koji zatvorenje skupa FZ ima u logičkom projektovanju baze podataka (npr. u utvrđivanju da li je neki skup atributa kandidat za primarni ključ), na ovom konceptu se neposredno zasniva i koncept pokrivanja.

Definicija 8.1 Neka je F neki skup FZ relacije R. *Zatvorenje od* F, u oznaci F^+, jeste najmanji skup FZ koji sadrži F i takav da se primenom Armstrongovih aksioma na F^+ ne dobija nijedna FZ koja već nije u F^+.

Skup F^+ je konačan i može da se odredi primenom aksioma F1, F2 i F6 i dodavanjem dobijenih FZ skupu F, ali je taj postupak jako dug. Zato se pitanje 1 zamenjuje pitanjem 2: kako konstruisati algoritam kojim se, bez određivanja celog skupa F^+, može utvrditi da li je proizvoljna FZ $X \to Y \in F^+$, tj. da li se ta FZ može *izvesti* iz F.

Definicija 8.2 Neka su X, Y neprazni skupovi atributa relacije R i neka je F neki skup FZ te relacije. Kaže se da je Y *zatvorenje od* X *nad* F (u oznaci $Y = X^+$) ako postoji FZ $X \to Y$ u F^+ i ako za svaku drugu FZ $X \to Z$ iz F^+ važi $Y \supseteq Z$ (Y je maksimalni skup atributa funkcionalno zavisnih od X).

Zbog refleksivnosti, zatvorenje X^+ uvek sadrži X.

Primer 8.3 Neka je $F = \{A \to D,\ AB \to DE,\ CE \to G,\ E \to H\}$. Tada je $(AB)^+ = ABDEH$.

Zatvorenje skupa atributa X nad skupom FZ F može se odrediti u nizu iteracija sledećim algoritmom:

Algoritam ZATVORENJE
Ulaz: skup atributa X i skup FZ F;
Izlaz: zatvorenje od X nad F;
ZATVORENJE(X, F)
BEGIN
    $stari := \emptyset$; $novi := X$;
    WHILE $novi \neq stari$ DO
        BEGIN
            $stari := novi$;
            FOR svaka FZ $W \to Z$ u F DO
                IF $novi \supseteq W$ THEN $novi := novi \cup Z$
        END;
    RETURN($novi$)
END.

Primer 8.4 Neka je $F = \{A \to D,\ AB \to E,\ BI \to E,\ CD \to I,\ E \to C\}$. Izvršenjem algoritma ZATVORENJE(AE, F) promenljiva *novi* dobija, redom, sledeće skupove atributa kao svoje vrednosti:
$novi = AE$
$novi = AEDC$
$novi = AEDCI = (AE)^+$.

Korišćenjem algoritma ZATVORENJE može se konstruisati i algoritam ČLAN kojim se ispituje da li zadata FZ pripada zatvorenju F^+ skupa FZ F.

Algoritam ČLAN
Ulaz: skup FZ F i FZ $X \to Y$;
Izlaz: TAČNO ako $F \models X \to Y$, NETAČNO u suprotnom;
ČLAN(F, $X \to Y$)
BEGIN
    IF $Y \subseteq$ ZATVORENJE (X, F) THEN RETURN(TAČNO)
    ELSE RETURN(NETAČNO)
END.

Primer 8.5 Za skup FZ F kao u primeru 8.4, i FZ $AE \to I$, algoritam ČLAN daće potvrdan odgovor o pripadnosti zadate FZ zatvorenju F^+ skupa F.

8.4 Pokrivanje skupa funkcionalnih zavisnosti

Složenost algoritama za kontrolu i održavanje integriteta baze podataka, zasnovanih na funkcionalnim zavisnostima, bitno zavisi od broja FZ čije važenje treba proveravati (i složenost algoritma ZATVORENJE, pa i algoritma ČLAN, zavisi od broja FZ u skupu F). Zbog toga se prirodno postavlja pitanje da li se dati skup FZ F može predstaviti drugim (manjim) skupom FZ F' koji "proizvodi" isti skup FZ kao i F (tj. za koji je $F^+ = F'^+$). Na primer, svaka FZ koja se može izvesti iz skupa $F = \{A \to B,\ B \to C,\ A \to C,\ AB \to C,\ A \to BC\}$, može se izvesti i iz skupa $F' = \{A \to B,\ B \to C\}$.

Definicija 8.3 Dva skupa FZ F i F' relacije R su *ekvivalentna* (u oznaci $F \equiv F'$) ako je $F^+ = F'^+$. Ako je $F \equiv F'$, onda je F *pokrivanje za* F' (F' je *pokrivanje za* F).

Mada definicija pokrivanja ne uključuje njegovu kardinalnost, podrazumevaće se pokrivanje specijalnog oblika koje ne sadrži veći broj FZ od skupa koji pokriva.

Iz definicije 8.3 sledi da za svaku FZ $X \to Y$ iz F'^+ (pa, dakle, i za svaku FZ $X \to Y \in F'$) važi $F \models X \to Y$. Kaže se da se skup FZ F' *izvodi* iz skupa F, tj. da je F' *logička posledica* od F, u oznaci $F \models F'$. Zbog simetričnosti ekvivalencije skupova funkcionalnih zavisnosti važi i $F' \models F$.

Primer 8.6 Skupovi FZ $F = \{A \to BC,\ A \to D,\ CD \to E\}$ i $G = \{A \to BCE,\ A \to ABD,\ CD \to E\}$ su ekvivalentni, ali skup F nije ekvivalentan sa skupom $G' = \{A \to BCDE\}$, jer $G' \not\models CD \to E$.

Provera izvodivosti jednog skupa FZ iz drugog, odnosno provera njihove ekvivalentnosti, može se sprovesti sledećim algoritmima:

Algoritam IZVODI
Ulaz: dva skupa FZ F, G;
Izlaz: TAČNO ako $F \models G$, NETAČNO u suprotnom;
IZVODI(F, G)
BEGIN
    v := TAČNO;
    FOR svaka FZ $X \to Y$ iz G DO
        $v := v$ AND ČLAN(F, $X \to Y$);
    RETURN(v)
END.

Algoritam EKVIV
Ulaz: dva skupa FZ F, G;
Izlaz: TAČNO ako $F \equiv G$, NETAČNO u suprotnom;
EKVIV(F, G)
BEGIN
    v := IZVODI(F, G) AND IZVODI(G, F);
    RETURN(v)
END.

Iz istog razloga kao i pokrivanja uopšte, od interesa su pokrivanja bez suvišnih FZ (tj. bez onih FZ koje se mogu izvesti iz preostalih FZ istog skupa).

Definicija 8.4 Skup FZ F je *neredundantan* ako ne sadrži pravi podskup $F' \subset F$ takav da je $F' \equiv F$; u suprotnom je skup F *redundantan*. Skup F je *neredundantno pokrivanje* skupa FZ G ako je F pokrivanje skupa G i F je neredundantan skup FZ.

Primer 8.7 Za dva skupa FZ
$G = \{AB \to C,\ A \to B,\ B \to C,\ A \to C\}$ i
$F = \{AB \to C, A \to B, B \to C\}$
važi $F \equiv G$, ali je F redundantan skup FZ jer je i skup $F' = \{A \to B,\ B \to C\}$ pokrivanje skupa G. Skup FZ F' jeste neredundantno pokrivanje skupa FZ G.

Sledeća definicija neredundantnog skupa ekvivalentna je definiciji 8.4, ali je operativnija od nje i koristi pojam suvišne funkcionalne zavisnosti:

Definicija 8.5 FZ $X \to Y$ je *suvišna* u skupu FZ F ako $F \setminus \{X \to Y\} \models X \to Y$. Skup FZ F je *neredundantan* ako u njemu nema suvišnih FZ.

Za svaki skup FZ G postoji neki podskup $F \subseteq G$ takav da je F neredundantno pokrivanje od G. Ako je G neredundantan skup FZ, onda je $F = G$. Ako je G redundantan skup, treba izbaciti iz njega sve suvišne FZ. Sledeći algoritam NEREDUND izgrađuje jedno neredundantno pokrivanje F datog skupa FZ G:

Algoritam NEREDUND
Ulaz: skup FZ G;
Izlaz: neredundantno pokrivanje F skupa G;
NEREDUND(G)
BEGIN
    $F := G$;
    FOR svaka FZ $X \to Y$ iz G DO
        IF ČLAN($F \setminus \{X \to Y\}$, $X \to Y$) THEN $F := F \setminus \{X \to Y\}$;
    RETURN (F)
END.

Primer 8.8 Neredundantno pokrivanje skupa FZ
$G = \{A \to B,\ B \to A,\ B \to C,\ A \to C\}$,
određeno algoritmom NEREDUND(G), jeste skup
$\{A \to B,\ B \to A,\ A \to C\}$.
Rezultat algoritma je osetljiv na redosled navođenja FZ u skupu G, pa je, tako, NEREDUND($\{A \to B,\ A \to C,\ B \to A,\ B \to C\}$) = $\{A \to B,\ B \to A,\ B \to C\}$. Dakle, skup FZ G može da ima više od jednog neredundantnog pokrivanja.

Od posebnog interesa je *minimalno pokrivanje* F skupa FZ G; ono se definiše kao pokrivanje skupa G koje ne sadrži više FZ od bilo kog drugog njemu ekvivalentnog skupa.

Primer 8.9 Skup FZ $G = \{A \to BC,\ B \to A,\ AD \to E,\ BD \to J\}$ je neredundantan, ali nije minimalan jer postoji skup FZ $F = \{A \to BC, B \to A, AD \to EJ\}$ takav da je $F \equiv G$ i $|F|^2 < |G|$. Skup F jeste minimalno pokrivanje skupa G.

Algoritam za određivanje minimalnog pokrivanja ima veliku vremensku složenost (eksponencijalnu), a sam problem je jedan od NP kompletnih problema ([48]). Algoritam zahteva definisanje nekih novih pojmova kao što su graf izvođenja FZ i relacije definisane nad takvim grafom.

Jedna vrsta neredundantnog pokrivanja koja se često koristi u procesu logičkog projektovanja baze podataka i dokazivanju korektnosti tog procesa, jeste kanoničko pokrivanje.

Definicija 8.6 *Kanoničko pokrivanje* skupa FZ G je neredundantno pokrivanje F koje zadovoljava sledeća dva uslova:
a) svaka FZ u skupu F je oblika $X \to A$, gde je A jedan atribut;
b) leva strana svake FZ iz F oslobođena je *nebitnih* atributa, tj. atributa čije udaljenje iz leve strane te FZ ne menja zatvorenje skupa FZ G.

[2] Oznaka $|F|$ označava kardinalnost skupa F.

Primer 8.10 Za skup FZ $G = \{A \to BCE,\ AB \to DE,\ BI \to J\}$ kanoničko pokrivanje je $F = \{A \to B,\ A \to C,\ A \to D,\ A \to E,\ BI \to J\}$.

U opštem slučaju, kanoničko pokrivanje datog skupa FZ G može se odrediti u tri koraka:

1. svaka FZ oblika $X \to Y$ zameni se skupom FZ $\{X \to A \,|\, A \in Y\}$;
2. za svaku FZ $X \to A$ iz dobijenog skupa FZ, iz leve strane X te FZ uklanja se svaki nebitni atribut B;
3. odredi se neredundantno pokrivanje F prethodno dobijenog skupa FZ.

Drugi pristup smanjenju složenosti algoritama koji proveravaju važenje FZ je onaj koji, umesto smanjenja broja FZ, ide ka smanjenju broja atributa u njima. Naime, ako je F neredundantan skup FZ, u njemu nema suvišnih FZ (ne može se smanjiti F udaljenjem neke FZ jer bi se dobio skup $F' \not\equiv F$.) "Veličina" skupa F može se smanjiti udaljenjem nebitnih atributa iz levih strana FZ, ali i iz desnih strana FZ (atribut A u desnoj strani FZ $X \to Y$ je nebitan ako važi da je $(F \setminus \{X \to Y\}) \cup \{X \to (Y \setminus \{A\})\} \models X \to A$, tj. ako se funkcionalna zavisnost $X \to A$ može izvesti iz skupa FZ koji se dobije kada se FZ $X \to Y$ zameni sa $X \to (Y \setminus \{A\})$).

Primer 8.11 U skupu FZ $G = \{A \to BC,\ B \to C,\ AB \to D\}$, atribut C je nebitan u FZ $A \to BC$, a atribut B je nebitan u FZ $AB \to D$. Ovi atributi mogu se ukloniti iz pripadnih FZ.

Ovaj pristup ima svoj cilj u definisanju *optimalnog pokrivanja* datog skupa FZ, koje se odnosi na minimizaciju broja atributa u pokrivanju tog skupa FZ ([48]). Dokazuje se da je optimalno pokrivanje istovremeno i minimalno pokrivanje bez nebitnih atributa.

8.5 Višeznačne zavisnosti

Funkcionalne zavisnosti su specijalni slučaj opštijih logičkih veza među atributima relacije, koje se zovu *višeznačne zavisnosti*. Opštost ovog tipa logičke veze ogleda se u sledećem: kod funkcionalne zavisnosti skupa atributa Y od skupa atributa X $(X \to Y)$, vrednost skupa atributa X određuje *jednu vrednost* skupa atributa Y, dok kod višeznačne zavisnosti Y od X, vrednost skupa atributa X određuje *skup vrednosti* skupa atributa Y.

Ilustrujmo koncept višeznačne zavisnosti sledećim primerom:

Neka relacija PNR ima atribute PREDMET, NASTAVNIK, REFERENCA, sa značenjem da NASTAVNIK predaje PREDMET i pri tome koristi REFERENCU kao literaturu. Ako među atributima ove relacije važi logički uslov da jednu istu referencu koriste svi nastavnici koji predaju isti predmet, onda skup referenci koje

se koriste u nastavi jednog predmeta zavisi samo od predmeta a ne i od nastavnika, tj. atribut REFERENCA višeznačno zavisi od atributa PREDMET. Neka relacija PNR ima sledeći sadržaj:

PNR

PREDMET	NASTAVNIK	REFERENCA
p1	n1	r1
p1	n1	r2
p1	n2	r1
p1	n2	r2
p2	n3	r3
p2	n3	r4
p2	n3	r5

Predmet p2 predaje samo jedan nastavnik pa razmatranje vrednosti njegovih referenci nije zanimljivo. Predmet p1 predaju nastavnici n1 i n2, i to oba nastavnika po istoj literaturi {r1, r2}, tj. skup referenci za predmet p1 koje koristi nastavnik n1 isti je kao i skup referenci za predmet p1 koje koristi nastavnik n2. Ova jednakost skupova referenci za jedan predmet, koje koriste razni nastavnici, označava postojanje višeznačne zavisnosti atributa REFERENCA od atributa PREDMET u relaciji PNR.

Višeznačna zavisnost može se formalno definisati na sledeći način:

Definicija 8.7 Neka su X, Y, Z neprazni skupovi atributa relacije R, pri čemu je $Z = Atr(R)\setminus XY$ i neka su x, z proizvoljni elementi projekcija relacije R na skupove atributa X, Z, redom. Uvedimo oznaku

$$Y_{xz} \stackrel{\text{def}}{=} \begin{cases} \{y \mid y \in R[Y] \text{ i } xyz \in R\} & \text{ako } X \cap Y = \emptyset, \\ \{y \mid y \in R[Y] \text{ i } x * yz \in R\} & \text{ako } X \cap Y \neq \emptyset. \end{cases}$$

($*$ u $x * y$ označava prirodno spajanje torki x i y po jednakim vrednostima zajedničkih atributa). U relaciji R važi *višeznačna zavisnost* (kraće: VZ) skupa Y od skupa X, u oznaci $X \rightarrow\!\!\!\rightarrow Y$, ako važi jednakost $Y_{xz} = Y_{xz'}$ za svako $x \in R[X]$ i $z, z' \in R[Z]$ za koje su $Y_{xz}, Y_{xz'}$ oba neprazna. Kaže se da skup atributa X *višeznačno određuje* skup atributa Y, tj. da skup atributa Y *višeznačno zavisi* od skupa atributa X.

Prethodna definicija ima i sledeće interpretacije. U relaciji R važi VZ $X \rightarrow\!\!\!\rightarrow Y$ ako pripadnost n-torki xyz, $xy'z' \in R$, za $x \in R[x]$, $y, y' \in R[Y]$, $z, z' \in R[Z]$ (tj. $x * yz$, $x * y'z' \in R$), povlači i pripadnost n-torki xyz', $xy'z \in R$ (tj. $x * yz'$, $x * y'z \in R$). Takođe, višeznačna zavisnost $X \rightarrow\!\!\!\rightarrow Y$ važi u R ako je skup Y_{xz} jednoznačno određen vrednošću x, tj. zavisi samo od x a ne i od z.

Dakle, Y_{xz} je skup svih elemenata projekcije relacije R na skup atributa Y, koji su u relaciji R sa x, z, tj. Y_{xz} je projekcija na skup atributa Y restrikcije relacije R po uslovu $X = x$ i $Z = z$. Koristeći ovu oznaku, za skup vrednosti višeznačno zavisnog atributa REFERENCA za predmet p1 dobijamo

$\text{REFERENCA}_{p1n1} = \{r|\ (\text{p1}, r, \text{n1}) \in \text{PNR}\} = \{\text{r1}, \text{r2}\}$,
$\text{REFERENCA}_{p1n2} = \{r|\ (\text{p1}, r, \text{n2}) \in \text{PNR}\} = \{\text{r1}, \text{r2}\}$.

U prethodnom primeru može se uočiti i sledeće: ako važi logički uslov da skup referenci koje jedan nastavnik koristi za jedan predmet zavisi samo od predmeta a ne i od nastavnika (važi višeznačna zavisnost PREDMET $\rightarrow\!\!\rightarrow$ REFERENCA), onda važi i logički uslov da skup nastavnika koji u nastavi jednog (određenog) predmeta koriste jednu (određenu) referencu zavisi takođe samo od predmeta a ne i od reference (važi višeznačna zavisnost PREDMET $\rightarrow\!\!\rightarrow$ NASTAVNIK). To nije slučajno, već predstavlja jednu od osobina višeznačnih zavisnosti – komplementarnost.

8.6 Osnovne osobine višeznačnih zavisnosti

Među osobinama višeznačnih zavisnosti ističu se sledeće osnovne osobine iz kojih se sve ostale mogu izvesti:

(a') *komplementarnost:* ako u relaciji R važi višeznačna zavisnost $X \rightarrow\!\!\rightarrow Y$, gde su X, Y neprazni podskupovi skupa $Atr(R)$, onda u relaciji R važi i višeznačna zavisnost $X \rightarrow\!\!\rightarrow Z$, gde je $Z = Atr(R) \setminus XY \neq \emptyset$;

(b') *proširenje:* ako u relaciji R važi višeznačna zavisnost $X \rightarrow\!\!\rightarrow Y$ (X, Y su neprazni podskupovi skupa $Atr(R)$), onda za svaka dva podskupa $V, W \subseteq Atr(R)$, takva da je $V \subseteq W$, važi višeznačna zavisnost $WX \rightarrow\!\!\rightarrow VY$;

(c') *tranzitivnost:* ako u relaciji R važe VZ $X \rightarrow\!\!\rightarrow Y$ i $Y \rightarrow\!\!\rightarrow Z$ ($X, Y, Z \subseteq Atr(R)$), onda važi i višeznačna zavisnost $X \rightarrow\!\!\rightarrow Z$ (ako je $Y \cap Z = \emptyset$), odnosno $X \rightarrow\!\!\rightarrow Z \setminus Y$ (ako je $Y \cap Z \neq \emptyset$).

Osim osnovnih osobina višeznačnih zavisnosti, od značaju su i sledeće dve osobine veze između funkcionalnih i višeznačnih zavisnosti:

(a") Ako su X, Y neprazni podskupovi skupa atributa relacije R i ako u relaciji R važi FZ $X \rightarrow Y$, onda u relaciji R važi i VZ $X \rightarrow\!\!\rightarrow Y$. Dakle, funkcionalne zavisnosti su specijalni slučaj višeznačnih zavisnosti. Definicija funkcionalne zavisnosti može se dobiti modifikovanjem definicije višeznačne zavisnosti dodatnim zahtevom da skup Y_{xz}, osim što zavisi samo od x, sadrži (najviše) jedan element. Da su ova dva logička uslova strogo inkluzivna pokazuje primer iz prethodnog odeljka u kome važi višeznačna ali ne važi funkcionalna zavisnost.

(b") Ako u relaciji R važi višeznačna zavisnost $X \rightarrow\!\!\rightarrow Y$ ($X, Y \subseteq Atr(R)$ i $X, Y \neq \emptyset$) i ako za dva skupa atributa $W, U \subseteq Atr(R)$, takva da je $U \subseteq Y$ i $W \cap Y = \emptyset$, važi funkcionalna zavisnost $W \rightarrow U$, onda u relaciji R važi i FZ $X \rightarrow U$.

Sve osobine funkcionalnih i višeznačnih zavisnosti (F1 – F6, (a') – (c')) i osobine veze funkcionalnih i višeznačnih zavisnosti (a"), (b") mogu se dokazati polazeći od

definicije funkcionalnih i višeznačnih zavisnosti; sve te osobine nazivaju se i aksiomama zavisnosti, a njihov značaj ogleda se u zajedničkom svojstvu potpunosti koje se formuliše sledećom teoremom.

Teorema 8.2 *Za dati skup funkcionalniih i višeznačnih zavisnosti D u relaciji R sa skupom atributa A, svojstva F1–F6, (a')–(c') i (a"), (b"), primenjena na zavisnosti iz D, omogućuju izvođenje skupa D^+ svih zavisnosti (funkcionalnih i višeznačnih) koje su logička posledica skupa D.*

Dokaz teoreme može se naći u [48], [65].

Značaj višeznačnih zavisnosti leži u činjenici da one daju potreban i dovoljan uslov da se relacija može dekomponovati u par svojih projekcija bez gubitka informacije, tj. da bude rezultat prirodnog spajanja svojih projekcija. Ovo svojstvo višeznačnih zavisnosti je izuzetno važno u metodologiji logičkog projektovanja baze podataka, i može se formulisati sledećim tvrđenjem:

Teorema 8.3 *Neka je R relacija i neka su X, Y, Z neprazni podskupovi skupa $Atr(R)$, pri čemu je $Z = Atr(R) \setminus XY$. Za relaciju R važi jednakost*

$$R = R[XY] * R[XZ] \tag{1}$$

ako i samo ako važi višeznačna zavisnost $X \rightarrow\rightarrow Y$.

Dokaz. Razlikujemo tri slučaja:

1. Neka je $Y \subseteq X$. Tada tvrđenje važi jer je projekcija $R[XZ]$ jednaka celoj relaciji R, a jednakost $Y_{xz} = Y_{xz'}$ je trivijalna.

2. Neka su skupovi X, Y disjunktni. Dokažimo prvo da je postojanje višeznačne zavisnosti $X \rightarrow\rightarrow Y$ dovoljan uslov za jednakost (1).

   Kao što je već naglašeno u dokazu teoreme 8.1, inkluzija $R \subseteq R[XY] * R[XZ]$ je trivijalna i uvek važi. Suprotna inkluzija sledi iz sledećeg razmatranja.

   Neka važi višeznačna zavisnost $X \rightarrow\rightarrow Y$. Tada za svaku n-torku $xyz \in R[XY] * R[XZ]$ postoje $y' \in R[Y]$ i $z' \in R[Z]$, takvi da $xy'z, xyz' \in R$. Zbog višeznačne zavisnosti $X \rightarrow\rightarrow Y$ važi da je $Y_{xz} = Y_{xz'}$, tj. $y' \in Y_{xz'}$ i $y \in Y_{xz}$, odakle sledi da $xyz, xy'z' \in R$. Dakle, ako je $xyz \in R[XY] * R[XZ]$, onda je i $xyz \in R$.

   Da bismo dokazali da je postojanje višeznačne zavisnosti potreban uslov za jednakost (1), pretpostavimo da važi jednakost (1) i da su xyz, $xy'z'$ dve proizvoljne n-torke iz $R\,(x \in R[X], y, y' \in R[Y], z, z' \in R[Z])$. Tada su, zbog jednakosti (1), i n-torke $xy'z, xyz' \in R$, što znači da važi implikacija $y \in Y_{xz}$, $y' \in Y_{xz'} \Rightarrow y \in Y_{xz'}$, $y' \in Y_{xz}$, tj. jednakost $Y_{xz} = Y_{xz'}$. Kako ova jednakost važi za proizvoljne x, z, z' za koje su oba skupa $Y_{xz}, Y_{xz'}$ neprazna, to u relaciji R važi, po definiciji, višeznačna zavisnost $X \rightarrow\rightarrow Y$.

3. Neka $Y \not\subseteq X$ i neka skupovi atributa X i Y imaju neprazan presek. Tada se skup Y može predstaviti kao $Y = Y'Y''$, gde je $Y' = X \cap Y$ i $Y'' \neq \emptyset$. Tada VZ $X \rightarrow\rightarrow Y$ važi ako i samo ako važi VZ $X \rightarrow\rightarrow Y''$ (u jednom smeru po definiciji VZ, u drugom po osobini proširenja VZ), pri čemu su skupovi X, Y'' disjunktni. Na osnovu tvrđenja teoreme u slučaju 2, druga VZ važi ako i samo ako je $R = R[XY''] * R[XZ]$ $(Z = Atr(R) \setminus XY'')$, tj. ako i samo ako je $R = R[XY] * R[XZ]$ (zbog $XY'' \equiv XY$).

Dakle, tvrđenje teoreme važi u sva tri slučaja.

Ako se definicija višeznačne zavisnosti modifikuje tako da skup atributa Z može biti prazan, dolazi se do definicije trivijalne višeznačne zavisnosti:

Definicija 8.8 *Trivijalne višeznačne zavisnosti* u relaciji $R(X, Y)$ sa disjunktnim, nepraznim podskupovima X, Y skupa $Atr(R)$ jesu zavisnosti $X \rightarrow\rightarrow \emptyset$, $X \rightarrow\rightarrow Y$.

Motivacija za ovakvu definiciju trivijalnih VZ je činjenica da je u zavisnosti $X \rightarrow\rightarrow \emptyset$ skup zavisnih atributa prazan pa je i skup njihovih vrednosti prazan, dok je VZ $X \rightarrow\rightarrow Y$ uvedena zbog osobine komplementarnosti, modifikovane za slučaj praznog skupa atributa. Ovako definisane trivijalne višeznačne zavisnosti važe u svakoj relaciji.

8.7 Zavisnosti spajanja

Opštiji oblik logičke veze među atributima relacije, u odnosu na višeznačne zavisnosti, predstavlja tzv. zavisnost spajanja ([65]). Definicija ovog oblika logičke veze zasniva se na uopštenju svojstva funkcionalnih odnosno višeznačnih zavisnosti da predstavljaju dovoljan odnosno potreban i dovoljan uslov za "binarnu" dekompoziciju relacije (dekompoziciju u dve projekcije) bez gubljenja informacije. Zavisnost spajanja se definiše upravo ovim svojstvom, ali proširenim na konačnu dekompoziciju relacije. Naime, postoje primeri relacija koje se ne mogu binarno dekomponovati bez gubljenja informacije, ali se mogu konačno dekomponovati (v. odeljak 9.6).

Definicija 8.9 Neka je $\{X_1, \ldots, X_m\}$ skup podskupova skupa atributa $Atr(R)$ relacije R takav da je $X_1 \cup X_2 \cup \cdots \cup X_m = Atr(R)$. U relaciji R važi *zavisnost spajanja*, u oznaci $*\{X_1, \ldots, X_m\}$ ako je

$$R = R[X_1] * R[X_2] * \cdots * R[X_m].$$

Zavisnost spajanja je *trivijalna* ako je $X_i = Atr(R)$ za neko $i \in \{1, 2, \ldots, m\}$.

Ovako definisana trivijalna zavisnost spajanja važi u svakoj relaciji.

Opštost zavisnosti spajanja u odnosu na prethodno definisane zavisnosti ogleda se u sledećoj osobini veze između višeznačnih zavisnosti i zavisnosti spajanja:

(a"') U relaciji R sa tri neprazna podskupa skupa atributa X, Y, Z ($Z = Atr(R) \setminus XY$) važi zavisnost spajanja $*\{XY, XZ\}$ ako i samo ako u relaciji R važi višeznačna zavisnost $X \rightarrow\rightarrow Y$.

8.8 Pitanja i zadaci

1. Definisati sledeće pojmove:

funkcionalna zavisnost (FZ)	kanoničko pokrivanje skupa FZ
trivijalna FZ	nebitan atribut
aksiome izvođenja	suvišna FZ
zatvorenje skupa atributa	višeznačna zavisnost (VZ)
zatvorenje skupa FZ	trivijalna VZ
pokrivanje skupa FZ	zavisnost spajanja (ZS)
neredundantno pokrivanje skupa FZ	trivijalna ZS

2. Formulisati i dokazati vezu FZ i dekompozicije relacije u skup projekcija.
3. U čemu je značaj aksioma izvođenja?
4. Dokazati da refleksivnost, proširenje i pseudotranzitivnost čine potpun sistem aksioma.
5. Objasniti algoritme ZADOVOLJAVA, ZATVORENJE, ČLAN, NEREDUND, IZVODI, EKVIV, kao i njihov značaj u teoriji zavisnosti.
6. Konstruisati algoritam REDUND koji ispituje da li je dati skup FZ redundantan. Preformulisati algoritam NEREDUND tako da, umesto algoritma ČLAN, koristi algoritam REDUND.
7. Navesti osnovne osobine višeznačnih zavisnosti.
8. Formulisati i dokazati vezu VZ i dekompozicije relacije u skup projekcija.
9. Neka je relacija R stepena n. Odrediti najveći broj funkcionalnih zavisnosti koje relacija R može da zadovoljava.
10. Odrediti zatvorenje skupa atributa $\{A, B\}$ nad skupom FZ $F = \{AB \rightarrow C,\ B \rightarrow D,\ CD \rightarrow E,\ CE \rightarrow GH,\ G \rightarrow A\}$.
11. Odrediti zatvorenje F^+ skupa FZ $F = \{AB \rightarrow C,\ C \rightarrow B\}$.
12. Odrediti neredundantno pokrivanje sledećeg skupa FZ relacije R(A,B,C,D,E,I):

$A \rightarrow C$	$AB \rightarrow C$
$C \rightarrow DI$	$CD \rightarrow I$
$EC \rightarrow AB$	$EI \rightarrow C$

13. Neka relacija BIBLIOTEKA ima sledeće atribute:

BIBLIOTEKA_BR	IZDANJE
MESTO	BR_PRIMERAKA
NAZIV	KAT_BROJ
K_SIF	ZAUZETOST
I_SIF	DATUM_IZD

Torka (*bb, mesto, naziv, k_sif, i_sif, izd, br_pr, kat_br, zauz, dat*) je element relacije BIBLIOTEKA ako biblioteka sa matičnim brojem *bb* i nazivom *naziv* u mestu *mesto* poseduje primerak pod kataloškim brojem *kat_br* knjige sa šifrom *k_sif*, u izdanju *izd* izdavača sa šifrom *i_sif*; primerak je jedan od *br_pr* primeraka te knjige u toj biblioteci, a može biti zauzet (*zauz* je 1) sa datumom izdavanja *dat*, ili slobodan (*zauz* je 0 a *dat* je NULL). Koje funkcionalne zavisnosti zadovoljava relacija biblioteka?

14. Neka relacija UCENIK ima sledeće atribute:

UCENIK_BR	RAZRED
IME	ODELJENJE
DATUM_ROĐENJA	RAZREDNI_STARESINA
SKOLA	PREDMET
SMER	OCENA

Atributi imaju uobičajeno značenje. Odrediti skup zavisnosti koje zadovoljava relacija UCENIK.

15. Neka relacija RASPORED_CASOVA ima sledeće atribute:

D – dan u nedelji (1–5)	U – broj učionice
SP – sat početka predavanja (8–19)	N – ime nastavnika
SZ – sat završetka predavanja (9–20)	P – naziv predmeta

Torka (D:d, SP:sp, SZ:sz, U:u, N:n, P:p) pripada relaciji RASPORED_CASOVA ako u vreme (D:d, SP:sp–SZ:sz) nastavnik N:n drži predavanje iz predmeta P:p u učionici U:u. Koje funkcionalne (i druge) zavisnosti važe u relaciji RASPORED_CASOVA?

9

Normalne forme

Logičko projektovanje može, kao jednim kriterijumom, da bude vođeno pravilima o grupisanju atributa u relacije na osnovu logičkih veza među atributima. Za relacije koje se dobiju primenom pojedinih takvih pravila kaže se da se nalaze u odgovarajućim *normalnim formama*. Zbog toga su normalne forme čisto sintaksna svojstva zasnovana na algebri zavisnosti (funkcionalnih, višeznačnih i zavisnosti spajanja).

Za sve relacije relacionog modela pretpostavlja se da imaju jednostavne domene atributa. Za takve relacije kaže se da su u *prvoj normalnoj formi*, u oznaci 1NF.

Više normalne forme zahtevaju da je relacija u 1NF i da zadovoljava dodatne uslove vezane za funkcionalne, višeznačne odnosno zavisnosti spajanja među atributima relacije. Ovi uslovi se nameću da bi se prevazišle teškoće u radu sa relacijama, poznate kao *anomalije* unošenja, brisanja i ažuriranja.

9.1 Druga normalna forma

Pre nego što damo definiciju druge normalne forme i algoritam dekompozicije relacije u skup relacija koje su u drugoj normalnoj formi, navešćemo primer relacije koja ispoljava anomalije unošenja, brisanja i ažuriranja, i objasniti uzrok anomalija kao i način njihovog otklanjanja.

Primer 9.1 Neka je data relacija R sa sledećim sadržajem:

R

I_SIF	K_SIF	NAZIV
i1	k1	Prosveta
i3	k2	Dečje novine
i3	k3	Dečje novine
i4	k1	Matica srpska
i5	k1	Prosveta
i5	k2	Prosveta

Značenje atributa relacije R je isto kao u odeljku 1.2. Neka u relaciji R važi funkcionalna zavisnost I_SIF $\rightarrow$ NAZIV, i neka je to jedina funkcionalna zavisnost u toj relaciji. Dakle, NAZIV $\not\rightarrow$ I_SIF, I_SIF $\not\rightarrow$ K_SIF , K_SIF $\not\rightarrow$ I_SIF. U relaciji R jedini ključ, pa i primarni ključ jeste par atributa (I_SIF, K_SIF).

Anomalija unošenja u relaciju R ogleda se u nemogućnosti unošenja podataka o šifri i nazivu izdavača dok se ne unese i podatak o bar jednoj knjizi koju taj izdavač izdaje (atribut K_SIF ne može imati nedostajuću vrednost s obzirom da je deo primarnog ključa).

Anomalija brisanja ogleda se u činjenici da se brisanjem informacije o izdanju neke knjige može izbrisati, bez takve namere, i informacija o nazivu odgovarajućeg izdavača. Tako, isključenje knjige k1 iz izdanja izdavača i4 može se izvršiti samo brisanjem trojke (i4, k1, Matica srpska), čime se briše i informacija o nazivu izdavača sa šifrom i4.

Anomalija ažuriranja ogleda se u nemogućnosti nezavisnog ažuriranja vrednosti atributa NAZIV (izdavača). Na primer, ako je potrebno promeniti naziv izdavača sa šifrom i5, sa "Prosveta" na "Nova Prosveta", onda se taj podatak ne može promeniti ("ažurirati") u jednoj trojci relacije, već se to mora učiniti u svakoj trojci koja se odnosi na izdavača sa šifrom i5, tj. onoliko puta koliko izdanja ima taj izdavač.

Uzrok anomalija koje ispoljava relacija R je u činjenici da su tom relacijom predstavljeni jedan tip entiteta IZDAVAČ i jedan odnos između tipa entiteta IZDAVAČ i tipa entiteta KNJIGA. Entitet IZDAVAČ može se predstaviti relacijom IZDAVAC sa atributima I_SIF, NAZIV, itd, a odnos – relacijom IZDANJA sa atributima I_SIF, K_SIF, itd. Ovaj semantički uzrok anomalija sintaksno se može prepoznati u funkcionalnoj zavisnosti atributa NAZIV od atributa I_SIF koji je pravi podskup primarnog ključa. Ovakva funkcionalna zavisnost zove se parcijalna, njeno postojanje čini da relacija R nije u drugoj normalnoj formi, a anomalije koje prouzrokuje eliminišu se pravilom po kome se relacija R zamenjuje svojim projekcijama na parove atributa (I_SIF, NAZIV), (I_SIF, K_SIF):

R [I_SIF, NAZIV] = IZDAVAC

I_SIF	NAZIV
i1	Prosveta
i3	Dečje novine
i4	Matica srpska
i5	Prosveta

R [I_SIF, K_SIF] = IZDANJA

I_SIF	K_SIF
i1	k1
i3	k2
i3	k3
i4	k1
i5	k1
i5	k2

Dobijene relacije nemaju pomenute anomalije, a njihovim prirodnim spajanjem po atributu I_SIF dobija se prvobitna relacija R. Dakle, dekompozicija relacije R u dve projekcije IZDAVAC i IZDANJA je korektna, bez gubitka informacije (s obzirom na FZ I_SIF $\rightarrow$ NAZIV, v. teoremu 8.1), što je prvi i obavezni uslov pri bilo kojoj transformaciji podataka.

Definicija 9.1 Za funkcionalnu zavisnost $X \rightarrow Y$ kaže se da je *potpuna* ako za svaki pravi podskup X' od X važi $X' \not\rightarrow Y$. Ako za neki skup $X' \subset X$ važi $X' \rightarrow Y$, funkcionalna zavisnost $X \rightarrow Y$ zove se *parcijalna*.

Definicija 9.2 Atribut A relacije R je *sporedan* u toj relaciji ako ne učestvuje ni u jednom ključu te relacije; u suprotnom je atribut A *ključni* atribut. Neka je X skup svih sporednih atributa relacije R. Relacija R je u *drugoj normalnoj formi*, u oznaci 2NF, ako svaki atribut iz X potpuno funkcionalno zavisi od svakog ključa relacije R.

Teorema 9.1 (2NF normalizacija) *Ako relacija $R(A_1, \ldots, A_n)$ nije u* 2NF*, onda postoji dekompozicija relacije R u skup njenih projekcija koje su u* 2NF*, a iz kojih se operacijom prirodnog spajanja može ponovo dobiti relacija R.*

Dokaz: Dokaz teoreme se izvodi konstrukcijom sledećeg algoritma dekompozicije.

Algoritam 2NF normalizacije

Ulaz: relacija R sa atributima $\{A_1, \ldots, A_n\}$ i skupom FZ F;

Izlaz: skup $R_1, \ldots, R_i$ $(i \geq 1)$ projekcija relacije R, koje su sve u 2NF i za koje važi: ako je $i = 1$, onda je $R_1 \equiv R$; ako je $i > 1$, onda je $R = R_1 * R_2 * \cdots * R_i$;

```
BEGIN
  i := 1;
  WHILE relacija R nije u 2NF DO
    BEGIN
        uočiti parcijalnu FZ X → A, (X ⊆ Atr(R), A ∈ Atr(R))
            sporednog atributa A od ključa X;
        neka je X = X'X'', gde je X' → A potpuna FZ;
        neka je Z = Atr(R) \ (X ∪ {A});
        zameniti relaciju R njenim projekcijama R[XZ], R[X'A];
        R_i := R[X'A];
        i := i + 1;
        R := R[XZ]
    END;
  R_i := R
END.
```

Spajanjem projekcija $R[X'A], R[XZ]$ kojima je zamenjena relacija R može se opet dobiti relacija R (zbog FZ $X' \rightarrow A$, prema teoremi 8.1). Relacija $R[X'A]$ je u

2NF (lako se dokazuje), a ako druga projekcija $R[XZ]$ nije u 2NF, ona se posmatra kao nova relacija R i sa njom se ponavlja isti postupak dekompozicije po parcijalnoj funkcionalnoj zavisnosti koju sadrži. Svaka projekcija dobijena dekompozicijom u nekom prolazu pridružuje se skupu relacija koje su u 2NF a čijim se spajanjem može dobiti polazna relacija. Postupak je konačan jer se broj atributa relacija – projekcija smanjuje pa se, ako ne pre, postupak završava kada se dođe do binarne relacije (relacije sa dva atributa) koja je uvek u 2NF. Naime, ako su oba atributa binarne relacije u ključu, onda ona nema sporednih atributa, a ako je samo jedan atribut u ključu, onda nema parcijalne zavisnosti od ključa; u svakom od ovih slučajeva relacija je u 2NF.

Dakle, algoritam 2NF normalizacije je korektan, tj. tvrđenje teoreme je dokazano.

Prethodni algoritam se može učiniti efikasnijim tako što se relacija R dekomponuje u projekcije $R[X'Y]$, $R[Atr(R) \setminus Y]$ (umesto $R[X'A]$, $R[XZ]$), gde je sada Y skup svih sporednih atributa koji su parcijalno zavisni od ključa X i potpuno zavisni od skupa X'.

9.2 Treća normalna forma

Kriterijum koji definiše drugu normalnu formu nije dovoljno strog, pa relacija koja je u 2NF i dalje može da ispoljava anomalije unošenja, brisanja i ažuriranja. Tada su anomalije posledica drugih nepoželjnih oblika funkcionalnih zavisnosti koje treba otkloniti dekompozicijom relacije bez gubljenja informacije, u skup projekcija koje su u tzv. trećoj normalnoj formi (v. definiciju 9.3).

Primer 9.2 Neka je relacijom S(P_SIF, I_SIF, DRZAVA) predstavljen tip entiteta IZDAVAČ (sa atributima I_SIF i DRZAVA) i odnos tog tipa entiteta sa tipom entiteta PISAC koji je predstavljen jedinstvenim identifikatorom P_SIF. Neka je u relaciji S primarni ključ atribut P_SIF, i neka u njoj važe FZ P_SIF $\rightarrow$ I_SIF i I_SIF $\rightarrow$ DRZAVA, dok I_SIF $\not\rightarrow$ P_SIF i DRZAVA $\not\rightarrow$ I_SIF (jedan pisac može imati samo jednog izdavača, dok jedan izdavač može izdavati dela većeg broja pisaca). Iz ovih funkcionalnih zavisnosti proističu sve ranije pomenute anomalije. Na primer, ne može se uneti informacija o državi u kojoj je registrovan novi izdavač dok se ne unese šifra bar jednog pisca čija dela taj izdavač izdaje; brisanjem poslednjeg pisca čija dela izdaje neki izdavač gubi se i informacija o državi u kojoj je izdavač registrovan; promena države datog izdavača mora se izvršiti u svakoj trojci koja se odnosi na nekog pisca tog izdavača. Sintaksno gledano, anomalije su posledica tzv. tranzitivne zavisnosti sporednog atributa DRZAVA od ključnog atributa P_SIF, a mogu se otkloniti zamenom relacije S njenim projekcijama S[P_SIF, I_SIF], S[I_SIF, DRZAVA].

Za novodobijene relacije važi da se njihovim spajanjem može dobiti relacija S (zbog FZ I_SIF $\rightarrow$ DRZAVA), pri čemu u njima nema tranzitivne zavisnosti pa su u tzv. trećoj normalnoj formi.

Definicija 9.3 Neka su X, Y, Z skupovi atributa relacije $R(A_1, \ldots, A_n)$. Kaže se da skup atributa Z *tranzitivno zavisi od* X ako važi $X \to Y, Y \not\to X, Y \to Z$. Relacija R je u *trećoj normalnoj formi* (u oznaci 3NF) ako je u 1NF i nijedan njen sporedni atribut ne zavisi tranzitivno ni od jednog njenog ključa.

Lako se pokazuje da, ako je relacija R u 3NF, onda je ona i u 2NF. Naime, ako relacija R nije u 2NF, onda postoji sporedni atribut A_i koji je parcijalno zavisan od ključa X, ali je onda sporedni atribut A_i i tranzitivno zavisan od ključa X jer postoji takav skup $X' \subset X$ da je $X \to X', X' \not\to X, X' \to A_i$. Dakle, relacija R tada nije ni u 3NF.

Teorema 9.2 (3NF normalizacija) *Ako relacija $R(A_1, \ldots, A_n)$ nije u* 3NF*, onda postoji dekompozicija relacije R u skup njenih projekcija koje jesu u* 3NF*, koje čuvaju sve polazne FZ relacije R, a iz kojih se operacijom prirodnog spajanja može ponovo dobiti relacija R.*

Dokaz: Slično dokazu teoreme o 2NF normalizaciji, dokaz teoreme se izvodi konstrukcijom sledećeg algoritma dekompozicije.

Algoritam 3NF normalizacije

Ulaz: relacija R sa atributima $\{A_1, \ldots, A_n\}$ i skupom FZ F;

Izlaz: skup $R_1, \ldots, R_m$ $(m \geq 1)$ projekcija relacije R, koje su sve u 3NF i za koje važi: ako je $m = 1$, onda je $R_1 \equiv R$; ako je $m > 1$, onda je $R = R_1 * R_2 * \cdots * R_m$;

```
BEGIN
  i := 1;
  WHILE relacija R nije u 3NF, DO
    BEGIN
        uočiti tranzitivnu zavisnost X → Y, Y → A, Y ↛ X sporednog atributa
          A od ključa X, za koju je Y → A potpuna funkcionalna zavisnost;
        neka je Z = Atr(R) \ (X ∪ {A});
        zameniti relaciju R projekcijama R[YA], R[XZ];
        R_i := R[YA];
        i  := i + 1;
        R := R[XZ]
    END;
  R_i := R
END.
```

Algoritam 3NF normalizacije zasniva se na dekompoziciji. Spajanjem projekcija $R[YA]$, $R[XZ]$ može se dobiti relacija R (zbog FZ $Y \to A$). Za projekciju $R[YA]$ lako se dokazuje da je u 3NF. Ako je i projekcija $R[XZ]$ u 3NF, postupak se završava, a ako nije, isti postupak 3NF normalizacije primenjuje se na tu projekciju kao novu relaciju R.

Kao i kod 2NF normalizacije, postupak je konačan jer se broj atributa relacija – projekcija smanjuje pa se, ako ne pre, postupak završava kada se dođe do binarne relacije koja je uvek u 3NF (nad dva atributa ne može postojati tranzitivna zavisnost sporednog atributa od ključa).

Dakle, algoritam 3NF normalizacije je korektan, tj. tvrđenje teoreme je dokazano.

I ovaj se algoritam može modifikovati tako da postane efikasniji, ako se relacija R dekomponuje u projekcije $R[YW]$, $R[Atr(R) \setminus W]$ (umesto $R[YA]$, $R[XZ]$), gde je W skup svih sporednih atributa koji tranzitivno zavise od ključa X i potpuno zavise od skupa Y ([65]).

Prethodni algoritmi normalizacije polaze od pretpostavke da je polazna relacija neka vrsta "univerzalne relacije" ([65]). To je imaginarna relacija koja obuhvata sva svojstva (atribute) informacionog okruženja koje se modelira, i sve podatke baze podataka. Postojanje univerzalne relacije znatno bi pojednostavilo upitni jezik, jer bi se upiti odnosili samo na atribute, a ne i na relacije kojima atributi pripadaju. Sve prirodnojezičke sumeđe sa bazom podataka se *de facto* odnose na univerzalnu relaciju, a upite postavljene na prirodnom jeziku sistem onda prevodi u upite nad postojećim relacijskim shemama.

Osim za potrebe komunikacije, koncept univerzalne relacije koristi se i u kontekstu logičkog projektovanja. Univerzalna relacija ima pridružene logičke veze koje impliciraju ključeve i sporedne atribute. Iz nje treba, na osnovu uočenih logičkih veza među atributima, izdvojiti "dobre" relacije bez anomalija. Prethodno navedeni algoritmi 2NF i 3NF normalizacije su konceptualno najjednostavniji, ali imaju niz loših svojstava. Jedno od tih svojstava je da rezultuju nepotrebno velikim brojem relacija, tj. nepotrebno usitnjenim relacijama čije održavanje (prostorno i vremenski) može biti skupo, a manipulisanje njima komplikovano.

Algoritam kojim se formalizuje 3NF normalizacija i prevazilaze nedostaci 3NF dekompozicije polazi samo od zadatog skupa funkcionalnih zavisnosti među atributima (ne uključuje pojam univerzalne relacije i predefinisanih ključeva); to je algoritam sinteze relacija koje su u trećoj normalnoj formi. Dokazuje se da je broj sintetizovanih relacija dobijenih ovim algoritmom minimalan ([7]). Algoritam se zasniva na algebri funkcionalnih zavisnosti, Armstrongovim aksiomama, pojmu zatvorenja i pokrivanja skupa funkcionalnih zavisnosti i pretpostavci o jedinstvenosti funkcionalne zavisnosti (v. glavu 8).

Algoritam sinteze 3NF relacija iz zadatog skupa funkcionalnih zavisnosti
Ulaz: skup F funkcionalnih zavisnosti;
Izlaz: minimalni skup relacija $\{R_1, \ldots, R_m\}$ koje su u 3NF i koje zadovoljavaju sve funkcionalne zavisnosti iz F;
koraci:

1. Eliminacija nebitnih atributa: eliminisati nebitne atribute iz levih strana FZ iz F; neka je G tako dobijeni skup FZ.

2. Nalaženje pokrivanja: naći neredundantno pokrivanje H skupa G.

3. Particioniranje: particionirati skup H u particije $H_1, \ldots, H_k$ tako da sve funkcionalne zavisnosti u jednoj particiji imaju identične leve strane.

4. Spajanje ekvivalentnih ključeva: za svaki par particija H_i, H_j sa levim stranama X, Y, redom, takvim da postoje obe FZ $X \to Y$, $Y \to X$ u H^+, spojiti particije H_i i H_j i dodati FZ $X \to Y$, $Y \to X$ skupu "dvostranih" FZ J. Iz H izbrisati svaku FZ oblika $X \to A$ za $A \in Y$, i svaku FZ oblika $Y \to B$ za $B \in X$.

5. Eliminacija tranzitivnih zavisnosti: naći skup FZ $H' \subseteq H$ takav da je $(H' \cup J)^+ = (H \cup J)^+$, i da nijedan pravi podskup od H' nema to svojstvo. Dodati svaku FZ iz J odgovarajućoj particiji iz H'.

6. Konstruisanje relacija: za svaku particiju FZ iz H' konstruisati relaciju koja sadrži sve atribute te particije. Leva strana svake funkcionalne zavisnosti je jedan ključ relacije, a svaki takav ključ zove se *sintetizovani ključ*.

Može se dokazati da sve relacije dobijene prethodnim algoritmom jesu u 3NF i da skup dobijenih relacija zadovoljava sve polazne funkcionalne zavisnosti skupa F. Takoďe se može dokazati da je broj dobijenih relacija isti bez obzira od kog se neredundantnog pokrivanja (kojih može biti više) poďe, tj. da je broj dobijenih relacija minimalan. Neredundantno pokrivanje može se odrediti primenom algoritma NEREDUND, a tranzitivne zavisnosti mogu se eliminisati primenom algoritma ČLAN (glava 8). Složenost algoritma je kvadratna u odnosu na dužinu niske kojom se zadaju funkcionalne zavisnosti polaznog skupa F. Dokaz ovih tvrďenja može se naći u [7].

Primer 9.3 Neka je dat skup FZ $F = \{A \to B, A \to C, B \to C, B \to D, D \to B, ABE \to F\}$. Primena algoritma sinteze 3NF relacija odvija se u sledećim koracima:

– (eliminacija suvišnih atributa):
$$G = \{A \to B,\ A \to C,\ B \to C,\ B \to D,\ D \to B,\ AE \to F\};$$

– (nalaženje pokrivanja):
$$H = \{A \to B,\ B \to C,\ B \to D,\ D \to B,\ AE \to F\};$$

– (particioniranje): $H_1 = \{A \to B\}$
$H_2 = \{B \to C,\ B \to D\}$
$H_3 = \{D \to B\}$
$H_4 = \{AE \to F\};$

– (spajanje ekvivalentnih ključeva):
$H_1 = \{A \to B\}$
$H_2 = \{B \to C\}$
$H_3 = \{AE \to F\}$
$J = \{B \to D,\ D \to B\}$

– (eliminacija tranzitivnih zavisnosti): $H' = H$
$H'_1 = \{A \to B\}$
$H'_2 = \{B \to C,\ B \to D,\ D \to B\}$
$H'_3 = \{AE \to F\}$

– (konstruisanje relacija): $R_1(A, B)$ (ključ A),
$R_2(B, C, D)$ (ključ B; ključ D),
$R_3(A, E, F)$ (ključ $\{A, E\}$).

9.3 Boyce-Codd normalna forma

Anomalije unošenja, brisanja i ažuriranja javljaju se i kod nekih relacija koje su u 3NF, kada mogu biti posledica tranzitivne zavisnosti ključnih atributa od ključa. Sledeći primer ilustruje takav slučaj.

Primer 9.4 Neka u relaciji R(I_SIF, NAZIV, K_SIF, TIRAZ) (značenje atributa je isto kao u odeljku 1.2) važe funkcionalne zavisnosti {I_SIF, K_SIF} → TIRAZ; I_SIF → NAZIV; NAZIV → I_SIF; {NAZIV, K_SIF} → TIRAZ. Jedan ključ relacije R je par atributa (I_SIF, K_SIF), a drugi ključ (kandidat za ključ) je par atributa (NAZIV, K_SIF). Relacija R je u 3NF, ali ispoljava anomalije. Na primer, promena naziva izdavača sa datom šifrom mora se izvršiti u onoliko n-torki koliko knjiga taj izdavač izdaje. Anomalije su posledica tranzitivne zavisnosti ključnog atributa NAZIV od ključa (I_SIF, K_SIF) (tj. funkcionalne zavisnosti I_SIF → NAZIV na čijoj levoj strani nije ceo ključ). Zamenom relacije R njenim projekcijama R[I_SIF, NAZIV], R[I_SIF, K_SIF, TIRAZ] uklanjanju se anomalije, a njihovim spajanjem dobija se relacija R. Dobijene projekcije su u tzv. Boyce-Coddovoj normalnoj formi.

Definicija 9.4 Relacija $R(A_1, \ldots, A_n)$ je u *Boyce-Codd-ovoj normalnoj formi* (u oznaci BCNF) ako egzistencija FZ $X \to Y$, gde je $X, Y \subseteq \{A_1, \ldots A_n\}$ i $X \cap Y = \emptyset$, povlači egzistenciju FZ $X \to A_i$, za svako $i = 1, 2, \ldots, n$.

Sintaksna interpretacija definicije BCNF je sledeća: sve FZ u relaciji koja je u BCNF jesu posledice ključa. Semantička interpretacija je da nijedan entitet nije sadržan, kao pravi podskup, u datoj relaciji; takav entitet bi se odrazio postojanjem FZ $X \to Y$, gde X nije ključ relacije. Ako relacija sadrži takav entitet, njega treba izdvojiti u postupku projektovanja, što se postiže dekompozicijom relacije u niz projekcija koje su u BCNF.

Teorema 9.3 (BCNF normalizacija) *Ako relacija $R(A_1, \ldots, A_n)$ nije u* BCNF*, onda postoji dekompozicija relacije R u skup njenih projekcija koje su u BCNF, a iz kojih se operacijom prirodnog spajanja može ponovo dobiti relacija R.*

Dokaz: Slično dokazu teoreme o 3NF normalizaciji, dokaz teoreme izvodi se konstrukcijom algoritma dekompozicije.

Algoritam BCNF normalizacije
Ulaz: relacija R sa atributima $\{A_1, \ldots, A_n\}$ i skupom FZ F;
Izlaz: skup $R_1, \ldots, R_m$ $(m \geq 1)$ projekcija relacije R, koje su sve u BCNF i za koje važi: ako je $m = 1$, onda je $R_1 \equiv R$; ako je $m > 1$, onda je $R = R_1 * R_2 * \cdots * R_m$;

```
BEGIN
  IF relacija R nije u BCNF, THEN
    BEGIN
        uočiti funkcionalnu zavisnost X → Y, u kojoj leva strana X nije
            ključ relacije R i X ∩ Y = ∅ ;
        neka je Z = Atr(R) \ XY;
        zameniti relaciju R njenim projekcijama R[XY], R[XZ];
        ponoviti rekurzivno algoritam za relaciju R[XY];
        ponoviti rekurzivno algoritam za relaciju R[XZ]
    END
  ELSE
    izdati relaciju R na izlazu
END.
```

Razmatranje analogno onom u dokazu teoreme 9.2 dokazuje korektnost algoritma, čime je tvrđenje teoreme dokazano.

Sve relacije dobijene algoritmom BCNF normalizacije jesu u BCNF, njihovim spajanjem se dobija polazna relacija R, ali ovaj algoritam ne garantuje očuvanje svih funkcionalnih zavisnosti polazne relacije.

Primer 9.5 U relaciji SPN(STUDENT, PREDMET, NASTAVNIK) važe FZ

$$\begin{aligned} \{\text{STUDENT,PREDMET }\} &\rightarrow \text{NASTAVNIK,} \\ \text{NASTAVNIK} &\rightarrow \text{PREDMET, ali} \\ \text{PREDMET} &\not\rightarrow \text{NASTAVNIK,} \end{aligned}$$

sa sledećim značenjem: jedan student sluša jedan predmet kod jednog nastavnika, jedan nastavnik predaje jedan predmet, jedan predmet može da predaje veći broj nastavnika. Primarni ključ relacije je par atributa (STUDENT, PREDMET), i relacija je u 3NF, ali nije u BCNF. Njenom dekompozicijom prema BCNF algoritmu dobile bi se projekcije

SPN[STUDENT, NASTAVNIK],
SPN[NASTAVNIK, PREDMET],

koje su u BCNF. Jedina netrivijalna funkcionalna zavisnost u ovim projekcijama je FZ NASTAVNIK → PREDMET, a iz nje se ne može izvesti funkcionalna zavisnost {STUDENT, PREDMET} → NASTAVNIK koja važi u polaznoj relaciji. Zato se dobijene projekcije ne mogu nezavisno ažurirati. Ovo je primer nepoželjne dekompozicije, jer se BCNF normalizacijom ne rešavaju problemi zbog kojih se ona i sprovodi. Neopravdanost BCNF dekompozicije ima objašnjenje i u semantičkoj činjenici da relacija SPN, iako ne sasvim normalizovana, ne sadrži u sebi, kao pravi podskup, nijedan entitet niti odnos između entiteta.

Odnos 2NF, 3NF i BCNF može se izraziti sledećim tvrđenjem koje se jednostavno dokazuje.

Teorema 9.4 *Neka je* 2NFRel *oznaka za skup relacija koje su u* 2NF*, slično* 3NFRel*,* BCNFRel*. Tada važe inkluzije* BCNFRel ⊂ 3NFRel ⊂ 2NFRel*. Ove inkluzije su stroge* (kao što pokazuju uvodni primeri za definicije 3NF i BCNF).

9.4 Dobre i loše dekompozicije

Relacija se često na više načina može zameniti skupom relacija (njenih projekcija) koje su u 3NF ili BCNF. Neke od tih dekompozicija su dobre, neke su nekorisne, a neke su pogrešne i stoga neprimenljive. Na primer, ako u relaciji IZDAVAC(I_SIF, STATUS, DRZAVA) važe FZ I_SIF → DRZAVA, DRZAVA → STATUS, onda u toj relaciji važi i FZ I_SIF → STATUS (prema aksiomi tranzitivnosti). Relacija IZDAVAC nije u 3NF upravo zbog tranzitivne zavisnosti atributa STATUS od ključa I_SIF, a njena dekompozicija u skup relacija koje jesu u 3NF može se izvršiti na jedan od sledeća tri načina:

A:	I1(I_SIF, DRZAVA)	=	IZDAVAC[I_SIF, DRZAVA]
	I2(DRZAVA, STATUS)	=	IZDAVAC[DRZAVA, STATUS]
B:	I1(I_SIF, DRZAVA)	=	IZDAVAC[I_SIF, DRZAVA]
	I3(I_SIF, STATUS)	=	IZDAVAC[I_SIF, STATUS]
C:	I2(DRZAVA, STATUS)	=	IZDAVAC[DRZAVA, STATUS]
	I3(I_SIF, STATUS)	=	IZDAVAC[I_SIF, STATUS].

Sve tri dekompozicije dovode do binarnih relacija koje su u BCNF. Dekompozicije A i B su bez gubitka informacije (zbog FZ DRZAVA $\rightarrow$ STATUS po kojoj je sprovedena dekompozicija A, odnosno zbog FZ I_SIF $\rightarrow$ STATUS po kojoj je sprovedena dekompozicija B). Ipak, dekompozicija A je bolja jer ona čuva obe polazne funkcionalne zavisnosti. Pri dekompoziciji B ne može se izvesti polazna FZ DRZAVA $\rightarrow$ STATUS. To dovodi do anomalija jer je "razbijen" entitet DRZAVA sa svojim atributom STATUS na dve relacije, pa se ne može uneti informacija o statusu neke države dok se neki izdavač ne pridruži toj državi. Zato je dekompozicija B nekorisna. Dekompozicija C je pogrešna jer nije sprovedena po FZ koja bi obezbedila očuvanje informacije; spajanjem projekcija I2 i I3 ne bi se mogla dobiti polazna relacija IZDAVAC, u šta je lako uveriti se analizom primera njenog mogućeg sadržaja.

Uopšte, pri dekompoziciji relacije u njene dve projekcije važno je poštovati dva pravila:

1. zajednički atributi projekcija moraju biti ključ u bar jednoj od projekcija (da bi se informacija sačuvala);
2. potrebno je da se sve FZ polazne relacije mogu izvesti iz FZ u projekcijama dobijenim dekompozicijom.

Kada su zadovoljena oba pravila, dobijene projekcije su nezavisne u smislu da se mogu nezavisno ažurirati; u slučaju da se drugo pravilo ne može primeniti (kao u nekim slučajevima BCNF dekompozicije), dekompozicija je nekorisna.

9.5 Četvrta normalna forma

Uzrok anomalija u jednoj relaciji mogu da budu, pored raznih vrsta funkcionalnih zavisnosti, i višeznačne zavisnosti. U tom slučaju relacija se dekomponuje u niz projekcija koje su u tzv. četvrtoj normalnoj formi.

Definicija 9.5 Relacija $R(A_1, \ldots, A_n)$ je u *četvrtoj normalnoj formi* (u oznaci 4NF) ako za svaku netrivijalnu VZ $X \rightarrow\!\!\!\rightarrow Y$, gde su $X, Y \subseteq Atr(R)$, važi i FZ $X \rightarrow A_i$, za svako $i = 1, 2, \ldots, n$.

Jedna interpretacija relacije u 4NF je sledeća: ona je u 3NF i sve VZ u njoj su u stvari specijalni slučaj – FZ. Drugim rečima, relacija je u 4NF ako su sve zavisnosti u njoj ključne, tj. na levoj strani imaju ceo ključ te relacije.

Ako relacija nije u 4NF, postupkom dekompozicije ona se može zameniti skupom svojih projekcija koje su u 4NF i čijim spajanjem se dobija polazna relacija. Algoritam je sličan algoritmu BCNF normalizacije, s tim što se u relaciji R koja nije u 4NF uočava VZ $X \rightarrow\rightarrow Y$ (i, zbog komplementarnosti, VZ $X \rightarrow\rightarrow Atr(R) \setminus XY$), pa se relacija R zamenjuje projekcijama $R[XY], R[X, Atr(R) \setminus XY]$, sa kojima se ponavlja isti postupak ako nisu u 4NF.

Na primer, u relaciji PNR(PREDMET, NASTAVNIK, REFERENCA) iz glave 8 važe VZ PREDMET $\rightarrow\rightarrow$ NASTAVNIK i PREDMET $\rightarrow\rightarrow$ REFERENCA, pa ona nije u 4NF. Relacija PNR može se zameniti svojim projekcijama na parove atributa (PREDMET, NASTAVNIK) i (PREDMET, REFERENCA), koje su u 4NF i čijim spajanjem se dobija relacija PNR.

Ceo postupak BCNF i 4NF normalizacije je ponekad nepoželjan, jer anomalije prouzrokovane tranzitivnim zavisnostima ključnih atributa od ključa, i višeznačnim zavisnostima, često jesu deo semantike realnosti koja se modelira a ne anomalije u pravom smislu reči. Drugi razlog zbog kojeg BCNF i 4NF normalizacija mogu biti nepoželjne je to što dobijene dekompozicije ne čuvaju sve polazne zavisnosti, pa se dobijene projekcije ne mogu nezavisno ažurirati. Preveliko usitnjavanje normalizovanih relacija – projekcija može takođe da bude razlog protiv ovih završnih oblika normalizacije. Zato se postupak normalizacije u praksi obično završava na trećoj normalnoj formi; ona predstavlja rezultat logičkog projektovanja kome teže i metode semantičkog modeliranja.

9.6 Peta normalna forma

Postoje slučajevi kada posmatrana relacija predstavlja veći broj entiteta ili njihovih odnosa (više od dva). Ako je svaki od tih entiteta i odnosa predstavljen samo skupovima svojih ključnih atributa, onda nije moguće dekomponovati relaciju u samo dve njene projekcije bez gubljenja informacije, ali je moguća njena dekompozicija u više projekcija. Na primer, relaciju

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_3	b_3	c_3
a_4	b_3	c_4
a_5	b_5	c_5
a_6	b_6	c_5

nije moguće dekomponovati samo u dve njene projekcije bez gubitka informacije, ali je moguće dekomponovati je u sledeće tri binarne projekcije (čijim se spajanjem dobija upravo relacija R):

R[A, B]

A	B
a_1	b_1
a_1	b_2
a_3	b_3
a_4	b_3
a_5	b_5
a_6	b_6

R[A, C]

A	C
a_1	c_1
a_1	c_2
a_3	c_3
a_4	c_4
a_5	c_5
a_6	c_5

R[B, C]

B	C
b_1	c_1
b_2	c_2
b_3	c_3
b_3	c_4
b_5	c_5
b_6	c_5

Ovo svojstvo relacije R označava prisustvo zavisnosti spajanja koja je zamenom relacije R njenim trima projekcijama eliminisana, a same projekcije su u tzv. petoj normalnoj formi. Ova normalna forma se još zove i normalna forma projektovanja/spajanja (engl. projection/join normal form, PJNF), a definiše se na sledeći način ([32]):

Definicija 9.6 Relacija R je u *petoj normalnoj formi* (u oznaci 5NF) ako ne sadrži netrivijalne zavisnosti spajanja, tj. ako za svaku ZS $*\{X_1, \ldots, X_m\}$ važi da je $X_i = Atr(R)$ za neko $1 \leq i \leq m$.

Kao i za prethodne normalne forme, postoji algoritam dekompozicije relacije koja nije u 5NF u skup njenih projekcija koje jesu u 5NF i čijim spajanjem se dobija polazna relacija. Relacija se sada dekomponuje prema uočenoj ZS $*\{X_1, \ldots, X_m\}$, i proizvodi se m projekcija u jednom koraku, $R[X_1], \ldots, R[X_m]$, sa kojima se nastavlja isti postupak ukoliko nisu u 5NF.

Za relaciju u 5NF važi da je istovremeno i u svim nižim normalnim formama. Štaviše, peta normalna forma je "najjača" normalna forma sintaksno zasnovanog logičkog projektovanja vođenog zavisnostima i algebrom zavisnosti (funkcionalnih, višeznačnih i zavisnosti spajanja), i praktično je ostala u domenu teorijskih razmatranja.

U neizvesnosti da li postoje drugi mehanizmi ograničenja vrednosti atributa, osim funkcionalnih, višeznačnih i zavisnosti spajanja, učinjen je pokušaj da se definiše normalna forma koja bi isključila anomalije bilo kog uzroka ([33]). To je tzv. domen/ključ normalna forma (engl. domain/key normal form, DKNF) koja zahteva da su sva ograničenja vrednosti atributa posledica ključa ili domena. Definicija je neoperacionalizovana, pa se samo za pojedinačne primere relacija, koje su već u nekoj od dobro definisanih normalnih formi, može utvrditi da su i u DKNF.

Normalne forme uključuju samo jedan, sintaksno orijentisan kriterijum logičkog projektovanja, koji se u drugim, pretežno semantičkim metodama logičkog projektovanja, često zanemaruje. U sledećoj glavi biće reči o takvim metodama logičkog projektovanja.

9.7 Pitanja i zadaci

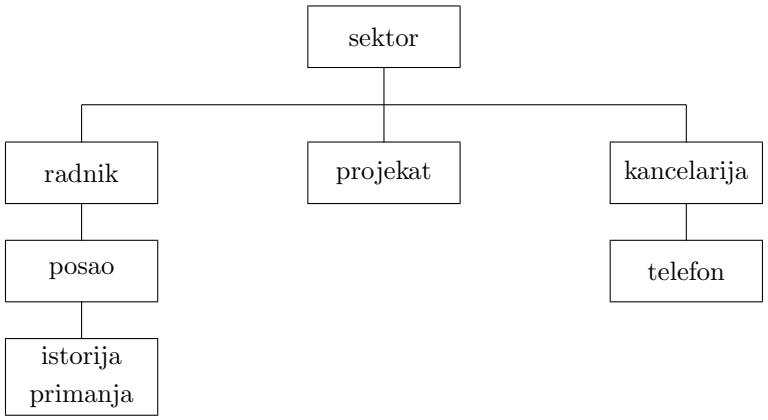
1. Definisati sledeće pojmove:

prva normalna forma	treća normalna forma
parcijalna zavisnost	Boyce-Codd normalna forma
druga normalna forma	četvrta normalna forma
tranzitivna zavisnost	peta normalna forma

2. Opisati algoritme 2NF, 3NF i BCNF normalizacije. Proceniti njihovu vremensku složenost.
3. Koje su prednosti a koji nedostaci normalizacije dekompozicijom?
4. Navesti primer relacije koja se ne može korisno (tj. sa očuvanjem polaznih FZ) dekomponovati u BCNF projekcije.
5. Opisati algoritam sinteze 3NF relacija iz zadatog skupa funkcionalnih zavisnosti.
6. Neka je dat sledeći skup funkcionalnih zavisnosti:

$A \rightarrow B$	$B \rightarrow D$
$B \rightarrow C$	$B \rightarrow A$
$C \rightarrow D$	$ABD \rightarrow E$

   Algoritmom sinteze 3NF relacija konstruisati shemu relacione baze podataka.
7. Na slici 9.1 prikazana je hijerarhijska (nenormalizovana) reprezentacija podataka u bazi podataka jedne kompanije. Slika se interpretira na sledeći način:



Slika 9.1: Nenormalizovana shema

   – kompanija ima više sektora;
   – svaki sektor uključuje skup radnika, skup projekata i skup kancelarija;

- za svakog radnika postoji opis svih poslova koje je radio; za svaki takav posao postoji i istorija primanja radnika dok je radio taj posao;
- svaka kancelarija ima skup (jedan ili više) telefona.

Baza podataka sadrži sledeće podatke.

- za svaki sektor: jedinstveni broj sektora, sredstva i jedinstveni broj rukovodioca;
- za svakog radnika: jedinstveni broj radnika, broj projekta na kome je angažovan, broj kancelarije i broj telefona; takođe, baza sadrži i naziv svakog posla koji je radnik obavljao, kao i datum i iznos svakog primanja na tom poslu;
- za svaki projekat: jedinstveni broj projekta i sredstva;
- za svaku kancelariju: jedinstveni broj kancelarije, površinu (u kvadratnim metrima) i brojeve svih telefona u toj kancelariji.

Projektovati relacijske sheme relacija koje su u 3NF i koje prikazuju ove podatke. Formulisati pretpostavke o odnosu uvedenih atributa, kao i zavisnosti koje formalizuju te pretpostavke.

8. Neka fakultetska baza podataka o nastavi sadrži podatke o nastavnicima, predmetima, mestu i vremenu održavanja predavanja.

   Potrebno je uključiti sledeće podatke:

   - za svaki predmet: šifru predmeta, naziv predmeta, smer na kome se predaje, semestar u kome se predaje i fond časova;
   - za svakog nastavnika: jedinstveni lični broj, ime, zvanje, naučni stepen;
   - za svako predavanje: jedinstveni broj nastavnika, šifru predmeta, vreme održavanja i broj sale.

   Definisati atribute kojima se opisuju ovi podaci, formulisati pretpostavke o odnosu uvedenih atributa, kao i zavisnosti koje formalizuju te pretpostavke. Projektovati relacijske sheme relacija koje su u 3NF i koje prikazuju ove podatke.

10

Semantičko modeliranje

Sistemi zasnovani na tradicionalnim modelima – hijerarhijskom, mrežnom i relacionom, imaju tendenciju da u značajnoj meri razgraničavaju unutrašnji (fizički) nivo podataka od konceptualnog (logičkog) i spoljašnjeg, ali ne u dovoljnoj meri i spoljašnji nivo od konceptualnog. Osim toga, ovi modeli ne podržavaju semantiku podataka, već je ona prepuštena korisniku.

Tako, na primer, relacioni sistemi "ne prepoznaju" da su "naslov knjige" i "država" semantički različiti atributi, pa je moguće, ako su definisani na istom domenu (niske simbola), izvršiti operaciju spajanja nad njima. Ovi modeli "ne prepoznaju" ni da je, na primer, tip entiteta "programer" specijalni slučaj tipa entiteta "radnik", a da je ovaj – specijalni slučaj tipa entiteta "osoba", odnosno da je ta vrsta hijerarhije tipova entiteta na bilo koji način specifična u odnosu na opšte pravilo referencijalnog integriteta (hijerarhija koju podržava hijerarhijski model je sasvim drugačijeg značenja). Tako se dolazi do potrebe za proširenjem postojećih modela semantičkim aspektom podataka, tj. do stvaranja novih modela koji uključuju semantičku komponentu.

Predstavljanje značenja podataka modelom podataka zove se *semantičko modeliranje*. Osnovni cilj semantičkog modeliranja je da se sistemi zasnovani na novim modelima ponašaju inteligentnije od sistema za upravljanje bazama podataka zasnovanih na klasičnim modelima. Svaki od modela do kojih se na taj način dolazi poseduje manju ili veću moć predstavljanja semantike podataka (izvesnu moć te vrste poseduju i klasični modeli), pa je svaki od njih u određenoj meri *semantički model*.

Među konceptima koji karakterišu semantičku komponentu podataka, i koji imaju specifičnu (logičku) reprezentaciju u svakom od semantičkih modela, od značaja su sledeći: entitet, svojstvo (atribut) i odnosi među entitetima kao što su asocijacija, podtip i nadtip. Postupci kojima se uspostavljaju odnosi među entitetima su agregacija, generalizacija i specijalizacija. Svaki od tih modela i dalje poseduje skupove operacija za manipulisanje reprezentacijama semantičkih koncepata i pravila integriteta nad tim reprezentacijama.

Semantički koncepti i modeli, čak i kada nemaju kompletnu podršku operacijama i/ili pravilima integriteta, predstavljaju izuzetno značajan okvir za logičko projektovanje, s obzirom da prevazilaze osnovni nedostatak sintaksno orijentisane metode normalizacije – nedostatak semantičke komponente.

U daljem tekstu ovog poglavlja biće prvo detaljnije predstavljeni osnovni koncepti semantičkih modela, a zatim će biti prikazana tri modela koji mogu da ponesu epitet semantičkih – *prošireni relacioni model*, *model entiteta i odnosa* i *semantički model baze podataka*.

Entitet, kao osnovni pojam koji se ne definiše, a čije je značenje stvar, biće, pojava, događaj od značaja, koji se može jednoznačno identifikovati, može da postoji na različitim nivoima samostalnosti. Tako, *samostalni entitet* karakteriše njegova egzistencijalna nezavisnost od drugih entiteta u bazi, tj. on je od interesa (u specifičnoj bazi) sam za sebe. S druge strane, *opisni entitet* (ili zavisni entitet) karakteriše njegova egzistencijalna zavisnost od drugog entiteta u bazi, tj. on opisuje, kao svojstvo, neki drugi entitet. Tako je u bazi o zaposlenima u nekom preduzeću, entitet RADNIK samostalni entitet, a entitet DETE_RADNIKA – opisni entitet. Svaka od ovih vrsta entiteta ima sopstvena svojstva (u terminima raznih modela – atribute, karakteristike). Potreba za izdvajanjem opisnih entiteta nastaje zbog strogo višeznačnih zavisnosti (onih koje nisu funkcionalne) među atributima nekih entiteta (v. odeljak 8.5 o višeznačnim zavisnostima).

Asocijacija je odnos M:N između dva tipa entiteta (ili više tipova entiteta), koji može imati i svoja sopstvena svojstva (oznaka "M:N" označava da jedan entitet prvog tipa može biti pridružen većem broju entiteta drugog tipa, i obratno). Tako se između entiteta tipa RADNIK i PREDUZEĆE može uspostaviti odnos – asocijacija ZAPOSLENJE, sa svojstvima kao što su datum zaposlenja, status, položaj, prihod itd. Osim što je asocijacija odnos između tipova entiteta, ona se može posmatrati i kao entitet specifičnog tipa – *asocijativni entitet*.

Između dva tipa entiteta X i Y važi odnos *podtip/nadtip* ako je svaki entitet tipa X obavezno i entitet tipa Y. U ovom odnosu, tip entiteta X je *podtip*, a tip entiteta Y je *nadtip*. Pri tome, tip entiteta Y može biti nadtip za veći broj tipova entiteta $X_1, \ldots, X_n$ u datom odnosu podtip/nadtip, dok je tip entiteta X podtip samo tipa entiteta Y u tom odnosu podtip/nadtip. Stoga se odnos podtip/nadtip može posmatrati kao preslikavanje nadtipa u uniju podtipova. Na primer, tip entiteta OSOBA može biti nadtip za tipove entiteta RADNIK, STUDENT, PENZIONER, NEZAPOSLENO_LICE, dok su pobrojani tipovi – podtipovi tipa entiteta OSOBA. Nad istim parom tipova entiteta X, Y može biti definisan veći broj odnosa tipa podtip/nadtip, pa se svaki takav odnos imenuje značenjem preslikavanja nadtipa u uniju podtipova. Tako bi navedeni odnos mogao da se zove "PO_ZANIMANJU_JE" (npr. osoba o_1 je po zanimanju radnik), dok bi drugi odnos podtip/nadtip nad istim tipovima entiteta mogao da bude "SUPRUŽNIK_JE_PO_ZANIMANJU" (npr. suprug(a) osobe o_1 je po zanimanju radnik).

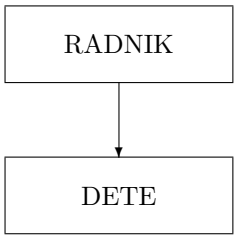
Pri izgradnji asocijativnog tipa entiteta, kao i pri izgradnji podtipa (nadtipa) entiteta, od značaja je, pre svega, način na koji iz jednog skupa tipova entiteta

nastaje drugi skup tipova entiteta, a zanemaruju se specifičnosti pojedinih tipova entiteta (njihova svojstva). Tako se "udruživanjem" dva ili više tipova entiteta dobija novi – asocijativni tip entiteta (tipovi entiteta RADNIK i PREDUZEĆE udružuju se u tip entiteta ZAPOSLENJE), "uopštavanjem" većeg broja tipova entiteta dobija se jedan novi tip entiteta – njihov nadtip, a "preciziranjem" jednog tipa entiteta dobija se veći broj novih tipova entiteta – njegovih podtipova.

Pogled na bazu podataka u kome se neki detalji namerno zanemaruju zove se *apstrakcija* te baze podataka ([58]). Apstrakcija kojom se odnos između dva ili više tipova entiteta definiše kao entitet nove vrste zove se *agregacija*, a apstrakcija kojom se skup sličnih tipova entiteta posmatra kao jedan *generički* tip entiteta zove se *generalizacija*; ti slični tipovi entiteta su *podtipovi* generičkog tipa. Apstrakcija koja je "inverzna" generalizaciji zove se *specijalizacija*.

I u okviru klasičnih modela podataka (u nekom bolje, u nekom lošije) moguće je predstaviti prethodno opisane semantičke koncepte. Zato oni, pre svega, služe kao sredstvo za modeliranje dela stvarnosti koji se prikazuje bazom podataka, tj. kao elementi semantičkih metoda logičkog projektovanja (relacione, hijerarhijske ili mrežne) baze podataka. Tako, na primer, samostalni entitet RADNIK i njegov opisni entitet DETE_RADNIKA mogu se predstaviti hijerarhijskom strukturom kao na dijagramu na slici 10.1, ili, na primer, nenormalizovanom relacijom

RADNIK(IME, DOHODAK, DETE(IME, GODINE)).



Slika 10.1: Samostalni i opisni entitet

U relacionom modelu ova dva tipa entiteta se predstavljaju dvema relacijama:

RADNIK (IME, DOHODAK)
DETE (IME, GODINE, IME_RODITELJA)

(primarni ključevi relacija su podvučeni; entitet DETE je, osim što je egzistencijalno zavisan od entiteta RADNIK, i identifikaciono zavisan od njega, jer atribut IME_RODITELJA, koji ima isto značenje kao atribut IME relacije RADNIK, ulazi u primarni ključ relacije DETE).

Asocijacija PREDAVANJE nad tipovima entiteta NASTAVNIK i PREDMET može se grafički (tzv. mrežnom strukturom) predstaviti kao na dijagramu na slici 10.2 (jedno predavanje uključuje tačno jednog nastavnika i tačno jedan predmet,

dok jedan nastavnik može učestvovati u većem broju predavanja; isto važi i za predmet).

Slika 10.2: Asocijativni entitet

Ista asocijacija bi se relacionim modelom predstavila trima relacijama:

NASTAVNIK	(IME, ZVANJE)
PREDMET	(NAZIV, FOND_ČASOVA)
PREDAVANJE	(IME, NAZIV, VREME, SALA)

(primarni ključevi su podvučeni; asocijativni entitet PREDAVANJE nastao je agregacijom entiteta NASTAVNIK i PREDMET, pa je i primarni ključ tog asocijativnog entiteta skup primarnih ključeva entiteta od kojih je nastao (IME, NAZIV). Ovi atributi u relaciji PREDAVANJE su i strani ključevi).

Generalizacija uključuje sledeći opšti uslov integriteta: za svaki primerak entiteta podtipa mora u bazi postojati i odgovarajući primerak entiteta nadtipa. Tako se generalizacija sa nadtipom OSOBA i podtipovima RADNIK, STUDENT, PENZIONER, NEZAPOSLENO_LICE može u relacionom modelu predstaviti sa nekoliko relacija. Jedna relacija odgovara nadtipu i sadrži zajedničke atribute svih podtipova, a po jedna relacija odgovara svakom podtipu i, osim primarnog ključa relacije – nadtipa, sadrži samo još specifične atribute za taj podtip. Tako naš primer može imati sledeću relacionu reprezentaciju:

OSOBA (MLB, IME, ADRESA, ZANIMANJE)
RADNIK (MLB, STRUKA, PREDUZECE, RADNO_MESTO)
STUDENT (MLB, FAKULTET, STATUS, GODINA, BR_IND)
PENZIONER (MLB, STATUS, STAZ, IZNOS)
NEZAPOSLENO_LICE (MLB, STATUS, NAKNADA).

Opšti uslov integriteta generalizacije bi se u relacionom sistemu mogao zadati korišćenjem odgovarajućeg iskaza, npr. u jeziku SQL2,

```
CREATE ASSERTION GEN1  CHECK
     ( NOT EXISTS    ( SELECT * FROM RADNIK
                       WHERE NOT EXISTS
                           ( SELECT * FROM OSOBA
                             WHERE OSOBA.MLB = RADNIK.MLB ) ) )
```

(ne postoji n-torka relacije RADNIK za koju ne postoji n-torka relacije OSOBA sa istim matičnim ličnim brojem). Slične uslove integriteta treba zadati i za relacije STUDENT, PENZIONER, NEZAPOSLENO_LICE.

Generalizacijom se u relacionom modelu izbegava druga vrsta nedostajuće vrednosti – neprimenljivo svojstvo (npr. svojstvo BR_IND za radnika, kada bi sve osobe bile predstavljene samo jednom relacijom sa unijom atributâ prethodnih relacija).

Druga strategija za predstavljanje generalizacije u relacionom modelu su pogledi. Tako relaciona baza može da se sastoji od relacija koje odgovaraju podtipovima i sadrže sve atribute odgovarajućih podtipova:

RADNIK (MLB, IME, ADRESA, STRUKA, PREDUZECE, RADNO_MESTO)
STUDENT (MLB, IME, ADRESA, FAKULTET, STATUS, GODINA, BR_IND)
PENZIONER (MLB, IME, ADRESA, STATUS, STAZ, IZNOS)
NEZAPOSLENO_LICE (MLB, IME, ADRESA, STATUS, NAKNADA).

Sada se relacija koja odgovara nadtipu koji uključuje samo zajedničke atribute podtipova može realizovati pogledom (u SQL2):

```
CREATE VIEW OSOBA AS
       SELECT MLB, IME, ADRESA
       FROM RADNIK
    UNION
       SELECT MLB, IME, ADRESA
       FROM STUDENT
    UNION
       SELECT MLB, IME, ADRESA
       FROM PENZIONER
    UNION
       SELECT MLB, IME, ADRESA
       FROM NEZAPOSLENO_LICE
```

Jedan od prvih rezultata na razvoju semantičkih modela jeste "Osnovni semantički model" ([54]). On prepoznaje razne nivoe samostalnosti entiteta kao i odnos asocijacije, ustanovljava pravila integriteta koja se odnose na egzistencijalnu zavisnost asocijativnog entiteta od komponentnih entiteta, i ostaje u okviru strukture n-arnih relacija relacionog modela.

10.1 Prošireni relacioni model RM/T

Prvu verziju proširenog relacionog modela RM/T definisao je Codd u radu [18] ("T" u oznaci modela nema značenje vezano za prirodu modela, već dolazi od geografskog pojma "Tasmanija", gde je autor boravio u vreme rada na proširenom

relacionom modelu). To je kompletan model jer poseduje sva tri neophodna dela modela.

Strukturni deo modela, koji je vrlo precizno definisan, bogat je i sadrži veći broj semantički različitih tipova relacija. Model ne pravi nepotrebne razlike između entiteta i odnosa, već se odnos posmatra kao specijalna vrsta entiteta. Integritetni deo modela je takođe precizno definisan kroz niz pravila za očuvanje integriteta, koja se odnose na pojedinačne tipove relacija (entiteta) i veze među njima. Najzad, RM/T ima specifični skup operacija za manipulisanje relacijama, kojima je proširen skup operacija osnovnog relacionog modela.

10.1.1 Strukturni deo modela

Prema klasifikacionoj shemi RM/T modela, svi entiteti su podeljeni u tri klase:

- osnovni entiteti (odgovaraju samostalnim entitetima u ranijem smislu);
- opisni entiteti (u ranijem smislu);
- asocijativni entiteti (u ranijem smislu), koji se mogu graditi nad osnovnim, opisnim ili asocijativnim tipovima entiteta.

Osim toga, kako među raznim tipovima entiteta mogu postojati različiti odnosi (asocijacija, podtip/nadtip), RM/T uključuje i formalnu katalošku strukturu kojom se ti odnosi mogu registrovati u sistemu.

U osnovnom relacionom modelu, entiteti predstavljeni n-torkama relacije jednoznačno se identifikuju pomoću primarnog ključa koji definiše korisnik. U modelu RM/T entiteti se identifikuju pomoću *surogata*, primarnih ključeva koje generiše i kontroliše sistem. Dakle, svi primarni i strani ključevi u RM/T su surogati. Surogati su jedinstvene reprezentacije entiteta; to su nepromenljive i neponovljive vrednosti, od kojih svaka odgovara pojedinačnoj n-torci relacije i koje se generišu pri unošenju n-torki kojima se predstavljaju entiteti.

Entiteti svih klasa predstavljeni su dvema vrstama relacija: E-relacije i P-relacije.

Svakom tipu entiteta (bez obzira na njegovu klasu) odgovara jedna E-relacija. E-relacija za dati tip entiteta je unarna relacija (sa jednom kolonom) koja sadrži surogate svih entiteta tog tipa, predstavljenih u bazi. Sa druge strane, svakom tipu entiteta odgovara jedna ili više P-relacija, od kojih svaka sadrži po dva ili više atributa: jedan od atributa je surogat odgovarajućeg tipa entiteta, a ostali atributi su neki (može i svi) atributi tog tipa entiteta. Dakle, E-relacija registruje egzistenciju entiteta, a P-relacije registruju njegova svojstva. Na primer, tip entiteta RADNIK sa atributima MLB, IME, ADRESA, STRUKA može se predstaviti jednom E-relacijom

RADNIK(RADNIK$)

i trima P-relacijama:

RADNIKBI(RADNIK$, MLB, IME),
RADNIKA (RADNIK$, ADRESA),
RADNIKS (RADNIK$, STRUKA).

Izbor načina za grupisanje svojstava u P-relacije vrši projektant baze (mogu se i svi atributi prikazati jednom P-relacijom), a krajnosti pri ovom izboru (binarne P-relacije ili sva svojstva u jednoj relaciji) odgovaraju opredeljenju za nesvodive (binarne) modele odnosno normalizovane modele.

U slučaju asocijativnog tipa entiteta, među svojstvima odgovarajućih P-relacija naći će se sistemski primarni ključevi (surogati) tipova entiteta – komponenti asocijativnog entiteta, a ne njihovi korisnički primarni ključevi. Tako bi za asocijativni entitet UČEŠĆE koji je izgrađen nad entitetima tipa RADNIK i PROJEKAT, u svaku njegovu P-relaciju ušao atribut – surogat RADNIK$ (a ne primarni ključ MLB), i slično za tip entiteta PROJEKAT. Asocijativni entiteti mogu imati svoje opisne entitete, i mogu učestvovati u drugim asocijativnim entitetima.

S obzirom na odnos podtip/nadtip koji je podržan modelom RM/T, svaki entitet ima bar jedan tip, ali može imati i više tipova koji preko ovog odnosa obrazuju hijerarhiju. Tako je svaki entitet tipa LEKAR ujedno i entitet tipa RADNIK odnosno i entitet tipa OSOBA (ako su ti tipovi entiteta predstavljeni u bazi). Za predstavljanje podtipa odnosno nadtipa važi isti mehanizam kao i za svaki tip entiteta: jedna E-relacija i jedna ili više P-relacija. P-relacija podtipa registruje samo specifična svojstva tog podtipa, dok su sva svojstva odgovarajućeg nadtipa automatski primenljiva i na taj podtip. Ova primenljivost, kao i važenje odgovarajućih pravila integriteta u slučaju da relacije predstavljaju nadtip/podtip odnos, obezbeđeni su registrovanjem ovakvog odnosa u sistemskom katalogu RM/T.

10.1.2 Integritetni deo modela

Osim dva opšta pravila integriteta koja važe za osnovni relacioni model (integritet entiteta i referencijalni integritet), RM/T poseduje još niz opštih pravila integriteta:

- Integritet entiteta u RM/T : E-relacije ne prihvataju nedostajuće vrednosti; nad E-relacijama moguće je vršiti operacije unošenja i brisanja, ali ne i operacije ažuriranja.
- Integritet svojstva: Ako n-torka t postoji u nekoj P-relaciji, onda odgovarajuća E-relacija mora imati registrovano postojanje entiteta koji t opisuje, tj. sistemski primarni ključ (surogat) n-torke t mora postojati u odgovarajućoj E-relaciji.
- Integritet opisnog entiteta: Ako opisni entitet postoji u bazi, onda u bazi mora postojati i entitet koji taj opisni entitet opisuje.
- Integritet asocijacije: Asocijativni entitet može postojati u bazi (kao i njegov surogat u odgovarajućoj E-relaciji), čak i ako je jedan ili više entiteta koji učestvuju u toj asocijaciji nepoznato. U tom slučaju surogat E-ω (surogat

nedostajuće vrednosti) označava da je entitet – učesnik nepoznat. Ovo pravilo ne važi u slučaju da postoji eksplicitni uslov integriteta koji to zabranjuje.

- Integritet podtipa: Ako surogat s pripada E-relaciji za tip entiteta S, onda s mora da pripada E-relaciji svakog tipa entiteta čiji je S podtip.

10.1.3 RM/T katalog

RM/T katalog je struktura relacijâ o relacijama, atributima i domenima u bazi podataka, koja olakšava korišćenje i transformaciju informacija. Mada bi svaka specifična implementacija dopunila ovu strukturu dodatnim atributima i relacijama, minimalni projekat RM/T kataloga uključio bi sledeće dve grupe relacija (zanemareno je razdvajanje na E i P-relacije, o čemu bi svaka implementacija povela računa):

I	DOMENI	(D$, DOM_IME, TIP_PODATAKA, UREĎENJE)
	RELACIJE	(R$, REL_IME, REL_TIP)
	ATRIBUTI	(A$, ATR_IME)
	REL_ATR	(RA$, R$, A$, KLJUC)
	ATR_DOM	(AD$, A$, D$).

Prve tri relacije opisuju sve domene, relacije i atribute u bazi, i sadrže po jednu n-torku za svaki domen, relaciju odnosno atribut, redom. Druge dve relacije predstavljaju veze relacijâ i pripadnih atributa, odnosno atributa i pripadnih domena.

Značenje pojedinih atributa (onih za koje to nije očevidno) je sledeće:

– UREĎENJE u relaciji DOMENI ukazuje na to da li je nad datim domenom definisana relacija poretka $>$ (DA/NE);

– REL_TIP je vrsta relacije – E-relacija (osnovna, opisna ili asocijativna, podtip), P-relacija ili kataloška relacija;

– KLJUC ukazuje na učešće atributa u korisnički definisanom ključu odgovarajućeg entiteta (DA/NE).

Druga grupa relacija odnosi se na predstavljanje različitih veza koje postoje među relacijama u bazi podataka (tzv. grafovske relacije, s obzirom da se odgovarajuće veze mogu predstaviti grafom). To su:

II PE (PE$, P-ime-relacije, E-ime-relacije)
(veza E-relacije i svih odgovarajućih P-relacija);

OS (OS$, Opis-E-ime-relacije, Osn-E-ime-relacije)
(veza E-relacija opisnog i njemu pripadnog osnovnog entiteta);

AS (AS$, Asoc-E-ime-relacije, Asoc-P-ime-atr, Komp-E-ime-relacije)
(veza E-relacije asocijacije i E-relacije komponente; za svaku komponentu svake asocijacije u bazi postoji jedna n-torka relacije AS koja, osim imena E-relacija asocijativnog i komponentnog entiteta, sadrži još i ime atributa asocijativnog entiteta (Asoc-P-ime-atr) koji identifikuje taj komponentni entitet (u ranijem primeru, atribut RADNIK$ identifikuje komponentni entitet RADNIK u asocijativnom entitetu UČEŠĆE);

PN (PN$, Podtip-E-ime-relacije, Nadtip-E-ime-relacije, Kriterijum)
(veza E-relacija svakog podtipa i nadtipa u bazi; uz imena ovih relacija, u svakoj n-torci relacije PN prisutan je kriterijum specijalizacije nadtip/podtip; npr. POSAO bi mogao da bude kriterijum specijalizacije nadtipa RADNIK u podtip LEKAR).

10.1.4 Manipulativni deo modela

Među specifičnim operacijama RM/T modela (kako su definisane u [18]), ističu se sledeće:

- PROPERTY – unarna operacija koja, za zadati tip entiteta kao operand, objedinjuje sve odgovarajuće P-relacije u jednu relaciju, eliminišući surogate, i proizvodi relaciju osnovnog relacionog modela koja odgovara tom tipu entiteta;
- DENOTE – unarna operacija koja, za zadato ime relacije kao operand, vraća samu relaciju kao rezultat;
- APPLY – binarna operacija čiji su operandi unarna operacija f koja preslikava relaciju u relaciju (npr. restrikcija), i neki skup relacija $Z = \{Z_1, \ldots, Z_k\}$; rezultat je skup relacija dobijenih primenom operacije f na svaku od relacija iz Z, tj. $\{f(Z_1), \ldots, f(Z_k)\}$;
- COMPRESS – binarna operacija čiji je jedan operand binarna asocijativna i komutativna operacija f koja preslikava par relacija u relaciju (npr. spajanje), a drugi operand je neki skup relacija $Z = \{Z_1, \ldots, Z_k\}$; rezultat operacije COMPRESS je relacija dobijena uzastopnom primenom operacije f na relacije iz Z, tj. $f(f(\ldots f(f(Z_1, Z_2), Z_3), \ldots), Z_k)$.

Na primer, neka na osnovu E-relacije i skupa P-relacija tipa entiteta RADNIK treba konstruisati jedinstvenu n-arnu relaciju RADNIK sa skupom svih atributa

tog tipa entiteta, isključujući surogat. Rezultat bi se u RM/T modelu dobio primenom operacije PROPERTY na tip entiteta RADNIK, tj. rezultat je vrednost izraza PROPERTY(RADNIK). Ova operacija zamenjuje čitav sledeći niz operacija osnovne relacione algebre i drugih operacija RM/T proširenja:

1. izvršiti restrikciju kataloške relacije PE po uslovu E-ime-relacije = 'RADNIK' (dobiju se n-torke koje se odnose na E i P relacije tipa entiteta RADNIK);

2. projektovati rezultat koraka 1 na atribut P-ime-relacije (dobijaju se imena svih P-relacija koje odgovaraju tipu entiteta RADNIK);

3. rezultatu koraka 2 dodati i ime E-relacije tipa entiteta RADNIK (unija);

4. izvršiti spoljašnje prirodno spajanje (v. tačku 2.1.3 – Proširena relaciona algebra) svih relacija čija su imena u skupu koji je rezultat koraka 3;

5. eliminisati iz rezultata koraka 4 atribut RADNIK$.

Od operacija koje ne pripadaju osnovnoj relacionoj algebri, u realizaciju ovog primera trebalo bi uključiti dve operacije: DENOTE (posle koraka 3 da bi se od skupa imena pripadnih relacija dobio skup samih relacija) i operaciju COMPRESS sa prvim operandom OUTER NATURAL JOIN (spoljašnje prirodno spajanje) i drugim operandom – skupom relacija dobijenih prethodnom primenom operacije DENOTE.

Prošireni relacioni model uključuje i jednu vrstu nedostajućih vrednosti i sve operacije koje se na njih odnose (v. tačku 2.1.3).

10.2 Prošireni model entiteta i odnosa (PMEO)

Model entiteta i odnosa definisao je Peter Chen u radu [13]. To i nije u pravom smislu model, jer ne poseduje manipulativni deo, dok je integritetni deo implicitno podržan. Uz model je razvijena i dijagramska tehnika za predstavljanje entiteta i odnosa među njima, sa svim njihovim karakteristikama. Bez obzira na nekompletnost ovog modela, on je doživeo veliku popularnost, zahvaljujući semantičkim konceptima koje podržava, kao i dijagramskoj tehnici koja svojom preglednošću pojednostavljuje predstavljanje entiteta i odnosa u modelu i olakšava razumevanje njihovog značenja.

Posebnu primenu model entiteta i odnosa ima u realizaciji efikasne i produktivne semantičke metode logičkog projektovanja. Shema baze podataka koja se dobije ovom metodom obično se (automatski) preslikava u shemu relacione baze podataka, na koju se primenjuje neki od relacionih jezika za definisanje i manipulisanje podacima. Zato ([27]), model entiteta i odnosa predstavlja samo tanki sloj nad relacionim modelom. Ako je proces projektovanja, zasnovan na modelu entiteta i odnosa, korektno sproveden, tj. ako su uočeni svi bitni entiteti i pravilno

uspostavljeni odnosi među njima, onda dobijena relaciona baza podataka ima svojstvo da su joj, gotovo uvek, sve relacije u nekoj od viših normalnih formi (bar u 3NF).

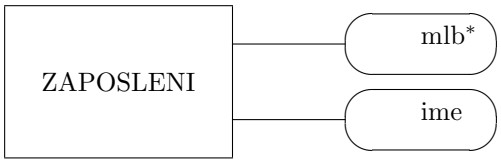
Model entiteta i odnosa je već dugi niz godina najzastupljenija metoda logičkog projektovanja baze podataka. Doživeo je veći broj različitih proširenja i modifikacija (u odnosu na Chen-ovu varijantu), tako da je i danas u upotrebi veći broj verzija tog modela. Ovde će biti prikazana jedna od tih verzija, *prošireni model entiteta i odnosa*, onako kako je izložen u [62].

10.3 Strukturni deo PMEO

Entitet, u smislu u kome je ranije uveden, opisuje se svojim značajnim, u datom kontekstu, *atributima*.

Klasa entiteta je skup entiteta sa istim skupom atributa (npr. klasa STUDENT predstavlja skup entiteta {S1, S2, S3, S4}). Klasa se može opisati tipom entiteta sa svojstvima (npr. broj indeksa, ime, godina studija, prosečna ocena). Svaki pojedinačni entitet (npr. student) je primerak datog tipa, tj. element date klase (npr. klase STUDENT). U dijagramskoj tehnici proširenog modela entiteta i odnosa (PMEO), klasa entiteta (tj. tip entiteta) predstavlja se pravougaonikom sa upisanim imenom, a atribut – elipsom ili zaobljenim pravougaonikom.

Atribut je preslikavanje skupa entiteta datog tipa u domen atributa (skup mogućih vrednosti). Ako je ovo preslikavanje 1–1, atribut je identifikator (primarni ključ) i označava se zvezdicom (slika 10.3; mlb je oznaka za matični lični broj). Ako više atributa ima svojstvo identifikatora, jedan se bira za primarni ključ.



Slika 10.3: Entitet i atributi u PMEO

Odnos prikazuje povezanost i kriterijum povezanosti pojedinih entiteta. I kod odnosa postoje koncepti klase, tipa i primerka odnosa. Klasa odnosa je u matematičkom smislu relacija nad nekim skupom klasa entiteta, i u dijagramskoj tehnici predstavlja se rombom. U ovom proširenju modela entiteta i odnosa dozvoljeni su samo *binarni odnosi* (odnosi nad dva tipa entiteta). Ostali tipovi odnosa se realizuju preko *agregiranih entiteta* (termin koji odgovara asocijativnom entitetu u ranijem smislu).

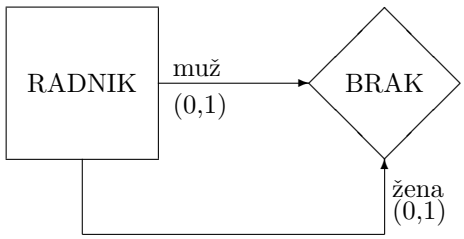
Svaki odnos između dve klase entiteta E1 i E2 definiše dva *imenovana preslikavanja* – iz E1 u E2 i iz E2 u E1. Ime se obično određuje kao uloga klase entiteta koja je domen preslikavanja.

Kardinalnost preslikavanja iz E1 u E2 je par (DG, GG), gde je DG (donja granica) najmanji mogući, a GG (gornja granica) najveći mogući broj dodeljenih elemenata klase E2 jednom elementu klase E1 (DG $\in \{0, 1, \text{konstanta} > 1\}$, GG $\in \{1, \text{konstanta} > 1, \text{M}\}$; M je celobrojna promenljiva).

Primer predstavljanja odnosa u PMEO dat je dijagramom na slici 10.4.

Slika 10.4: Odnos u PMEO

Odnos se može uspostaviti i nad jednim tipom entiteta, kao na slici 10.5.



Slika 10.5: Odnos nad jednim tipom entiteta u PMEO

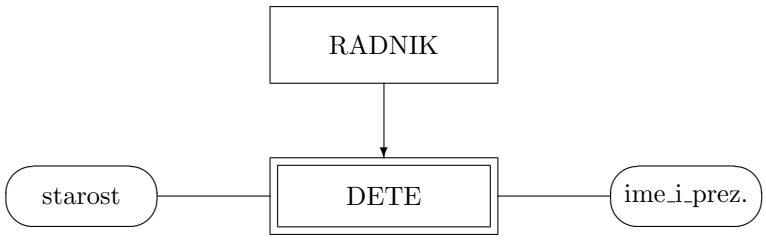
Pri izboru tipova entiteta vodi se računa o tome da svaki atribut može imati najviše jednu vrednost u jednom primerku entiteta (primerak entiteta odgovara n-torci relacije relacionog modela). To se ponekad postiže proglašavanjem atributa za novi entitet i uspostavljanjem odnosa između starog i novog entiteta. Na primer, jedno svojstvo tipa entiteta KNJIGA je IME_AUTORA, ali, kako jedna knjiga može imati više autora, uvodi se novi tip entiteta AUTOR i odnos KNJIGA-AUTOR sa preslikavanjima "napisana" (1,M) i "napisao" (0,M) (slika 10.6).

Regularni (*nezavisni*) *entitet* je termin koji odgovara samostalnom entitetu u ranijem smislu i koji se može identifikovati pomoću sopstvenih atributa (npr. entitet ZAPOSLENI(mlb, ime)).

Slabi entitet je termin koji odgovara opisnom entitetu u ranijem smislu; on se ne može identifikovati samo pomoću sopstvenih atributa, već se moraju koristiti

Slika 10.6: Atribut u funkciji entiteta u PMEO

odnosi sa drugim entitetima. Pri tome se koriste samo funkcionalni odnosi (jedno preslikavanje definisano odnosom ima kardinalnost (1,1)), a identifikovanje pomoću odnosa može se primeniti i rekurzivno, dok se ne dode do regularnog entiteta. Ovi odnosi se predstavljaju specifično, bez romba. Slabi entitet je egzistencijalno (i identifikaciono) zavisan od entiteta pomoću kojeg se identifikuje, pa proizvodi anomalije brisanja i unošenja, samo što to sada nisu anomalije već karakteristike značenja podataka. Primer slabog entiteta DETE_RADNIKA sa atributima ime i prezime, starost, predstavljen je na slici 10.7.

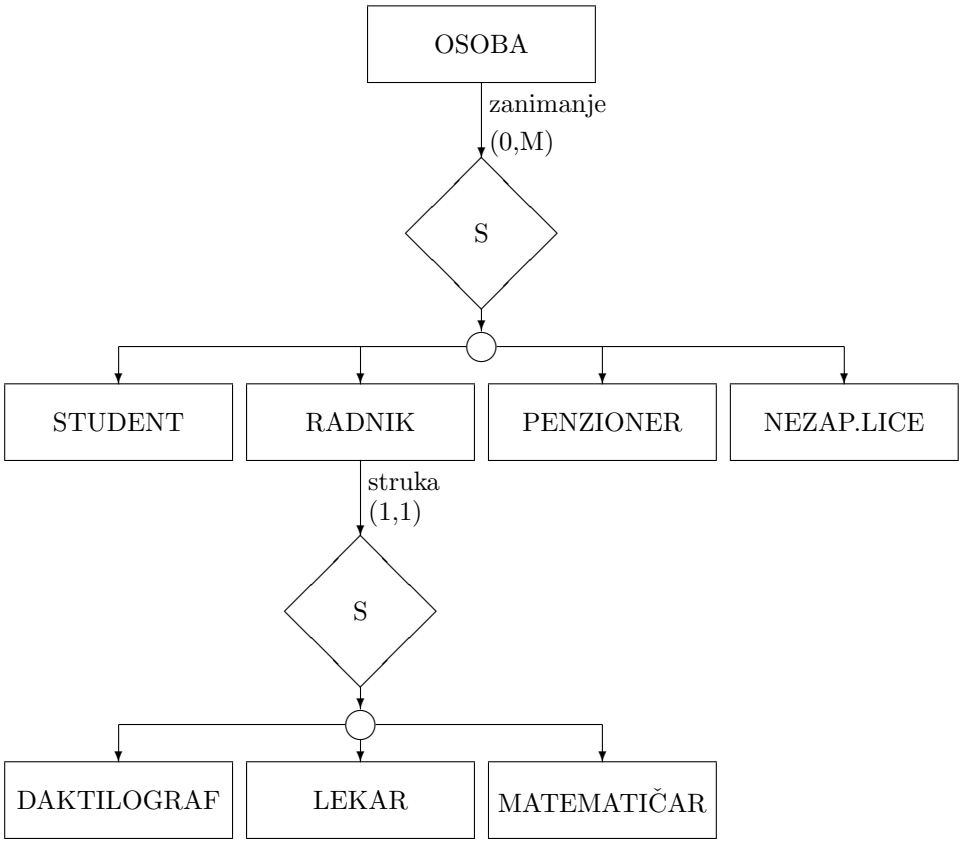


Slika 10.7: Regularni i slabi entitet u PMEO

Generalizacija i specijalizacija su dva smera odnosa podtip/nadtip. Postupak *generalizacije* predstavlja se preslikavanjem PODTIP → NADTIP čija je kardinalnost (1,1). Primerci podtipa su egzistencijalno i identifikaciono zavisni od primeraka nadtipa (ova zavisnost se predstavlja strelicom, kao i kod slabog entiteta).

Specijalizacija, kao i ranije, je postupak definisanja podtipova za neki tip entiteta. U primeru u kome se tip entiteta OSOBA (nadtip) specijalizuje tipovima entiteta RADNIK, STUDENT, PENZIONER, NEZAPOSLENO_LICE (podtipovi), ovi drugi imaju zajedničke atribute – npr. mlb, ime, starost, adresu, i zajedničke operacije – npr. preseljenje lica. Tip entiteta RADNIK se može dalje specijalizovati u podtipove DAKTILOGRAF, LEKAR, MATEMATIČAR, sa specifičnim atributima i operacijama. Specijalizacija se predstavlja preslikavanjem NADTIP → UNIJA_PODTIPOVA, koje može biti različite kardinalnosti (obavezno se navodi). Odnos podtip/nadtip predstavlja se rombom sa oznakom "S" (engl. subtype – podtip).

Kardinalnost preslikavanja koje karakteriše specijalizaciju određuje vrstu specijalizacije. *Ekskluzivna specijalizacija* je ona sa kardinalnošću GG=1 (npr. jedan radnik može imati najviše jednu struku u kojoj radi). *Neekskluzivna specijalizacija* je ona sa kardinalnošću GG>1 (npr. jedna osoba može biti i student i radnik). Ne mora se svaki primerak nadtipa specijalizovati (može biti DG=0). Na primer, učenik je osoba ali se ne specijalizuje ni u jedan od postojećih podtipova. *Kriterijum specijalizacije* je naziv preslikavanja NADTIP → UNIJA_PODTIPOVA (takvih kriterijuma i preslikavanja može biti više), a pri prevođenju u relacioni model taj kriterijum postaje jedan atribut u relaciji koja odgovara nadtipu. Kardinalnost preslikavanja PODTIP → NADTIP se ne označava jer je ona uvek (1,1). Primer odnosa nadtip – podtip (tj. generalizacija/specijalizacija) predstavljen je na slici 10.8.



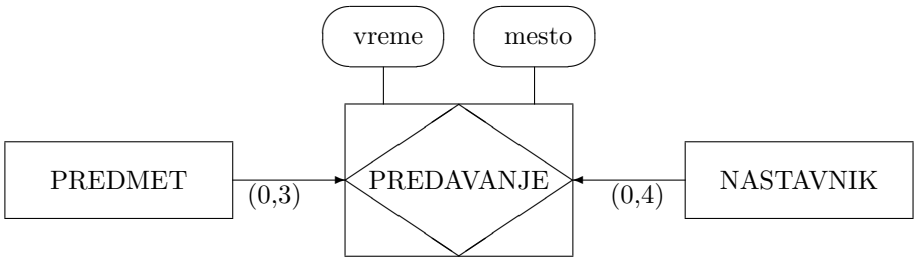
Slika 10.8: Generalizacija/specijalizacija u PMEO

Agregacija je termin koji odgovara asocijaciji u ranijem smislu. Asocijativni entitet koji se dobija u slučaju da odnos asocijacije ima sopstvene atribute, zove

se u PMEO *agregirani entitet*, ili mešovit tip entitet – odnos. Agregirani entitet predstavlja se rombom u pravougaoniku.

Inverzni postupak agregaciji je *dekompozicija*. Tipovi entiteta nad kojima je izgrađen agregirani entitet zovu se *komponente agregacije*. Agregirani entitet je egzistencijalno zavisan od komponenata, i njegovo preslikavanje u komponentni tip je uvek (1,1); zato se ovo preslikavanje i njegova kardinalnost na dijagramu ne označavaju. Kardinalnost preslikavanja komponente u agregirani entitet se mora navesti jer može biti proizvoljna.

Primer agregiranog entiteta je tip entiteta PREDAVANJE, dobijen iz tipova entiteta NASTAVNIK i PREDMET (to su komponente agregacije), sa sopstvenim atributima mesto i vreme (slika 10.9).



Slika 10.9: Agregirani entitet u PMEO

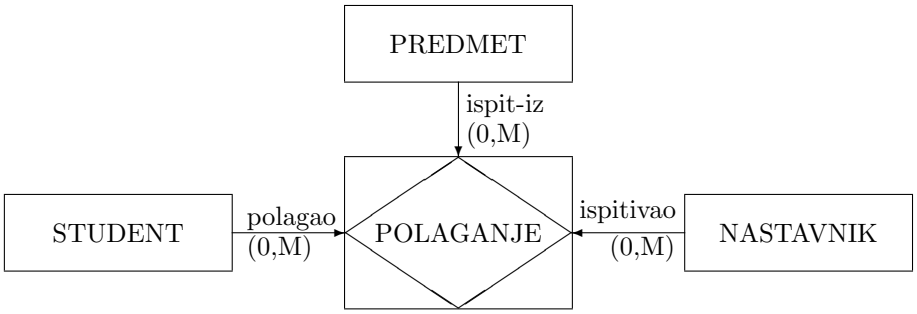
Primarni ključ agregiranog entiteta sastoji se od primarnih ključeva komponentnih entiteta, inače se agregirani entitet ponaša kao i svaki drugi entitet, tj. može da bude u odnosu sa nekim drugim entitetom, može da se specijalizuje itd.

Agregirani entitet (može i bez sopstvenih atributa) koristi se i kada je potrebno predstaviti odnos između većeg broja tipova entiteta (više od dva, slika 10.10).

Skup tipova entiteta i odnosa kojima se u proširenom modelu entiteta i odnosa modeliraju podaci iz baze podataka, zajedno sa njihovim atributima, primarnim ključevima, preslikavanjima koja definišu i kardinalnošću tih preslikavanja, obrazuje shemu baze podataka proširenog modela entiteta i odnosa. Ova se shema najpreglednije predstavlja dijagramom PMEO.

Pravila logičkog projektovanja relacione baze podataka, tj. pravila prevođenja sheme baze podataka PMEO u shemu relacione baze podataka, su sledeća:

- regularni entitet se prevodi u relaciju sa svim njegovim atributima;
- slabi entitet se prevodi u relaciju koja sadrži sve njegove atribute i primarni ključ entiteta od kojeg zavisi;



Slika 10.10: Višestruki odnos u PMEO

- agregirani entitet se prevodi u relaciju koja sadrži sve njegove atribute i primarne ključeve komponenata;
- podtip nekog nadtipa entiteta prevodi se u relaciju koja sadrži njegove specifične atribute i primarni ključ odgovarajućeg nadtipa;
- odnos se prevodi u relaciju koja sadrži primarne ključeve entiteta koji grade odnos;
- ako se u nekom odnosu jedan tip entiteta preslikava u drugi tip entiteta preslikavanjem kardinalnosti (1,1), onda se prvom tipu entiteta kao atribut dodeljuje primarni ključ drugog tipa entiteta; odnos između njih se ne prevodi u posebnu relaciju.

10.3.1 Primer logičkog projektovanja

Razmotrimo uporedni primer logičkog projektovanje baze podataka koja se odnosi na univerzitetsku nastavu, metodom normalnih formi i primenom proširenog modela entiteta i odnosa.

Neka baza podataka sadrži podatke o predavanjima, sa informacijama o predavaču (lični broj, ime, zvanje, naučni stepen), predmetu (šifri predmeta, nazivu, smeru, godini predavanja, fondu časova), vremenu i mestu održavanja. Logičko projektovanje relacione baze metodom normalnih formi polazi od univerzalne relacije koja sadrži sve navedene informacije kao svoje atribute i koja je u 1NF (atributi primarnog ključa su podvučeni):

predavanje(predavač#, ime, zvanje, stepen, šifra_predmeta, naziv, šifra_smera, naziv_smera, godina, fond_časova, vreme, br.sale, sprat)

(atribut "vreme" ulazi u primarni ključ jer se predavanja iz jednog predmeta kod jednog predavača mogu održavati više puta nedeljno).

Funkcionalne zavisnosti u ovoj relaciji su:

1. predavač# → ime, zvanje, stepen
2. šifra_predmeta → naziv
3. šifra_predmeta, šifra_smera → godina, fond_časova
4. šifra_predmeta, šifra_smera, predavač#, vreme → br.sale, sprat
5. šifra_smera → naziv_smera
6. br.sale → sprat.

Iz funkcionalnih zavisnosti pod brojem 1, 2, 3 i 5, sledi parcijalna zavisnost sporednih atributa ime, zvanje, stepen, naziv, godina, fond_časova i naziv_smera, od primarnog ključa. Prema ovim zavisnostima, polazna relacija **predavanje** može se zameniti sledećim skupom relacija (svojih projekcija) koje su u 2NF:

predavač	(predavač#, ime, zvanje, stepen)
predmet	(šifra_predmeta, naziv)
predmet-smer	(šifra_predmeta, šifra_smera, godina, fond_časova)
	(ova se relacija može nazvati i "nastavni plan")
smer	(šifra_smera, naziv_smera)
predavanje2	(predavač#, šifra_predmeta, šifra_smera, vreme, br.sale, sprat)

Relacije **predavač**, **predmet**, **predmet-smer** i **smer** su i u 3NF. U relaciji **predavanje2** prisutna je tranzitivna zavisnost sporednog atributa "sprat" od primarnog ključa ({predavač#, šifra_predmeta, šifra_smera, vreme} → br.sale; br.sale → sprat). Zato se relacija **predavanje2** može zameniti, bez gubljenja informacije, sledećim relacijama koje su u 3NF:

sala	(br.sale, sprat)
predavanje3	(predavač#, šifra_predmeta, šifra_smera, vreme, br.sale).

Dakle, shema relacione baze podataka dobijena metodom normalnih formi sastoji se od relacijskih shema za relacije (koje su u 3NF):

predavač, **predmet**, **predmet-smer**, **smer**, **sala**, **predavanje3**.

Logičko projektovanje relacione baze zasnovano na proširenom modelu entiteta i odnosa polazi od uočavanja entiteta i njihovih odnosa, koje treba predstaviti. Uočimo regularne tipove entiteta (sa atributima):

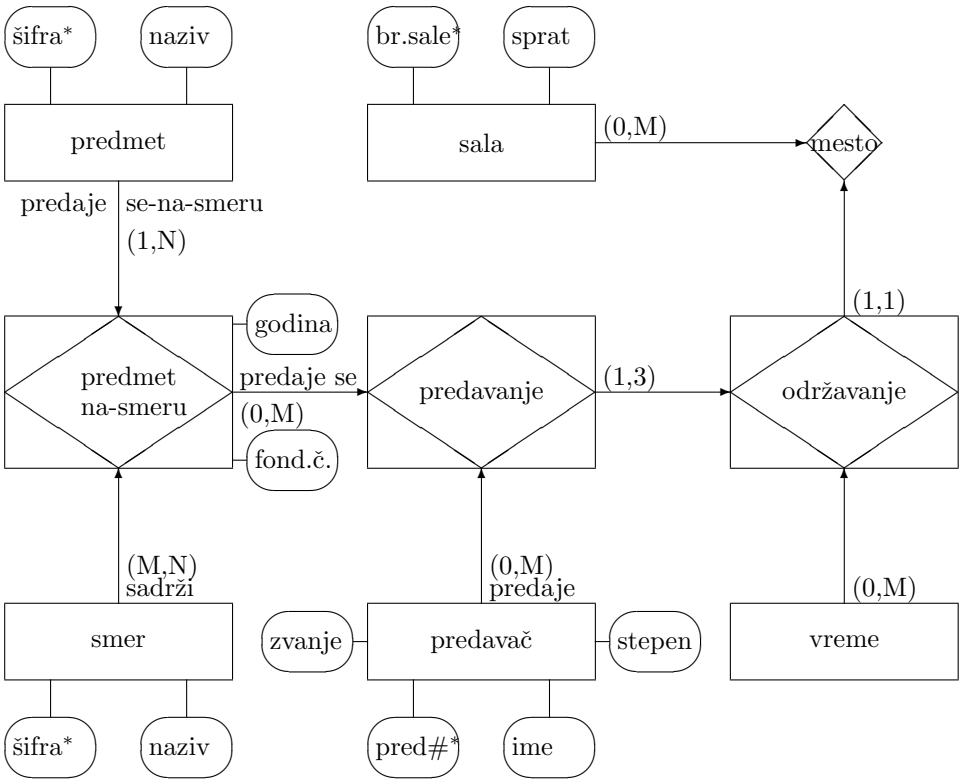
predavač:	predavač#, ime, zvanje, stepen
predmet:	šifra_predmeta, naziv
smer:	šifra_smera, naziv_smera
sala:	br.sale, sprat

i agregirane tipove entiteta:

predavanje – nad komponentnim tipovima entiteta **predavač**, **predmet**, **smer** i **sala**, sa sopstvenim atributom *vreme*, i

predmet-na-smeru (nastavni plan) – nad komponentnim tipovima entiteta **smer** i **predmet**, sa sopstvenim atributima *godina*, *fond_časova*.

Atribut "vreme" agregiranog entiteta "predavanje" uzima više vrednosti za jedan primerak tog entiteta, pa se izdvaja u entitet koji je sa entitetom "predavanje" u odnosu "održavanje", predstavljenom ključnim atributima tih entiteta. Odnos "održavanje", posmatran kao agregirani entitet, u odnosu je (1,1) prema entitetu "sala"; taj odnos se može nazvati "mesto" (slika 10.11).



Slika 10.11: PMEO dijagram baze podataka o nastavi

Do sheme relacione baze podataka može se sada doći primenom pravila za transformaciju entiteta i odnosa u relacije. Tako, samostalni entiteti **predavač**, **predmet**, **smer** i **sala**, predstavljaju se relacijama sa sopstvenim atributima.

Agregirani entitet **predmet-na-smeru** takoďe se predstavlja posebnom relacijom, sa atributima – primarnim ključevima komponentnih entiteta **predmet** i **smer**, i sopstvenim atributima.

Dalje, kako entitet "vreme" ima samo jedan atribut, ne predstavlja se posebnom relacijom, već ulazi u relaciju kojom se predstavlja odnos (agregirani entitet) "održavanje". S druge strane, odnos "održavanje" sadrži sve atribute agregiranog entiteta "predavanje" (jer su svi u primarnom ključu), pa se ni agregirani entitet "predavanje" ne predstavlja relacijom.

Kako je agregirani entitet "održavanje" u odnosu (1,1) prema entitetu "sala", primarni ključ entiteta "sala" ulazi kao atribut u relaciju koja predstavlja entitet "održavanje", a odnos "mesto" ne predstavlja se posebnom relacijom.

Tako se dolazi do sheme relacione baze podataka:

predmet	(šifra, naziv)
predavač	(predavač#, ime, zvanje, stepen)
smer	(šifra, naziv)
sala	(br.sale, sprat)
predmet_na_smeru	(šifra_predmeta, šifra_smera, godina, fond_časova)
održavanje	(šifra_predmeta, šifra_smera, predavač#, vreme, br.sale).

Može se uočiti da su projektovane sheme relacione baze podataka identične. Nije slučajno što se metodom semantičkog modeliranja došlo takoďe do normalizovanih relacija. To je i najčešće slučaj kada se projektovanje proširenim modelom entiteta i odnosa, odnosno metodom normalnih formi, sprovodi korektno i precizno, prema pravilima odgovarajuće metode.

10.4 Semantički model baze podataka

Semantički model baze podataka (SDM – A Semantic Database Model, [36]) može se smatrati jednim od prvih modela podataka iz nove generacije – generacije objektnih modela ([43]). To je formalizam visokog nivoa za struktuiranje i opisivanje baze podataka, zasnovan na semantici podataka u bazi, koji se izgraďuje nad pojmovima klase, entiteta, atributa klase i atributa entiteta, vezâ meďu klasama i posebno vezâ nadklasa/podklasa meďu klasama.

Osnovni principi organizacije baze podataka u skladu sa semantičkim modelom SDM su sledeći:

1. Baza podataka se sastoji od *entiteta* koji odgovaraju stvarnim objektima koji se obraďuju aplikacijom.

2. Entiteti su organizovani u *klase*, koje predstavljaju skupove entiteta koji nose odreďeno značenje. Struktura i organizacija SDM baze podataka zadaje se SDM *shemom*, kojom se identifikuju klase u bazi podataka. Klasa ima *ime* (ili više sinonimnih imena) i skup *članova* – entiteta, homogene strukture.

3. Klase nisu logički nezavisne, već su međusobno povezane vezama različitih vrsta. Postoje u osnovi dve vrste veza među klasama u SDM: jedna definiše podklase (analogon generalizaciji), a druga podržava grupisanje klasa (analogon agregaciji). Klasa može biti *bazna* ili *ne-bazna*. Bazna klasa je ona koja se definiše nezavisno od drugih klasa u bazi (odgovara joj pojam samostalnog tipa entiteta u prethodnom razmatranju). Ne-bazna klasa je ona koja se definiše u terminima drugih klasa u bazi, i koja nema nezavisno postojanje, već ima pridruženu jednu vezu između klasa. Na primer, klasa koja je podklasa druge klase (u smislu inkluzije pripadnog skupa entiteta), jeste ne-bazna klasa. Veze između klasa predstavljaju blokove za izgradnju složenijih (izvedenih) atributa i složenijih veza među klasama, kao i mogućnost različitih pogleda na iste informacije.

4. Entiteti i klase imaju atribute koji opisuju njihova svojstva i povezuju ih sa drugim entitetima u bazi. Vrednost atributa može biti izvedena iz drugih vrednosti u bazi. Za jednu klasu vezane su dve vrste atributa: *atributi članova klase* i *atributi klase*. Prvi opisuju svojstva pojedinog entiteta (npr. naziv knjige za klasu KNJIGA), dok drugi opisuju svojstva klase kao celine (npr. broj entiteta klase KNJIGA). Svaka bazna klasa poseduje listu grupâ atributa članova od kojih svaka jednoznačno identifikuje član klase (analogoni kandidata za ključeve u relacionom modelu).

Semantički model SDM, mada poseduje semantičke koncepte objektnog modela podataka (v. dodatak A), u velikoj meri je vezan za osnovne principe relacionog modela. Ova dualnost je osnova jednog od pravaca u kojima se vidi razvoj sistema baza podataka nove generacije ([20]), što daje novu dimenziju značaju SDM-a.

10.5 Pitanja i zadaci

1. Definisati sledeće pojmove:

semantičko modeliranje	podtip/nadtip
asocijativni entitet	generalizacija
agregacija	specijalizacija

2. Kako se u relacionom modelu modeliraju apstrakcije agregacija i generalizacija? Navesti primere.

3. Kojim sredstvima prošireni relacioni model RM/T realizuje koncepte semantičkog modeliranja?

4. Opisati operacije manipulativnog dela RM/T modela.

5. Koja su opšta pravila integriteta RM/T modela?

6. Opisati strukturu RM/T kataloga.

7. Opisati elemente strukturnog dela proširenog modela entiteta i odnosa.

8. Kako se dijagramom entiteta i odnosa predstavljaju regularni entitet, slabi entitet, agregirani entitet, nadtip/podtip? Kakva je kardinalnost preslikavanja koja definiše agregacija odnosno generalizacija/specijalizacija?

9. Kako se u proširenom modelu entiteta i odnosa predstavlja odnos između tri ili više tipova entiteta?

10. Navesti pravila transformacije sheme baze podataka proširenog modela entiteta i odnosa u shemu relacione baze podataka.

11. Nacrtati dijagram entiteta i odnosa PMEO modela baze podataka kompanije iz zadatka 7 glave 9.

12. Neka baza podataka sadrži podatke o trgovačkim predstavnicima, oblastima prodaje i proizvodima. Svaki predstavnik zadužen je za jednu ili više oblasti, i za jedan ili više proizvoda. Za svaku oblast i za svaki proizvod zadužen je po jedan ili više predstavnika. Svaki se proizvod prodaje u svakoj oblasti, ali ni u jednoj oblasti dva ili više predstavnika ne prodaju isti proizvod. Svaki predstavnik prodaje isti skup proizvoda u svakoj oblasti za koju je zadužen. Nacrtati dijagram entiteta i odnosa koji predstavlja shemu opisane baze podataka u PMEO modelu. Prevesti dobijenu shemu u shemu relacione baze podataka.

Deo IV

Fizička organizacija podataka

Deo **IV** odnosi se na fizičku reprezentaciju podataka baze podataka, i na metode pristupa podacima i obrade podataka nad tim reprezentacijama. Fizička reprezentacija podataka je način predstavljanja podataka strukturama podataka u spoljašnjoj memoriji, obično na disku. Strukture podataka (različiti tipovi datoteka), zajedno sa metodama pristupa i obrade podataka, zovu se fizička organizacija podataka.

Osnovna motivacija za izučavanje različitih fizičkih organizacija je činjenica da je vreme pristupa disku mnogo veće (za četiri do pet redova veličine) nego vreme pristupa unutrašnjoj memoriji. Zato izbor fizičke organizacije ima za cilj minimizaciju broja pristupa disku pri radu sa podacima.

Ne postoji optimalna fizička organizacija, jer se različite fizičke organizacije pokazuju manje ili više efikasnim u različitim aplikacijama. Zato sistemi za upravljanje bazama podataka obično podržavaju veći broj fizičkih organizacija, tako da se za razne delove baza podataka mogu koristiti najpogodnije strukture i algoritmi.

Deo **IV** sastoji se od tri glave:

- U glavi 11, **Strukture podataka i algoritmi**, daje se pregled različitih fizičkih organizacija i prikazuju dve najjednostavnije fizičke organizacije – sekvencijalna i indeks-sekvencijalna.
- Glava 12, **Indeksna organizacija**, prikazuje ovu, najčešće korišćenu i često najefikasniju fizičku organizaciju. Razmatra se struktura balansiranog drveta kojom se realizuje indeks, i izlažu se algoritmi pristupa podacima i indeksu.
- Glava 13, **Direktna organizacija**, odnosi se na strukture datoteka kojima se pristupa primenom funkcije direktnog pristupa (heš funkcije), kao i na algoritme obrade takvih struktura.

11

Strukture podataka i pristupne metode

Najniži nivo ANSI/SPARC arhitekture sistema baza podataka je unutrašnji nivo. To je fizička reprezentacija (obično na disku) cele baze podataka, niskog nivoa, koja se sastoji od velikog broja *fizičkih slogova* (ili, kraće, *slogova*[1]) različitih tipova. Fizički slogovi su prema svom tipu, s tačke gledišta SUBP, grupisani u *datoteke* koje imaju jedinstvene identifikatore.

Fizički slog odgovara logičkom slogu koji predstavlja (npr. n-torci specifične relacije), ima jedinstveni identifikator i sastoji se od niza bajtova. Prvih nekoliko bajtova sadrži informaciju o tipu fizičkog sloga, sledećih nekoliko bajtova je rezervisano za dužinu sloga u bajtovima ako je tip sloga promenljive dužine, a zatim slede oznake i vrednosti *fizičkih polja*.

Zadatak SUBP je da zahteve koje korisnik postavlja na konceptualnom nivou nad logičkim tipovima podataka (npr. relacijama i n-torkama) preslika, na unutrašnjem nivou, u zahteve nad datotekama i fizičkim slogovima (preslikavanje skupa relacija u skup datoteka ne mora biti 1–1; v. sledeći odeljak). Zahtevi koje SUBP može da proizvede na unutrašnjem nivou su sledeći:

1. pronaći fizički slog s iz datoteke d;
2. izmeniti fizički slog s u datoteci d;
3. dodati novi fizički slog datoteci d i dobiti njegov identifikator;
4. izbrisati fizički slog s iz datoteke d;
5. kreirati novu datoteku d;
6. ukloniti datoteku d.

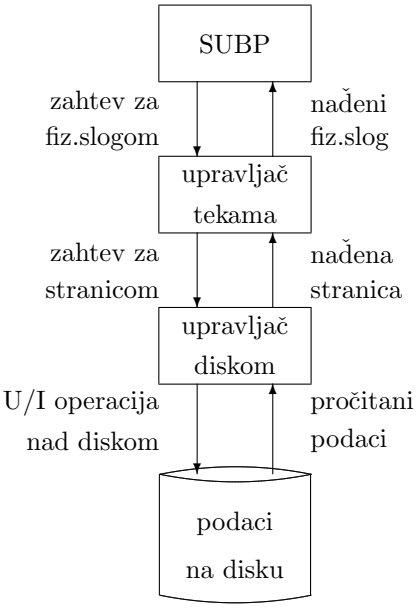
[1] Terminologija, kao i rešenja pojedinih problema u ovoj oblasti bitno se razlikuju od sistema do sistema, ali principi fizičke reprezentacije i pristupa podacima uglavnom su standardni.

Da bi se ovi zahtevi izvršili, SUBP u svojstvu korisnika operativnog sistema koristi funkcionalnost komponente operativnog sistema – *upravljača tekama* (engl. file manager), a upravljač tekama svoju funkcionalnost izgrađuje koristeći funkcije komponente operativnog sistema nižeg nivoa – *upravljača diskom*.

Opišimo, ukratko, ove dve komponente operativnog sistema.

Upravljač diskom ima informacije o svim fizičkim adresama na disku, i odgovoran je za sve fizičke ulazno/izlazne operacije sa diskom (slika 11.1, [27]). Za upravljač diskom (kao i za upravljač tekama), struktura fizičkog sloga nije od interesa i on predstavlja samo niz bajtova na specifičnim adresama.

Upravljač tekama "vidi" disk kao skup blokova fiksne veličine koji se zovu *stranice*. Stranice su numerisane tako da linearno uređenje skupa stranica odgovara njihovom fizičkom redosledu. Veličina stranice je obično izražena brojem bajtova, tj. brojem znakova. Stranice su logički grupisane u *grupe stranica* od kojih svaka ima svoj redni broj (tj. identifikator). Stranice (odnosno grupe stranica) su, s tačke gledišta upravljača tekama, dalje grupisane u disjunktne *stranične skupove* (engl. page sets). Svaki stranični skup ima jedinstveni identifikator. Jedan stranični skup može da sadrži fizičke slogove jednog ili raznih tipova, odnosno može da uključi i fizičke slogove koji, s tačke gledišta SUBP, pripadaju različitim datotekama.



Slika 11.1: SUBP, upravljač tekama i upravljač diskom

Preslikavanje stranice u fizičke adrese na disku koje odgovaraju toj stranici, ostvaruje upravljač diskom (slika 11.1). Zato je upravljač tekama u mogućnosti da postavi (a upravljač diskom da izvrši) sledeće vrste zahteva:

1. pretražiti stranicu p iz straničnog skupa st;
2. izmeniti stranicu p u straničnom skupu st;
3. dodati novu stranicu straničnom skupu st i dobiti broj te stranice, p;
4. ukloniti stranicu p iz straničnog skupa st.

Preslikavanje datoteke u stranične skupove ostvaruje upravljač tekama. Tako se zahtevi koji se odnose na fizički slog, a koje SUBP postavlja na unutrašnjem nivou ANSI arhitekture, realizuju pozivom upravljača tekama. On određuje broj stranice koja sadrži traženi fizički slog i poziva upravljač diskom koji realizuje ulazno/izlazne operacije nad podacima koji odgovaraju traženoj stranici (slika 11.1).

Mada ne mora da učestvuje u upravljanju stranicama, SUBP mora da "zna" za njihovo postojanje, da bi izborom odgovarajuće fizičke reprezentacije povećao efikasnost pristupa podacima (v. sledeći odeljak).

Odnos između načinâ na koje upravljač tekama i SUBP "vide" podatke na disku ilustrovan je slikom 11.2. Stranice sa brojem 1 i 3 (sa ove slike) pripadaju jednom straničnom skupu, a stranice sa brojem 2, 4 i sve ostale stranice – drugom straničnom skupu, npr. skupu slobodnih stranica. Oznake "f.k1" – "f.k5" odnosno "f.i1" – "f.i10" označavaju fizičke slogove kojima su predstavljene na disku n-torke relacije K sa vrednostima k1 – k5 atributa K_SIF, odnosno n-torke relacije I sa vrednostima i1 – i10 atributa I_SIF. Stranice diska (kako ih vidi upravljač tekama) obeležene su brojevima 1, 2, 3, 4, itd. Isti stranični skup na unutrašnjem nivou SUBP vidi se kao dve datoteke, a na konceptualnom nivou – kao dve relacije.

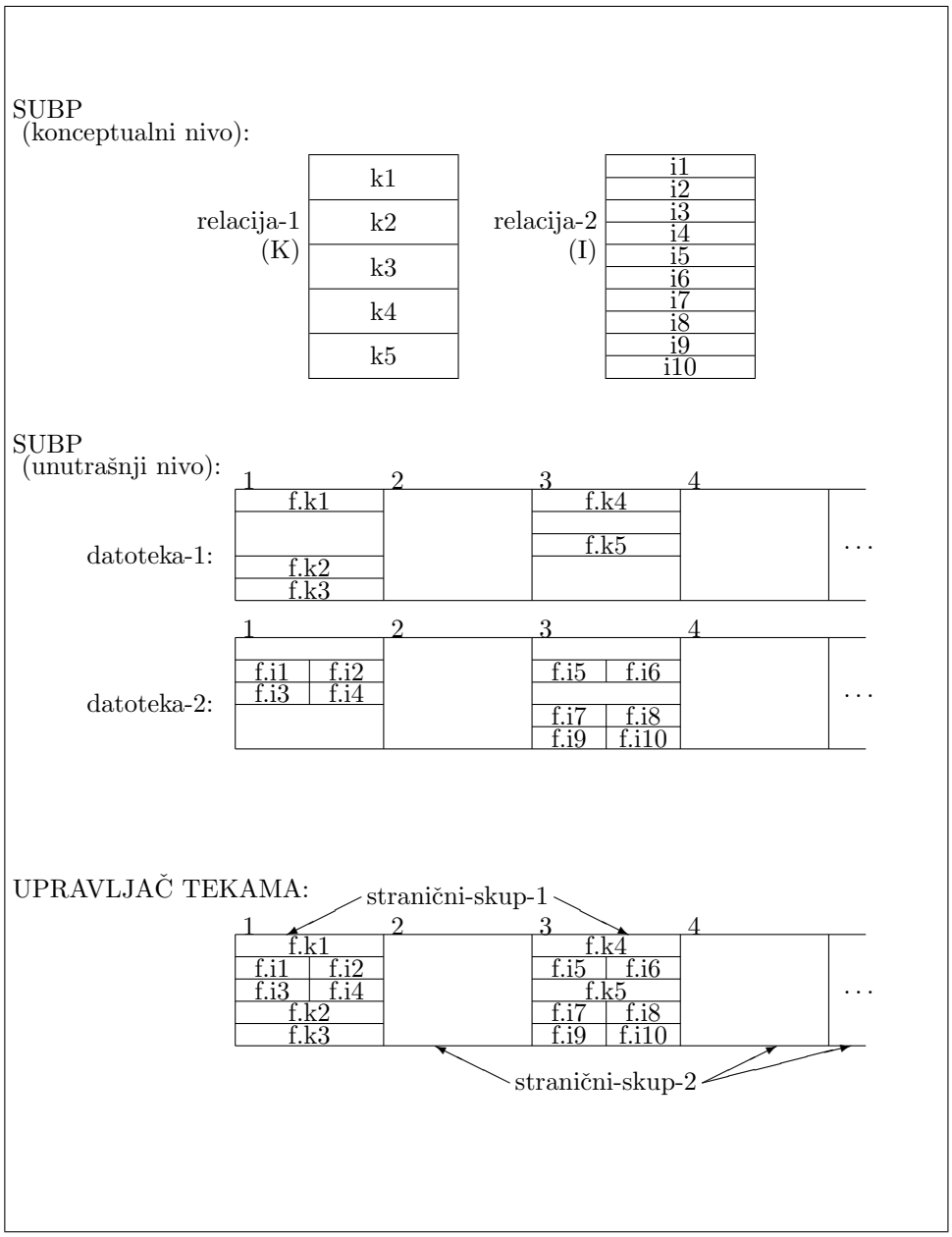
Mada upravljač tekama postoji kao komponenta operativnog sistema, veći broj SUBP izgrađuje svoj sopstveni upravljač tekama da bi povećao efikasnost pristupa podacima.

Pored fizičke reprezentacije podataka, fizička organizacija podrazumeva i algoritme obrade podataka. Zato će u ovoj glavi, osim elemenata fizičke reprezentacije, biti skicirani i odgovarajući algoritmi obrade.

11.1 Fizička reprezentacija relacije

Jedna relacija može imati fizičku reprezentaciju koja se sastoji od većeg broja datoteka (s tačke gledišta SUBP), tj. od većeg broja straničnih skupova (s tačke gledišta operativnog sistema). U tekstu koji sledi biće razmatrana samo fizička reprezentacija s tačke gledišta SUBP koji uključuje funkcije upravljača tekama.

Fizička reprezentacija jedne relacije uključuje, obavezno, *datoteku podataka*, u oznaci DP, koja sadrži fizičke slogove koji odgovaraju podacima koji se "vide" na



Slika 11.2: Razlike u predstavljanju podataka: SUBP i upravljač tekama

logičkom nivou (npr. n-torkama relacije). Međutim, da bi se postigla što veća efikasnost obrade podataka u datoteci podataka, u zavisnosti od prirode podataka i vrste obrade, fizička reprezentacija može da uključi i druge datoteke – *datoteku promena*, *datoteku prekoračenja*, u koje može da se "širi" datoteka podataka, i datoteke koje ubrzavaju pristup podacima u datoteci podataka i koje se zovu *indeksne datoteke*. Tako se, uz odgovarajuće algoritme obrade podataka, dolazi do različitih fizičkih organizacija podataka – sekvencijalne, indeks-sekvencijalne, indeksne, direktne.

Efikasnost pojedine fizičke organizacije podataka izražava se vremenom obrade traženog fizičkog sloga, a to vreme uključuje vreme pristupa grupi stranica, vreme pristupa stranici unutar grupe (koja sadrži traženi slog), vreme donošenja odabrane stranice sa diska u unutrašnju memoriju odnosno vreme upisa stranice iz unutrašnje memorije na odabranu poziciju, kao i vreme obrade sloga u unutrašnjoj memoriji. Vreme pristupa grupi stranica i vreme pristupa stranici unutar grupe zavisi od karakteristika uređaja (brzina pomeranja glava diska), i neće se uzimati u obzir pri daljem razmatranju. Zato će se vreme čitanja, odnosno upisa stranice smatrati jednakim za sve stranice na disku. Takođe, vreme obrade stranice u unutrašnjoj memoriji je za nekoliko redova veličine manje od vremena čitanja, odnosno upisa stranice na disk, pa se ni ono neće uzimati u obzir. Tako će jedina mera efikasnosti pojedine fizičke organizacije biti broj stranica koje je potrebno upisati, odnosno pročitati sa diska, u realizaciji upita pri toj fizičkoj organizaciji.

Razmotrimo sledeći primer. Neka datoteka podataka sadrži n slogova grupisanih u stranice, od kojih svaka može da primi po m slogova. Dakle, datoteka podataka sadrži (logički, jer stranični skup koji joj odgovara može da sadrži i slogove drugog tipa) n/m stranica. Neka je vreme čitanja, odnosno upisa pojedine stranice jednako t. Pretpostavimo da je potrebno obraditi sve slogove iz te datoteke podataka. Ako se slogovima pristupa serijski (redom, kako su upisani), za obradu svih slogova potrebno je vreme $(n/m) * t$, tj. vreme obrade pojedinačnog sloga pri ovakvom pristupu je $((n/m) * t)/n = t/m$. Ako postoji mogućnost direktnog pristupa slogu (po nekom svojstvu tog sloga), i ako slogovi nisu grupisani po tom svojstvu, realna je pretpostavka da se dva sloga koja se obrađuju jedan za drugim nalaze na različitim stranicama. Tada je za obradu cele datoteke podataka potrebno vreme $n * t$, tj. vreme obrade pojedinačnog sloga pri ovakvom pristupu je $n * t/n = t$. Ovo razmatranje dovodi do zaključka da, kada je potrebno obraditi veliki broj slogova iz datoteke podataka, vreme obrade po slogu, pa i celokupno vreme obrade svih slogova daleko je manje u slučaju da se slogovima pristupa sekvencijalno – u fizičkom redosledu stranica, nego u slučaju da je pristup stranicama slobodan ("preko reda").

Indeksna datoteka, u oznaci ID, sa logičke tačke gledišta odgovara funkciji koja za datu vrednost iz domena skupa atributa određuje skup slogova u datoteci podataka DP sa datom vrednošću tih atributa (ili neki nadskup tog skupa slogova). Dakle, za $A \subseteq \{A_1, A_2, \ldots, A_n\}$ indeksna datoteka ID predstavlja logički sledeće preslikavanje:

$$\mathrm{ID} : dom(\mathrm{A}) \rightarrow \mathcal{P}(\mathrm{DP}).$$

($\mathcal{P}$ je oznaka za partitivni skup).

Indeksna datoteka ID može predstavljati indeks kod indeks-sekvencijalne reprezentacije ili indeks kod indeksne reprezentacije. U prvom slučaju, indeksna datoteka ID, tj. indeks izgrađuje se nad primarnim ključem relacije i predstavlja (logički) tabelu koja sadrži uređene parove oblika

(redni-broj-stranice, maksimalna-vrednost-ključa-u-stranici).

U drugom slučaju indeks ID može se izgraditi nad bilo kojim atributom relacije (ne samo nad primarnim ključem) ili nad kombinacijom atributa. Logički, on predstavlja tabelu u kojoj je određenoj kombinaciji vrednosti indeksnih atributa pridružen skup rednih brojeva stranica u kojima se nalaze slogovi sa tom kombinacijom vrednosti indeksnih atributa. Dakle, indeks je u ovom slučaju (logički) tabela koja sadrži uređene parove oblika

(vrednost-indeksnih-atributa, {r-br-stranice$_1$, ..., r-br-stranice$_k$}).

Umesto indeksne datoteke, za pristup slogovima datoteke podataka može se koristiti i algoritam direktnog pristupa –"heš" algoritam (engl. hash), koji transformiše vrednost nekog atributa (atribut direktnog pristupa, heš atribut) u redni broj stranice sloga sa datom vrednošću tog atributa. Algoritam direktnog pristupa zadaje se analitički; to je funkcija f koja preslikava domen atributa direktnog pristupa (ili proizvod domenâ skupa atributa direktnog pristupa) u datoteku podataka, tj. u odgovarajući stranični skup.

Ova glava posvećena je razmatranju jednostavnih fizičkih organizacija koje se izgrađuju nad različitim fizičkim reprezentacijama podataka, kao i metodâ pristupa i algoritama obrade. Naredne dve glave biće posvećene sofisticiranijim i efikasnijim fizičkim organizacijama.

Različiti algoritmi obrade datoteke podataka, pri različitim fizičkim organizacijama, podrazumevaju sortiranost datoteke podataka po nekom atributu (*atribut sortiranja*). S obzirom da je to datoteka na disku, za njeno sortiranje primenjuje se neki od algoritama spoljašnjeg sortiranja. U sledećem odeljku biće opisan jedan jednostavan ali dosta efikasan algoritam – algoritam spoljašnjeg sortiranja balansiranim spajanjem ([55], [1]).

11.2 Sortiranje sekvencijalne datoteke

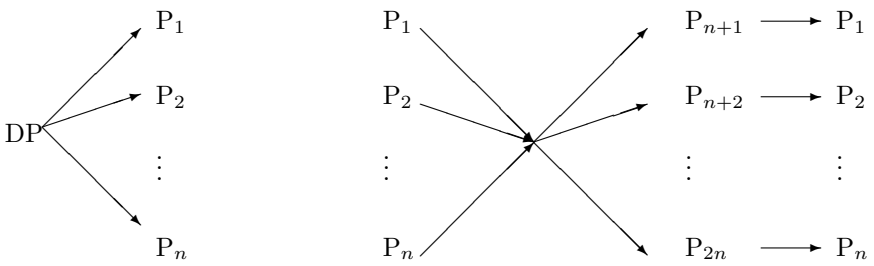
Sekvencijalna datoteka je datoteka u koju se slogovi upisuju redom, jedan za drugim (ne može se upisati slog između dva već upisana sloga), i u kojoj se slogovima

pristupa redom kojim su upisani (jednom slogu se može pristupiti samo tako što se prethodno pristupi svim slogovima, od početka datoteke, koji mu prethode).

Algoritam sortiranja balansiranim spajanjem polazi od neuređene (nesortirane) sekvencijalne datoteke podataka DP i njenog atributa sortiranja A (ili kombinacije atributa), i sastoji se od unutrašnjeg sortiranja delova datoteke DP i spoljašnjeg spajanja tih sortiranih delova. Algoritam koristi dva skupa sekvencijalnih pomoćnih datoteka, P_1, P_2, ..., P_n i P_{n+1}, P_{n+2}, ..., P_{2n}. Pretpostavlja se da je u unutrašnjoj memoriji, s obzirom na njen kapacitet i veličinu slogova, moguće sortirati po m slogova.

Prvi korak u radu algoritma zove se *početna distribucija sekvenci*. Sastoji se od čitanja iz datoteke DP, redom, po m slogova u unutrašnju memoriju, gde se sortiraju po atributu A nekom od metoda unutrašnjeg sortiranja, i od upisa tako sortiranih nizova slogova – *sekvenci* dužine m, kružno, u prvi skup pomoćnih datoteka P_1, P_2, ..., P_n. Naredni koraci algoritma predstavljaju ponavljanje sledećeg postupka: odgovarajuće (prve, druge, itd.) sekvence iz pomoćnih datoteka prvog skupa se spajaju i dobijene sekvence se kružno distribuiraju u pomoćne datoteke drugog skupa; zatim se sadržaj drugog skupa pomoćnih datoteka prepiše u prvi. Postupak se zove *spajanje sa distribucijom sekvenci* i ponavlja se sve dok se ne dobije jedna jedina sortirana sekvenca u pomoćnoj datoteci P_1 (ostale su prazne).

Grafički se koraci algoritma mogu predstaviti kao na slici 11.3



a) početna distribucija b) spajanje, distribucija i prepisivanje

Slika 11.3: Spoljašnje sortiranje balansiranim spajanjem

Na primer, pretpostavimo da sekvencijalna datoteka DP sadrži slogove sa vrednostima atributa sortiranja 1, 5, 31, 7, 9, 3, 13, 6, 17, 8, 25, 27, 2, 10, da unutrašnja memorija može da sortira tri sloga, i da je na raspolaganju šest pomoćnih sekvencijalnih datoteka. Tada se postupak sortiranja datoteke DP odvija kroz sledeće faze (u datotekama DP, P_1, ..., P_6 označavaju se samo vrednosti atributa sortiranja, ali se u njih u stvari upisuju celi slogovi sa tim vrednostima atributa sortiranja):

početna distribucija:

P_1: 1, 5, 31, 8, 25, 27
P_2: 3, 7, 9, 2, 10
P_3: 6, 13, 17

spajanje i distribucija:

P_4 : 1, 3, 5, 6, 7, 9, 13, 17, 31
P_5 : 2, 8, 10, 25, 27
P_6 : –

prepisivanje drugog u prvi pomoćni skup:

P_1 : 1, 3, 5, 6, 7, 9, 13, 17, 31
P_2 : 2, 8, 10, 25, 27
P_3 : –

spajanje i distribucija:

P_4 : 1, 2, 3, 5, 6, 7, 8, 9, 10, 13, 17, 25, 27, 31
P_5 : –
P_6 : –

prepisivanje drugog u prvi pomoćni skup:

P_1 : 1, 2, 3, 5, 6, 7, 8, 9, 10, 13, 17, 25, 27, 31
P_2 : –
P_3 : –

DP := P_1.

Dakle, posle dve primene spajanja sa distribucijom, datoteka DP postaje sortirana.

Globalni algoritam sortiranja sekvencijalne datoteke balansiranim spajanjem može se opisati sledećim pseudoprogramom:

Algoritam sortiranja balansiranim spajanjem

ulaz: sekvencijalna datoteka DP, atribut sortiranja A;
izlaz: sortirana datoteka DP;
pomoćne strukture: sekvencijalne datoteke $P_1, \ldots, P_{2n}$;

```
BEGIN
    izvršiti početnu distribuciju sekvenci iz DP u P_1, ..., P_n;
    REPEAT
        izvršiti spajanje sekvenci iz datoteka P_1, ..., P_n
```

```
            i njihovu distribuciju u datoteke P_{n+1}, ..., P_{2n};
            (algoritmom spajanja sa distribucijom sekvenci)
        prepisati datoteke {P_{n+1}, ..., P_{2n}} u {P_1, ..., P_n};
        isprazniti datoteke {P_{n+1}, ..., P_{2n}}
    UNTIL u P_1 samo jedna sekvenca, a ostale pomoćne datoteke prazne;
    prepisati sadržaj datoteke P_1 u datoteku DP
END.
```

Globalni algoritmi početne distribucije, odnosno spajanja sa distribucijom, koji se koriste u prethodnom algoritmu, mogu se opisati sledećim pseudoprogramima ([55], [1]):

Algoritam početne distribucije sekvenci

ulaz: sekvencijalna datoteka DP, atribut sortiranja A i "kapacitet" memorije m;
izlaz: sekvencijalne datoteke $P_1, \ldots, P_n$ koje sadrže sekvence datoteke DP;

```
BEGIN
    pripremiti datoteku DP za čitanje i datoteke P_1, ...,P_n za upis;
    i := 1; /*indeks pomoćne datoteke*/
    REPEAT
        učitati sledeći niz od m slogova iz datoteke DP u unutrašnju memoriju
            (ili k < m ako je ostalo samo k),
            sortirati ga i sortiranu sekvencu upisati u datoteku P_i;
        i := (i mod n) + 1
    UNTIL kraj datoteke DP
END.
```

Algoritam spajanja sa distribucijom sekvenci

ulaz: niz sekvencijalnih datoteka $P_1, \ldots, P_n$, atribut sortiranja A;
izlaz: niz sekvencijalnih datoteka $P_{n+1}, \ldots, P_{2n}$ koje sadrže spojene nizove (od po n) sekvenci ulaznog niza datoteka;

```
BEGIN
    broj sekvenci u izlaznom nizu datoteka je 0;
    otvoriti ulazne datoteke P_1, ..., P_n za čitanje ;
    otvoriti izlazne datoteke P_{n+1}, ..., P_{2n} za upis;
    postaviti indeks izlazne datoteke, i, na n + 1;
    REPEAT
        spojiti sledeće sekvence iz P_1, ..., P_n (po jednu iz svake datoteke do
                    čijeg se kraja još nije došlo)
            i upisati rezultujuću sekvencu u izlaznu datoteku P_i;
        broj sekvenci u izlaznom nizu datoteka uvećati za 1;
        indeks sledeće izlazne datoteke, i, postaviti na (i mod n) + n + 1
    UNTIL kraj svih ulaznih datoteka P_1, ..., P_n
END.
```

Spajanje niza od n sekvenci, koje se koristi u algoritmu spajanja sa distribucijom sekvenci ("*spojiti*"), može se izvršiti jednim čitanjem sekvenci, i to prema algoritmu sekvencijalnog spajanja sledećeg (globalnog) oblika:

Algoritam sekvencijalnog spajanja

ulaz: n sekvenci $s_1, \ldots, s_n$;
izlaz: sekvenca s koja se sastoji od slogova sekvenci $s_1, \ldots, s_n$;
pomoćne strukture: niz $tekući_i, i = 1, \ldots, n$, slogova, po jedan za svaku ulaznu sekvencu; $tekući_i$ sadrži tekući slog sekvence s_i ako nije kraj te sekvence, inače sadrži indikator kraja;

```
BEGIN
    b := 0; /* broj obrađenih (pročitanih i spojenih) sekvenci ulaznog niza */
    FOR svaku sekvencu s_i, i = 1,...,n, DO
        IF sekvenca s_i prazna
        THEN tekući_i := prvi slog sekvence s_i
        ELSE
            BEGIN
                tekući_i := indikator kraja;
                b := b +1
            END;
    WHILE b < n − 1 DO /* dok su bar dve sekvence neobrađene*/
        BEGIN
            odrediti 1 ≤ k ≤ n tako da je tekući_k = min {tekući_i, i = 1,...,n};
              /* slogovi se porede po vrednosti atributa sortiranja, */
              /* i to samo oni koji ne sadrže indikator kraja */
            dodati slog tekući_k izlaznoj sekvenci s;
            IF nije kraj sekvence s_k
            THEN tekući_k := sledeći slog sekvence s_k
            ELSE
                BEGIN
                    tekući_k := indikator kraja;
                    b := b +1
                END
        END;
    upisati u s sve neupisane slogove neobrađene sekvence (ako postoji)
END.
```

Vreme potrebno za sortiranje balansiranim spajanjem sekvencijalne datoteke DP sa n slogova je proporcionalno sa n^2 ([55]). Postoje poboljšanja ovog algoritma koja se pre svega odnose na broj potrebnih pomoćnih datoteka, ali se složenost algoritma ne može bitno popraviti.

11.3 Sekvencijalna organizacija

Sekvencijalna organizacija podataka karakteriše se isključivo sekvencijalnim (rednim) pristupom slogovima u datoteci podataka. Ova organizacija ima dve varijante

– neuređenu, koja se zove i "gomila" (engl. *heap*), i uređenu varijantu. U prvoj varijanti fizička reprezentacija relacije uključuje samo jednu datoteku – datoteku podataka (DP), u koju se slogovi unose hronološkim redom – u redosledu kojim stižu. U drugoj varijanti, slogovi datoteke podataka su sortirani u redosledu nekog atributa sortiranja. Slogovi koji se unose posle početnog sortiranja unose se u drugu datoteku – tzv. datoteku promena (DPR), i to hronološki, kao i u prvoj varijanti. Slogovi koje treba brisati ne izbacuju se (ni u jednoj varijanti), već se obeležavaju nevažećim, sve do povremene reorganizacije kada se fizički izbacuju svi obeleženi slogovi. Pri reorganizaciji se, u drugoj varijanti, i objedinjuju i sortiraju datoteka podataka i datoteka promena.

11.3.1 Efikasnost izvršavanja jednorelacionih upita

Pretraživanje. Izbor pojedinačnog sloga iz datoteke DP (DPR) sa n slogova, u obe varijante sekvencijalne organizacije, zahteva u proseku pretraživanje pola te datoteke u slučaju uspeha, i cele datoteke u slučaju neuspeha. Ako stranica prima m slogova, onda je vreme pretraživanja pojedinačnog sloga proporcionalno sa $t*n/2m$, tj. vreme je $O(t*n/m)$ – proporcionalno sa brojem stranica u datoteci (t je, kao i ranije, vreme pristupa stranici). Dakle, organizacija je neadekvatna za pretraživanje pojedinačnog sloga.

Primer ovakvog izbora je upit: naći status i državu izdavača sa šifrom i2. Ovaj upit se u SQL-u izražava sledećim SELECT blokom:

```
SELECT STATUS, DRZAVA
FROM   I
WHERE  I_SIF = 'i2'
```

Obrada svih slogova datoteke DP (DPR) u proizvoljnom redosledu, u obe varijante sekvencijalne organizacije, zahteva u proseku samo dva puta duže vreme od uspešne obrade pojedinačnog sloga, pa je vreme obrade svih slogova istog reda veličine, tj. $O(t * n/m)$. Sekvencijalna organizacija je adekvatna ako je grupna obrada u proizvoljnom redosledu najčešća obrada podataka.

Primer grupne obrade je upit: prikazati naziv i status svih izdavača. U SQL-u,

```
SELECT NAZIV, STATUS
FROM   I
```

U ovom primeru obrada je pretraživanje (obrada može biti i ažuriranje, unošenje, brisanje), i uključuje samo operaciju projekcije. Zato ovaj upit predstavlja specijalni slučaj opštijeg jednorelacionog upita grupnog pretraživanja koji može da uključi i restrikciju. Algoritam po kome se izvršava grupna sekvencijalna restrikcija i projekcija uključuje sekvencijalni prolaz kroz sve slogove datoteke DP (DPR) i, za svaki slog koji zadovoljava uslov restrikcije, projektovanje na atribute projekcije.

Primer upita grupne obrade sa restrikcijom je: naći nazive jugoslovenskih izdavača čiji status nije manji od 20. Ovaj upit u SQL-u ima oblik:

```
SELECT NAZIV
FROM   I
WHERE  STATUS >= 20 AND DRZAVA = 'Jugoslavija'
```

Grupna obrada slogova u zadatom redosledu zadatog atributa zahteva sortiranje DP datoteke u neuređenoj varijanti, odnosno DP i DPR datoteka (zajedno), u uređenoj varijanti, i to u zadatom redosledu zadatog atributa. Uređena varijanta ima značajne prednosti u slučaju da je datoteka podataka već sortirana po atributu koji određuje redosled, jer se onda zahteva samo sortiranje datoteke promenâ i sekvencijalno spajanje dve sortirane datoteke (v. prethodni odeljak).

Na primer, upit "prikazati naziv i status svih izdavača u abecednom poretku naziva" predstavlja grupnu obradu u zadatom redosledu:

```
SELECT NAZIV, STATUS
FROM   I
ORDER  BY NAZIV
```

Za sortiranje DP datoteke, odnosno DP i DPR datoteka, koje se odnose na relaciju I, može se koristiti neki od algoritama spoljašnjeg sortiranja (v. odeljak 11.2).

Izbor sloga, tj. operacija pretraživanja, potrebna je pri izvršavanju svake od operacija ažuriranja odnosno brisanja. Zbog toga je efikasnost izvršenja ovih operacija direktno zavisna od efikasnosti izbora (koja je razmotrena u ovom paragrafu). Sledeći paragraf odnosi se na razmatranje upravo tih operacija.

Ažuriranje, brisanje i unošenje. Ažuriranje i brisanje sloga, odnosno grupe slogova analogno je izboru, pa i vreme potrebno za ažuriranje, odnosno brisanje odgovara vremenu potrebnom za izbor. Umesto fizičkog brisanja, slog se obeležava nevažećim, s tim što vreme potrebno za brisanje pojedinačnog sloga uključuje još i vreme periodične reorganizacije (pri kojoj se slogovi *de facto* brišu), izračunato po slogu.

Unošenje novog sloga realizuje se kao hronološko dodavanje sloga datoteci podataka odnosno datoteci promena; u slučaju uređene varijante, vreme unošenja uključuje i vreme povremene reorganizacije, izračunato po pojedinačnom slogu, kojom se datoteka podataka i datoteka promena objedinjuju u novu datoteku podataka, a datoteka promena prazni.

11.3.2 Izvršavanje višerelacionih upita

Svi višerelacioni upiti uključuju neki oblik operacije spajanja, a najčešće i neki oblik restrikcije i projekcije. Zato osnovni algoritam izvršavanja dvorelacionih upita (analogno višerelacionih), za sekvencijalnu fizičku reprezentaciju relacijâ, ima dva globalna koraka:

1. Sortirati obe datoteke podataka (i datoteke promena, ako postoje) po atributima spajanja; u prvom prolazu kroz datoteke pri sortiranju izvršiti restrikcije i projekcije na izlazne atribute i atribute spajanja;
2. Izvršiti sekvencijalno prirodno spajanje sortiranih datoteka, uz istovremenu projekciju na izlazne atribute.

Za relacije koje ne učestvuju u prvom koraku (npr. kod uređene varijante po atributu spajanja), pripadne restrikcije vrše se u drugom koraku, istovremeno sa spajanjem.

Prvi korak je izložen u prethodnom odeljku. Drugi je predstavljen sledećim algoritmom ([57], [1]):

Algoritam sekvencijalnog prirodnog spajanja po atributima sortiranja

ulaz: sortirane datoteke P_1 i P_2;
izlaz: sortirani rezultat operacije prirodnog spajanja datoteka P_1 i P_2;

```
BEGIN
  otvoriti datoteke P1 i P2 za čitanje;
  pročitati prvi slog s1 datoteke P1 i prvi slog s2 datoteke P2;
  REPEAT
    uporediti vrednosti atributa spajanja slogova s1 i s2;
    IF vrednost atributa spajanja sloga s1 manja od
          vrednosti atributa spajanja sloga s2
    THEN
        BEGIN
          čitati slogove iz P1 dok imaju istu vrednost atributa spajanja;
          pročitati sledeći slog s1 datoteke P1
        END
    ELSE IF vrednost atributa spajanja sloga s2 manja od
              vrednosti atributa spajanja sloga s1
          THEN BEGIN
                  čitati slogove iz P2 dok imaju istu vrednost atributa spajanja;
                  pročitati sledeći slog s2 datoteke P2
               END
          ELSE BEGIN
                  (jednaki su) spojiti sve slogove iz datoteka P1, P2
                     sa istom vrednošću atributa spajanja kao s1 (tj. s2);
                  pročitati sledeći slog s1 datoteke P1 i sledeći slog s2 datoteke P2
```

```
            END
  UNTIL kraj jedne od datoteka P1, P2
END.
```

Spajanje dva skupa slogova sa istom vrednošću atributa spajanja (poslednji korak prethodnog algoritma) je spajanje svakog sloga iz prvog skupa sa svakim slogom iz drugog skupa. To je radnja koja se najčešće izvršava bar delimično u unutrašnjoj memoriji, jer bi inače zahtevala vraćanje kroz jedan od skupova, što nije karakteristično za sekvencijalni pristup.

Na primer, u izvršavanju upita

```
SELECT NAZIV, K_SIF
FROM   I, KI
WHERE  I.I_SIF = KI.I_SIF AND TIRAZ > 3000
```

pri čemu su obe relacije (I, KI) fizički organizovane sekvencijalno, prvo bi se sortirala datoteka podataka (i datoteka promena, ako postoji) relacije I, uz projektovanje na atribute I_SIF, NAZIV, odnosno datoteka podataka relacije KI, uz restrikciju "TIRAZ $>$ 3000" i projekciju na atribute K_SIF, I_SIF; zatim bi se rezultati ovih operacija sekvencijalno prirodno spojili uz projekciju na atribute NAZIV, K_SIF.

Osim opisanog osnovnog algoritma spajanja sekvencijalnih datoteka prethodnim sortiranjem, moguće je operaciju spajanja izvršiti i algoritmom "ugnježdenih petlji". Prema ovom algoritmu, jedna sekvencijalna datoteka se čita jedanput, slog po slog. Za svaki slog te datoteke čitaju se, redom, svi slogovi druge sekvencijalne datoteke, svaki se poredi sa slogom prve datoteke i, ako ispunjava uslov spajanja, spaja se sa njim.

Algoritam koji će se primeniti u konkretnom slučaju bira se na osnovu procene cene, uzimajući u obzir učestalost promena sekvencijalnih datoteka. Ako su datoteke "sporopromenljive", tj. ako se veći broj operacija spajanja nad tim datotekama izvrši između dva ažuriranja, cena sortiranja (izračunata po jednom spajanju) je opravdana i prihvatljiva. Ako se pak između svaka dva spajanja sadržaj relacija promeni, sortiranje može biti neopravdano skupa operacija.

Prednost uređene varijante u izvršavanju dvorelacionih (i višerelacionih) upita postoji samo u slučaju da je atribut spajanja baš atribut sortiranja bar jedne od relacija koje učestvuju u tom upitu, jer ta relacija ne prolazi kroz korak sortiranja.

Može se zaključiti da su u slučaju sekvencijalne reprezentacije potrebna česta sortiranja, čak i kada se radi o uređenoj varijanti, s obzirom da je u njoj jedan od atributa privilegovan.

11.4 Indeks-sekvencijalna organizacija

Indeks-sekvencijalna organizacija podrazumeva reprezentaciju podataka koju karakteriše, sa jedne strane, sortiranost datoteke podataka po primarnom ključu i mogućnost sekvencijalnog pristupa slogovima u poretku primarnog ključa, a sa druge strane, postojanje indeksa po primarnom ključu, koji omogućuje brz pristup slogu na osnovu vrednosti primarnog ključa (otuda i naziv – indeks-sekvencijalna). Ove dve bitne karakteristike indeks-sekvencijalne organizacije realizuju se specifičnom strukturom indeksa i algoritmima za održavanje datoteke podataka i indeksne datoteke.

Indeks kod ove reprezentacije zadat je datotekom sa slogovima oblika[2]

(redni-broj-stranice, maksimalna-vrednost-ključa-u-stranici),

i sortiran je po primarnom ključu.

Mada svojstvo sekvencijalnog pristupa i sortiranost po jednom atributu – ovde primarnom ključu, podseća na uređenu varijantu sekvencijalne organizacije, indeks-sekvencijalna organizacija primenjuje drugi algoritam unošenja novih slogova, i time povećava efikasnost pristupa odnosno izbora. Naime, posle početnog unošenja slogova u datoteku podataka (u poretku primarnog ključa), novi slogovi se unose u stranice datoteke podataka, u poretku primarnog ključa, ili (kada je odgovarajuća stranica puna) u posebnu datoteku koja se u ovoj organizaciji zove *zona prekoračenja*, u oznaci ZP. Poslednji slog u stranici datoteke podataka i svi slogovi u zoni prekoračenja koji su tamo uneti zbog prekoračenja kapaciteta te stranice datoteke podataka, povezuju se pokazivačima u poretku primarnog ključa (v. tačku 11.4.1, paragraf Ažuriranje, brisanje i unošenje). Tako se pokazivačima realizuje funkcija

$$\mathrm{DP} \to \mathcal{P}(\mathrm{ZP})$$

Niz slogova povezanih pokazivačima, od kojih je jedan u datoteci podataka a ostali u zoni prekoračenja, zove se *lanac prekoračenja*.

Zbog mogućeg produžavanja lanaca prekoračenja, koje značajno degradira efikasnost pristupa slogu po vrednosti primarnog ključa, periodično se nad ovom reprezentacijom vrši reorganizacija, koja objedinjuje datoteku podataka i zonu prekoračenja u jedinstvenu sortiranu datoteku podataka, a zonu prekoračenja prazni.

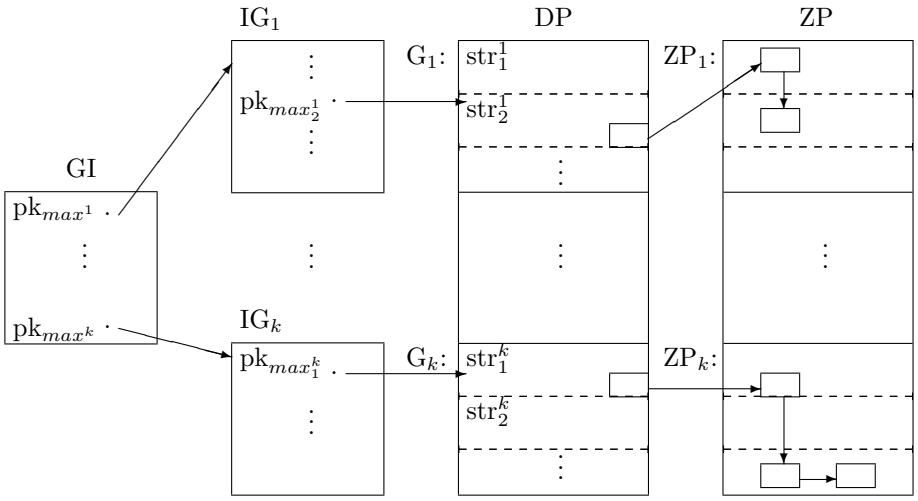
Osim specifične strukture indeksa, za indeks kod indeks-sekvencijalne organizacije je karakteristično da je statički, tj. da se ne menja u periodu između dve reorganizacije. Ovaj indeks tek u procesu reorganizacije postaje potpuno ažuran i samo do prve izmene baze potpuno odražava njeno stanje.

[2]Ovakav indeks zove se i redak indeks (engl. nondense), jer u njemu, za razliku od gustog indeksa (v. sledeću glavu), nisu prisutne sve vrednosti atributa indeksiranja.

Mada indeksna datoteka ima mnogo manje slogova, a slogovi manje polja nego datoteka podataka, i za njeno sekvencijalno pretraživanje potrebno je vreme koje može da postane nezanemarljivo. Zbog toga je poželjno i nad indeksnom datotekom imati indeks koji bi omogućio efikasni pristup slogu (odnosno stranici) u indeksnoj datoteci. Poboljšanje efikasnosti indeks-sekvencijalne reprezentacije postiže se fizičkim grupisanjem delova (stranica) datoteke podataka, zone prekoračenja i indeksne datoteke, pri čemu se deo indeksa u grupi, koji se odnosi na stranice datoteke podataka i zone prekoračenja u grupi, zove *indeks grupe*. Grupe se obično realizuju po cilindrima diska. Pored indeksa grupâ, postoji i glavni indeks (glavna indeksna datoteka) koji sadrži parove oblika

(redni-broj-grupe, maksimalna-vrednost-ključa-u-grupi)

(slika 11.4; GI je oznaka za glavni indeks; IG_1 – IG_k su indeksi grupa G_1 – G_k; DP je datoteka podataka; ZP je zona prekoračenja, a njeni delovi koji pripadaju grupama G_1 – G_k su ZP_1 – ZP_k; redni brojevi grupa odnosno stranica u indeksu su, radi preglednosti, predstavljeni pokazivačima; pk je oznaka za vrednost primarnog ključa).



Slika 11.4: Glavni indeks i indeksi grupâ kod indeks-sekvencijalne reprezentacije

11.4.1 Efikasnost izvršavanja jednorelacionih upita

Pretraživanje. Izbor pojedinačnog sloga u ovoj reprezentaciji efikasna je operacija samo u slučaju izbora po primarnom ključu, inače indeks nije od koristi. U slučaju izbora po primarnom ključu, izbor se sastoji od sledećih radnji:

- pretražiti glavni indeks; naći slog sa najmanjom vrednošću primarnog ključa koja nije manja od vrednosti primarnog ključa koja se traži; iz nađenog sloga pročitati redni broj grupe;
- pretražiti indeks grupe; naći slog sa najmanjom vrednošću primarnog ključa koja nije manja od vrednosti primarnog ključa koja se traži; iz nađenog sloga pročitati redni broj stranice;
- pročitati stranicu iz datoteke podataka i naći traženi slog – ako je na njoj;
- eventualno pročitati jednu ili više stranica iz zone prekoračenja (prateći pokazivače u lancu prekoračenja), dok se ne dođe do traženog sloga ili ustanovi da ga nema.

Od veličine glavnog indeksa i indeksa grupe zavisi broj stranica koje treba pročitati dok se ne dođe do stranice u datoteci podataka. Od dužine lanaca prekoračenja zavisi broj stranica koje treba pročitati od stranice datoteke podataka do stranice u zoni prekoračenja koja sadrži traženi slog. Indeks je sortiran ali se pretražuje sekvencijalno, pa vreme pretraživanja indeksa (uspešnog ili neuspešnog) jeste $\frac{1}{2} * t * Bs$, gde je Bs broj stranica indeksa (glavnog odnosno indeksa grupe), a t – vreme čitanja (odnosno upisa) proizvoljne stranice na disku.

Obrada svih slogova (ili velike grupe njih) u proizvoljnom redosledu ne zahteva upotrebu indeksa i odvija se po odgovarajućem algoritmu grupne obrade sekvencijalne datoteke.

Obrada svih slogova u redosledu primarnog ključa je efikasna (bez upotrebe indeksa), dok se u slučaju redosleda po bilo kom drugom atributu zahteva sortiranje datoteke podataka i zone prekoračenja u redosledu tog atributa. Obrada svih slogova u redosledu primarnog ključa je efikasnija nego kod uređene varijante sekvencijalne reprezentacije, jer su i slogovi u zoni prekoračenja povezani pokazivačima sa odgovarajućim slogovima iz datoteke podataka.

Posebno karakterističan za ovu reprezentaciju je algoritam restrikcije tipa intervala, koji predstavlja puni izraz efikasnosti ove reprezentacije, s obzirom da koristi oba njena koncepta sadržana u nazivu: indeks i sekvencijalnost. Ovaj algoritam je kombinacija izbora pojedinačnog (tj. prvog iz intervala) sloga uz pomoć indeksa, i sekvencijalnog pretraživanja datoteke podataka i zone prekoračenja po algoritmu grupne obrade. Da bi se izbor pojedinačnog sloga mogao izvršiti pomoću indeksa, potrebno je da uslov restrikcije bude zadat nad primarnim ključem relacije.

Primer upita tipa intervalne restrikcije je: izdati nazive i statuse jugoslovenskih izdavača sa šifrom od i500 do i1000. U SQL-u upit se može izraziti SELECT upitnim blokom

```
SELECT NAZIV, STATUS
FROM  I
WHERE I_SIF >= 'i500' AND I_SIF <= 'i1000' AND DRZAVA  = 'Jugoslavija'
```

Restrikcija tipa intervala, uz projektovanje na izlazne atribute, kod indeks-sekvencijalne reprezentacije sastoji se iz dva koraka. Prvi je pronalaženje (po indeksu) sloga sa najmanjom vrednošću primarnog ključa koja nije manja od donje granice intervala. Drugi je sekvencijalna obrada svih slogova sa vrednošću primarnog ključa koja je manja ili jednaka gornjoj granici intervala; obrada sloga uključuje proveru ostalih uslova restrikcije i, ukoliko su ispunjeni, projekciju na izlazne atribute.

Ažuriranje, brisanje i unošenje. Operacija brisanja se sastoji od izbora i obeležavanja sloga nevažećim, jer se slog, zbog indeksa i lanaca prekoračenja, fizički ne izbacuje.

Novi slog se unosi na odgovarajuće mesto u sortirani niz slogova, u stranicu datoteke podataka ili u lanac prekoračenja zone prekoračenja. Zbog unošenja sloga u punu stranicu datoteke podataka može se dogoditi da slog sa prethodno najvećom vrednošću primarnog ključa (koja je i u indeksu pridružena stranici) bude premešten iz stranice datoteke podataka u zonu prekoračenja.

Operacija unošenja sastoji se od sledećih radnji:

- pronaći stranicu datoteke podataka u koju treba uneti slog (pretraživanjem glavnog indeksa i indeksa grupe, kao kod operacije pretraživanja);
- AKO je vrednost primarnog ključa sloga koji se unosi veća od najveće vrednosti primarnog ključa u stranici, ONDA
  1. upisati slog u prvu nepopunjenu stranicu zone prekoračenja;
  2. pretražiti niz stranica zone prekoračenja iste grupe, preko lanca prekoračenja, radi lociranja sloga – prethodnika po vrednosti primarnog ključa;
  3. u novi slog postaviti pokazivač na slog na koji pokazuje locirani slog – prethodnik;
  4. u slog – prethodnik postaviti pokazivač na novi upisani slog;
- INAČE
  1. sortirati slogove stranice datoteke podataka zajedno sa slogom koji se unosi;
  2. upisati sortirani sadržaj stranice u datoteku podataka;
  3. AKO je došlo do prekoračenja kapaciteta stranice, ONDA
     – uneti slog koji je izazvao prekoračenje u prvu nepopunjenu stranicu zone prekoračenja iste grupe;
     – povezati ga u lanac prekoračenja, između poslednjeg sloga u stranici datoteke podataka i sloga na koji je prethodno pokazivao poslednji slog u stranici datoteke podataka.

Vrednosti atributa koji nisu u primarnom ključu ažuriraju se izborom sloga i izmenom vrednosti u samom slogu u datoteci podataka ili zoni prekoračenja. Vrednost primarnog ključa se ažurira kao dvoetapna operacija brisanja sloga koji se ažurira (tj. njegovog obeležavanja nevažećim) i unošenja novog sloga u zonu prekoračenja, uz uvezivanje u odgovarajući lanac prekoračenja.

Kada se "prepuni" zona prekoračenja ili kada se zbog obeleženih a neizbačenih slogova, ili predugih lanaca prekoračenja, suviše degradira efikasnost organizacije, vrši se reorganizacija. Ona se odvija bez korišćenja indeksa, a sortirana (DP, ZP) datoteka postaje nova DP datoteka. Kreira se nova indeksna datoteka, a zona prekoračenja postaje prazna.

11.4.2 Izvršavanje višerelacionih upita

Ključna operacija kod algoritama izvršavanja višerelacionih upita je operacija spajanja, pa je i efikasnost izvršavanja upita zavisna od efikasnosti izvršavanja ove operacije.

Kada se relacije spajaju po primarnim ključevima relacijâ u indeks-sekvencijalnoj reprezentaciji, operacija se izvodi po algoritmu sekvencijalnog prirodnog spajanja po atributima sortiranja, kao kod uređene varijante sekvencijalne datoteke (v. odeljak 11.3)

U slučaju da je u jednoj relaciji sa indeks-sekvencijalnom reprezentacijom atribut spajanja primarni ključ a da u drugoj nije, relacije se spajaju prema algoritmu *indeksnog spajanja* koji će biti izložen u okviru glave o indeksnoj organizaciji (v. odeljak 12.2)

Ako se spajanje relacija (fizički organizovanih indeks-sekvencijalno) vrši po atributima koji nisu primarni ključ ni u jednoj relaciji, indeksi nisu od koristi, pa se spajanje izvodi kao kod neuređene varijante sekvencijalne organizacije.

11.5 Pitanja i zadaci

1. Definisati sledeće pojmove:

unutrašnji nivo ANSI arhitekture	spoljašnje sortiranje
fizički slog	fizička reprezentacija podataka
datoteka	fizička organizacija podataka
stranični skup	sekvencijalna organizacija
stranica	indeks-sekvencijalna organizacija

2. Opisati softverske komponente koje učestvuju u pristupu podacima u bazi podataka.
3. Proceniti vremensku složenost algoritama koji ulaze u sastav algoritma spoljašnjeg sortiranja balansiranim spajanjem.
4. Implementirati u programskom jeziku algoritam spoljašnjeg sortiranja balansiranim spajanjem.
5. Kako se izvršavaju jednorelacioni upiti nad sekvencijalnom reprezentacijom?
6. Opisati algoritam sekvencijalnog prirodnog spajanja po atributima sortiranja i proceniti njegovu vremensku složenost.
7. Koje su glavne karakteristike indeks-sekvencijalne reprezentacije?
8. Opisati postupak pretraživanja kod indeks-sekvencijalne reprezentacije.

9. Opisati postupak unošenja novog sloga kod indeks-sekvencijalne reprezentacije.

10. Implementirati u programskom jeziku algoritam unošenja kod indeks-sekvencijalne reprezentacije.

11. Sekvencijalna reprezentacija je često zadovoljavajuća fizička reprezentacija podataka u bazi podataka. Za koju vrstu upita su algoritmi obrade sekvencijalne reprezentacije efikasniji od algoritama obrade indeks-sekvencijalne reprezentacije?

12. Za koje vrste upita je indeks-sekvencijalna organizacija efikasnija od sekvencijalne? Navesti primere.

13. Indeks kod indeks-sekvencijalne reprezentacije moguće je izgraditi i kao uređen skup parova oblika

    (redni-broj-stranice, minimalna-vrednost-ključa-u-stranici).

    Kako bi se u tom slučaju modifikovali algoritmi pretraživanja i unošenja nad ovom reprezentacijom?

12

Indeksna organizacija

Indeksna organizacija dobila je ime po pristupnoj datoteci u sastavu fizičke reprezentacije – indeksu. Nad jednom relacijom (na fizičkom nivou – nad datotekom podataka) može da postoji jedan ili više indeksa. Indeksi imaju specifičnu strukturu i karakteristike u odnosu na indeks kod indeks-sekvencijalne reprezentacije. Osnovna razlika između ove dve vrste indeksa je u tome što su u ovom indeksu prisutne sve vrednosti atributa nad kojim se gradi indeks.[1] Takođe, za razliku od statičkog indeksa kod indeks-sekvencijalne reprezentacije, indeks kod indeksne reprezentacije je dinamički i neposredno odražava sve promene nad datotekom podataka.

Logički, indeks I nad atributom (ili skupom atributa) A relacije R u indeksnoj reprezentaciji ostvaruje preslikavanje

$$dom(\mathrm{A}) \rightarrow \mathcal{P}(R)$$

(fizička organizacija indeksa biće opisana u odeljku 12.3). Atribut A zove se *atribut indeksiranja*.[2]

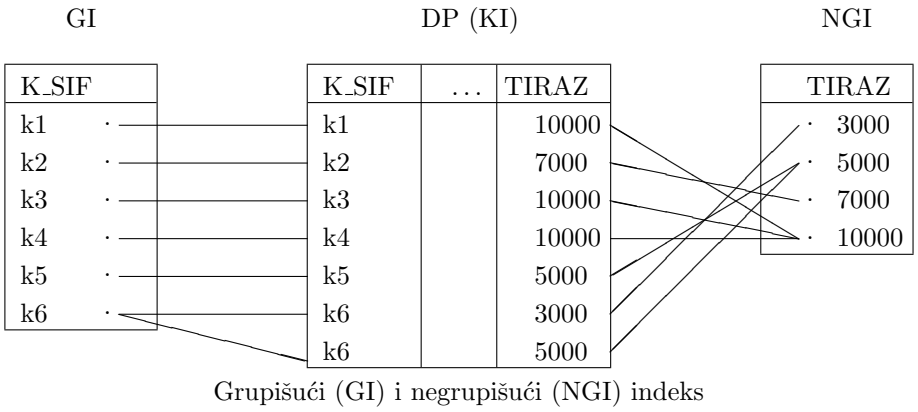
Indeksna organizacija je, zbog efikasnosti pristupa po raznim grupama atributa, najzastupljenija fizička organizacija relacijâ u svim sistemima za upravljanje bazama podataka.

Među indeksima izgrađenim nad jednom datotekom podataka, najviše jedan indeks može biti *grupišući* (engl. clustering), što znači da je izgrađen nad atributom po kome su uređeni fizički slogovi datoteke podataka, kao kod indeks-sekvencijalne reprezentacije. Ostali indeksi su "obični", negrupišući, i po njima je moguć efikasan

[1] Ovakav indeks zove se i *gust* indeks (engl. dense).

[2] Na logičkom nivou indeks se definiše nad atributom (ili skupom atributa) indeksiranja neke relacije, u cilju efikasnog pristupa n-torkama te relacije; na fizičkom nivou indeks je izgrađen nad *fizičkim poljem indeksiranja* (koje odgovara atributu indeksiranja) odgovarajuće datoteke podataka, i koristi se za efikasan pristup fizičkim slogovima; u kontekstu u kome razgraničenje nivoa nije bitno, parovi odgovarajućih termina (atribut indeksiranja, polje indeksiranja) i (relacija, datoteka podataka) upotrebljavaju se u sinonimnom značenju.

pristup slogovima. Grupišući indeks ne mora biti izgrađen nad primarnim ključem. To je privilegovan indeks, omogućuje vrlo efikasnu obradu (restrikciju, spajanje) koja uključuje atribut indeksiranja, i izgrađuje se u slučaju da je takva obrada relacije najčešća (v. sliku 12.1; grupišući indeks (GI) i jedan negrupišući indeks (NGI) izgrađeni su nad atributom K_SIF, odnosno atributom TIRAZ datoteke podataka (DP) relacije KI). Održavanje fizičkog uređenja u slučaju grupišućeg indeksa zahteva, kao kod indeks-sekvencijalne reprezentacije, korišćenje posebne datoteke – zone prekoračenja – za unošenje elemenata posle početnog sortiranja datoteke podataka po atributu indeksiranja. Međutim, za razliku od indeks-sekvencijalne organizacije, indeks kod indeksne organizacije omogućuje neposredan pristup svakom od slogova (odnosno svakoj stranici) i iz zone prekoračenja, bez korišćenja lanaca prekoračenja.



Slika 12.1: Indeksna reprezentacija

Ako postoji indeks nad primarnim ključem PK relacije R, onda on (logički) realizuje preslikavanje $dom(\text{PK}) \rightarrow R$. Isti je slučaj i sa bilo kojim drugim indeksom čijim različitim vrednostima iz domena atributa indeksiranja odgovaraju različite n-torke relacije (tzv. jedinstveni indeks).

12.1 Efikasnost izvršavanja jednorelacionih upita

12.1.1 Pretraživanje

Indeks kod indeksne reprezentacije može se koristiti samo za pristup slogovima po vrednosti atributa indeksiranja (inače je pristup sekvencijalan). Indeksna reprezentacija relacije efikasna je samo u slučaju izbora pojedinačnog sloga, dok je u slučaju obrade svih (ili velike grupe) slogova, u bilo kom redosledu, sekvencijalni pristup slogovima efikasniji nego pristup preko indeksa.

Kako operacija restrikcije vrši izbor jednog ili grupe slogova po vrednosti atributa, efikasnost ove operacije u slučaju indeksne reprezentacije zavisi od tipa restrikcije i od atributâ po kojima postoje indeksi. Tip restrikcije ukazuje na količinu slogova čiji se izbor vrši tom restrikcijom, i može biti intervalna (po uslovu $\Theta \in \{<, <=, >, >=\}$) ili specifična (po uslovu =).

Primer 12.1 Upit kojim se iz relacije P (pisac) dobijaju imena jugoslovenskih pisaca sa više od 10 naslova, može se SQL-om izraziti kao:

```
SELECT IME
FROM   P
WHERE  DRZAVA = 'Jugoslavija' AND BR_NASLOVA > 10
```

Ako je nad datotekom podataka relacije P izgrađen indeks po atributu DRZAVA, prethodna restrikcija će se izvršiti kao izbor svih slogova sa vrednošću atributa DRZAVA = 'Jugoslavija', za kojim sledi provera intervala za atribut BR_NASLOVA. Dakle, potrebno je pretražiti indeks da bi se dobio skup svih rednih brojeva stranica na kojima su slogovi čija je vrednost atributa DRZAVA – 'Jugoslavija'. U slučaju da je indeks negrupišući, može se pretpostaviti da je svaki takav slog na posebnoj stranici; ako je broj takvih slogova (i stranica) veliki, ovakva restrikcija je skupa operacija. Ako je indeks grupišući, svi slogovi (bar iz datoteke podataka) sa istom vrednošću atributa DRZAVA nalaze se zajedno na stranici ili susednim stranicama, pa je pristup znatno efikasniji.

Ako nad relacijom P postoji indeks po atributu BR_NASLOVA, a ne postoji indeks po atributu DRZAVA, upotreba indeksa je dobra samo ako je indeks grupišući. Indeks se koristi za nalaženje prvog sloga iz odgovarajuće datoteke podataka, sa vrednošću atributa BR_NASLOVA većom od 10, a zatim se pretraživanje datoteke podataka nastavlja sekvencijalno (sa lancima prekoračenja u zoni prekoračenja) do kraja. Restrikcija se u ovom slučaju izvodi prema algoritmu intervalne restrikcije kao kod indeks-sekvencijalne organizacije.

Ako nad relacijom P nema nijednog od ovih indeksa, pretražuju se svi slogovi datoteka DP i ZP, uz ispitivanje uslova restrikcije za svaki slog.

12.1.2 Ažuriranje, unošenje i brisanje

Unošenje sloga pri indeksnoj organizaciji u slučaju da nad datotekom podataka postoji grupišući indeks razlikuje se od slučaja kada su svi indeksi negrupišući. Ako postoji grupišući indeks, unošenje se sastoji od sledećih koraka:

1. novi slog se unosi u prvu slobodnu stranicu zone prekoračenja;
2. pretraživanjem grupišućeg indeksa pronalazi se element podataka – par (vrednost atributa indeksiranja, {redni broj stranice}), koji se odnosi na slog sa istom ili prvom manjom vrednošću atributa indeksiranja od odgovarajuće vrednosti unetog sloga;

3. ako vrednost atributa indeksiranja u indeksu već postoji, dodaje se broj stranice novog unetog sloga skupu brojeva stranica u nađenom elementu podataka; u suprotnom, unosi se u indeks novi element podataka koji se odnosi na novi uneti slog (v. odeljak 12.3, "Struktura indeksa");

4. novi slog se "uvezuje" u odgovarajući lanac prekoračenja (kao kod indeks-sekvencijalne organizacije) iza poslednjeg sloga sa istom (ili prvom manjom) vrednošću atributa indeksiranja (njegova adresa je dobijena u koraku 2);

5. pretraživanjem svakog od negrupišućih indeksa pronalazi se mesto za upis novog elementa podataka (kao u koraku 2), i element podataka (ili samo redni broj stranice zone prekoračenja) se tamo upisuje.

U slučaju da ne postoji grupišući indeks, nema ni zone prekoračenja, već je datoteka podataka jedina koja sadrži fizičke slogove koji odgovaraju n-torkama relacije; unošenje novog sloga u tom slučaju sastoji se od upisa novog sloga u datoteku podataka (odgovara koraku 1 prethodnog postupka) i upisa odgovarajućeg elementa podataka u svaki od negrupišućih indeksa (korak 5).

Brisanje sloga (slogova) sa zadatom vrednošću nekog atributa sastoji se od sledećih radnji:

1. nalaženje sloga (preko indeksa, ako postoji nad datim atributom, ili sekvencijalnim pretraživanjem);

2. obeležavanje sloga nevažećim ili fizičko brisanje sloga sa uklanjanjem sloga iz odgovarajućeg lanca prekoračenja (u slučaju postojanja grupišućeg indeksa);

3. za svaki postojeći indeks nad datotekom podataka, brisanje sloga koji se odnosi na slog izbrisan iz datoteke podataka, uz dodatne neophodne izmene na indeksnim datotekama.

Ažuriranje sloga sastoji se od sledećih radnji:

1. nalaženje sloga (najbolje preko grupišućeg indeksa, ali može i preko nekog drugog indeksa);

2. ažuriranje sloga i upis ažurirane stranice (ako atribut indeksiranja grupišućeg indeksa nije ažuriran), odnosno obeležavanje sloga nevažećim i upis novog sloga sa ažuriranim atributima (ako se ažurira i atribut indeksiranja grupišućeg indeksa);

3. za svaki indeks, odgovarajuće ažuriranje indeksa.

12.2 Izvršavanje višerelacionih upita

Algoritmi izvršavanja dvorelacionih, pa i višerelacionih upita koji se na njima zasnivaju, zavise od pristupnih – indeksnih datoteka nad relacijama koje učestvuju u upitu.

Primer 12.2 Za relacije P i KP (pisac i autor, glava 1), upit "naći imena svih jugoslovenskih pisaca koji se pojavljuju kao drugi autor neke knjige", koji se u SQL-u izražava kao SELECT blok

```
SELECT IME
FROM   P, KP
WHERE  P.P_SIF = KP.P_SIF AND P.DRZAVA = 'Jugoslavija'
       AND KP.R_BROJ = 2
```

izvršiće se na razne načine za različite indekse nad relacijama P i KP. Tako, ako postoje indeksi po atributima specifične restrikcije – P.DRZAVA i KP.R_BROJ, a ne postoje indeksi po atributima spajanja P_SIF ni u jednoj relaciji, upit se izvršava tako što se prvo vrši indeksna restrikcija (relacije P po uslovu P.DRZAVA = 'Jugoslavija' i relacije KP po uslovu KP.R_BROJ = 2), a zatim sortiranje (ako je potrebno) uz preostale restrikcije, i sekvencijalno prirodno spajanje (po P_SIF) dobijenih sortiranih datoteka, uz projektovanje na atribute projekcije (IME).

Ako pak postoji indeks po atributu specifične restrikcije jedne relacije (npr. po atributu P.DRZAVA) i indeks po atributu spajanja druge relacije (npr. po atributu KP.P_SIF), onda se upit izvršava u dva koraka. U prvom koraku izvršava se indeksna restrikcija (relacije P po uslovu P.DRZAVA = 'Jugoslavija'). U drugom koraku datoteka – rezultat te restrikcije sekvencijalno se pretražuje, i svaki njegov slog, korišćenjem indeksa po atributu spajanja druge relacije (KP), spaja se sa odgovarajućim n-torkama te druge relacije, tj. sa slogovima odgovarajuće datoteke. U drugom koraku izvode se i preostale restrikcije (npr. relacije KP po uslovu KP.R_BROJ = 2), kao i projekcija na atribute projekcije (IME).

Spajanje koje je opisano prethodnim postupkom zove se *indeksno spajanje*. Objasnimo njegov algoritam nešto detaljnije.

Indeksno spajanje je prirodno spajanje dve datoteke koje odgovaraju dvema relacijama, od kojih jedna (npr. prva) ima indeks po atributu spajanja. Fizičko polje slogova datoteke, koje odgovara atributu spajanja, zvaćemo polje spajanja. Datoteke se spajaju tako što se druga datoteka čita sekvencijalno, i za svaki njen slog, tj. njegovu vrednost polja spajanja, pristupa se, korišćenjem indeksa, slogovima prve datoteke koji imaju istu vrednost polja spajanja kao slog iz druge datoteke. Slog druge i odgovarajući slogovi prve datoteke se spajaju.

12.3 Struktura indeksa

Indeks nad atributom (ili skupom atributa) A relacije R u indeksnoj reprezentaciji fizički je predstavljen datotekom ID čije stranice obrazuju strukturu *drveta* ([55]).

Listovi tog drveta su stranice koje sadrže skupove slogova sortiranih po vrednostima iz domena atributa indeksiranja. Ovakva struktura čini pretraživanje indeksa maksimalno efikasnom radnjom.

Struktura indeksnog drveta čiji su čvorovi – stranice, vrlo je specifična, kao i struktura i druga svojstva samih stranica. Zato ćemo prvo podsetiti na opštu definiciju strukture grafa i drveta kao specifične strukture grafa, a zatim ćemo uvesti definicije specifičnih struktura drveta, i posebno B-drveta kao jedne fizičke reprezentacije indeksa.

Definicija 12.1 (Graf) *Usmereni graf* Γ je uređeni par (A, r), gde je A – konačni skup a r – binarna relacija nad skupom A. Elemente skupa A zovemo *čvorovi*, a elemente skupa r – *potezi*. Ako je $(a_i, a_j) \in r$ poteg, kaže se da je čvor a_i – *neposredni prethodnik* čvora a_j, a da je čvor a_j – *neposredni sledbenik* čvora a_i; takoďe, čvor a_i je *izlazni čvor* potega (a_i, a_j), a čvor a_j je *ulazni čvor* potega (a_i, a_j). *Ulazni red* čvora a_i je broj potega kojima je taj čvor – ulazni čvor, a *izlazni red* čvora a_i je broj potega kojima je taj čvor – izlazni čvor. *Putanja dužine* n od čvora a_i do čvora a_k u grafu Γ je niz čvorova $(a_i, a_{i+1}, \ldots, a_{i+n})$, takvih da je $a_{i+n} = a_k$ i $(a_j, a_{j+1}) \in r$ za svako $j = i, i+1, \ldots, i+n-1$. Putanja od čvora a_i do čvora a_k, dužine veće od 1, za koju važi da je $a_i = a_k$, zove se *ciklus*.

Definicija 12.2 (Drvo) Graf Γ je *drvo* ako zadovoljava sledeće uslove:

- ulazni red svakog čvora (osim jednog) je 1;
- ulazni red jednog čvora je 0, i taj čvor se zove *koren* drveta;
- izlazni red svakog čvora je ≥ 0 i konačan;
- graf Γ ne sadrži cikluse.

Pri tome,

– čvorovi bez neposrednih sledbenika zovu se *listovi* drveta (izlazni red svakog lista je 0);

– *visina* drveta je maksimalna dužina putanje od korena do nekog lista drveta.

Posebno poželjna osobina drveta, kada je u pitanju efikasnost njegovog pretraživanja, jeste *balansiranost* koja označava da je dužina puta od korena do svakog lista jednaka. Jedna specifična struktura balansiranog drveta, poznata kao B-drvo, poslužila je kao osnov za izgradnju niza struktura podataka koje se koriste u implementaciji indeksa. U daljem tekstu biće izložena struktura B-drveta kao i osnovni algoritmi za unošenje u B-drvo i pretraživanje B-drveta.

Definicija 12.3 (*B*-drvo) *B-drvo* je balansirano drvo sa sledećim dvema grupama karakteristika:

I struktura drveta:

- čvorovi drveta su stranice;
- visina balansiranog drveta je $h \geq 0$;
- svaka stranica, osim korena i listova, ima najmanje $n + 1$ neposrednih sledbenika ($n > 0$); koren je ili list ili ima najmanje dva neposredna sledbenika;
- svaka stranica ima najviše $2n + 1$ neposrednih sledbenika.

II struktura stranice – čvora drveta:

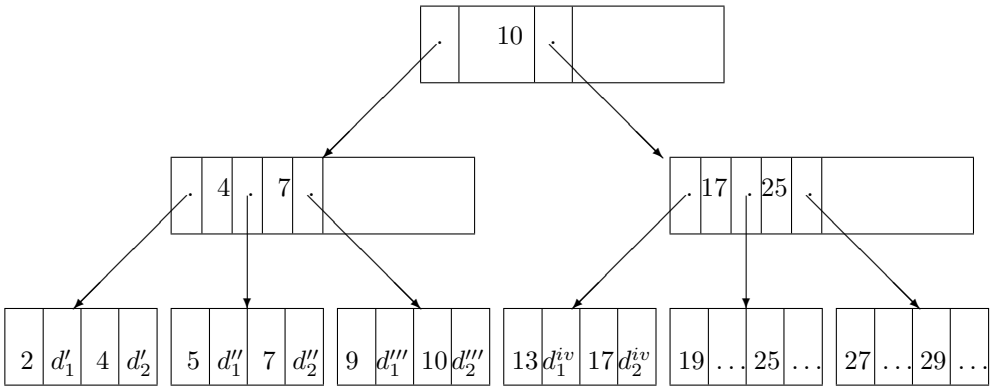
- svaka stranica koja nije list predstavlja niz

$$(p_0, (k_1, p_1), (k_2, p_2), \ldots, (k_m, p_m)),$$

  gde je uređeni par (k_i, p_i) – *indeksni element*, koji se sastoji od vrednosti iz domena atributa indeksiranja (k_i) i pokazivača na stranicu neposrednog sledbenika (p_i), $i = 1, 2, \ldots, m$. Pokazivač na stranicu – neposrednog sledbenika je i p_0;
- svaka stranica koja nije list, osim korena, ima najmanje n indeksnih elemenata, a koren koji nije list – najmanje jedan. Svaka stranica koja nije list ima najviše $2n$ indeksnih elemenata;
- za svaku vrednost k_i iz indeksnog elementa važi sledeće: pokazivač koji joj prethodi (p_{i-1}) pokazuje na poddrvo u kome za svaku vrednost k_j iz domena atributa indeksiranja važi $k_j \leq k_i$; pokazivač koji za njom sledi (p_i) pokazuje na poddrvo u kome za svaku vrednost k_j iz domena atributa indeksiranja važi $k_j > k_i$;
- stranice – listovi sadrže niz *elemenata podataka* – parova oblika (k_i, d_i), gde su k_i kao i u indeksnom elementu, a d_i – pridruženi podaci. Na primer, d_i može biti skup adresa slogova sa vrednošću atributa indeksiranja k_i (ili skup rednih brojeva stranica na kojima su ti slogovi). List sadrži najmanje n', a najviše $2n'$ ($n' > 0$) elemenata podataka;
- elementi stranice (indeksni i elementi podataka) sortirani su u rastućem redosledu vrednosti iz domena atributa indeksiranja.

Na primer, za $h = 2, n = 2, n' = 1$, grafički izgled jednog *B*-drveta predstavljen je slikom 12.2.

Osnovne radnje nad indeksom su:



Slika 12.2: Primer B-drveta

– *pretraživanje indeksa*: pronalaženje podataka d (npr. adresâ slogova) koji odgovaraju datoj vrednosti atributa indeksiranja k;

– *unošenje u indeks* (odnosno brisanje i ažuriranje): dodavanje para (k, d) indeksu (odnosno uklanjanje iz indeksa, izmena); d je podatak koji odgovara datoj vrednosti k iz domena atributa indeksiranja.

Ove radnje ne obavlja korisnik baze podataka, već su one ugrađene u algoritme koje SUBP primenjuje u obradi podataka.

12.3.1 Pretraživanje indeksa

Ako je indeks predstavljen strukturom B-drveta, onda se traženi podatak nalazi u stranici – listu do kojeg se dolazi putanjom preko odgovarajućih indeksnih elemenata; podatak koji treba da se unese, unosi se u stranicu – list, pri čemu se unose (ili modifikuju) i indeksni elementi koji leže na putanji koja vodi do tog lista. Algoritam pretraživanja indeksa opisuje se sledećim pseudoprogramom.

Pretraživanje indeksa

ulaz: vrednost k iz domena atributa indeksiranja;
izlaz: skup podataka d pridružen vrednosti k ako par (k, d) postoji u indeksu, inače poruka "u indeksu ne postoji vrednost k";

```
BEGIN
  neka je tekuća stranica – koren B-drveta;
  WHILE tekuća stranica nije list DO
    BEGIN pretražiti tekuću stranicu;
        IF za neko i, k ≤ k_i, THEN
          BEGIN
            neka je i' najmanje takvo i;
            tekuća stranica neka je stranica na koju pokazuje pokazivač p_{i'-1}
          END
        ELSE tekuća stranica neka je stranica na koju pokazuje pokazivač p_m
    END;
  pretražiti list drveta;
  IF pronađen par (k, d) THEN rezultat je d
  ELSE poruka "u indeksu ne postoji vrednost k"
END.
```

Na primer, ako iz prethodnog drveta treba naći podatak d koji odgovara vrednosti $k = 13$ atributa indeksiranja, izvršavanje prethodnog algoritma se odvija kroz sledeće korake:

- tekuća stranica je koren;
- kako ne postoji i za koje je $k(= 13) < k_i$, tekuća stranica postaje stranica sa sadržajem .17.25. jer na nju pokazuje pokazivač $p_m(\equiv p_1)$;
- kako je na tekućoj stranici $k(= 13) < k_i$ za $i = 1$, tekuća stranica postaje stranica na koju pokazuje pokazivač $p_{i-1}(\equiv p_0)$, tj. stranica sa sadržajem $(13, d_1^{iv}; 17, d_2^{iv})$;
- tekuća stranica je list; njenim pretraživanjem pronalazi se element podataka – par $(13, d_1^{iv})$, pa je podatak d_1^{iv} – rezultat izvršavanja algoritma.

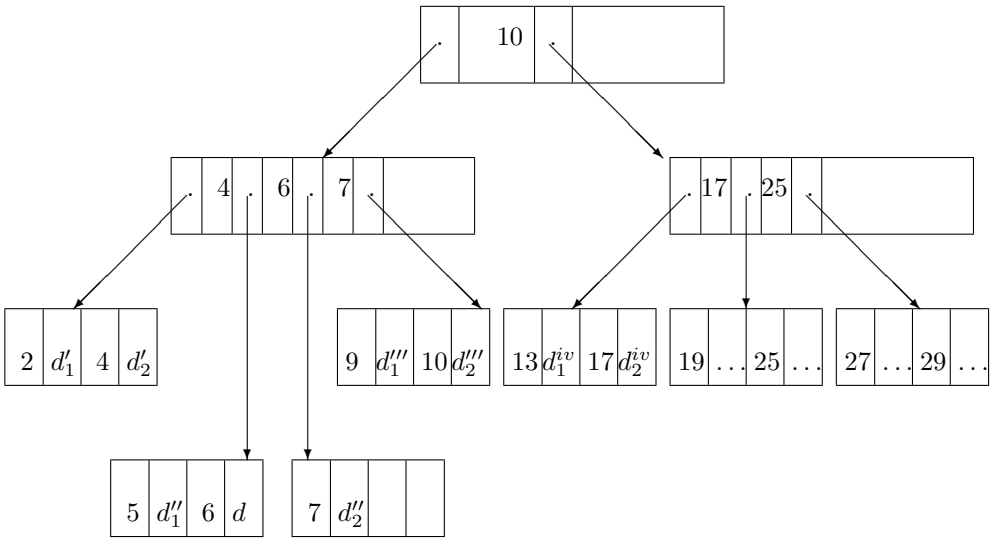
12.3.2 Unošenje u indeks, brisanje i ažuriranje

B-drvo je dinamički indeks. U toku ažuriranja baze podataka, u njega se podaci unose i iz njega brišu. Ako je potrebno uneti novi par (k, d) u indeks, koji označava da podatak d određuje položaj slogova sa vrednošću atributa indeksiranja k, onda se prethodnim algoritmom locira stranica u indeksu u koju treba uneti novi element podataka. Ako u toj stranici nema mesta (stranica je maksimalno popunjena), primenjuje se algoritam "cepanja" stranice, tj. dodeljivanje nove indeksne stranice

B-drvetu i ravnomerna podela elemenata podataka između locirane i nove stranice. Sada treba ažurirati stranicu prethodnika unošenjem novog indeksnog elementa koji se odnosi na novu stranicu, pri čemu sa cepanjem stranicâ može da se nastavi (ako na stranici – prethodniku nema mesta za unošenje novog indeksnog elementa). Cepanje stranice može da se nastavi sve do korena drveta, kada se povećava visina drveta. Algoritam unošenja novog elementa podataka može se naći u [67], [65], [1], a ovde će biti ilustrovan primerom. Algoritam obezbeđuje popunjenost stranice od 50% do 100% (u proseku 75%), i podložan je mnogim prostornim i vremenskim poboljšanjima.

Primer 12.3 U prethodno B-drvo uneti element podataka $(6, d)$.

Algoritmom pretraživanja drveta locira se stranica sa sadržajem $(5, d_1''), (7, d_2'')$ kao stranica na koju treba smestiti novi element podataka. Kako je ta stranica maksimalno popunjena (sadrži $2n'$ elemenata podataka, za $n' = 1$), stranica se "cepa" tako što se uvodi nova stranica; na "staroj" stranici ostaju dva (uređena) elementa podataka $(5, d_1''), (6, d)$, a na novoj stranici – jedan element podataka $(7, d_2'')$. U opštem slučaju na "staroj" stranici ostavlja se $n' + 1$ (oko 50%) elemenata podataka, a ostalih n' se upisuje na novu stranicu. Takođe se u stranicu – prethodnik (stranicu sa sadržajem .4.7.) dodaje novi indeksni element, pa njen sadržaj postaje .4.6.7. Tako se dobija novo B-drvo indeksa, grafički predstavljeno na slici 12.3.



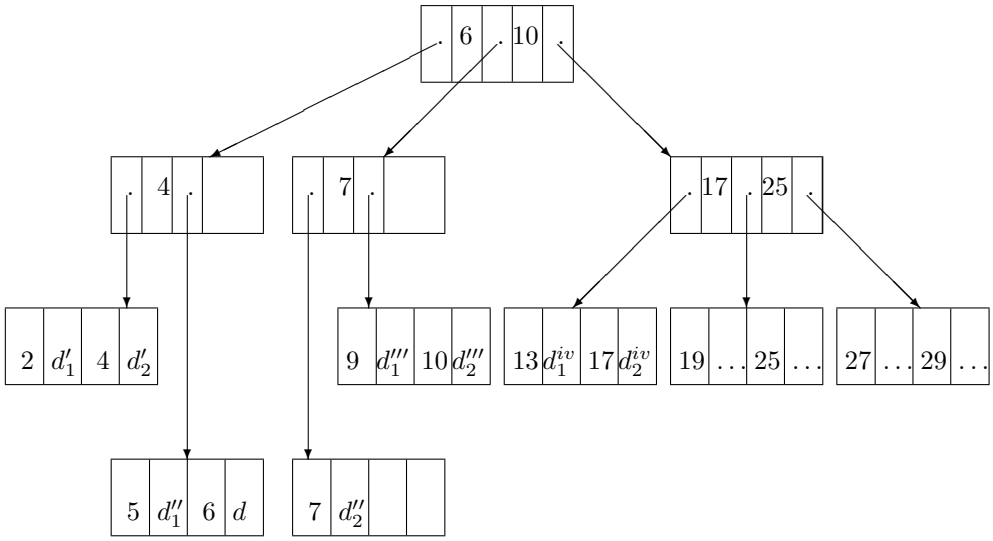
Slika 12.3: Cepanje stranice – lista B-drveta

U slučaju da je u prethodnom primeru parametar B-drveta n bio $n = 1$, stranica sa sadržajem .4.7. sadrži maksimalni broj indeksnih elemenata, $2n$. Pri pokušaju

da se u tu stranicu unese indeksni element 6. javila bi se potreba za cepanjem i te stranice. Cepanje stranice koja nije list unekoliko se razlikuje od cepanja stranice – lista, i odvija se po sledećem postupku:

- neka je s stranica koja nije list a koja se "cepa"; u našem primeru to je stranica sa sadržajem .4.7.;
- u unutrašnjoj memoriji sortirati skup vrednosti indeksa iz $2n + 1$ indeksnog elementa: 4,6,7;
- formirati novu stranicu s'; u stranici s ostaviti pokazivač p_0 i prvih n indeksnih elemenata iz sortiranog niza (u našem primeru .4.); u stranicu s' uneti $(n+1)$-vi pokazivač (p_{n+1}) i indeksne elemente od $(n+2)$-og do poslednjeg (u našem primeru .7.);
- u prethodnika stranice s uneti $(n + 1)$-vu vrednost (u našem primeru 6), i pokazivač p' na stranicu s'.

Grafički, unošenje u ovom primeru može se predstaviti kao na slici 12.4.



Slika 12.4: Cepanje stranice B-drveta koja nije list

Sličan postupak se sprovodi u slučaju da je potrebno "cepati" korenu stranicu. Tada se visina drveta povećava za 1.

Pri brisanju vrednosti iz indeksa primenjuje se inverzni postupak koji može dovesti do spajanja stranica. Vrednost atributa indeksiranja ažurira se u indeksu parom operacija (brisanje, unošenje).

Postoje dva problema sa osnovnom strukturom B-drveta, zbog kojih se češće koriste neke modifikacije te strukture. Jedan problem je nedovoljna popunjenost stranica. Naime, nedovoljno popunjeno indeksno drvo produžuje vreme pretraživanja, i zato popunjenost stranice od 75% (u proseku) ne zadovoljava. Rešenje ovog problema nalazi se u "prelivanju" elemenata podataka između susednih stranica, umesto cepanja stranice. Tako se susedne stranice, ako je to moguće, dodatno popunjavaju, a nova stranica (koja je popunjena samo sa 50%) uvodi se samo kada je to zaista neophodno.

Drugi problem sa osnovnom strukturom B-drveta je nemogućnost pretraživanja datoteke podataka u redosledu vrednosti atributa indeksiranja, a što je često veoma poželjno. Modifikacija koja omogućuje ovakvo pretraživanje je povezivanje stranica – listova B-drveta u listu koja odražava sortiranost atributa indeksiranja, i obezbeđuje pristup slogovima datoteke podataka po rastućim (ili opadajućim) vrednostima atributa indeksiranja.

12.4 Pitanja i zadaci

1. Koje su karakteristike indeksa kod indeksne organizacije?
2. Šta je grupišući indeks?
3. Opisati postupak unošenja sloga kod indeksne organizacije.
4. Kako se vrši operacija spajanja kod indeksne organizacije?
5. Napisati pseudoprogram za algoritam restrikcije kod indeksne organizacije.
6. Napisati pseudoprogram za algoritam indeksne restrikcije.
7. Definisati graf, drvo i balansirano drvo.
8. Opisati strukturu B-drveta.
9. Objasniti algoritam pretraživanja indeksa.
10. Napisati skup programskih procedura za implementaciju algoritma pretraživanja indeksa. Uvesti neophodne pretpostavke i ograničenja.
11. Objasniti algoritam unošenja u indeks.
12. Konstruisati algoritam brisanja iz indeksa.
13. Koje su prednosti indeksne organizacije?
14. Koji bi atribut (ili grupa atributa) bio najpogodniji kao atribut indeksiranja relacije KI (glava 1)? Obrazložiti.

15. Neka relacija KI sadrži 50000 n-torki i neka njen sadržaj ima sledeću strukturu: u relaciji je 10000 različitih knjiga; svaku knjigu izdalo je u proseku 2.5 izdavača; svaki izdavač izdao je jednu istu knjigu u proseku u po 2 izdanja. Neka je relacija KI predstavljena indeksnom reprezentacijom sa indeksom po atributu K_SIF, tako da polje K_SIF u indeksu zauzima 5 bajtova, koliko zauzima i pokazivač na odgovarajući slog u datoteci podataka. Neka je element podataka oblika (K_SIF, pokazivač) i neka je veličina stranice 1024 bajta. Izračunati broj stranica indeksa potrebnih za elemente podataka i broj stranica indeksa potrebnih za indeksne elemente. Izračunati visinu indeksnog drveta.

13

Direktna organizacija

Potpuno drugačiji pristup fizičkoj organizaciji relacije, u odnosu na indeks-sekvencijalnu i indeksnu organizaciju, predstavlja direktna (heš, rasejana, rasuta) organizacija (engl. hash).

Odlike direktne organizacije su:

- specifičnom slogu se pristupa uz pomoć algoritma koji se zove *algoritam direktnog pristupa* ili *heš algoritam*
- algoritam direktnog pristupa sastoji se u izvoďenju aritmetičkih transformacija date vrednosti nekog atributa A (*atribut direktnog pristupa, heš atribut*), tj. polja u slogu (*polje direktnog pristupa, heš polje*), u broj stranice na kojoj se slog nalazi; tako algoritam direktnog pristupa ostvaruje preslikavanje

$$\text{H: } dom(\text{A}) \rightarrow \text{skup-brojeva-stranica}$$

- svaki slog smešta se na stranicu čiji je broj određen algoritmom direktnog pristupa; sâm broj stranice zove se *adresa direktnog pristupa* ili *heš adresa* (ili samo *adresa*, u kontekstu u kome se značenje tog termina može jednoznačno razlikovati od fizičke adrese na disku);
- jedna relacija može imati više indeksnih atributa, ali samo jedan atribut direktnog pristupa.

Na primer, neka je relacija I (I_SIF, NAZIV, STATUS DRZAVA) predstavljena fizički direktnom reprezentacijom, i neka atribut I_SIF uzima vrednosti i100, i200, i300, i400, i500. Neka je algoritam direktnog pristupa zadat funkcijom (*funkcija direktnog pristupa, heš funkcija*)

$$\text{broj\_stranice(I\_SIF)} = num\ (\text{I\_SIF})\ mod\ 13,$$

gde *num* označava numerički deo vrednosti atributa I_SIF, a *mod* je funkcija izračunavanja ostatka pri celobrojnom deljenju. Prethodna funkcija proizvodi sledeće brojeve stranica datoteke podataka, u koje će se upisati slogovi sa navedenim vrednostima atributa I_SIF: 9, 5, 1, 10, 6. Ta heš funkcija je jedna modifikacija najčešće korišćene heš funkcije, tzv. *količnik/ostatak* heš funkcije: $h(v) = v \ mod$ M. Ovom funkcijom se izračunava adresa direktnog pristupa sloga sa vrednošću heš atributa v, pri veličini datoteke podataka od M stranica.

Ako se zna da su vrednosti atributa direktnog pristupa različiti celi brojevi od 1 do N, slog sa vrednošću atributa direktnog pristupa i mogao bi da se upiše na i-tu stranicu datoteke podataka. Heširanje (direktna reprezentacija) je uopštenje ovog postupka kada ne postoji tako specifično znanje o vrednostima atributa direktnog pristupa. Naime, domen atributa direktnog pristupa po pravilu ima mnogo više elemenata od broja stranica u datoteci DP, pa funkcija direktnog pristupa h nije injektivna, tj. može biti $h(v_1) = h(v_2)$ i za $v_1 \neq v_2$. U prethodnom primeru, ako postoji izdavač sa šifrom I_SIF=i1400, broj stranice na koju će se smestiti slog sa tom vrednošću atributa I_SIF je $1400 \ mod \ 13 = 9$, a to je ista stranica na koju se smešta i slog sa vrednošću atributa I_SIF=i100. Ova pojava kod direktne reprezentacije zove se *kolizija*.

Dakle, prvi korak u izgradnji direktne reprezentacije je izračunavanje heš funkcije koja transformiše vrednost atributa direktnog pristupa u broj stranice. Ne postoji savršena heš funkcija, i uvek može da se dogodi da dve ili više različitih vrednosti atributa direktnog pristupa dovedu do kolizije. Drugi korak u izgradnji direktne reprezentacije je razrešavanje kolizije.

Jedna od metoda razrešavanja kolizije koristi povezane liste (ulančavanje) i pogodna je u nekim dinamičkim situacijama kada broj različitih vrednosti atributa direktnog pristupa nije poznat unapred. U našem primeru, ako na stranici 9 ima mesta za oba sloga, oni će se tamo upisati bez problema. Ako nema mesta na stranici (npr. svaki slog zauzima celu stranicu), bira se prva slobodna stranica pomoćne datoteke – zone prekoračenja (u ovom slučaju stranica broj 13), tamo se upisuje slog i povezuje se pokazivačem sa stranicom datoteke podataka čiji je broj "dupliran" (u našem primeru – sa stranicom 9). Dalji slogovi sa istom adresom ulančavaju se kao kod indeks-sekvencijalne datoteke.

Direktna reprezentacija je dobar primer nagodbe oko prostora i vremena. Ako nema ograničenja memorije (veličine datoteke podataka), onda svako pojedinačno pretraživanje sloga može da se izvede u jednom koraku, korišćenjem vrednosti atributa direktnog pristupa kao adrese (heš funkcija h je identitet). Kada ne bi bilo vremenskog ograničenja, mogla bi se primeniti sekvencijalna reprezentacija uz korišćenje minimuma memorije. Direktna reprezentacija omogućuje upotrebu razumne količine prostora i vremena. Osnovni princip metode heširanja je efikasno korišćenje raspoložive memorije i brz pristup memoriji.

Tako se, korišćenjem heš atributa I_SIF relacije I, efikasno izvršavaju upiti sa specifičnom restrikcijom oblika

```
SELECT *
FROM   I
WHERE  I_SIF = 'i2000'
```

dok se taj heš atribut ne može iskoristiti u upitima sa intervalnom restrikcijom oblika

```
SELECT *
FROM   I
WHERE  I_SIF > 'i500'
```

Da bi se smanjila frekvencija kolizije (u slučaju da se ostavi mali broj stranica), a i da bi se smanjilo nepotrebno trošenje stranica (kada se ostavi mnogo stranica), obično se za smeštaj slogova ostavlja oko 20% više prostora u datoteci podataka i zoni prekoračenja, nego što je to potrebno s obzirom na broj i veličinu slogova.

Metode direktne reprezentacije i direktnog pristupa, bilo da se odnose na trajne strukture kao što su spoljašnje datoteke baza podataka, ili na memorijske strukture kao što su razne vrste tabela i njihovo efikasno pretraživanje, dosta dobro su istražene, uključujući oba problema: heš funkcije i metode razrešavanja kolizije. U sledećim odeljcima biće više reči o ovim problemima i njihovom rešavanju.

13.1 Funkcija direktnog pristupa

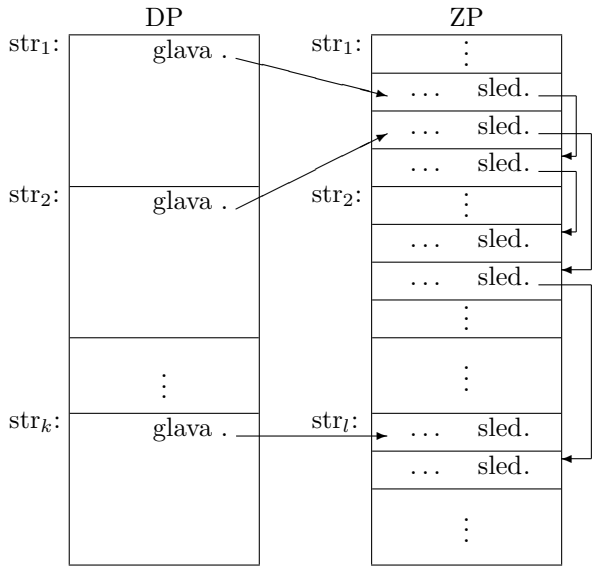
Kod direktne fizičke organizacije relacija, definisanje i izračunavanje funkcije direktnog pristupa predstavlja prvi problem realizacije ove organizacije. Funkcija treba da transformiše vrednosti atributa direktnog pristupa (obično cele brojeve ili kratke niske znakova) u cele brojeve iz intervala 0..M–1, gde je M broj stranica datoteke podataka. Problem sa heš funkcijom je u tome što ona treba da preslika veliki domen mogućih vrednosti heš atributa u mali skup brojeva stranica. Pri tome kolizija treba da se izbegne ili da se minimizuje.

Nalaženje "idealne" heš funkcije – bez kolizije, čak i kada ona teorijski postoji, vrlo je težak posao. Na primer, ako imamo trideset slogova sa jedinstvenim vrednostima polja KLJUČ, i ako na jednu stranicu može da stane tačno jedan slog a na raspolaganju je 40 stranica datoteke podataka (40 raznih adresa), onda ima 40^{30} načina da se izračunaju adrese tih 30 slogova, tj. ima 40^{30} različitih heš funkcija. Od tog broja, samo $40 * 39 * \cdots * 11 = \frac{40!}{10!}$ funkcija daje različite adrese za različite vrednosti heš polja KLJUČ. Dakle, jedna od oko 10 miliona funkcija je idealna i ne proizvodi koliziju. Zbog toga se dobrom heš funkcijom smatra ona koja se lako izračunava i aproksimira "slučajnu" funkciju: za proizvoljni argument, svaki je rezultat podjednako moguć.

Prvi korak u izračunavanju funkcije direktnog pristupa je transformacija vrednosti atributa direktnog pristupa u broj (jer su funkcije aritmetičke). Drugi korak je sámo izračunavanje adrese direktnog pristupa, pri čemu se često koristi metoda izbora prostog broja M za broj raspoloživih stranica datoteke podataka, i neka modifikacija heš funkcije $h(v) = v \ mod$ M.

13.2 Rešavanje kolizije – posebno ulančavanje

Suština najjednostavnije metode za rešavanje problema kolizije je u održavanju povezane liste slogova za svaku stranicu datoteke podataka; u takvu listu će se "ulančati" svi slogovi koji se heš funkcijom preslikavaju u broj te stranice datoteke podataka, a za koje nema mesta u toj stranici. Tom prilikom dodeljena stranica zone prekoračenja može se koristiti za unošenje slogova sa različitim heš adresama, a nova stranica zone prekoračenja se ne dodeljuje sve dok se već dodeljena ne popuni, ili ne popuni do neke unapred zadate granice (slika 13.1).



Slika 13.1: Razrešavanje kolizije – posebno ulančavanje sa listama slogova

Svaka stranica datoteke podataka, u ovom slučaju, ima pokazivač *glava* na prvi slog u zoni prekoračenja sa istom heš adresom, a svaki slog stranice zone prekoračenja ima pokazivačko polje *sled* na sledeći slog u odgovarajućoj listi (na istoj ili različitoj stranici zone prekoračenja).

Ilustrujmo sledećim primerom postupak unošenja kod direktne reprezentacije sa posebnim ulančavanjem.

Primer 13.1 Neka u relaciji R atribut direktnog pristupa jeste SLOVO sa domenom – skupom velikih slova abecede. Neka je i-to slovo u abecedi predstavljeno brojem i i neka se koristi heš funkcija $h(v) = v \ mod$ M (M je broj stranica datoteke podataka). Tada se za zadati skup vrednosti atributa SLOVO i M=11 dobijaju sledeće heš adrese:

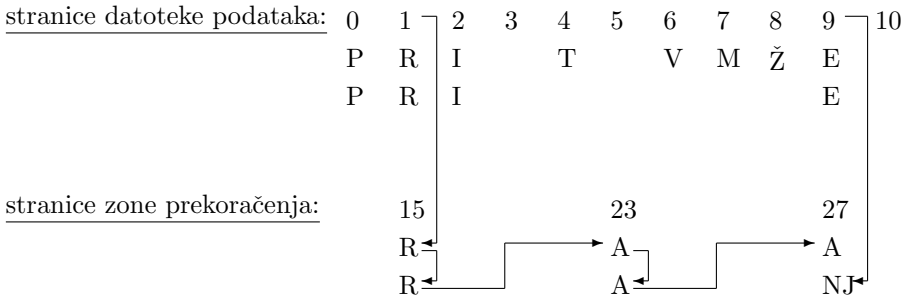
SLOVO:	P	R	I	M	E	R	P	R	E	T	R	A	Ž	I	V	A	NJ	A
i:	22	23	13	18	9	23	22	23	9	26	23	1	30	13	28	1	20	1
heš:	0	1	2	7	9	1	0	1	9	4	1	1	8	2	6	1	9	1

Ako se slogovi sa ovim vrednostima heš polja SLOVO redom unose u datoteku podataka koja je na početku prazna, stranice datoteke podataka sa brojevima od 0 do 10 treba da sadrže slogove sa sledećim vrednostima polja SLOVO:

0	1	2	3	4	5	6	7	8	9	10
P	R	I		T		V	M	Ž	E	
P	R	I							E	
	R								NJ	
	R									
	A									
	A									
	A									

Ako stranica datoteke podataka može da primi najviše dva sloga, onda se za upis trećeg sloga sa istom heš adresom dodeljuje (ili bira nepopunjena) stranica zone prekoračenja ZP, u nju se upisuje taj slog, a u odgovarajuću stranicu datoteke podataka postavlja se pokazivač na upisani slog. Ostali slogovi, za koje nema mesta u stranici datoteke podataka, upisuju se u već dodeljenu (ili novu) stranicu zone prekoračenja, pri čemu se postavljaju pokazivači na sledeći slog u lancu.

Neka su tri dodeljene stranice zone prekoračenja – stranice sa brojevima 15, 23, 27. Tada se fizička reprezentacija relacije R može predstaviti slikom 13.2. Stranica sa brojem 23 dodeljuje se tek kada je stranica sa brojem 15 popunjena; slično važi i za stranicu sa brojem 27. Stranica sa brojem 27 sadrži slogove sa različitim heš adresama.



Slika 13.2: Primer razrešavanja kolizije posebnim ulančavanjem

Globalni algoritmi unošenja i pretraživanja kod direktne organizacije sa posebnim ulančavanjem u razrešavanju kolizije mogu se predstaviti sledećim pseudoprogramima.

Algoritam heš unošenja sa posebnim ulančavanjem

ulaz: slog s sa vrednošću v heš atributa A;
izlaz: adresa stranice x na koju je unet slog s, ili na kojoj je nađen slog s;
BEGIN
  $x := h(v)$;
  IF na stranici(x) nađen slog s THEN
      izdati poruku da je slog već zapisan na stranici(x)
  ELSE IF stranica(x) popunjena THEN
      BEGIN
        WHILE postoji sledeći slog, *s_slog*, u lancu prekoračenja, DO
          BEGIN
            x := adresa stranice sloga *s_slog*;
            IF *s_slog* = s THEN EXIT(slog s nađen na stranici(x))
          END;
        IF stranica(x) popunjena THEN
            naći nepopunjenu stranicu u zoni prekoračenja ZP
              i njenu adresu dodeliti x;
        uneti slog s na stranicu x;
        IF slog s prvi slog u lancu prekoračenja THEN
            stranica($h(v)$).*glava* neka pokazuje na slog s
        ELSE *s_slog.sled* neka pokazuje na slog s
      END
    ELSE uneti slog s na stranicu x;
  vratiti adresu stranice x
END.

Algoritam heš pretraživanja sa posebnim ulančavanjem

ulaz: heš atribut A i vrednost v tog atributa;
izlaz: skup brojeva stranica $x_1, x_2, \ldots, x_n$ koje sadrže slogove
  sa vrednošću v heš atributa; za neuspešno pretraživanje – poruka;

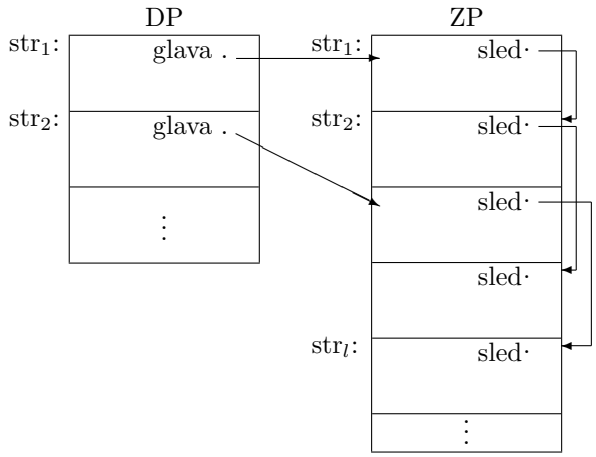
```
BEGIN
  i := 0;      /*broj pronađenih slogova*/
  x := h(v);
  IF stranica(x) sadrži slog sa vrednošću v atributa A THEN
      BEGIN i := 1; x_1 := x END;
  IF stranica(x) popunjena
  THEN WHILE postoji sledeći slog, s_slog, u lancu prekoračenja DO
            BEGIN
                x := adresa stranice sloga s_slog;
                IF s_slog.A = v i x_i <> x       /* za i > 0 */
                THEN BEGIN i := i + 1; x_i = x END
            END;
  IF i = 0 THEN izdati poruku o neuspešnom pretraživanju
  ELSE vratiti i, (x_j, j = 1, 2, ..., i)
END.
```

Vreme pretraživanja datoteke podataka i zone prekoračenja, izraženo brojem pročitanih stranica, zavisi od dužine listâ stranicâ. Uređenost slogova unutar listâ je manje važna nego kod sekvencijalnog pretraživanja jer su liste kraće. Za neuspešno pretraživanje, vreme je jednako prosečnom broju stranica listâ slogova (ali ne manje od 1), što u primeru 13.1 daje (1+4+1+0+1+0+1+1+1+2+0)/11 = 12/11 = 1.091. Ako su liste uređene, ovo vreme se skraćuje na polovinu. Za uspešno pretraživanje u primeru 13.1, 12 slogova bi se našlo u prvom koraku (čitanjem jedne stranice), 3 sloga u drugom koraku (čitanjem dve stranice), 2 u trećem, 1 u četvrtom, pa je vreme jednako (12∗1+3∗2+2∗3+1∗4)/18 = 1.555. Ako je N (ukupni broj slogova) mnogo veće od M, i ako na stranicu datoteke (DP i ZP) staje k slogova, onda je srednja dužina liste aproksimirana sa N/(M∗k). To skraćuje vreme sekvencijalnog pretraživanja za faktor M.

Umesto održavanja povezanih lista slogova, posebnim ulančavanjem mogu se održavati povezane liste stranica. Svakoj stranici datoteke podataka odgovara po jedna takva lista, u koju se "ulančavaju" sve stranice sa slogovima koji se funkcijom direktnog pristupa preslikavaju u broj te stranice datoteke podataka. U tom slučaju svakoj stranici datoteke podataka dodeljuje se, po potrebi, skup stranica zone prekoračenja koje mogu da prime samo slogove sa heš adresom jednakom adresi stranice datoteke podataka (slika 13.3).

Za ovaj slučaj mogu se prilagoditi i prethodno opisani algoritmi pretraživanja i unošenja sa posebnim ulančavanjem, i to tako da svaka stranica (i datoteke podataka i zone prekoračenja) ima pokazivač na sledeću stranicu u lancu, dok sami slogovi nemaju takve pokazivače. Novi slog se unosi u poslednju stranicu u lancu koji odgovara njegovoj heš adresi, a traži se samo u lancu stranica koji odgovara toj adresi. Stranice u lancu pretražuju se sekvencijalno.

U slučaju da se lanac slogova (ili stranica) za neku stranicu datoteke podataka suviše produži, moguće je na slogove tog lanca primeniti novu (bolju) heš funkciju. To je ideja metode *virtuelnog heširanja*, koja podrazumeva premeštanje izvesnog



Slika 13.3: Razrešavanje kolizije – posebno ulančavanje sa listama stranica

broja (ali ne svih) slogova na nove adrese.

13.3 Rešavanje kolizije – otvoreno adresiranje

Ako je moguće proceniti broj slogova N, onda održavanje lista stranica, odnosno slogova, može biti neefikasno. U tom slučaju koriste se metode koje smeštaju svih N slogova u datoteku podataka. Veličina datoteke podataka je M stranica, gde je M $>$ N/k, a k je kapacitet stranice izražen brojem slogova koje može da primi. Prazne stranice ili prazna mesta u stranicama datoteke podataka pomažu pri koliziji. Ove metode se zovu *metode heširanja sa otvorenim adresiranjem.*

Najjednostavnija od tih metoda je *linearni pokušaj*: kada se pojavi kolizija, pokušati sa sledećom (susednom) stranicom u datoteci podataka. Ako se tražena vrednost nađe u toj stranici, pretraživanje je uspešno i slog sa tom vrednošću atributa direktnog pristupa je rezultat pretraživanja (pretraživanje se može nastaviti do pronalaženja svih takvih slogova). Ako ta stranica nije popunjena a tražena vrednost na njoj nije nađena, pretraživanje je neuspešno. Ako je stranica popunjena a tražena vrednost na njoj nije nađena, pretraživanje se nastavlja na sledećoj stranici. U slučaju neuspešnog pretraživanja, slog sa datom vrednošću može se uneti na prvu nepopunjenu stranicu.

Algoritmi unošenja i pretraživanja koji realizuju direktnu organizaciju primenom metode linearnog pokušaja, opisuju se sledećim pseudoprogramima; primer 13.2 ilustruje algoritam unošenja.

Algoritam unošenja sa otvorenim adresiranjem

ulaz: slog s sa vrednošću v heš atributa A;
izlaz: broj stranice x na koju je unet ili na kojoj je nađen slog s;

```
BEGIN
  x := h(v);
  IF stranica(x) sadrži slog s
  THEN izdati poruku o postojanju sloga s
  ELSE BEGIN
         WHILE stranica(x) popunjena DO
           BEGIN x := (x + 1) mod M;
                 IF stranica(x) sadrži slog s THEN
                       EXIT(slog s nađen na strnici x)
           END;
         upisati slog s u stranicu x
       END;
  vratiti x
END.
```

Primer 13.2 Za skup vrednosti atributa SLOVO kao u primeru 13.1, ista heš funkcija ($h(v) = v \ mod$ M) za M=23 proizvodi sledeće heš adrese:

SLOVO:	P	R	I	M	E	R	P	R	E	T	R	A	Ž	I	V	A	NJ	A
adresa:	22	0	13	18	9	0	22	0	9	3	0	1	7	13	5	1	20	1

Ako na jednu stranicu staje samo jedan slog (k=1), sledeća tabela prikazuje postupno unošenje slogova sa navedenim vrednostima atributa SLOVO, u stranice datoteke podataka sa naznačenim brojem:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
																						P
R																						P
R													I									P
R													I					M				P
R									E				I					M				P
R	R								E				I					M				P
R	R	P							E				I					M				P
R	R	P	R						E				I					M				P
R	R	P	R						E	E			I					M				P
R	R	P	R	T					E	E			I					M				P
R	R	P	R	T	R				E	E			I					M				P
R	R	P	R	T	R	A			E	E			I					M				P
R	R	P	R	T	R	A	Ž		E	E			I					M				P
R	R	P	R	T	R	A	Ž		E	E			I	I				M				P
R	R	P	R	T	R	A	Ž	V	E	E			I	I				M				P
R	R	P	R	T	R	A	Ž	V	E	E	A		I	I				M				P
R	R	P	R	T	R	A	Ž	V	E	E	A		I	I				M		NJ		P
R	R	P	R	T	R	A	Ž	V	E	E	A	A	I	I				M		NJ		P

Algoritam heš pretraživanja sloga sa otvorenim adresiranjem

ulaz: heš atribut A i vrednost tog atributa v;
izlaz: skup brojeva stranica $x_1, \ldots, x_i$ datoteke podataka koje sadrže slog sa vrednošću v heš atributa; za neuspešno pretraživanje – poruka;

```
BEGIN
  i := 0;  x := h(v);
  IF stranica(x) sadrži slog sa vrednošću atributa A =v THEN
    BEGIN i := i + 1; x_i := x END;
  WHILE stranica(x) popunjena DO
    BEGIN x := (x + 1) mod M;
      IF stranica(x) sadrži slog sa vrednošću atributa A =v THEN
          BEGIN i := i + 1; x_i := x END
    END;
  IF i = 0 THEN izdati poruku da slog sa datom vrednošću v
      atributa A ne postoji
  ELSE vratiti x_1, ..., x_i
END.
```

Nedostatak metode linearnog pokušaja je što se pri traženju sloga sa datom vrednošću heš atributa ne prelazi samo preko vrednosti koje se preslikavaju u istu heš adresu, već i preko drugih vrednosti, naročito kad rezervisani skup stranica

počne da se popunjava. Tako, ako se pretražuju podaci uneti u primeru 13.2, pri traženju V-a prelazi se i preko R, A, Ž, koji nemaju istu heš adresu. Unošenje sloga sa jednom vrednošću heš adrese može drastično da uveća vreme pretraživanja slogova sa drugom vrednošću heš adrese. Umetanje na stranicu 2 u primeru 13.2 značajno uvećava vreme pretraživanja sloga čija je heš adresa 1. Ovo je problem grupisanja (clustering) kod linearnog pokušaja, koji se rešava drugom metodom otvorenog adresiranja, *dvostrukim heširanjem*. Umesto ispitivanja svake naredne stranice za stranicom u koliziji, uvodi se druga heš funkcija za fiksni korak (engl. increment) nad adresama, $u := h_2(v)$; pri tome se dodela $x := (x + 1)\ mod$ M (u algoritmima unošenja i pretraživanja sa otvorenim adresiranjem) zamenjuje dodelom $x := (x + u)\ mod$ M. Funkcija h_2 mora da se izabere pažljivo. Prvo, u mora biti različito od nule. Zatim, M i u treba da su uzajamno prosti (inače su mogući ciklusi).

Sledeći primer ilustruje metodu dvostrukog heširanja.

Primer 13.3 Neka je na skup vrednosti atributa SLOVO iz primera 13.1 i 13.2 primenjena osnovna heš funkcija $h(v)$=$v\ mod$ M, i druga heš funkcija $h_2(v)$ =M–2– $v\ mod$ (M-2). Za M=23, sledeće dve tabele prikazuju h i h_2 heš adrese, odnosno redosled unošenja slogova na stranice datoteke podataka:

SLOVO:	P	R	I	M	E	R	P	R	E	T	R	A	Ž	I	V	A	NJ	A
h :	22	0	13	18	9	0	22	0	9	3	0	1	7	13	5	1	20	1
h_2 :	20	19	8	3	12	19	20	19	12	16	19	20	12	8	14	20	1	20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
																						P
R																						P
R													I									P
R													I					M				P
R									E				I					M				P
R									E				I					M	R			P
R									E				I			P		M	R			P
R									E				I		R	P		M	R			P
R									E				I		R	P		M	R		E	P
R			T						E				I		R	P		M	R		E	P
R			T						E		R		I		R	P		M	R		E	P
R	A		T						E		R		I		R	P		M	R		E	P
R	A		T				Ž		E		R		I		R	P		M	R		E	P
R	A		T			I	Ž		E		R		I		R	P		M	R		E	P
R	A		T		V	I	Ž		E		R		I		R	P		M	R		E	P
R	A		T		V	I	Ž		E		R	A	I		R	P		M	R		E	P
R	A		T		V	I	Ž		E		R	A	I		R	P		M	R	NJ	E	P
R	A		T		V	I	Ž		E		R	A	I		R	P	A	M	R	NJ	E	P

Broj rezervisanih stranica kod metodâ otvorenog adresiranja veći je nego kod posebnog ulančavanja, jer je M > N/k (u primerima 13.1 i 13.2, za k=1 i N=18, kod posebnog ulančavanja potrebno je tačno 18 stranica, a kod otvorenog adresiranja 23). Međutim, ukupna potrošena memorija manja je nego kod posebnog ulančavanja jer nema pokazivača.

Broj zauzetih stranica datoteke podataka kod metode dvostrukog heširanja isti je kao i kod metode linearnog pokušaja. Isto važi i za prosečnu popunjenost (tzv. *faktor opterećenja*, a) stranice datoteke podataka. U primerima 13.2 i 13.3, za M=23, N=18, k=1, a=18/23/1∗100 = 78%. Vreme potrebno za uspešno pretraživanje kod metode dvostrukog heširanja je kraće; u primeru 13.3 ono iznosi 43/18=2.4, dok u primeru 13.2 (metodom linearnog pokušaja) ono iznosi 62/18=3.44. Heš funkcije treba da obezbede dobro vreme za uspešno pretraživanje pri popunjenosti datoteke podataka i do 90%. Ipak, tako pun skup stranica treba izbegavati.

Metode otvorenog adresiranja mogu se primeniti i u slučaju da se broj slogova ne može predvideti. Tada je moguće povremeno ponovno heširanje u veću datoteku podataka.

Za poboljšanje efikasnosti pretraživanja (i unošenja) kod otvorenog adresiranja, bez obzira da li se koristi metoda linearnog pokušaja ili dvostrukog heširanja, mogu se primeniti i ulančavanja. Tako, u slučaju da se slog ne nađe u stranici sa njegovom heš adresom, a da je stranica puna, traženje ne mora da se nastavi u svim narednim stranicama. Neka slog koji se traži ima heš adresu a. Tada slog može da se traži u lancu slogova (odnosno stranica) povezanih pokazivačima, koji počinje od hronološki prvog sloga sa istom heš adresom a. Naime, nijedan slog sa heš adresom a ne mora da bude na stranici sa adresom a (u vreme unošenja takvih slogova stranica je mogla da bude već popunjena slogovima sa heš adresama različitim od a). Zato svaka stranica datoteke podataka (npr. stranica sa brojem x) ima jedan pokazivač na hronološki prvi slog čija je heš adresa jednaka x; taj pokazivač možemo da označimo sa stranica(x).*prvi*. Svaki slog koji odgovara n-torci relacije sa vrednošću heš atributa v sadrži pokazivač na sledeći slog (na istoj ili različitoj stranici) sa istom heš adresom $h(v)$; taj pokazivač možemo da označimo sa slog(v).*sled*.

Algoritam pretraživanja za slučaj ulančavanja kod otvorenog adresiranja predstavlja analogon algoritma pretraživanja kod posebnog ulančavanja; za razliku od tog algoritma, ovaj algoritam koristi jednu fizičku datoteku – datoteku podataka, sa unapred određenim skupom stranica. Slično važi i za algoritam unošenja.

Sledeći pseudoprogrami prikazuju ova dva globalna algoritma.

Algoritam pretraživanja kod otvorenog adresiranja sa ulančavanjem

ulaz: heš atribut A i vrednost v atributa A;
izlaz: skup brojeva stranica $x_1, x_2, \ldots, x_n$ datoteke podataka, koje sadrže slogove sa vrednošću v atributa A; za neuspešno pretraživanje – poruka;

```
BEGIN
  i := 0;
  x := h(v);
  uzeti pokazivač stranica(x).prvi na hronološki prvi slog sa heš adresom x;
  WHILE postoji sledeći slog u lancu DO
        /* u prvom prolazu "sledeći" znači "prvi" */
    BEGIN broj stranice slog-a dodeliti x;
      IF vrednost atributa A slog-a = v i x <> x_i  /* za i > 0 */
      THEN /*stranica nije već uzeta u obzir */
        BEGIN i := i + 1; x_i := x END
    END;
  IF i = 0 THEN izdati poruku o neuspešnom pretraživanju
  ELSE vratiti x_1, ..., x_i
END.
```

Algoritam unošenja sloga kod otvorenog adresiranja sa ulančavanjem i kontrolom duplikata

ulaz: slog s sa vrednošću v heš atributa A;
izlaz: broj stranice x na koju je unet ili na kojoj je nađen slog s;

```
BEGIN
  x := h(v); /* heš funkcija */
  uzeti pokazivač stranica(x).prvi na hronološki prvi slog sa heš adresom x;
  WHILE postoji sledeći slog u lancu DO
    BEGIN broj stranice slog-a dodeliti x;
      IF slog = s
      THEN EXIT(slog s nađen na stranici x)
    END;
  WHILE stranica(x) popunjena DO x := (x + 1) mod M;
  uneti slog s na stranicu sa brojem x;
  postaviti polje sled poslednjeg slog-a u lancu (ili polje stranica(h(v)).prvi,
    ako je s prvi slog sa heš adresom h(v)) da pokazuje na slog s;
  vratiti x
END.
```

13.4 Efikasnost izvršavanja operacija

Izbor pojedinačnog sloga kod direktne reprezentacije je vrlo efikasna operacija (jedina efikasna operacija nad ovom reprezentacijom), i to u slučaju da se izbor vrši po heš atributu. Heš atribut može biti ma koji atribut ili kombinacija atributâ relacije. To ne mora biti primarni ključ, niti vrednosti heš atributa moraju biti jedinstvene unutar relacije. Ipak, poželjno je za heš atribut izabrati što je moguće "jedinstveniji" atribut. Na primer, atribut POL neke relacije, sa vrednostima {muški,

ženski}, nije dobar kandidat za heš atribut, dok lični broj to svakako jeste. Zato je važno maksimalno uskladiti izbor heš atributa sa atributom specifične restrikcije koja se najčešće javlja u kvalifikaciji upita. Najčešće se za heš atribut bira primarni ključ relacije.

Ako je faktor opterećenja $a = \mathrm{N}/(\mathrm{M}*\mathrm{k})$ (N je broj slogova, M – broj stranica datoteke podataka, k – kapacitet stranice u slogovima), onda se analitički izrazi za vreme utrošeno pri neuspešnom i uspešnom pretraživanju kod direktne organizacije mogu prikazati na sledeći način ([44], [55]):

	neuspešno	uspešno
posebno ulančavanje:	$1 + \frac{a}{2}$	$\frac{(a+1)}{2}$
linearni pokušaj:	$\frac{1}{2} + \frac{1}{2(1-a)^2}$	$\frac{1}{2} + \frac{1}{2(1-a)}$
dvostruko heširanje:	$\frac{1}{(1-a)}$	$-\frac{ln(1-a)}{a}$

U slučaju posebnog ulančavanja može biti $a > 1$, dok u slučaju otvorenog adresiranja (linearni pokušaj i dvostruko heširanje) mora biti $a < 1$.

Za malo a, pokazuje se da se svi navedeni izrazi za utrošeno vreme svode na osnovni rezultat – oko $1+\mathrm{N}/(\mathrm{M}*\mathrm{k})$ za neuspešno i $1+\mathrm{N}/(2\mathrm{M}*\mathrm{k})$ pokušaja za uspešno pretraživanje. Kada je a blisko jedinici, tj. pri velikoj popunjenosti stranica, otvoreno adresiranje daje veoma loše rezultate.

Iz efikasnosti izbora proističe efikasnost i svih ostalih operacija nad direktnom reprezentacijom.

Grupna obrada (ili obrada svih slogova) u proizvoljnom ili specifičnom redosledu vrši se na isti način kao kod neuređene varijante sekvencijalne datoteke.

Novi slog se unosi prema nekom od prethodnih algoritama (u zavisnosti od varijante direktne reprezentacije).

Brisanje sloga u slučaju posebnog ulančavanja svodi se na izbor i stvarno brisanje, uz izmene nad lancem pokazivača. Brisanje sloga u slučaju otvorenog adresiranja mora biti pažljivo izvedeno. Nije dozvoljeno prosto obrisati slog zbog moguće promene karakteristike stranice koja se pretražuje algoritmom pretraživanja: popunjena postaje nepopunjena, što može da sugeriše neuspešno pretraživanje slogova sa drugim vrednostima heš atributa; zbog toga se u ovoj varijanti direktne reprezentacije mora postaviti neki indikator "poluslobodne" lokacije.

Vrednost ne-heš atributa ažurira se na isti način na koji se vrši i izbor. Vrednost heš atributa se ažurira operacijom brisanja sloga za kojom sledi unošenje.

Prirodno spajanje dveju direktnih reprezentacija vrši se ili kao i kod sekvencijalne datoteke, pri čemu se zahteva sortiranje po atributima spajanja uz obavljanje restrikcije i projekcije, ili indeksnim spajanjem ako je to moguće, pri čemu se umesto

indeksa koristi heš algoritam za transformaciju vrednosti atributa direktnog pristupa u adresu, kada je taj atribut ujedno i atribut spajanja.

Vreme pristupa slogu kod direktne reprezentacije može biti vrlo malo, u srednjem je dosta dobro (naročito u slučaju dovoljno velike datoteke podataka, skoro konstantno vreme), ali u najgorem slučaju može biti i vrlo veliko (kada se veliki broj vrednosti heš atributa preslikava heš funkcijom u jednu istu adresu); prethodno navedena tabela sa analitičkim rezultatima vezanim za vreme pristupa to dobro ilustruje. Zbog toga se direktna reprezentacija smatra nestabilnom reprezentacijom. Poseban nedostatak direktne reprezentacije predstavlja činjenica da su slogovi u datoteci podataka DP u proizvoljnom redosledu (nesortiranom, nehronološkom, onom koji diktira heš funkcija), i da nema fizičkog grupisanja slogova pogodnog za grupne operacije. Ako ovi nedostaci nisu od velike važnosti za aplikaciju koja intenzivno pretražuje relaciju po specifičnim vrednostima nekog atributa, onda je heširanje po tom atributu pravi izbor fizičke reprezentacije relacije.

Najbolju metodu heširanja u konkretnoj situaciji je teško odabrati, ali često svaka od metoda radi dovoljno dobro. Svaka od metoda podržana je čitavim nizom tehnika ([44]) koje mogu biti adekvatne pod određenim uslovima. Navedene formule vremena pretraživanja ne opravdavaju izbor posebnog ulančavanja u odnosu na dvostruko heširanje. Ipak, u principu se na relacije nepredvidive veličine primenjuje neka od tehnika metode posebnog ulančavanja, a na relacije predvidive veličine neka od metoda otvorenog adresiranja (na primer, dvostruko heširanje).

13.5 Pitanja i zadaci

1. Opisati karakteristike direktne organizacije.
2. Šta je heš funkcija i koji su problemi u njenom konstruisanju?
3. Opisati postupak razrešavanja kolizije posebnim ulančavanjem.
4. Navesti primer primene algoritma pretraživanja sa posebnim ulančavanjem.
5. U čemu se sastoji postupak otvorenog adresiranja?
6. Navesti primer primene algoritma unošenja kod otvorenog adresiranja.
7. Koje su prednosti otvorenog adresiranja sa ulančavanjem?
8. Diskutovati efikasnost operacija nad direktnom reprezentacijom.
9. Direktna reprezentacija može da koristi i skup atributa direktnog pristupa (umesto jedinstvenog heš atributa). Tada se heš adresa sloga izračunava primenom heš funkcije na kombinaciju vrednosti skupa atributa direktnog pristupa.

   Neka relacija KI ima istu strukturu i sadržaj kao u zadatku 15 prethodne glave, i neka je predstavljena direktnom reprezentacijom sa parom (K_SIF,

I_SIF) kao skupom atributa direktnog pristupa. Konstruisati heš funkciju za ovu reprezentaciju i diskutovati prednosti i nedostatke različitih metoda razrešavanja kolizije.

Deo V

Upravljanje transakcijama

Deo **V** odnosi se na komponente sistema za upravljanje bazama podataka koje omogućuju istovremno, korektno i bezbedno izvršavanje zahteva većeg broja korisnika. Skup radnji koje predstavljaju logičku jedinicu posla zove se transakcija. Tako bi, na primer, u sistemu baza podataka jedne banke, prenošenje novca sa jednog računa na drugi predstavljalo jednu transakciju. Ona se sastoji od bar dve radnje: isplate sa jednog i uplate na drugi račun, ali su te radnje logički nedeljive i tek se obe zajedno smatraju jednim poslom. Na efekte transakcije ne sme da utiče eventualno istovremeno izvršenje druge transakcije, ili nestanak struje.

Deo **V** sastoji se od dve glave:

- U glavi 14, **Transakcija, integritet i konkurentnost**, prvo se uvodi pojam transakcije kao niza radnji koji ispunjava uslove integriteta baze podataka. Zatim se daje pregled problema koji nastaju pri istovremenom (konkurentnom) izvršavanju skupa transakcija, kao i načina za rešavanje tih problema.
- Glava 15, **Oporavak**, odnosi se na restauriranje podataka baze podataka u slučaju da dođe do greške u transakciji, u sistemu ili u mediju (disku). Razmatraju se aktivnosti koje komponenta SUBP – upravljač oporavkom preduzima da bi oporavljeni (restaurirani) sadržaj baze podataka zadovoljio (privremeno narušene) uslove integriteta baze.

14

Transakcija, integritet i konkurentnost

14.1 Transakcija i integritet

Transakcija je logička jedinica posla pri radu sa podacima. Ona predstavlja niz, tj. sekvencijalnu kompoziciju radnji koja ne narušava uslove integriteta. Po izvršenju kompletne transakcije stanje baze[1] treba da je *konzistentno*, tj. da su ispunjeni uslovi integriteta. Dakle, posmatrana kao jedinica posla, transakcija prevodi jedno konzistentno stanje baze u drugo takvo stanje baze, dok u međukoracima transakcije konzistentnost podataka može biti i narušena. Transakcija tako predstavlja i bezbedno sredstvo za interakciju korisnika sa bazom.

Transakcija se karakteriše sledećim važnim svojstvima (poznatim kao ACID svojstva):

- atomičnost (engl. atomicity): transakcija se izvrši u celosti ili se uopšte ne izvrši (ni jedna njena radnja);
- konzistentnost (engl. consistency): transakcija prevodi jedno konzistentno stanje baze u drugo konzistentno stanje baze;
- izolacija (engl. isolation): efekti izvršenja jedne transakcije su nepoznati drugim transakcijama sve dok se ona uspešno ne kompletira;
- trajnost (engl. durability): svi efekti uspešno kompletirane transakcije su trajni, tj. mogu se poništiti samo drugom transakcijom.

Primer 14.1 Neka je relacija KI (odeljak 1.2) proširena atributom VR_IZDANJA koji se odnosi na vrednost jednog izdanja (u dinarima) jedne knjige jednog izdavača,

[1] Stanje beze podataka je sadržaj baze u jednom momentu, kao što je i stanje relacije – sadržaj relacije u jednom momentu, v. glavu 8.

i neka je relacija I proširena atributom VREDNOST koji se odnosi na vrednost (u dinarima) svih izdanja jednog izdavača. Sada uslov integriteta baze može biti uslov da je vrednost atributa VREDNOST relacije I, za svakog pojedinačnog izdavača, jednaka zbiru vrednosti svih izdanja svih knjiga tog izdavača u relaciji KI. Kako se izdavač identifikuje vrednošću atributa I_SIF, ovaj uslov integriteta se može izraziti kao zahtev da je vrednost atributa VREDNOST relacije I, za svaku pojedinačnu vrednost atributa I_SIF, jednaka zbiru vrednosti atributa VR_IZDANJA svih izdanja u relaciji KI koja odgovaraju toj vrednosti atributa I_SIF.

Navedeni uslov integriteta može se eksplicitno zadati u SQL2 iskazom (koji u sistemu DB2 nije podržan)

```
CREATE ASSERTION PROVERA CHECK
 (NOT EXISTS ( SELECT *
               FROM   I
               WHERE  VREDNOST <> ( SELECT SUM (VR_IZDANJA)
                                    FROM   KI
                                    WHERE  I_SIF = I.I_SIF) ) )
```

Ako se aktivira SET CONSTRAINT iskazom, ovako zadat uslov integriteta se proverava i mora da važi na kraju svake transakcije. Obezbeđenje njegovog važenja postiže se odgovarajućom strukturom i sadržajem transakcije. Sledeći primer daje strukturu jedne transakcije.

Primer 14.2 Neka je transakcija T_1 – promena vrednosti atributa VR_IZDANJA drugog Prosvetinog izdanja knjige sa šifrom k1 u relaciji KI, kao i odgovarajuća promena vrednosti atributa VREDNOST u relaciji I. U nekim SQL dijalektima ova transakcija može se izraziti na sledeći način (sintaksa transakcije u sistemu DB2 je nešto drugačija):

```
       BEGIN TRANSACTION
(A)         UPDATE KI
                  SET VR_IZDANJA = VR_IZDANJA + 4000
                  WHERE K_SIF = 'k1' AND I_SIF = 'i1' AND IZDANJE = 2;
(B)         UPDATE I
                  SET VREDNOST  = VREDNOST  + 4000
                  WHERE I_SIF = 'i1';
       END TRANSACTION
```

U toku izvršavanja transakcije dozvoljena je privremena nekonzistentnost baze. Tako se transakcija ponaša kao jedinstvena radnja sa tačke gledišta uslova integriteta, što ne važi za njene pojedinačne radnje.

U sistemu DB2, za sadržaj transakcije i proveru uslova integriteta odgovoran je programer, tj. važenje uslova integriteta obezbeđuje aplikacija. Uobičajeno je da u radu sa transakcijama SUBP koristi usluge drugog sistema – upravljača transakcijâ. Sistem za upravljanje transakcijama obezbeđuje da se, bez greške, upisuju u bazu

podataka efekti izvršenja svih radnji od kojih se transakcija sastoji, ili nijedne, što je jedan aspekt kontrole konzistentnosti baze.

RSUBP DB2 koristi dve radnje upravljača transakcijâ, COMMIT i ROLLBACK (to su i iskazi SQL-a), od kojih svaka označava i kraj transakcije: COMMIT označava uspešan kraj transakcije i trajan upis efekata svih radnji transakcije u bazu, dok ROLLBACK označava neuspešan kraj transakcije i poništenje svih efekata koje su proizvele radnje te transakcije. Tako je u sistemu DB2 transakcija deo aplikacije od početka programa do prve radnje COMMIT ili ROLLBACK, odnosno između dve susedne takve radnje u programu, odnosno od poslednje takve radnje do kraja programa. Ako se nijedna od radnji COMMIT, ROLLBACK ne uključi u program, onda čitav program predstavlja jednu transakciju; na kraju programa se sistemski generiše jedna komanda COMMIT ako je kraj programa uspešan, odnosno jedna komanda ROLLBACK ako je kraj programa neuspešan. Ugnježdene transakcije u sistemu DB2 nisu moguće.

Deo programa u aplikativnom SQL-u, koji realizuje prethodnu transakciju u sistemu DB2 i koji odražava zadati uslov integriteta nad relacijama KI, I, može imati sledeći izgled:

```
        EXEC SQL WHENEVER SQLERROR GO TO PONISTI;
        EXEC SQL UPDATE KI
                 SET VR_IZDANJA = VR_IZDANJA + 4000
                 WHERE K_SIF = 'k1' AND I_SIF = 'i1' AND IZDANJE = 2;
        EXEC SQL UPDATE I
                 SET VREDNOST  = VREDNOST  + 4000
                 WHERE I_SIF = 'i1';
        EXEC SQL COMMIT;
                 GO TO ZAVRSENA;
PONISTI:  EXEC SQL ROLLBACK;
ZAVRSENA: RETURN;
```

14.2 Konkurentnost

Kako bazu podataka koristi istovremeno veći broj korisnika, nad bazom podataka se istovremeno može izvršavati više transakcija. Mada svaka od njih može ispunjavati uslove integriteta baze podataka ("dobro su definisane", kao što je bio slučaj sa transakcijom iz prethodnog primera), ponekad njihovo istovremeno izvršenje može da naruši te uslove. Izvršenje u kome se transakcije izvršavaju jedna za drugom je *serijsko* izvršenje, i ono ispunjava uslove integriteta baze, tj. ostavlja bazu u konzistentnom stanju. Pod *izvršenjem skupa transakcija* ovde se podrazumeva niz radnji od kojih se te transakcije sastoje. U tom nizu redosled radnji iz svake transakcije je nepromenjen, ali dve uzastopne radnje ne moraju pripadati istoj transakciji. Pojedinačna radnja izvršenja izvršava se u jednom *koraku*.

Da bi se definisalo izvršenje skupa transakcija koje ispunjava uslove integriteta a koje nije serijsko, potrebno je preciznije definisati izvršenje skupa transakcija i

ekvivalentnost dva izvršenja.

Primer 14.3 U ilustraciji definicija koje slede koristićemo, uz transakciju T_1 iz prethodnog primera, i transakciju T_2 kojom se povećava vrednost svih izdanja izdavača i1 za 10%. Transakcija T_2 se u SQL-u može zapisati kao:

```
        BEGIN TRANSACTION
(A')        UPDATE KI
                  SET VR_IZDANJA = VR_IZDANJA  * 1.1
                  WHERE  KI.I_SIF  = 'i1';
(B')        UPDATE I
                  SET VREDNOST = VREDNOST * 1.1
                  WHERE I.I_SIF = 'i1';
        END TRANSACTION
```

14.2.1 Izvršenja skupa transakcija

Kao osnovne komponente transakcije posmatraćemo *objekte* (npr. slogove) i dve radnje: *čitanje* i *upis* (tj. ažuriranje) ([30], [1]). Dve radnje u ovom modelu su *konfliktne* ako se obavljaju nad istim objektom a jedna od njih je radnja upisa. To znači da parovi konfliktnih radnji mogu biti samo parovi (čitanje, upis) i (upis, upis).[2]

Neformalno govoreći, dva izvršenja datog skupa transakcija su ekvivalentna ako imaju isti redosled izvršavanja svakog para konfliktnih radnji. Intuitivno je jasno da će dva ekvivalentna izvršenja proizvesti isti efekat na bazu, tj. da će oba, polazeći od stanja baze s, dovesti bazu u isto stanje s_1.

Sa jedne strane, potrebno je da izvršenje skupa transakcija ostavi bazu u konzistentnom stanju (dakle, da proizvede efekat serijskog izvršenja). Sa druge strane, potrebno je da veći broj transakcija (odnosno korisnika baze) istovremeno, neserijski, pristupi podacima u bazi. Zbog toga su od posebnog interesa ona izvršenja skupa transakcija koja su ekvivalentna serijskom a različita od serijskog.

Za skup transakcija T, skup objekata O nad kojima transakcije obavljaju radnje, i neko izvršenje I skupa transakcija T, redosled izvršavanja konfliktnih radnji karakteriše se tzv. relacijom zavisnosti.

Definicija 14.1 *Relacija zavisnosti* izvršenja I skupa transakcija T= $\{T_1, T_2\}$, u oznaci Z(I), jeste podskup Dekartovog proizvoda T×O×T, koji sadrži sve i samo trojke (T', o, T") (za T', T" $\in \{T_1, T_2\}$ i o∈O), sa sledećim svojstvima:

1. u i-tom koraku izvršenja I transakcija T' obavlja čitanje ili upis nad objektom o, a u kasnijem koraku j ($i < j$) transakcija T" obavlja čitanje ili upis nad tim istim objektom;
2. jedna od dve uočene radnje je upis;

[2] Ovi parovi su neuređeni, tj. poredak komponenti je nebitan.

3. između i-tog i j-tog koraka izvršenja I nema radnje upisa nad objektom o.

Analogno se definiše relacija zavisnosti izvršenja skupa transakcija T={T_1, T_2, ..., T_n}.

Sada se relacija ekvivalencije na skupu izvršenja skupa transakcija može formalno definisati sledećom definicijom.

Definicija 14.2 Dva izvršenja I_1, I_2 skupa transakcija T su *ekvivalentna* ako su im jednake relacije zavisnosti, tj. ako je $Z(I_1) = Z(I_2)$.

Definicija 14.3 Izvršenje I skupa transakcija T je *linearizovano* ako je ekvivalentno nekom serijskom izvršenju.

Primer 14.4 U transakcijama T_1, T_2 iz primera 14.2 i 14.3, radnje UPDATE (ažuriranja) mogu se smatrati radnjama upisa. Za skup tih transakcija {T_1, T_2} posmatrajmo sledeća tri izvršenja:

	I_1:		I_2:		I_3:	
korak:	T_1:	T_2:	T_1:	T_2:	T_1:	T_2:
1.		upis KI		upis KI		upis KI
2.	upis KI		upis KI			upis I
3.		upis I	upis I		upis KI	
4.	upis I			upis I	upis I	

Relacije zavisnosti ovih izvršenja su:

$Z(I_1) = \{(T_2, KI, T_1), (T_2, I , T_1)\}$,
$Z(I_2) = \{(T_2, KI, T_1), (T_1, I , T_2)\}$,
$Z(I_3) = \{(T_2, KI, T_1), (T_2, I , T_1)\}$,

i pri tom važi $Z(I_3) = Z(I_1)$. Izvršenje I_1 je linearizovano jer je ekvivalentno serijskom izvršenju I_3. Izvršenje I_2 nema ekvivalentno serijsko, pa ono nije linearizovano.

Za serijsko (tj. linearizovano) izvršenje I skupa transakcija T važi da je njegova relacija zavisnosti $Z(I)$ "antisimetrična", tj. da za proizvoljna dva objekta o_1, o_2 važi: $(T_1, o_1, T_2) \in Z(I) \Rightarrow (T_2, o_2, T_1) \notin Z(I)$. Relacija zavisnosti takvog izvršenja definiše na skupu transakcija T linearno uređenje $<$ na sledeći način: $T_1 < T_2$ ako je $(T_1, o, T_2) \in Z(I)$ za neki objekat o.

14.3 Problemi konkurentnosti

Pri *konkurentnom* (istovremenom) izvršenju skupa transakcija mogu nastati sledeći problemi.

1. problem: nekonzistentna analiza

Ako jedna transakcija (iz primerâ 14.2, 14.3) počne po završetku druge (redosled radnji: A, B, A', B' ili A', B', A, B), ili ako im se radnje izvršavaju naizmenično (redosled radnji: A, A', B, B' ili A', A, B', B), njihovo izvršenje je serijsko ili ekvivalentno serijskom izvršenju T_1;T_2 (ili T_2;T_1). Posle tog izvršenja baza će biti u konzistentnom stanju. Ako se jedna transakcija kompletno izvrši između prve i druge radnje druge transakcije (redosled radnji: A, A', B', B ili A', A, B, B'), baza će se posle izvršenja tog skupa transakcija naći u nekonzistentnom stanju.

Da bismo se uverili u ovo tvrđenje, razmotrimo efekte koje na bazu proizvodi redosled radnji A, A', B', B. Pretpostavimo, pri tom, da je vrednost 2. izdanja knjige k1 izdavača i1 jednaka a, a da je vrednost svih izdanja izdavača i1 jednaka b (i u relaciji I – vrednost atributa VREDNOST, i u relaciji KI – suma vrednosti atributa VR_IZDANJA za izdavača i1). Da bi izvršenje skupa transakcija $\{T_1, T_2\}$ ostavilo bazu u konzistentnom stanju, iznos b u obe relacije mora da se promeni za istu vrednost. Modifikovanom sintaksom relacione algebre, promene nad relacijama I i KI možemo da posmatramo na sledeći način:

- stanje baze pre izvršenja transakcija T_1, T_2:

  KI [K_SIF=k1,I_SIF=i1,IZDANJE=2][VR_IZDANJA] $= a$;
  SUM (KI [VR_IZDANJA]) za I_SIF = i1 $= b$;
  I [I_SIF=i1] [VREDNOST] $= b$;

- posle izvršenja radnje A:

  KI [K_SIF=k1,I_SIF=i1,IZDANJE=2][VR_IZDANJA] $= a$+4000;
  SUM (KI [VR_IZDANJA]) za I_SIF = i1 $= b$+4000;

- posle izvršenja radnje A':

  KI [K_SIF=k1,I_SIF=i1,IZDANJE=2][VR_IZDANJA] $= (a + 4000) * 1.1$
  $= a * 1.1 + 4400$
  SUM (KI [VR_IZDANJA]) za I_SIF = i1 $= (b$+4000$)$*1.1
  $= b * 1.1 + 4400$

  (svim izdanjima izdavača i1 vrednost je uvećana za 10%);

- posle izvršenja radnje B':

  I [I_SIF=i1] [VREDNOST] $= b * 1.1$;

- posle izvršenja radnje B:

  I [I_SIF=i1] [VREDNOST] $= b * 1.1 + 4000$.

Dakle, vrednost atributa VREDNOST relacije I za izdavača sa šifrom i1 je b∗1.1+4000, a zbir vrednosti svih izdanja (VR_IZDANJA) tog izdavača u relaciji KI je $b * 1.1 + 4400$. Prva vrednost (VREDNOST) uvećana je za 10% (zbog uvećanja vrednosti svih izdanja tog izdavača za 10%) i za 4000 dinara (jer je 2. izdanju knjige k1 izdavača i1 uvećana vrednost i za 4000) ali nije uvećana za 10% od tih 4000 dinara (mada je vrednost 2. izdanja knjige k1 izdavača i1 porasla i za 10% na uvećanje od 4000 dinara). Zato je vrednost atributa VREDNOST relacije I, za izdavača sa šifrom i1, manja za 400 dinara nego što bi trebalo da bude.

2. problem: izgubljeno ažuriranje

Neka se svaka od transakcija T_1, T_2 sastoji od čitanja i upisa istog sloga datoteke, pri čemu se prvo izvrše radnje čitanja, redom, transakcijâ T_1 i T_2, a zatim radnje upisa u istom redosledu. Na primer, transakcija T_1 može da pročita slog koji se odnosi na izdavača sa šifrom I_SIF = i2, zatim transakcija T_2 može da pročita isti slog (obe transakcije čitaju identične podatke; posebno, za atribut STATUS, obe čitaju vrednost 20). Posle toga transakcija T_1 može da promeni vrednost atributu STATUS tog sloga (sa 20 na 30) i da završi sa radom, npr. da zatvori datoteku, a zatim transakcija T_2 može da promeni vrednost atributu STATUS (sa pročitane vrednosti 20 na 40) i da završi sa radom (npr. da zatvori svoju verziju iste datoteke). U tom slučaju ostaju zapamćene samo promene transakcije T_2 koja je poslednja izvršila upis i zatvorila datoteku (vrednost polja STATUS sloga o izdavaču i2 ostaje 40), a transakciji T_1 se bez obaveštenja poništava efekat upisa, tj. gubi se vrednost 30 polja STATUS posmatranog sloga.

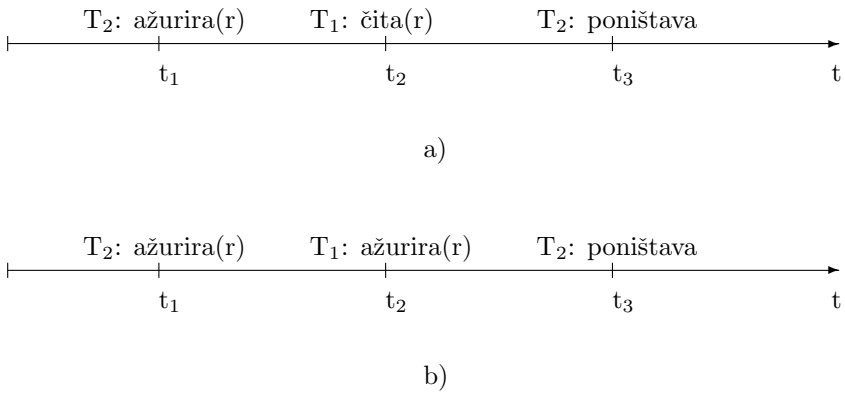
3. problem: zavisnost od poništenog ažuriranja

Pretpostavimo da transakcija T_1 uključuje čitanje nekog sloga r, a da transakcija T_2 uključuje upis (tj. ažuriranje) istog sloga. Neka pri konkurentnom izvršavanju ovih transakcija prvo transakcija T_2 ažurira slog r, a zatim transakcija T_1 pročita slog r i završi svoje izvršavanje. U nekom kasnijem trenutku, iz bilo kog razloga, transakcija T_2 poništi sve efekte koje je do tog trenutka proizvela na bazu, tj. poništi ažuriranje sloga r (slika 14.1 a)). U tom slučaju transakcija T_1 pročitala je ažuriranu vrednost sloga r koja nije ni trebalo da bude upisana u bazu.

Slično, neka transakcija T_1 uključuje ažuriranje nekog sloga r, a transakcija T_2 se sastoji od istih radnji kao i u prethodnom slučaju. Pri istom redosledu izvršavanja radnji i poništenju transakcije T_2 (slika 14.1 b)) izgubilo bi se, bez obaveštenja, ažuriranje transakcije T_1, jer bi poništavanjem efekata transakcije T_2 vrednost sloga r bila vraćena na vrednost pre ažuriranja (od strane transakcije T_2).

Prva dva problema konkurentnosti vezana su za redosled konkurentnog izvršavanja radnji dve ili više transakcija. Treći problem je, osim za konkurentnost, vezan i za oporavak baze podataka pri poništavanju transakcije u konkurentnom izvršenju (v. sledeću glavu). Do poništavanja transakcije može doći, na primer, zbog greške u transakciji ili zbog pada sistema.

Sva tri problema rešavaće se (i rešavaju se u RSUBP) pomoću mehanizama *zaključavanja*. Ovi mehanizmi obezbeđuju konkurentno izvršenje skupa transakcija



Slika 14.1: Zavisnost a) čitanja b) ažuriranja od poništenog ažuriranja

koje je ekvivalentno serijskom. Za serijsko izvršenje važi da ne narušava integritet i konzistentnost baze ako to isto važi za svaku pojedinačnu transakciju u posmatranom skupu transakcija.

14.4 Zaključavanje

Problemi konkurentnosti opisani u prethodnom odeljku mogu se rešiti mehanizmom dinamičkog zaključavanja i otključavanja objekata, koji obezbeđuje linearizovanost izvršenja. Linearizovana izvršenja, s obzirom da su ekvivalentna serijskim, ne poznaju ni jedan od pomenutih problema.

Moguće je realizovati različitu *granularnost*, tj. veličinu objekata zaključavanja; na primer, objekti zaključavanja mogu biti cele relacije, pojedinačne n-torke relacije, delovi n-torki, grupe atributa, skupovi n-torki koje zadovoljavaju zadati uslov, itd.

Zaključavanje objekta (postavljanje *katanca* na objekat) je postupak koji obezbeđuje transakciji pristup objektu, i kojim transakcija istovremeno sprečava druge transakcije da pristupe tom objektu. Svaka transakcija na kraju svog izvršavanja otključava sve objekte koje je sama zaključala (a koje nije već otključala). Pretpostavimo da u jednostavnom modelu postoji samo jedna vrsta katanca, i da pri izvršenju skupa transakcija nijedna transakcija ne može zaključati već zaključani objekat. (U realnim modelima transakcijâ postoji više vrsta katanaca, od kojih su neki deljivi a neki ekskluzivni; pri tom više transakcija može da postavi deljivi katanac na jedan objekat, ali najviše jedna može da postavi ekskluzivni katanac).

Svojstvo transakcijâ koje, koristeći mehanizam zaključavanja, obezbeđuje linearizovanost izvršenja skupa transakcija, jeste *dvofaznost*. Transakcija je dvofazna

ako se sastoji od dve serijske faze: faze zaključavanja objekata i faze otključavanja objekata. U fazi zaključavanja nema nijedne radnje otključavanja objekta, odnosno u fazi otključavanja više nema radnji zaključavanja (naravno, obe ove faze uključuju i druge operacije nad objektima kao što su čitanje, obrada, upis objekata). Faza otključavanja objekata počinje prvom radnjom otključavanja.

Teorema 14.1 *Dovoljan uslov za linearizovanost izvršenja I skupa transakcija* $\{T_1, T_2\}$ *je da su one dvofazne.*

Dokaz: Pretpostavimo da za neki objekat o trojka (T_1, o, T_2) jeste element relacije zavisnosti izvršenja I, tj. $(T_1, o, T_2) \in Z(I)$. Pokazaćemo da je onda izvršenje I ekvivalentno serijskom $T_1; T_2$, tj. da je linearizovano (sasvim analogno se pokazuje da, ako je trojka $(T_2, o, T_1) \in Z(I)$, onda je izvršenje I ekvivalentno serijskom $T_2; T_1$, tj. opet je linearizovano).

U suprotnom, neka postoji objekat o' takav da je $(T_2, o', T_1) \in Z(I)$. To znači da:

1. zbog $(T_1, o, T_2) \in Z(I)$, postoje koraci $k, m\,(k < m)$ izvršenja I u kojima transakcija T_1 otključava objekat o (korak k), odnosno transakcija T_2 zaključava objekat o (korak m);
2. zbog $(T_2, o', T_1) \in Z(I)$, postoje koraci $p, q\,(p < q)$ izvršenja I u kojima transakcija T_2 otključava objekat o' (korak p), odnosno transakcija T_1 zaključava objekat o' (korak q);
3. iz dvofaznosti transakcija T_1 i T_2 sledi da je $q < k$ i $m < p$;
4. iz 1) i 3) sledi $q < k < m < p$ što je u kontradikciji sa 2). To znači da ne postoji objekat o' takav da je $(T_2, o', T_1) \in Z(I)$, tj. za svaki objekat o' nad kojim operišu obe transakcije T_1, T_2 važi: $(T_1, o', T_2) \in Z(I)$. Dakle, izvršenje I je linearizovano.

Indukcijom se može dokazati važenje prethodnog tvrđenja i za svaki konačan skup transakcija $\{T_1, T_2, \ldots, T_n\}$.

Dvofaznost skupa transakcija nije potreban uslov za linearizovanost proizvoljnog izvršenja, što se može jednostavno dokazati primerom.

Osim što obezbeđuje linearizovanost, dvofaznost transakcija može proizvesti i neželjeni efekat poznat kao *uzajamno blokiranje transakcija* (engl. deadlock).

Primer 14.5 Za dve dvofazne transakcije

T_1:	zaključati KI	T_2:	zaključati I
	upisati KI		upisati I
	zaključati I		zaključati KI
	upisati I		upisati KI
	otključati KI		otključati I
	otključati I		otključati KI,

jedno njihovo delimično izvršenje je:

T_1: zaključati KI
T_1: upisati KI
T_2: zaključati I
T_1: upisati I.

Nijedna transakcija ne može da nastavi sa radom, pa se jedna transakcija nasilno prekida uz poništenje svih njenih radnji i oslobađanje katanaca; zatim se ta transakcija ponovno aktivira.

Postoji niz specifičnosti vezanih za realne modele transakcija i zaključavanja u odnosu na izloženi jednostavni model. Na primer, za model zaključavanja u sistemu DB2 važi sledeće:

- Zbog povećanja konkurentnosti, svojstvo izolovanosti transakcije (I u ACID) realizuje se u nekoliko nivoa koji se među sobom razlikuju po stepenu u kome operacije jedne transakcije mogu da utiču na izvršenje operacija konkurentnih transakcija, odnosno stepen u kome operacije drugih konkurentnih transakcija mogu da utiču na izvršenje operacija posmatrane transakcije. To su tzv. *nivoi izolovanosti transakcije* (tj. celokupne aplikacije).
  - Najviši nivo izolovanosti aplikacije (i njenih transakcija) u sistemu DB2 je nivo *ponovljivog čitanja* – RR (engl. repeatable read). Ovaj nivo obezbeđuje zaključavanje svih vrsta kojima se transakcija obraća (npr. cele tabele), a ne samo onih vrsta koje zadovoljavaju uslov upita. Tako se rezultat izvršenja jednog upita (čitanja) transakcije T ne može promeniti od strane druge transakcije pre nego što se transakcija T završi. S druge strane, ovaj nivo izolovanosti obezbeđuje i da transakcija nema uvid u nepotvrđene promene drugih transakcija.
  - Sledeći nivo izolovanosti jeste nivo *stabilnosti čitanja* – RS (engl. read stability). Za razliku od prethodnog, ovaj nivo izolovanosti obezbeđuje zaključavanje samo onih vrsta koje zadovoljavaju uslov upita, tj. ne dopušta promenu (od strane drugih transakcija) vrste koju je pročitala transakcija T, sve do završtka transakcije T. Mada će, pri ovom nivou izolovanosti, vrsta koju je pročitala transakcija T ostati nepromenjena od strane drugih transakcija sve do završtka transakcije T, rezultati ponovljenih izvršenja istog upita od strane transakcije T mogu da se razlikuju (može da dođe do dodavanja, od strane drugih transakcija, vrsta koje će zadovoljiti taj upit, tj. do fantomskih redova).
  - Nivo *stabilnosti kursora* – CS (engl. cursor stability) obezbeđuje zaključavanje samo one vrste koju transakcija T trenutno čita. Sve vrste koje je transakcija T prethodno pročitala (ali ne i menjala) – otključane su i mogu ih druge transakcije menjati i pre okončanja transakcije T. Transakcija T i dalje nema uvid u promene drugih transakcija pre okončanja tih transakcija.

  - Najslabiji nivo izolovanosti transakcije je nivo nepotvrđenog čitanja – UR (engl. uncommitted read). Ovaj nivo omogućuje transakciji T da "vidi" (pročita) nepotvrđene promene drugih transakcija, kao i drugim transakcijama da pristupe podacima koje transakcija T upravo čita.

- Objekat koji transakcija sistemski zaključava zavisi od nivoa izolovanosti transakcije i operacije i može biti različite *granularnosti* – vrsta tabele, tabela, prostor tabela. Jedini objekat koji korisnik može eksplicitno da zaključa – deljivim (S) ili isključivim (X) katancem (v. sledeći paragraf) jeste tabela (LOCK TABLE iskaz); održavanje jednog ključa nad celom tabelom je efikasnije od zaključavanja pojedinačnih vrsta, ali smanjuje konkurentnost obrade; zato je zaključavanje cele tabele adekvatno u slučaju da transakcija obrađuje veliki broj slogova te tabele.

- Dva najvažnija načina upotrebe katanaca su *deljivi katanac* – S-katanac (engl. shared), koji se postavlja pri čitanju objekta, i *isključivi katanac* – X-katanac (engl. exclusive), koji se postavlja pri upisu u objekat. Ovi se katanci mogu postaviti kako na pojedinačnu vrstu tako i na celu tabelu. Takav je još samo tzv. katanac ažuriranja – U-katanac (engl. update), koji omogućuje menjanje objekta (tabele ili vrste) i podrazumeva dodelu X-katanca promenjenoj vrsti. Neki katanci nad istim objektom su kompatibilni (može ih istovremeno postaviti više transakcija), a neki nisu. Sledeća tabela ilustruje kompatibilnost S, U, X katanaca nad istim objektom od strane dve transakcije (jedna poseduje katanac, druga ga traži):

T_1 \ T_2	S	U	X
S	D	D	N
U	D	N	N
X	N	N	N

- Postoje i druge vrste katanaca u sistemu DB2, koje izražavaju nameru transakcije da pročita vrednost objekta, odnosno da promeni vrednost objekta (npr. namera da čita podatke iz tabele – IS, engl. intent share, namera da promeni vrednost u tabeli – IX, engl. intent exclusive, čitanje sa namerom promene – SIX, engl. shared with intent exclusive), pravo menjanja podataka uz dopuštenje drugim transakcijama da čitaju podatke – U, engl. update, pri čemu svako menjanje proizvodi isključivi katanac na promenjenoj vrsti, itd. [38].

- Katanci se drže do kraja transakcije, tj. do njene potvrde ili poništenja (osim u slučaju nivoa izolovanosti koji ne obezbeđuju u potpunosti ACID svojstva transakcija – CS i UR). Pri ovakvom pristupu dvofaznost transakcija

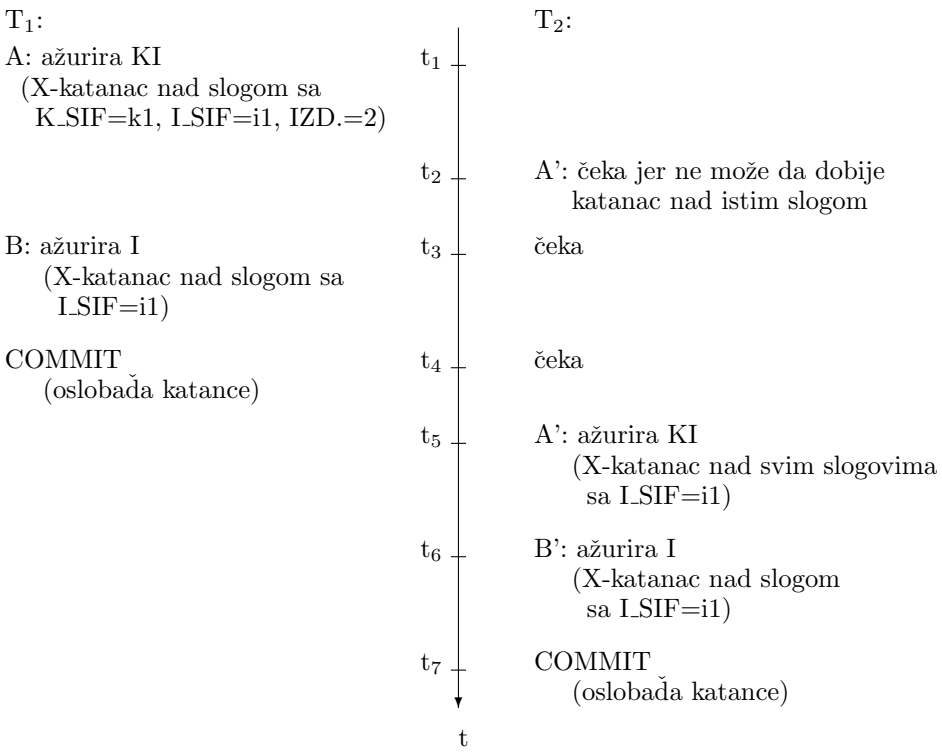
je očita, ali je izvršenje transakcija nad istim objektima gotovo serijsko (smanjena konkurentnost). Stoga se pribegava "usitnjavanju" transakcija, što opet povećava izglede za uzajamno blokiranje.

- Transakcija dobija katance od sistema, automatski (bez eksplicitnog zahteva): kad transakcija otvori kursor za čitanje, automatski dobija IS katanac nad odgovarajućom tabelom; kada uspešno obavi čitanje vrste, automatski dobija S-katanac na toj vrsti; kada otvori kursor za ažuriranje, dobija IX ili U katanac nad tabelom; kada uspešno obavi ažuriranje vrste, dobija X-katanac na vrsti, itd. Pojedinačne vrste mogu biti zaključane samo S, U ili X katancima, dok se "katanci namere" mogu postaviti samo na tabele ili prostore tabela.

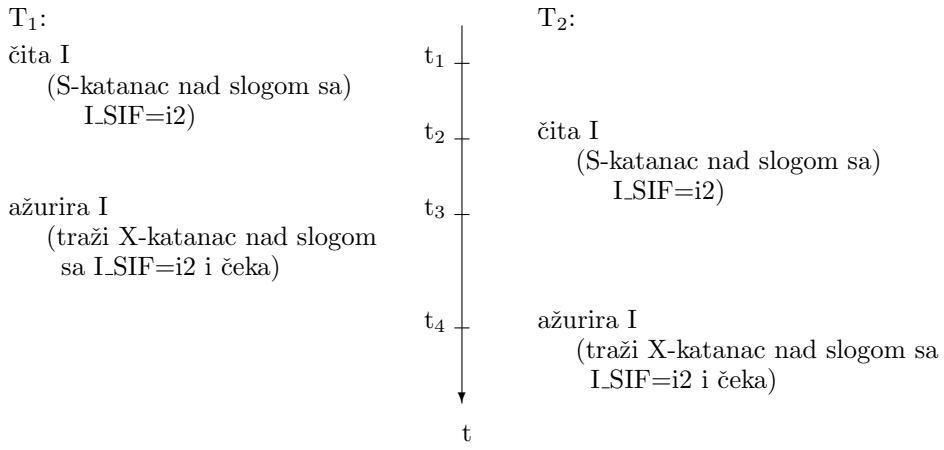
14.4.1 Rešenje problema konkurentnosti

Primenom mehanizma zaključavanja uz dvofaznost transakcija, rešenja tri problema konkurentnosti mogu se grafički predstaviti na sledeći način (t predstavlja vremensku osu):

Rešenje problema nekonzistentne analize:

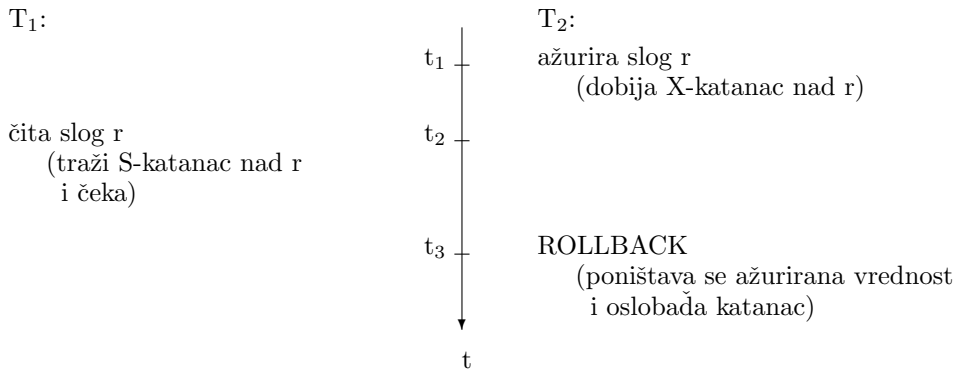


Rešenje problema izgubljenog ažuriranja:



Dakle, došlo je do uzajamnog blokiranja, pa je rešavanje problema izgubljenog ažuriranja dovelo do novog problema. Sistem DB2 rešava problem uzajmnog blokiranja tako što ga ne sprečava već ga, kada se dogodi, detektuje. Tada izabira jednu transakciju za "žrtvu", poništava je ili zahteva od nje da sama sebe poništi, pri čemu se oslobađaju svi katanci koje je ta transakcija držala.

Rešenje problema zavisnosti od poništenog ažuriranja:



Sledeći primer ilustruje problem uzajamnog blokiranja u realističnoj situaciji izvršenja većeg broja transakcija nad većim brojem objekata.

Primer 14.6 Neka se konkurentno izvršavaju transakcije T_1, T_2, ..., T_{12} nad slogovima obeleženim sa A,B,C,D,E,F,G,H. U sledećoj tabeli navedene su radnje koje

ove transakcije izvršavaju u navedenim vremenskim trenucima nad navedenim slogovima ([24]) (odgovarajući katanci se dobijaju pri čitanju odnosno ažuriranju, a oslobađaju se samo pri uspešnom kompletiranju ili poništenju transakcije).

vreme	trans.	radnja	vreme	trans.	radnja
t_1:	T_1:	čitaj A	t_{19}:	T_9:	upiši G
t_2:	T_2:	čitaj B	t_{20}:	T_8:	čitaj E
t_3:	T_1:	čitaj C	t_{21}:	T_7:	COMMIT
t_4:	T_4:	čitaj D	t_{22}:	T_9:	čitaj H
t_5:	T_5:	čitaj A	t_{23}:	T_3:	čitaj G
t_6:	T_2:	čitaj E	t_{24}:	T_{10}:	čitaj A
t_7:	T_2:	upiši E	t_{25}:	T_9:	upiši H
t_8:	T_3:	čitaj F	t_{26}:	T_6:	COMMIT
t_9:	T_2:	čitaj F	t_{27}:	T_{11}:	čitaj C
t_{10}:	T_5:	upiši A	t_{28}:	T_{12}:	čitaj D
t_{11}:	T_1:	COMMIT	t_{29}:	T_{12}:	čitaj C
t_{12}:	T_6:	čitaj A	t_{30}:	T_2:	upiši F
t_{13}:	T_5:	ROLLBACK	t_{31}:	T_{11}:	upiši C
t_{14}:	T_6:	čitaj C	t_{32}:	T_{12}:	čitaj A
t_{15}:	T_6:	upiši C	t_{33}:	T_{10}:	upiši A
t_{16}:	T_7:	čitaj G	t_{34}:	T_{12}:	upiši D
t_{17}:	T_8:	čitaj H	t_{35}:	T_4:	čitaj G
t_{18}:	T_9:	čitaj G	t_{36}:	...	...

Pitanje koje se postavlja je: da li u trenutku t_{36} ima uzajamnog blokiranja?

Iz analize transakcijâ sledi da su transakcije T_1, T_7 i T_6 uspešno završile rad, a da je transakcija T_5 poništena. Iz grafa preostalih transakcija i katanaca koje one drže nad pojedinim slogovima (slika 14.2) sledi da u trenutku t_{36} nijedna od njih ne radi ništa korisno, tj. da su uzajamno blokirane.

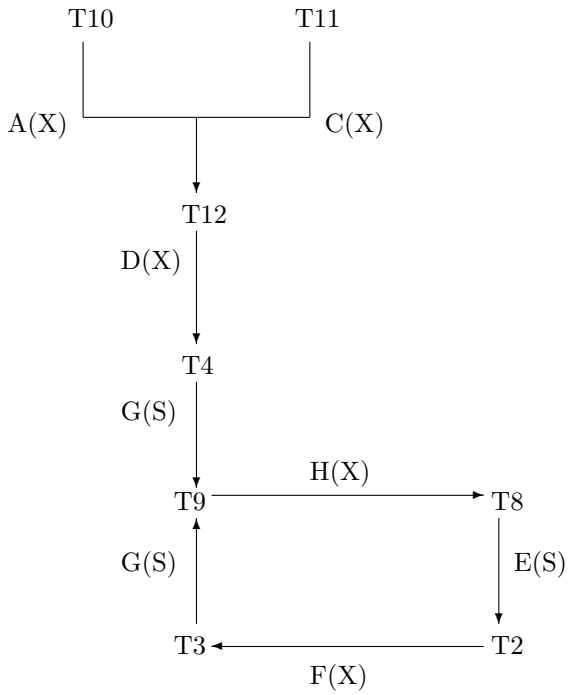
14.5 Pitanja i zadaci

1. Definisati sledeće pojmove i navesti primere:

transakcija	linearizovano izvršenje
integritet	zaključavanje
konkurentnost	katanac
izvršenje skupa transakcija	granularnost
relacija zavisnosti	dvofaznost transakcija
ekvivalentna izvršenja	uzajamno blokiranje transakcija
serijsko izvršenje	graf čekanja

2. Objasniti ACID svojstva transakcija.

3. Formulisati i objasniti probleme konkurentnosti.



Slika 14.2: Graf čekanja transakcijâ na X i S katance

4. Kako se rešavaju problemi konkurentnosti?

5. Date su transakcije T_1, T_2 i T_3:

T_1:	čitaj r	T_2:	čitaj s	T_3:	čitaj t
	ažur. r		čitaj t		čitaj s
	čitaj s		ažur. t		ažur. s
	čitaj t		čitaj r		čitaj r
	ažur. t				ažur. r

   (r, s i t su slogovi).

   Navesti sva moguća izvršenja skupa transakcija $\{T_1, T_2, T_3\}$ u slučaju da se ne primenjuje mehanizam zaključavanja. Koja su od tih izvršenja linearizovana?

6. Neka su date transakcije kao u prethodnom primeru. Navesti moguća izvršenja (ili delimična izvršenja) tog skupa transakcija, ako se koriste deljivi i ekskluzivni katanci. Koja od tih izvršenja dovode do uzajamnog blokiranja? Predložiti kriterijume poništavanja kojima se razrešava uzajamno blokiranje.

7. Konstruisati jedno linearizovano izvršenje skupa transakcija od kojih bar jedna nije dvofazna.

15

Oporavak

Oporavak podrazumeva aktivnost koju sistem preduzima u slučaju da se, u toku izvršenja jedne ili više transakcija, otkrije neki razlog koji onemogućava njihovo uspešno kompletiranje. Taj razlog može biti:

- u samoj transakciji, kao što je prekoračenje neke od dozvoljenih vrednosti (*pad transakcije*),
- u sistemu, npr. prestanak električnog napajanja (*pad sistema*), ili
- u disku na kome je baza podataka, npr. oštećenje glavâ diska (*pad medija*).

Kako je transakcija logička jedinica posla, njene radnje moraju ili sve uspešno da se izvrše ili nijedna, pa je u slučaju nemogućnosti uspešnog kompletiranja neophodno poništiti efekte parcijalno izvršenih transakcija. Sa druge strane, može se dogoditi da u trenutku pada sistema neki efekti uspešno kompletiranih transakcija još nisu upisani u bazu (npr. ažurirani slogovi su još u unutrašnjoj memoriji – u baferu podataka čiji se sadržaj pri padu sistema gubi); zato može postojati potreba za ponovnim (delimičnim) izvršenjem nekih prethodno uspešno kompletiranih transakcija. To poništavanje odnosno ponovno izvršavanje u cilju uspostavljanja konzistentnog stanja baze predstavlja sistemsku aktivnost *oporavka* od pada transakcije ili sistema. Transakcija je, dakle, i logička jedinica oporavka. U slučaju pada medija, oporavak uključuje pre svega prepisivanje arhivirane kopije baze (npr. sa trake) na ispravan medij, a zatim, eventualno, ponovno izvršenje transakcija kompletiranih posle poslednjeg arhiviranja a pre pada medija.

Ovako koncipiran oporavak nužno se zasniva na postojanju ponovljenih (dupliranih) podataka i informacija na različitim mestima ili medijima, tj. na mogućnosti rekonstrukcije informacije na osnovu druge informacije, ponovljeno smeštene na drugom mestu u sistemu. Kao "drugo" mesto u sistemu, na koje se (osim u bazu) upisuju informacije o izvršenim radnjama, obično se koristi tzv. *log datoteka* (*sistemski log*, *dnevnik ažuriranja*).

Komponenta SUBP odgovorna za oporavak baze podataka od pada transakcije, sistema ili medija je *upravljač oporavka*. Njegova aktivnost se globalno sastoji od sledećih radnji:

- periodično prepisuje (engl. dump) celu bazu podataka na medij za arhiviranje;
- pri svakoj promeni baze podataka, upisuje slog promene u log datoteku, sa tipom promene i sa novom i starom vrednošću pri ažuriranju, tj. sa novom vrednošću pri unošenju u bazu i starom vrednošću pri brisanju iz baze;
- u slučaju pada transakcije ili sistema, stanje baze podataka može biti nekonzistentno; upravljač oporavka koristi informacije iz log datoteke da poništi dejstva parcijalno izvršenih transakcija odnosno da ponovo izvrši neke kompletirane transakcije; arhivirana kopija baze podataka se ne koristi;
- u slučaju pada medija (npr. diska na kome je baza podataka), "najsvežija" arhivirana kopija baze podataka se prepisuje na ispravni medij (disk), a zatim se koriste informacije iz log datoteke za ponovno izvršenje transakcija kompletiranih posle poslednjeg arhiviranja a pre pada medija.

Da bi se omogućilo upravljaču oporavka da poništi odnosno da ponovo izvrši transakcije, potrebno je obezbediti procedure kojima se poništavaju odnosno ponovo izvršavaju pojedinačne radnje tih transakcija kojima se menja baza podataka. Baza podataka menja se operacijama ažuriranja (u užem smislu), unošenja ili brisanja podataka, pa pored procedura za izvršenje tih operacija, SUBP mora biti snabdeven i odgovarajućim procedurama za poništavanje odnosno ponovno izvršenje tih operacija, na osnovu starih odnosno novih vrednosti zapamćenih u sistemskom logu. Drugim rečima, SUBP poseduje, osim tzv. DO-logike ("uradi"), i tzv. UNDO ("poništi") i REDO ("ponovo uradi") logiku.

Pad sistema ili medija može se dogoditi i u fazi oporavka od prethodnog pada. Zato može doći do ponovnog poništavanja već poništenih radnji, odnosno do ponovnog izvršavanja već izvršenih radnji. Ova mogućnost zahteva da UNDO i REDO logika imaju svojstvo idempotentnosti ([35]), tj. da je UNDO(UNDO(x)) $\equiv$ UNDO(x) i REDO(REDO(x)) $\equiv$ REDO(x) za svaku radnju x.

U log datoteci pokazivačima su povezani slogovi koji se odnose na jednu transakciju. To podrazumeva čuvanje log datoteke na mediju sa direktnim pristupom, npr. na disku. S druge strane, količina podataka koji se upisuju u log datoteku može biti veoma velika (nekoliko stotina miliona bajtova dnevno). Zato se log datoteka obično organizuje tako da ima svoj aktivni deo na mediju sa direktnim pristupom, i arhivirani deo (npr. na traci), u koji se prepisuje aktivni log kadgod se prepuni.

Pretpostavlja se da log datoteka nikada, ili veoma retko, "pada", tj. da je, zahvaljujući sopstvenom dupliranju, tripliranju, itd, na raznim medijima, uvek dostupna.

15.1 Oporavak od pada transakcije

Jedan aplikativni program može se sastojati od većeg broja transakcija. Izvršenje jedne transakcije može da se završi planirano ili neplanirano. Do planiranog završetka dolazi izvršenjem COMMIT operacije kojom se uspešno kompletira transakcija, ili eksplicitne ROLLBACK operacije, koja se izvršava kada dođe do greške za koju postoji programska provera (tada se radnje transakcije poništavaju a program nastavlja sa radom izvršenjem sledeće transakcije). Neplanirani završetak izvršenja transakcije događa se kada dođe do greške za koju ne postoji programska provera; tada se izvršava implicitna (sistemska) ROLLBACK operacija, radnje transakcije se poništavaju a program prekida sa radom.

Izvršavanjem operacije COMMIT, efekti svih ažuriranja transakcije postaju trajni, tj. više se ne mogu poništiti procedurom oporavka. U log datoteku se upisuje odgovarajući slog o kompletiranju transakcije (COMMIT slog), a svi katanci koje je transakcija držala nad objektima – oslobađaju se. Izvršenje COMMIT operacije ne podrazumeva fizički upis svih ažuriranih podataka u bazu (neki od njih mogu biti u baferu podataka, pri čemu se, efikasnosti radi, sa fizičkim upisom u bazu čeka dok se baferi ne napune). Činjenica da efekti ažuriranja postaju trajni znači da se može garantovati da će se upis u bazu dogoditi u nekom narednom trenutku; u slučaju pada sistema, na primer, posle COMMIT operacije a pre fizičkog upisa podataka iz bafera u bazu, upis se garantuje REDO-logikom.

Pad transakcije nastaje kada transakcija ne završi svoje izvršenje planirano. Tada sistem izvršava implicitnu – prinudnu ROLLBACK operaciju, tj. sprovodi aktivnost oporavka od pada transakcije.

Pojedinačne radnje ažuriranja u okviru jedne transakcije, kao i operacija početka transakcije (eksplicitna ili implicitna BEGIN TRANSACTION operacija), izvršavaju se na način koji omogućuje oporavak baze u slučaju pada transakcije. Oporavak podataka u bazi i vraćanje baze u konzistentno stanje omogućuje se upisom svih informacija o ovim radnjama i operacijama u sistemski log.

Pri izvršavanju operacija ažuriranja (u užem smislu), osim što se ažurira baza, u log se beleže vrednosti svakog ažuriranog objekta pre i posle ažuriranja (uz ostale informacije o tom ažuriranju).

Izvršavanjem operacije BEGIN TRANSACTION (bilo da je eksplicitna ili implicitna) u log datoteku se upisuje slog početka transakcije.

Operacija ROLLBACK, bilo eksplicitna ili implicitna, sastoji se od poništavanja učinjenih promena nad bazom. Izvršava se čitanjem unazad svih slogova iz log datoteke koji pripadaju toj transakciji, do BEGIN TRANSACTION sloga. Za svaki slog, promena se poništava primenom odgovarajuće UNDO-operacije.

Aktivnost oporavka od pada transakcije ne uključuje REDO logiku.

15.2 Oporavak od pada sistema

U slučaju pada sistema, sadržaj unutrašnje memorije je izgubljen. Zato se, po ponovnom startovanju sistema, za oporavak koriste podaci iz log datoteke da bi se poništili efekti transakcija koje su bile u toku u trenutku pada sistema. Ove transakcije se mogu identifikovati čitanjem sistemskog loga unazad, kao transakcije za koje postoji BEGIN TRANSACTION slog ali ne postoji COMMIT slog.

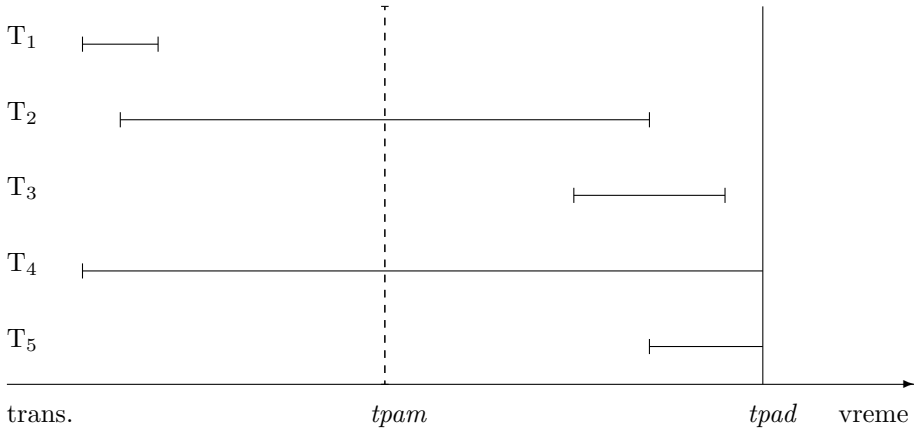
Da bi se smanjila količina posla vezana za poništavanje transakcija, uvode se, u pravilnim intervalima, obično posle određenog broja upisanih slogova u log datoteku, *tačke pamćenja*. U momentu koji određuje tačku pamćenja, fizički se upisuju podaci i informacije iz log bafera i bafera podataka (koji su u unutrašnjoj memoriji) u log datoteku i u bazu podataka, redom, i upisuje se slog tačke pamćenja u log datoteku. Ovaj slog sadrži informaciju o svim aktivnim transakcijama u momentu tačke pamćenja, adrese poslednjih slogova tih transakcija u log datoteci, a može sadržati i niz drugih informacija o stanju baze podataka u tom trenutku. Adresa sloga tačke pamćenja upisuje se u *datoteku ponovnog startovanja* (engl. restart file).

Fizički upis u tački pamćenja znači da će se sadržaj bafera prepisati na spoljni medij bez obzira da li su baferi puni ili ne. To dalje znači da se fizički upis svih ažuriranih podataka uspešno kompletirane transakcije može garantovati tek u tački pamćenja koja sledi za uspešnim kompletiranjem transakcije. Tako dolazimo do zaključka da su kompletiranje transakcije (upis COMMIT sloga u log datoteku) i definitivni upis svih ažuriranja te transakcije u bazu – dve odvojene radnje, za koje se ne sme dogoditi da se jedna izvrši a druga ne. Mehanizam koji to obezbeđuje je *protokol unaprednog upisivanja u log* (engl. WAL – Write Ahead Log). Prema ovom protokolu, pri izvršenju operacije COMMIT prvo se odgovarajući slog fizički upisuje u log datoteku, pa se zatim podaci upisuju iz bafera podataka u bazu. Ako dođe do pada sistema posle upisa COMMIT sloga u log datoteku a pre nego što je sadržaj bafera podataka prepisan u bazu, taj se sadržaj može restaurisati iz sistemskog loga REDO logikom.

Pri ponovnom startovanju sistema, posle pada sistema, moguće je prema sadržaju log datoteke identifikovati neuspele transakcije – kandidate za poništavanje, i uspele transakcije – kandidate za ponovno izvršavanje. Oporavak baze podataka tada se vrši prema protokolu koji se može opisati sledećim pseudoprogramom ([35]):

```
BEGIN
    naći slog poslednje tačke pamćenja u log datoteci
            (iz datoteke ponovnog startovanja);
    IF u poslednjoj tački pamćenja nema aktivnih transakcija
       i slog tačke pamćenja je poslednji slog u log datoteci,
       oporavak je završen
    ELSE
        BEGIN
            formirati dve prazne liste transakcija, "uspele" i "neuspele";
            sve transakcije aktivne u poslednjoj tački pamćenja
                staviti u listu neuspelih;
            čitati redom log datoteku od tačke pamćenja do kraja:
                kada se naiđe na slog "početak transakcije",
                    dodati transakciju listi neuspelih;
                kada se naiđe na slog "kompletirana transakcija",
                    dodati (premestiti) tu transakciju listi uspelih;
            čitati unazad log datoteku (od kraja)
                i poništiti akcije i efekte neuspelih transakcija;
            čitati log datoteku od poslednje tačke pamćenja unapred
                i ponovo izvršiti uspele transakcije
        END
END.
```

Primer 15.1 Razmotrimo izvršenje skupa transakcija predstavljeno slikom 15.1.

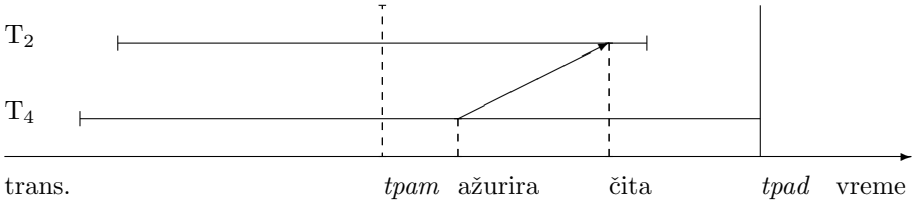


Slika 15.1: Pad sistema pri izvršenju skupa transakcija

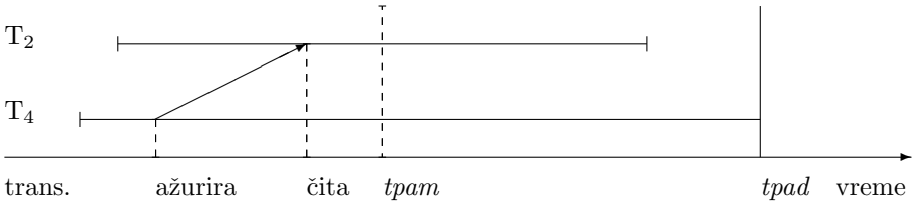
Na vremenskoj osi označene su dve tačke: *tpam* – tačka pamćenja, i *tpad* – tačka pada sistema. Transakcija T_1 završila je rad pre poslednje tačke pamćenja, pa su njena ažuriranja trajno upisana u bazu aktivnošću postavljanja te tačke, i zato ona

ne ulazi u proces oporavka. Transakcije T_2 i T_3 završile su sa radom posle momenta *tpam* a pre pada sistema. U momentu pada sistema aktivne su transakcije T_4 i T_5. Zbog toga prethodni protokol svrstava transakcije T_4 i T_5 u listu neuspelih (čija dejstva poništava), a transakcije T_2 i T_3 u listu uspelih (koje zatim ponovo izvršava od tačke *tpam*). Time je završena aktivnost oporavka od pada sistema.

Upravljač oporavka podrazumeva da transakcija oslobađa sve X-katance pri izvršenju COMMIT operacije. Najčešće je isti slučaj i sa S-katancima. Ovo je saglasno sa svojstvom izolacije transakcije i rešava problem zavisnosti od poništenog ažuriranja, bilo da se ono dogodilo pre ili posle tačke pamćenja. Dakle, nisu moguća izvršenja prikazana na slikama 15.2, 15.3.



Slika 15.2: Zavisnost od poništenog ažuriranja posle tačke pamćenja



Slika 15.3: Zavisnost od poništenog ažuriranja pre tačke pamćenja

15.2.1 Poboljšanje procedure oporavka od pada sistema

Postoji niz poboljšanja i modifikacija izloženog konceptualnog postupka oporavka od pada sistema. Tako je moguće sva upisivanja jedne transakcije u bazu ostaviti za trenutak izvršenja COMMIT operacije te transakcije, čime se eliminiše potreba za UNDO logikom. Takođe je moguće oslabiti zahtev za izolacijom transakcije, tj. zahtev za dvofaznošću, što nosi opasnost nelinearizovanog izvršenja.

Jedno evidentno poboljšanje protokola oporavka od pada sistema odnosi se na aktivnosti vezane za tačku pamćenja. Naime, prethodno opisani protokol poništava efekte neuspelih transakcija koji možda nisu ni bili ubeleženi u bazu (već samo u bafer podataka ako su se dogodili posle tačke pamćenja), odnosno ponovo izvršava

radnje uspelih transakcija od tačke pamćenja, čak i ako su efekti tih radnji, zbog popunjenosti bafera podataka, upisani u bazu pre pada sistema.

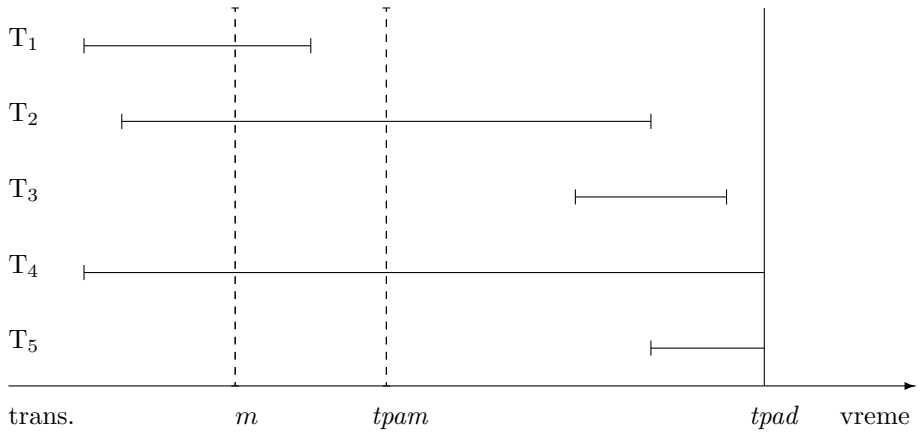
Moguće je eliminisati fizičko upisivanje bafera podataka u bazu podataka u tački pamćenja, a u fazi oporavka od pada sistema poništavati samo one radnje neuspelih transakcija čiji su efekti upisani u bazu, odnosno ponovo izvršavati samo one radnje uspelih transakcija čiji efekti nisu upisani u bazu. U tom cilju uvodi se, u svaki slog sistemskog loga, u vreme njegovog upisa u log, jedinstvena oznaka – serijski broj u logu (engl. LSN – Log Sequence Number). Serijski brojevi su u rastućem poretku. Osim toga, kadgod se ažurirana stranica fizički upisuje u bazu podataka, u stranicu se upisuje i LSN sloga sistemskog loga koji odgovara tom ažuriranju. U sistemskom logu može biti više slogova koji odgovaraju ažuriranju iste stranice baze podataka. Ako je serijski broj upisan u stranicu P veći ili jednak serijskom broju sloga loga, efekat ažuriranja kome odgovara taj slog fizički je upisan u bazu; ako je manji, efekat nije upisan.

Da bi se slog iz baze podataka ažurirao, potrebno je pročitati (iz baze) stranicu na kojoj se slog nalazi. Posle ažuriranja sloga, stranica je (u baferu podataka) spremna za upis u bazu, pri čemu i dalje nosi serijski broj svog prethodnog upisa u bazu.

Sada se u proceduru registrovanja tačke pamćenja, osim što se eliminiše fizički upis bafera podataka u bazu, može dodati upis, u slog tačke pamćenja, vrednosti LSN m najstarije stranice bafera podataka, tj. najmanjeg LSN stranicâ iz bafera podataka, koje su ažurirane ali još nisu upisane u bazu.

Slog sistemskog loga sa serijskim brojem m odgovara tački u sistemskom logu od koje treba, umesto od tačke pamćenja, pri oporavku od pada sistema ponovo izvršavati uspele transakcije. Kako tačka m prethodi tački pamćenja, može se desiti da neka transakcija koja je uspešno kompletirana pre tačke pamćenja, "upadne" u skup aktivnih (uspelih) transakcija (transakcija T_1, slika 15.4). Na takvu transakciju primeniće se procedura oporavka, mada se to ne bi dogodilo u prethodnoj varijanti oporavka; to je "cena" smanjenog broja poništavanja, ponovnog izvršavanja i fizičkog upisa koje obezbeđuje ovaj postupak.

Procedura oporavka od pada sistema sada ima sledeći izgled:



Slika 15.4: Oporavak od pada sistema korišćenjem serijskih brojeva

BEGIN
formirati dve prazne liste transakcija, "uspele" i "neuspele";
sve transakcije aktivne u tački m staviti u listu neuspelih;
čitati redom log datoteku od tačke m do kraja:
kada se naiđe na slog "početak transakcije",
dodati transakciju listi neuspelih;
kada se naiđe na slog "kompletirana transakcija",
dodati (premestiti) tu transakciju listi uspelih;
čitati unazad log datoteku (od kraja)
i poništiti efekte onih radnji neuspelih transakcija
za koje je $p \geq r$, gde je r – LSN tekućeg sloga
a p – LSN odgovarajuće stranice baze podataka;
čitati log datoteku od tačke m unapred
i ponovo izvršiti one radnje uspelih transakcija
za koje je $p < r$ (p, r – kao u prethodnom koraku)
END.

15.3 Pitanja i zadaci

1. Definisati sledeće pojmove:

pad transakcije	REDO logika
pad sistema	tačka pamćenja
pad medija	protokol unaprednog upisivanja u log
log datoteka	
upravljač oporavka	
UNDO logika	dnevnik ažuriranja

2. U čemu se sastoji funkcija upravljača oporavka?

3. Opisati protokol oporavka baze podataka od pada sistema.

4. U čemu se sastoji poboljšanje prethodnog protokola pomoću serijskog broja u logu? Opisati odgovarajuću proceduru.

5. Neka se transakcije sastoje samo od radnji čitanja i upisivanja slogova, i neka njihova sintaksa uključuje eksplicitni BEGIN TRANSACTION i END TRANSACTION iskaz. Skicirati programsku implementaciju procedure za naizmenično izvršavanje skupa transakcija ovoga tipa (npr. po dve radnje svake transakcije), uz beleženje odgovarajućih informacija u posebnu datoteku ("log datoteku"). U slučaju uzajamnog blokiranja, procedura treba da izabere transakciju za poništavanje i da poništi njene efekte uz ažuriranje log datoteke. Uvesti neophodne pretpostavke i ograničenja.

6. Skicirati programsku implementaciju procedure koja realizuje protokol oporavka baze podataka od pada sistema. Dopuniti proceduru iz prethodnog zadatka tako da simulira pad sistema i poziva proceduru za oporavak baze podataka.

7. Zašto je potrebno primeniti protokol unaprednog upisivanja u log datoteku?

Deo VI

Distribuirane baze podataka

U delu **VI** razmatraju se distribuirane baze podataka, sistemi za upravljanje distribuiranim bazama podataka, specifični problemi u realizaciji pojedinih funkcija tih sistema, kao i načini za rešavanje tih problema.

Sistem za upravljanje distribuiranim bazama podataka je i sam distribuiran, pa se označava sa DSUBP – distribuirani SUBP. Distribuirane baze podataka i DSUBP zajedno obrazuju sistem distribuiranih baza podataka. Za obe vrste sistema koristi se i kraći termin *distribuirani sistem*, kada je iz konteksta jasno na koju vrstu sistema se termin odnosi.

Neke od poznatih prototipskih implementacija DSUBP su SDD-1 (System for Distributed Databases, Computer Corporation of America [53]), R* (IBM Research Lab. [68]) i distribuirani INGRES (University of California, Berkeley [59]). Najpoznatiji komercijalni DSUBP su INGRES/STAR, ORACLE7/DDO ("distributed database option"), DB2/DDF ("distributed database facility"). Svi nabrojani sistemi su relacioni, s obzirom da nerelacioni DSUBP ne mogu biti uspešni (v. [27]).

Osnovna karakteristika distribuiranih sistema je velika količina poruka i podataka koji se prenose preko komunikacione mreže. Zato je glavni cilj u postizanju efikasnosti distribuiranih sistema – smanjenje mrežne komunikacije. Ovaj cilj se projektuje na sve ključne funkcije SUBP; one moraju da se razmatraju iz novog ugla, a problemi u njihovoj realizaciji zahtevaju nova rešenja.

Deo **VI** sastoji se od dve glave:

- Glava 16, **Problemi distribuiranih sistema**, bavi se nekim od problema distribuiranih sistema, kao što su fragmentacija podataka, distribuirana obrada upita, distribuirano ažuriranje i upravljanje katalogom. Ova glava sadrži i kratak prikaz heterogenih distribuiranih sistema.
- Za upravljanje transakcijama (kontrolu konkurentnosti i oporavka) u distribuiranom okruženju neophodan je novi pristup, bitno različit od onog u centralizovanom slučaju. Zbog toga je glava 17, **Distribuirano upravljanje transakcijama**, posvećena metodama za rešavanje problema distribuiranog izvršenja skupa transakcija i problema oporavka.

16

Problemi distribuiranih sistema

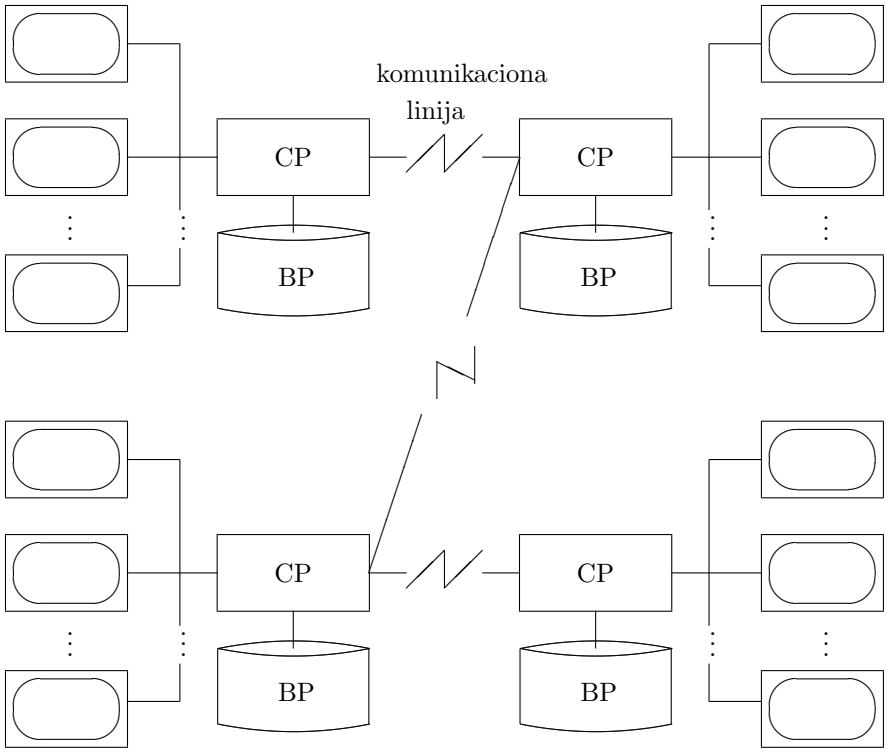
16.1 O distribuiranim sistemima

Neformalno govoreći, *distribuirana* baza podataka je baza podataka koja se ne nalazi u celosti na jednoj fizičkoj lokaciji (na jednom računaru), već je razdeljena na više lokacija koje su povezane komunikacionom mrežom. Svaka lokacija, koja se zove i *čvor* komunikacione mreže, poseduje svoj sopstveni, autonomni sistem za upravljanje bazama podataka, sa sopstvenom kontrolom, upravljačem transakcija i oporavka od pada, i ostalim značajnim funkcijama, a ima i svoj centralni procesor i ulazno/izlazne uređaje.

Osnovna pretpostavka za uspešnost sistema za upravljanje distribuiranim bazama podataka je *nevidljivost lokacije*. Naime, ovaj sistem treba da obezbedi još jedan nivo fizičke nezavisnosti podataka: korisnik i njegov program ne treba da znaju na kojoj se lokaciji u distribuiranom sistemu nalaze podaci koji su im potrebni. Za korisnika, distribuirani sistem treba da izgleda identično sa nedistribuiranim (centralizovanim), tj. pristup podacima treba da je isti kao da su svi podaci smešteni u lokalnom čvoru korisnika. Sistem odlučuje (ne uključujući korisnika) o tome da li će potrebne podatke, ukoliko su na udaljenoj lokaciji, doneti na lokalni čvor za obradu, ili će obradu preneti na udaljenu lokaciju.

Sistem distribuiranih baza podataka može se predstaviti grafom čiji su čvorovi – lokacije, a grane – komunikacione linije. Svaka lokacija sadrži centralni procesor (CP), lokalnu bazu podataka (BP) i pripadni SUBP, kao i skup terminala. Dve lokacije su povezane ako među njima postoji direktna veza koja funkcioniše, ili ako postoji treća lokacija koja je sa njima u direktnoj vezi (slika 16.1).

Na različitim lokacijama, lokalni sistemi za upravljanje bazama podataka mogu biti različiti. Tada se DSUBP naziva *heterogenim*. Ako su SUBP na svim lokacijama isti, DSUBP se naziva *homogenim*.



Slika 16.1: Struktura sistema distribuiranih baza podataka

Prednosti koje donosi distribuirani koncept baze podataka su:

- Lokalna autonomija podataka, upavljanja i kontrole. Okruženje u kome se distribuirane baze primenjuju, obično je i samo logički i fizički distribuirano (npr. univerzitet, fakulteti, odseci, odeljenja; centralna biblioteka, matične biblioteke, ogranci; ministarstvo, regionalni centri, itd). Distribuiranje baza podataka kao i sistema za upravljanje njima, omogućuje pojedinim grupama da lokalno kontrolišu sopstvene podatke, uz mogućnost pristupa podacima na drugim lokacijama kada je to potrebno.
- Veći kapacitet i postupni rast. Čest razlog za instaliranje distribuiranog sistema je nemogućnost jednog računara da primi i obrađuje sve potrebne podatke. U slučaju da potrebe nadmaše postojeće kapacitete, dodavanje čvora distribuiranom sistemu je znatno jednostavnije nego zamena centralizovanog sistema većim.
- Pouzdanost i raspoloživost. Distribuirani sistemi mogu da nastave svoje funkcionisanje i kada neki od čvorova privremeno izgube funkcionalnost.

- Efikasnost i fleksibilnost. Podaci su fizički blizu onome ko ih stvara i koristi, pa je znatno smanjena potreba za udaljenom komunikacijom.

Jedan jednostavan specijalni slučaj distribuiranog sistema je *klijent/server* sistem. To je distribuirani sistem u kome su neki čvorovi *klijenti* a neki *serveri*, pri čemu su na serverima smešteni podaci (i sistemi za upravljanje podacima) a na klijentima se izvršavaju aplikacije. Korisnik, odnosno aplikacija vodi računa o tome na kom serveru su smešteni relevantni podaci, što znači da u ovim sistemima nije ostvarena nevidljivost lokacije.

Klijent/server sistemi zasnovani su na klijent/server arhitekturi, unekoliko različitoj od ANSI/SPARC arhitekture sistema baza podataka prikazane u glavi 1. Ova arhitektura se odnosi na jednostavnu logičku podelu odgovornosti unutar sistema na dva dela. Jedan deo (prvobitno u engleskoj terminologiji back-end, *pozadinski*), server, u stvari je SUBP sa bazama podataka, i odgovoran je za definisanje podataka, manipulisanje podacima, bezbednost, integritet, oporavak, itd. Drugi deo (prvobitno je korišćen engleski termin frontend, *prednji sistem*), klijent, odgovoran je za rad aplikacija, kako korisničkih tako i sistemskih (na primer, procesor upitnog jezika, generator aplikacija, generator izveštaja, itd). Ovakva podela posla ne zahteva razdvajanje odgovornosti na razne računare. Međutim, termin "klijent/server" odnosi se danas gotovo isključivo na slučaj kada su klijent i server na odvojenim računarima, jer se mogućnost izdvajanja funkcijâ na odvojene računare pokazala veoma privlačnom i korisnom.

Osnovni klijent/server sistemi, dakle, podrazumevaju veći broj klijenata koji dele isti server. Moguće je, međutim, da jedan klijent pristupi većem broju servera, pri čemu je svaki pojedinačni zahtev (ili transakcija) upućen tačno jednom serveru (korisnik mora da zna na kom serveru se koji podaci nalaze). Detaljnije informacije o klijent/server sistemima mogu se naći, npr. u [27].

Klijent/server sistemi doživljavaju veliku komercijalnu ekspanziju, pre svega zbog značajnog pojednostavljenja funkcija i problema vezanih za njihovu implementaciju, u odnosu na prave sisteme za upravljanje distribuiranim bazama podataka. Bez obzira na aktuelno stanje na tržištu, u preostalom tekstu ove glave, kao i u sledećoj glavi, biće razmotrena puna funkcionalnost "pravih" DSUBP, jer osnova za razvoj novih arhitektura sistema, pa i za usmeravanje tržišta, jeste upravo osvajanje pune funkcionalnosti DSUBP.

Dok se kod centralizovanih sistema baza podataka efikasnost sistema postizala pre svega smanjenjem broja obraćanja disku (jer je komunikacija sa diskom bila najsporija), kod distribuiranih sistema efikasnost se postiže pre svega smanjenjem količine podataka i poruka koje se prenose preko mreže. Naime, komunikacione veze kroz mrežu su obično znatno sporije (i do 40 puta) od komunikacije sa lokalnim diskom, i obično imaju nezanemarljivo vreme kašnjenja. Osim toga, upravljanje porukama (slanje i prijem poruka) uključuje veliki broj operacija centralnog procesora (u proseku 5000 – 10000 instrukcija operativnog sistema i komunikacione kontrole po poruci; v. [21]).

Da bi se postigla efikasnost DSUBP, sve njegove komponente treba realizovati na način koji smanjuje mrežnu komunikaciju. Najznačajniji problemi koje tom prilikom treba rešiti su:

- fragmentacija podataka
- distribuirana obrada upita
- distribuirano (preneto) ažuriranje
- upravljanje katalogom
- distribuirano izvršenje skupa transakcija, što uključuje konkurentnost, integritet, oporavak i protokole kompletiranja transakcija.

16.2 Fragmentacija podataka

Podaci u distribuiranom sistemu mogu biti *particionirani* (engl. partitioned) ili *ponovljeni* (engl. replicated) u fizičkoj memoriji.

U slučaju ponavljanja podataka, jedan logički objekat može imati više fizičkih reprezentacija (veći broj kopija) na većem broju lokacija. Ponovljenost podataka povećava raspoloživost podataka i efikasnost pristupa podacima, ali u značajnoj meri usložnjava ažuriranje podataka, koji moraju biti konzistentni u svim svojim kopijama. Složenost koju nosi sobom strategija ponavljanja podataka mora biti sakrivena od korisnika, tj. mora biti obezbeđena *nevidljivost ponavljanja podataka*.

U slučaju particioniranja podataka, logički skup podataka (skup podataka sa logičkog nivoa) treba na neki način podeliti, a zatim delove – *fragmente* (eventualno sa ponovljenim kopijama) razdeliti po raznim lokacijama. Logički skup podatka u relacionom sistemu je relacija, a prirodni fragment relacije (koji je opet relacija) jeste neki njen podskup definisan uslovom projekcije i restrikcije. Fragmentacija mora biti izvedena tako da se spajanjem fragmenata može dobiti polazna relacija.

Na primer, relacija I sa sadržajem iz glave 1 može se SQL-olikim jezikom particionirati na sledeće fragmente:

```
DEFINE FRAGMENT JU1
       AS SELECT I_SIF, NAZIV
          FROM   I
          WHERE  DRZAVA = 'Jugoslavija'

DEFINE FRAGMENT JU2
       AS SELECT I_SIF, STATUS, DRZAVA
          FROM   I
```

```
            WHERE  DRZAVA = 'Jugoslavija'

DEFINE FRAGMENT AM1
       AS SELECT I_SIF, NAZIV
          FROM   I
          WHERE  DRZAVA = 'Amerika'


DEFINE FRAGMENT AM2
       AS SELECT I_SIF, STATUS, DRZAVA
          FROM   I
          WHERE  DRZAVA = 'Amerika'
```

Najčešće primenjivana tehnika u obezbeđivanju očuvanja informacija pri fragmentaciji podataka jeste uvođenje sistemskih identifikatora n-torki (tzv. *nametnutih* ključeva) kao primarnih ključeva, koji se pamte uz svaki deo pojedine n-torke, i omogućuju njenu rekonstrukciju. Ova tehnika je primenjena u distribuiranim sistemima R* i SDD-1.

16.3 Distribuirana obrada upita

Distribuirana obrada upita podrazumeva distribuiranu optimizaciju kao i distribuirano izvršavanje upita. Strategije optimizacije upita nad distribuiranom bazom podataka imaju za cilj da minimizuju cenu obrade i vreme za koje će korisnik dobiti odgovor. U troškovima obrade najveću stavku čine troškovi mrežne komunikacije, tj. prenosa podataka kroz mrežu, dok su troškovi komunikacije sa ulazno/izlaznim uređajima i korišćenja procesora niži za nekoliko redova veličine. Zbog toga je veoma značajno, u zavisnosti od propusnosti mreže (količine podataka koje može da primi u sekundi) i vremena kašnjenja, pravilno odabrati relacije i njihove fragmente koji će biti prenošeni sa jedne lokacije na drugu u cilju obrade upita (*globalna optimizacija*). Razlog za prenošenje podataka može biti to što su podaci na lokaciji različitoj od one na kojoj se postavlja upit, ili što u upitu učestvuje veći broj relacija sa različitih lokacija. Izbor strategije za izvršenje operacija na jednoj lokaciji poznat je kao *lokalna optimizacija*.

Ako se n relacija $R_1, R_2, \ldots, R_n$ koje učestvuju u upitu nalaze na k različitih lokacija $l_1, l_2, \ldots, l_k$, pri čemu je svaka relacija R_i u celosti na lokaciji l_j, onda se osnovna strategija distribuirane obrade upita sastoji od sledeća dva koraka:

1. maksimalna redukcija svake relacije na njenoj lokaciji (lokalna restrikcija i projekcija na atribute spajanja i izlazne atribute),

2. prenošenje dobijenih relacija na jednu lokaciju, ili na više lokacija, redom, na kojima je moguće izvršiti pojedinačna spajanja i projekciju na izlazne atribute.

Za drugi korak osnovne strategije vezana je odluka o tome koje se relacije prenose i na koje lokacije. Ta odluka se donosi na osnovu procene količine podataka koji se prenose između lokacija u svakom pojedinačnom slučaju; izbor relacija i lokacija vrši se tako da minimizuje protok podataka kroz mrežu. Sledeći primer ilustruje mogućnosti koje postoje pri donošenju te odluke.

Primer 16.1 Neka se relacije K, I, KI (iz glave 1) nalaze na lokacijama l_1, l_2, l_3, redom, i neka se na lokaciji l_1 postavi sledeći SQL upit (naći naslove romana jugoslovenskih izdavača kao i brojeve izdanja i nazive tih izdavača):

```
SELECT K.NASLOV, I.NAZIV, KI.IZDANJE
FROM   K, KI, I
WHERE  I.DRZAVA='Jugoslavija' AND  K.OBLAST='roman' AND
       K.K_SIF=KI.K_SIF AND KI.I_SIF=I.I_SIF
```

U upitu su prisutne operacije restrikcije nad relacijama K i I, dve operacije spajanja, jedna nad relacijama K i KI a druga nad relacijama KI i I, kao i operacija projekcije na izlazne atribute NASLOV, NAZIV i IZDANJE.

Prvi korak osnovne strategije distribuiranog izvršavanja ovog upita sastoji se u izvršavanju, na lokaciji l_1, jednorelacionog upita

```
SELECT NASLOV, K_SIF
FROM   K
WHERE  OBLAST='roman'
```

i u izvršavanju, na lokaciji l_2, jednorelacionog upita

```
SELECT NAZIV, I_SIF
FROM   I
WHERE  DRZAVA='Jugoslavija'
```

Prvi upit proizvodi relaciju K'(NASLOV, K_SIF) na lokaciji l_1, a drugi upit proizvodi relaciju I'(NAZIV, I_SIF) na lokaciji l_2.

Pred drugi korak osnovne strategije postavlja se sledeći izbor:

a) Moguće je preneti relacije K' i I' na lokaciju l_3 (analogno, relacije K' i KI na lokaciju l_2, odnosno relacije I' i KI na lokaciju l_1) i tamo izvršiti oba spajanja i projekciju na izlazne atribute:

```
SELECT K'.NASLOV, I'.NAZIV, KI.IZDANJE
FROM   K', I', KI
WHERE  K'.K_SIF=KI.K_SIF AND KI.I_SIF=I'.I_SIF
```

a zatim rezultat preneti na lokaciju l_1.

b) Moguće je, takoďe, pojedinačna spajanja vršiti na različitim lokacijama. Na primer, relacija I' može se preneti na lokaciju l_3, tamo izvršiti spajanje sa relacijom KI, uz projekciju na izlazne atribute celog upita i atribute spajanja preostalog upita:

```
SELECT I'.NAZIV, KI.K_SIF, KI.IZDANJE
FROM   I', KI
WHERE  I'.I_SIF=KI.I_SIF
```

Neka je dobijena relacija – rezultat IKI(NAZIV, K_SIF, IZDANJE). Sada se relacija IKI može preneti na lokaciju l_1, i tamo izvršiti spajanje sa relacijom K' i projekcija na izlazne atribute:

```
SELECT K'.NASLOV, IKI.NAZIV, IKI.IZDANJE
FROM   IKI, K'
WHERE  IKI.K_SIF=K'.K_SIF
```

(rezultat se nalazi na lokaciji l_1).

Prethodni upiti spajanja tipa b) predstavljaju jednu mogućnost pojedinačnih spajanja na različitim lokacijama. Ti upiti se mogu predstaviti parom izraza relacione algebre sa naznakom lokacije na kojoj se izvršavaju,

IKI = (I' ∗ KI) [NAZIV, K_SIF, IZDANJE] (l_3)
REZ = (IKI ∗ K') [NASLOV, NAZIV, IZDANJE] (l_1).

Analogne mogućnosti pojedinačnih spajanja na različitim lokacijama tipa b) opisuju se i sledećim parovima izraza relacione algebre, sa naznakom lokacija:

IKI = (I' ∗ KI) [NAZIV, K_SIF, IZDANJE] (l_2)
REZ = (IKI ∗ K') [NASLOV, NAZIV, IZDANJE] (l_1),

KKI = (K' ∗ KI) [NASLOV, I_SIF, IZDANJE] (l_1)
REZ = (KKI ∗ I') [NASLOV, NAZIV, IZDANJE] (l_2),

KKI = (K' ∗ KI) [NASLOV, I_SIF, IZDANJE] (l_3)
REZ = (KKI ∗ I') [NASLOV, NAZIV, IZDANJE] (l_2).

Razmotrimo efikasnost nabrojanih varijanti koraka 2 osnovne strategije distribuirane obrade upita.

Neka su veličine relacija date sa

K (K_SIF, NASLOV, OBLAST) – 10000 n-torki
I (I_SIF, NAZIV, STATUS, DRZAVA) – 1000 n-torki
KI (K_SIF, I_SIF, IZDANJE, GODINA, TIRAZ) – 50000 n-torki

i neka svaka n-torka zauzima po 100 bita. Neka je, dalje, procenjeno da ima 1000 romana u relaciji K, 100 jugoslovenskih izdavača u relaciji I i 20000 jugoslovenskih

izdanja u relaciji KI. Ako se još pretpostavi da je propusnost mreže 50000 bita u sekundi, a vreme kašnjenja 0.1 sekunda, onda se za optimalnu strategiju proglašava ona koja ima minimalnu vrednost vremena komunikacije

t = kašnjenje + (količina podataka za prenos / propusnost), tj.
t = (broj poruka $*$ 0.1sec) + (broj bita za prenos / 50000)sec.

Primena raznih varijanti mogućnosti navedenih pod a) i b) rezultovala bi različitim ukupnim vremenima komunikacije. Na primer, u slučaju a) biće

t $\approx$ 0.2 + (1100$*$100) / 50000 = 2.4 sekunde (bez prenošenja rezultata na lokaciju l_1), dok u slučaju b) dobijamo

t $\approx$ 0.2 + (100 + 20000)$*$ 100 / 50000 = 40.4sec.

Jedna od varijanti b) mogla bi da bude i prenošenje relacije KI na lokaciju l_2, spajanje sa relacijom I' i projekcija na atribute K_SIF, IZDANJE, NAZIV, zatim prenošenje rezultata na lokaciju l_1 i spajanje sa relacijom K'. U ovom slučaju ukupno vreme komunikacije je

t $\approx$ 0.2 + (70000)$*$ 100 / 50000 = 140.2 sekunde, što je znatno više od prethodnih vremena.

Na sličan način mogu se izračunati ukupna vremena komunikacije i u drugim varijantama, pri čemu je zadatak globalne optimizacije da izabere varijantu sa najmanjim vremenskim utroškom.

Znatno poboljšanje efikasnosti može se postići primenom sofisticiranih algoritama za izvoďenje nekih operacija. Jedan od takvih algoritama, primenjen u sistemu SDD-1, odnosi se na izvršenje operacije spajanja i uključuje operaciju *poluspajanja* (engl. semijoin). Rezultat operacije poluspajanja relacija R i S jednak je rezultatu spajanja tih relacija, projektovanom na atribute relacije R. Tako, ako je potrebno izvršiti spajanje relacije R (sa lokacije l_1) i relacije S (sa lokacije l_2), umesto da se cela relacija R prenese na lokaciju l_2 i tamo spoji sa relacijom S, moguće je izvršiti sledeći niz radnji:

- izračunati projekciju relacije S po atributu spajanja, na lokaciji l_2 (rezultat je relacija TEMP1);
- preneti relaciju TEMP1 na lokaciju l_1;
- izvršiti poluspajanje relacije R i TEMP1 po atributu spajanja, na lokaciji l_1 (rezultat je relacija TEMP2);
- preneti relaciju TEMP2 na lokaciju l_2;
- izvršiti spajanje relacija TEMP2 i S po atributu spajanja, na lokaciji l_2.

Ovaj niz radnji smanjuje količinu prenosa podataka samo ako je

broj bitova (TEMP1) + broj bitova (TEMP2) < broj bitova (R),

i nije efikasan ako se spajanje izvodi po primarnom ključu relacije R koji je strani ključ relacije S (npr. u slučaju relacija I', KI).

16.4 Preneto ažuriranje

Kao što je u odeljku o fragmentaciji podataka već rečeno, ponavljanje podataka podrazumeva da jedan logički objekat (npr. relacija ili jedan njen fragment) može imati više fizičkih reprezentacija (kopija) na većem broju lokacija. Posledica ove ideje je da se, s obzirom na potrebu za konzistentnošću podataka u svim kopijama, ažuriranje jednog logičkog objekta mora preneti i na sve fizičke kopije tog objekta. Međutim, momentalno prenošenje ažuriranja na sve kopije može da onemogući (ili da nedopustivo dugo odloži) uspešno izvršenje ažuriranja u slučaju da je bilo koja od lokacija u padu; time ponavljanje podataka smanjuje umesto da povećava raspoloživost podataka.

Jedan široko prihvaćeni pristup prenošenju ažuriranja oslanja se na koncept *primarne kopije*, i sastoji se u sledećem postupku:

- jedna kopija svakog ponovljenog objekta proglašava se za primarnu kopiju tog objekta, pri čemu primarne kopije različitih objekata mogu biti na različitim lokacijama;

- operacija ažuriranja objekta smatra se logički izvršenom čim se izvrši ažuriranje primarne kopije tog objekta; ažuriranje ostalih kopija je sada u nadležnosti lokacije na kojoj je primarna kopija, ali se mora izvršiti pre kompletiranja transakcije. Ovaj postupak zahteva primenu protokola dvofaznog kompletiranja transakcije (v. odeljak 17.4), koji se pak ne može uspešno sprovesti ako je bar jedna relevantna lokacija u padu, što je nezanemarljivo čest slučaj. Ukoliko se dopusti ažuriranje kopijâ i posle kompletiranja transakcije, ne može se garantovati konzistentnost podataka u svim njihovim kopijama. Ipak, neki komercijalni distribuirani sistemi pribegavaju tom rešenju jer puno poštovanje ažuriranja svih kopija pre kompletiranja transakcije može bitno da poveća vreme obrade.

Detaljnija razmatranja problema prenetog ažuriranja mogu se naći u [27], [21].

16.5 Upravljanje katalogom

Katalog je, kako je već istaknuto u odeljku 7.1, sistemska baza podataka koja sadrži podatke o baznim relacijama, pogledima, indeksima, korisnicima, itd, a u slučaju distribuiranog sistema, i o načinu i lokacijama na koje su podaci razdeljeni i (eventualno) ponovljeni. Sam katalog u distribuiranom sistemu može biti *centralizovan* (samo na jednoj lokaciji), *potpuno ponovljen* (na svim lokacijama po jedna kopija kataloga), *particioniran* (na svakoj lokaciji je deo kataloga koji se odnosi na objekte sa te lokacije) ili *kombinovan* (katalog je particioniran, ali na jednoj lokaciji postoji i jedna centralna kopija kompletnog kataloga).

S obzirom na nedostatke koje ispoljava svaki od navedenih pristupa (zavisnost od centralne lokacije, visoka cena prenošenja ažuriranja kataloga ili skup pristup

udaljenoj lokaciji), implementirani sistemi koriste druge strategije. Tako, na primer, u sistemu R* ([68]) svaka lokacija sadrži sledeće kataloške informacije:

- slog kataloga za svaki objekat "roďen" na toj lokaciji (tj. čija je prva kopija kreirana na toj lokaciji); ovaj slog sadrži i informaciju o tekućoj lokaciji objekta (ako je premešten);
- slog kataloga za svaki objekat koji je trenutno smešten na toj lokaciji;
- tabelu *sinonima* za svakog korisnika prijavljenog na toj lokaciji, koja preslikava sinonime objekata, definisane iskazom kreiranja sinonima, u sistemske identifikatore objekata, jedinstvene u celom distribuiranom sistemu.

Sistemski identifikator objekta, koji se nikada ne menja (dok se objekat ne ukloni), sastoji se od identifikatora korisnika koji je kreirao prvu kopiju tog objekta, lokacije sa koje je kreirana, lokalnog imena objekta (koje mu je dao korisnik pri roďenju) i lokacije na kojoj je roďen. Pronalaženje objekta, kada je dato njegovo lokalno ime ili sinonim, počinje preslikavanjem lokalnog imena (automatski), odnosno sinonima (pregledanjem tabele sinonima), u sistemski identifikator objekta. Zatim se, prema sistemskom identifikatoru, nalazi lokacija roďenja objekta, pristupa joj se i u slogu koji odgovara tom objektu, pronalazi se lokacija na kojoj je trenutno smešten. U sledećem koraku pristupa se (u opštem slučaju udaljenoj) lokaciji na kojoj je objekat smešten, i lokalnim operacijama pronalazi objekat.

16.6 Heterogeni distribuirani sistemi

Pretpostavka o homogenosti DSUBP, tj. pretpostavka da sve lokacije u DSUBP poseduju isti SUBP pokazuje se kao prejako ograničenje u današnjim uslovima, kada značajne količine podataka i aplikacija postoje na raznim računarima, pod različitim operativnim sistemima i pod kontrolom različitih SUBP. Potreba za istovremenim pristupom ovakvim podacima unutar jedne aplikacije, ili čak i jedne transakcije, postavlja zahtev pred proizvoďače DSUBP da obezbede podršku heterogenim DSUBP. To znači da, pored nezavisnosti pristupa podacima i obrade podataka od lokacije, fragmentacije, ponavljanja podataka, mašine, operativnog sistema i mrežnog protokola, heterogeni distribuirani sistem mora da obezbedi i nezavisnost od SUBP na pojedinim lokacijama.

Postoje dva bitno različita pristupa rešavanju ovog problema. Jedan je izgradnja tzv. sistema multibaza podataka, SMBP (engl. multidatabase system, [43]). SMBP je programski sistem koji se sastoji od niza komponenti. Jedna od tih komponenti je jedinstveni jezik za kreiranje podataka i manipulisanje podacima koji su pod kontrolom heterogenih SUBP. SMBP je, pre svega, jedinstvena sumeďa, kroz koju korisnici i aplikacije mogu da komuniciraju sa raznorodnim sistemima.

Druga komponenta SMBP je globalni upravljač transakcija. SMBP obezbeďuje, pored lokalnih transakcija (nad jednim SUBP), i upravljanje globalnim transakcijama. Globalne transakcije se sastoje od većeg broja podtransakcija koje se

izvršavaju nad pojedinačnim (različitim) SUBP, i sa njihovog aspekta se ponašaju kao lokalne transakcije.

SMBP uključuje i skupa servera, po jedan za svaki lokalni SUBP, koji se ponašaju kao veza između globalnog upravljača transakcija i lokalnog SUBP. Svaka globalna transakcija predaje globalnom upravljaču transakcija operacije čitanja i upisa, a ovaj može da ih preda na obradu lokalnim SUBP (preko odgovarajućih servera), da odloži ili da prekine transakciju. Kada globalni upravljač transakcija odluči da kompletira globalnu transakciju, on upućuje komandu za kompletiranje lokalnim SUBP. Globalni upravljač transakcija je odgovoran i za funkcionalnost i održavanje svojstava globalnih transakcija (ACID svojstva, globalna linearizovanost izvršenja skupa transakcija, izbegavanje ili razrešavanje uzajamnog blokiranja, oporavak od sistemskih padova).

Oblast sistema multibaza je još uvek otvorena istraživačka oblast. Osnovni cilj ovih sistema je da kroz upravljanje globalnim transakcijama održavaju konzistentnost multibaze. Detaljniji opis obrade transakcija multibaze, kao i način na koji se ostvaruju funkcije upravljača transakcija u ovom okruženju, mogu se naći u [43].

Drugi pristup heterogenim DSUBP je manje ambiciozan, ali i komercijalno zastupljeniji. Njegova suština je u izgradnji aplikativnih programa, tzv. *prolaza* (engl. gateway) sistema $SUBP_1$ prema sistemu $SUBP_2$, koji omogućuju korisniku $SUBP_1$ (na lokaciji l_1) da komunicira sa $SUBP_2$ (koji je na lokaciji l_2), istom sumedom kojom komunicira sa sistemom $SUBP_1$.

Realizacija aplikativnog programa prolaza sistema $SUBP_1$ prema sistemu $SUBP_2$ je u nadležnosti sistema $SUBP_1$, a program se izvršava nad sistemom $SUBP_2$. Na primer, ako je $SUBP_1$ – sistem Oracle a $SUBP_2$ – sistem DB2, onda bi prolaz sistema Oracle prema sistemu DB2 omogućio korisniku sistema Oracle da komunicira sa sistemom DB2, "lažno" predstavljajući sistem DB2 kao Oracle.

Aplikativni program prolaz ostvaruje sledeće komponente i zadatke ([27]):

- protokol za razmenu informacija između $SUBP_1$ i $SUBP_2$,
- relacioni server za $SUBP_2$,
- preslikavanja između tipova podataka i upitnih jezika dva sistema,
- preslikavanje strukture kataloga sistema $SUBP_2$ u strukturu kataloga sistema $SUBP_1$,
- učešće u dvofaznom protokolu kompletiranja transakcija,
- doslednu primenu mehanizma zaključavanja, itd.

S obzirom na značaj koji ima realizacija kvalitetnih programa prolaza, postoji razvijena aktivnost standardizacije odgovarajućih protokola ([43]). U implementaciji pune funkcionalnosti prolaza javljaju se značajni problemi, pa zato komercijalni proizvodi ovog tipa ne podržavaju sve potrebne funkcije ([27]).

16.7 Pitanja i zadaci

1. Definisati sledeće pojmove:

distribuirana baza podataka	globalna optimizacija
DSUBP	lokalna optimizacija
nevidljivost lokacije	sistem multibaza
heterogeni DSUBP	prolaz
homogeni DSUBP	particionirani podaci
klijent/server arhitektura	ponovljeni podaci
fragmentacija podataka	nevidljivost particioniranja

2. Koje su prednosti a koji nedostaci DSUBP?
3. Navesti primer fragmentacije podataka.
4. Koji su problemi distribuiranih sistema?
5. Navesti primer distribuirane obrade upita.
6. Operacija poluspajanja relacija R i S nije efikasna ako se izvodi po primarnom ključu relacije R koji je strani ključ relacije S (v. odeljak 16.3). Objasniti zašto.
7. Kako se rešava problem prenetog ažuriranja?
8. Koje su specifičnosti upravljanja katalogom u distribuiranom sistemu?
9. Distribuirani SUBP su gotovo isključivo relacioni sistemi. Objasniti razloge za to.

17

Distribuirano upravljanje transakcijama

Pod transakcijom u distribuiranom sistemu podrazumevaće se, kao i u centralizovanom slučaju, vremenski uređen niz radnji koji prevodi jedno konzistentno stanje baze u drugo konzistentno stanje baze. Posmatraćemo opet jednostavni model transakcija u kome se one sastoje samo od radnji čitanja i upisa (ažuriranja). Ipak, pojam transakcije u distribuiranom sistemu je kompleksniji, s obzirom da transakcija može da izvršava radnje svog programa na raznim lokacijama; zato transakciju, koja predstavlja logičku jedinicu posla (i oporavka), može da izvršava veći broj procesa na većem broju lokacija. Startovanjem jedne transakcije bira se jedan upravljač transakcija (na jednoj lokaciji) koji služi kao *koordinator* procesâ te transakcije. Svakoj transakciji dodeljuje se i njen *privatni radni prostor* (koji može da bude razdeljen na više lokacija), iz kojeg će transakcija čitati i u koji će upisivati vrednosti objekata.

Pri distribuiranoj obradi transakcija, transakcija koja čita vrednost objekta – čita vrednost samo jednog lokalnog primerka tog objekta, dok ažuriranje jednog objekta sprovodi nad svim primercima tog objekta (na svim lokacijama distribuiranog sistema na kojima primerak tog objekta postoji).

Transakcija se, kao i u centralizovanom sistemu, karakteriše ACID-svojstvima (v. glavu 14).

17.1 Konkurentnost

Neka je dat skup transakcija $\mathrm{T}=\{\mathrm{T}_i\}_{i=1}^{n}$ nad bazom podataka distribuiranom na k lokacija.

Lokalno izvršenje skupa T na lokaciji l je niz I_l trojki oblika
(T_j, pročitaj, x_l), odnosno
(T_j, ažuriraj, x_l),

gde je $T_j \in T$, x_l je primerak objekta x na lokaciji l, a radnja (pročitaj x) odnosno (ažuriraj x) je radnja transakcije T_j. Poredak trojki u ovom nizu odgovara poretku radnji pojedine transakcije.

Distribuirano izvršenje skupa T je niz lokalnih izvršenja $I = \{I_l\}_{l=1}^{k}$, takav da važi:

(a) (čitanje samo jednog primerka): ako je (T_i, pročitaj, x_j) $\in I_j$, tada (T_i, pročitaj, x_l) $\notin I_l$, za $l \neq j$, gde su x_j, x_l – primerci objekta x na lokacijama j, l, redom;

(b) (ažuriranje svih primeraka): ako je (T_i, ažuriraj, x_j) $\in I_j$, tada je (T_i, ažuriraj, x_l) $\in I_l$ za sve lokacije l na kojima postoji primerak x_l objekta x;

(c) svakoj radnji svake transakcije T_j odgovara bar jedna trojka sa prvom komponentom T_j.

Primer 17.1 Neka je distribuirana baza podataka particionirana na dve lokacije (1 i 2), i neka su njena tri logička objekta x, y, z predstavljena sledećim rasporedom svojih primeraka na tim lokacijama:

1: x_1, y_1
2: x_2, y_2, z_2.

Neka su dve transakcije, T_1 i T_2, koje se izvršavaju nad tom bazom podataka, sastavljene od sledećih radnji:

T_1: (pročitaj x; ažuriraj y)
T_2: (pročitaj y; ažuriraj z).

Jedno (ali ne i jedino) distribuirano izvršenje transakcija T_1 i T_2 nad ovom bazom podataka (označimo ga sa I), predstavlja skup sledećih lokalnih izvršenja I_1, I_2 na lokacijama 1, 2:

I_1: ((T_1, pročitaj, x_1), (T_1, ažuriraj, y_1), (T_2, pročitaj, y_1))
I_2: ((T_1, ažuriraj, y_2), (T_2, ažuriraj, z_2)).

U ovom primeru distribuiranog izvršenja, transakcija T_2 je odabrala da vrednost objekta y pročita na lokaciji 1 (a ne na lokaciji 2), tj. odabrala je da pročita primerak y_1 (a ne y_2).

Kao kod centralizovanih sistema, i kod distribuiranih sistema mogu se definisati serijska distribuirana izvršenja, ekvivalentna distribuirana i linearizovana (ekvivalentna serijskim) distribuirana izvršenja skupa transakcija.

Serijsko distribuirano izvršenje skupa transakcija $T = \{T_i\}_{i=1}^{n}$ je distribuirano izvršenje I za koje se na skupu T može definisati uređenje $<$ za koje važi: ako je $T_i < T_j$, onda sve radnje transakcije T_i prethode svim radnjama transakcije T_j u svakom lokalnom izvršenju I_l u kojem se pojavljuju radnje obeju transakcija.

Prethodni primer prikazuje serijsko distribuirano izvršenje skupa transakcija $\{T_1, T_2\}$ upravo u tom poretku.

Distribuirano izvršenje može da ne bude serijsko bilo zbog toga što neko od lokalnih izvršenja nije serijsko, bilo zbog toga što redosled transakcija u (inače serijskim) lokalnim izvršenjima nije isti.

Dve radnje skupa transakcija T su *konfliktne u distribuiranom izvršenju* ako su konfliktne (u centralizovanom smislu, v. glavu 14) u nekom (bilo kome) lokalnom izvršenju.

Dva distribuirana izvršenja su *ekvivalentna* ako su, za svaku lokaciju, njihova lokalna izvršenja ekvivalentna (u centralizovanom smislu).

Distribuirano izvršenje je *linearizovano* ako je ekvivalentno nekom serijskom izvršenju.

17.2 Dvofazno zaključavanje

Jedna metoda za postizanje linearizovanog distribuiranog izvršenja je, kao i kod centralizovanog linearizovanog izvršenja, *dvofaznost transakcija*, koja se u distribuiranom slučaju proširuje sledećim zahtevima:

- pri čitanju logičkog objekta x, dovoljno je da transakcija zaključa deljivim katancem jedan primerak objekta x (onaj koji stvarno čita);
- pri ažuriranju logičkog objekta x, transakcija mora da postavi ekskluzivne katance na sve primerke objekta x, na svim lokacijama u distribuiranoj bazi;
- ako transakcija ne može da dobije traženi katanac, ona staje u red za čekanje za onaj primerak objekta nad kojim treba da izvrši radnju.

Glavni nedostatak prethodnog postupka dvofaznog zaključavanja je što transakcija, pri ažuriranju jednog logičkog objekta, mora dobiti ekskluzivni katanac na svim primercima tog objekta, na raznim lokacijama, a o dodeljivanju i oslobađanju katanaca odlučuju lokalni moduli (upravljači zaključavanja). Zato taj postupak zahteva puno komunikacije (slanja i primanja poruka), i može se pojednostaviti na nekoliko načina (od kojih svaki ima svoje prednosti i nedostatke). Jedna tehnika dvofaznog zaključavanja u distribuiranom sistemu je *centralizacija upravljača zaključavanja*, koja podrazumeva da samo jedna lokacija upravlja postavljanjem i oslobađanjem katanaca u celom sistemu. U tom slučaju je ažuriranje pojednostavljeno, jer se skup svih primeraka jednog logičkog objekta može tretirati (u svrhe zaključavanja) kao jedinstveni objekat. Nedostatak ove tehnike je što se lokacija koja upravlja zaključavanjem brzo preopterećuje, a u slučaju njenog pada, pada i ceo distribuirani sistem.

Druga tehnika je tehnika *primarnog primerka objekta*. U ovoj tehnici svaki logički objekat ima jedan svoj primarni primerak i, moguće je, veći broj ostalih

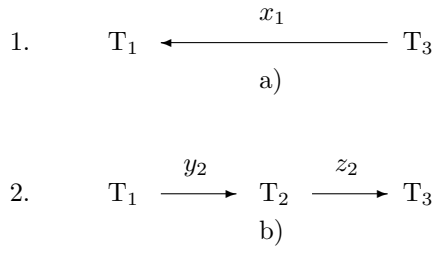
primeraka. Razni logički objekti mogu imati primarne primerke na raznim lokacijama. Upravljač zaključavanja lokacije na kojoj je primarni primerak logičkog objekta x obrađuje zahteve za zaključavanjem objekta x (i na svim drugim lokacijama). I u ovom slučaju svi primerci jednog objekta ponašaju se, u svrhe zaključavanja, kao jedinstveni objekat, ali ova tehnika ne pokazuje nedostatke centralizovanog upravljača zaključavanja.

Nedostatak metode dvofaznosti transakcija u obezbeđivanju linearizovanosti konkurentnog distribuiranog izvršenja, kao i u slučaju centralizovanih SUBP, jeste to što može doći do uzajamnog blokiranja transakcija. Na primer, ako bi se skupu transakcija iz primera 17.1 dodala i transakcija T_3: (pročitaj z; ažuriraj x), onda bi jedan skup parcijalnih lokalnih izvršenja (na lokacijama 1 i 2) mogao da bude:

I_1: (T_1, pročitaj, x_1)
I_2: (T_2, pročitaj, y_2), (T_3, pročitaj, z_2).

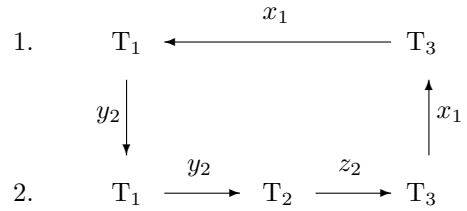
U tom trenutku transakcija T_1 drži deljivi katanac nad primerkom x_1 objekta x na lokaciji 1, transakcija T_2 drži deljivi katanac nad primerkom y_2 objekta y na lokaciji 2, i transakcija T_3 drži deljivi katanac nad primerkom z_2 objekta z na lokaciji 2. Pošto su transakcijama T_1, T_2, T_3, da bi nastavile sa radom, potrebni ekskluzivni katanci nad svim primercima objekata y, z, x, redom, s obzirom na dvofaznost transakcija, nijedna od transakcija ne može još da oslobodi katanac koji drži, a ne može ni da dobije katanac koji joj je potreban. Dakle, došlo je do uzajamnog blokiranja.

Lokalni graf čekanja na lokaciji 1 (odnosno 2) predstavljen je na slici 17.1 a) (odnosno 17.1 b)), a *globalni graf čekanja* na lokacijama 1 i 2 predstavljen je na slici 17.2. Da bi, na primer, transakcija T_1 na lokaciji 1 mogla da ažurira primerak y_1, ta transakcija mora, na svim lokacijama gde postoji primerak objekta y, imati katanac nad primerkom tog objekta; dakle, ona mora da sačeka da na lokaciji 2 dobije katanac na primerku y_2. Slično važi i za transakciju T_3 koja treba da ažurira objekat x, odnosno za transakciju T_2 koja treba da ažurira objekat z.



Slika 17.1: Lokalni grafovi čekanja na lokacijama 1 i 2

U prethodnom primeru došlo je do tzv. *globalnog uzajamnog blokiranja*, jer su u odnos uzajamnog blokiranja uključeni objekti na više lokacija (u ovom slučaju dve



Slika 17.2: Globalni graf čekanja na lokacijama 1 i 2

lokacije). Upravljači zaključavanja na pojedinačnim lokacijama nisu u stanju da identifikuju globalno uzajamno blokiranje, jer nema ciklusa u lokalnim grafovima čekanja.

Globalno uzajamno blokiranje moguće je detektovati ili sprečiti.

Jedan način za detekciju uzajamnog blokiranja je "centralizovanje" jedne lokacije i periodično prebacivanje lokalnih grafova čekanja sa svih drugih lokacija na centralnu lokaciju. Ovo rešenje je vrlo ranjivo zbog mogućnosti da centralna lokacija padne. Kompleksnije, ali i efikasnije rešenje može da uključi dodavanje novog čvora u lokalni graf čekanja pojedinačne lokacije i, koji predstavlja svaku transakciju koja čeka na kompletiranje neke transakcije sa lokacije i, odnosno svaku transakciju na čije kompletiranje čeka neka od transakcija lokacije i. Ukoliko nijedan tako proširen lokalni graf čekanja lokacije ne sadrži ciklus, globalnog uzajamnog blokiranja nema u celom sistemu; ako prošireni lokalni graf čekanja lokacije i sadrži ciklus, to je indikator mogućeg globalnog uzajamnog blokiranja u celom sistemu. Detaljniji opis ovog algoritma može se naći u [21], [49].

17.3 Vremenske oznake

Kod centralizovanih sistema sprečavanje uzajamnog blokiranja je bila skupa operacija pa se najčešće pribegavalo njegovom detektovanju i poništavanju nekih transakcija koje su u njemu učestvovale. Kod distribuiranih sistema operacija poništavanje transakcija je skuplja nego kod centralizovanih sistema, pa se primenjuje i druga mogućnost rešavanja problema uzajamnog blokiranja – njegovo sprečavanje. Ono se postiže po cenu smanjene konkurentnosti izvršenja skupa transakcija.

Jedna metoda konkurentnog izvršavanja skupa transakcija koja isključuje mogućnost uzajamnog blokiranja jeste *metoda vremenskih oznaka*. U ovoj metodi transakcijama se dodeljuju oznake vremena njihovog početka, i dopuštaju se konfliktne radnje samo u redosledu vremenskih oznaka pripadnih transakcija (pokušaj da se konfliktne radnje izvrše u drugačijem redosledu proizvodi poništavanje transakcije). Dok metoda zaključavanja, uz detekciju i razrešavanje (eventualno) nastalog uzajamnog blokiranja, obezbeđuje ekvivalentnost konkurentnog izvršenja sa

nekim (unapred se ne zna kojim) serijskim izvršenjem, metoda vremenskih oznaka obezbeđuje ekvivalentnost konkurentnog izvršenja sa specifičnim, unapred poznatim serijskim izvršenjem.

Osnovna ideja metode vremenskih oznaka sastoji se u sledećem:

- Svakoj transakciji dodeljuje se jedinstvena (u celom sistemu) vremenska oznaka. Ona se dobija dopisivanjem rednog broja lokacije oznaci vremena aktiviranja transakcije (oznaka lokacije je neophodna jer se više transakcija na više lokacija može istovremeno aktivirati).
- Ažuriranja se fizički upisuju u bazu tek pri uspešnom kompletiranju transakcije.
- Svaki fizički primerak logičkog objekta u bazi nosi vremensku oznaku transakcije koja ga je poslednja čitala, kao i transakcije koja ga je poslednja ažurirala (raznim optimizacijama vremenske oznake objekata u bazi mogu da se izostave).
- Svaki zahtev starije transakcije T_1 za operacijom odbacuje se, a transakcija se poništava i ponovo aktivira, ukoliko je taj zahtev u konfliktu sa operacijom koju je već, nad istim objektom, izvršila mlađa transakcija T_2. Operacija transakcije T_1 je u konfliktu sa operacijom transakcije T_2 ako se izvršavaju nad istim fizičkim objektom, i bar jedna od tih operacija je ažuriranje.
- Ponovno aktiviranoj transakciji dodeljuje se nova vremenska oznaka.

Kao što je za dvofazne transakcije u centralizovanom slučaju dokazana teorema o linearizovanosti njihovog konkurentnog izvršenja, može se i za distribuirani sistem dokazati slično tvrđenje, za slučaj da transakcije nose vremenske oznake.

Teorema 17.1 *Izvršenje koje poštuje redosled vremenskih oznaka je linearizovano.*

Dokaz: Dati skup transakcija $\{T_1, \ldots, T_n\}$ linearno je uređen vremenskim oznakama (jer su i same vremenske oznake linearno uređene). Kako na svakoj lokaciji važi da se izvršenje konfliktnih radnji obavlja u redosledu vremenskih oznaka transakcija, to za relaciju zavisnosti $Z(I)$ distribuiranog izvršenja I važi da je $(T_i, x_l, T_j) \in Z(I)$ (x_l je primerak logičkog objekta x na lokaciji l) ako i samo ako je $i < j$, tj. ako i samo ako je vremenska oznaka $(T_i) <$ vremenska oznaka (T_j). Ako postoji trojka $(T_i, x_l, T_j) \in Z(I)$, transakcije T_i i T_j su u *relaciji prethođenja*, $T_i \rightarrow T_j$. Relacija prethođenja je aciklična (ne može se dogoditi da je $(T_i, x_l, T_j) \in Z(I)$, $(T_j, y_k, T_p) \in Z(I)$ i $(T_p, z_m, T_i) \in Z(I)$, jer bi iz prethodnog sledilo da je $i < i$). Zato je i celo distribuirano izvršenje ekvivalentno serijskom $T_1, \ldots, T_n$.

Jedna od tehnikâ metode vremenskih oznaka, koja povećava konkurentnost izvršavanja i koja je primenjena u distribuiranom sistemu SDD-1, zasniva se na *klasama transakcija* i *analizi grafa konflikta*. U ovoj tehnici odbija se aktiviranje transakcije čiji zahtevi za podacima mogu da dođu u konflikt sa već aktiviranim transakcijama.

17.3.1 Klase transakcija

Tehnika zasnovana na klasama transakcija, kojom se postiže linearizovanost distribuiranog izvršenja skupa transakcija, može se opisati sledećim karakteristikama.

- Da bi se ograničio skup transakcija koje uopšte mogu da dođu u konflikt, uočavaju se transakcije koje mogu da operišu (čitaju ili ažuriraju) nad istim objektom. Zato se definiše *skup čitanja* i *skup ažuriranja* transakcije kao skup logičkih objekata koje ta transakcija čita odnosno ažurira.

- *Klasa transakcija* se definiše skupom čitanja i skupom ažuriranja. Pri tome, jedna transakcija pripada nekoj klasi transakcija ako je njen skup čitanja podskup skupa čitanja te klase transakcija, a skup ažuriranja – podskup skupa ažuriranja te klase transakcija. Tako, ako je T transakcija izmene statusa izdavača iz Jugoslavije, njen skup čitanja i skup ažuriranja definisan je sledećim upitom:

  ```
  SELECT *
  FROM   I
  WHERE  DRZAVA = 'Jugoslavija'
  ```

  (ili odgovarajućom projekcijom).

  Tada je transakcija T u klasi transakcija definisanoj skupom čitanja (i skupom ažuriranja)

  ```
  SELECT *
  FROM   I
  ```

  jer transakcija T čita, odnosno ažurira neki podskup skupa slogova svih izdavača.

  Jedna transakcija može biti istivremeno *član* većeg broja klasa transakcija (svih onih klasa čiji je skup čitanja nadskup njenog skupa čitanja, a skup ažuriranja nadskup njenog skupa ažuriranja).

- Dve klase transakcija su u konfliktu ako skup ažuriranja bilo koje od tih klasa ima neprazan presek sa skupom čitanja ili skupom ažuriranja druge klase. Transakcije koje pripadaju klasama koje nisu u konfliktu mogu nesmetano da se odvijaju paralelno, dok je za transakcije iz iste klase, ili za transakcije konfliktnih klasa, potrebna sinhronizacija (npr. vremenskim oznakama), tj. specifični protokoli.

- Administrator baze podataka (ili sam sistem, kao u slučaju sistema SDD-1) definiše inicijalno klase transakcija koje, u odnosu na predviđene transakcije, treba da imaju najmanji mogući skup čitanja i skup ažuriranja, da bi se smanjila mogućnost konflikta. Može se pretpostaviti da je svaka lokacija "domaćin" za tačno jednu klasu transakcija (jer se to može postići logičkim umnožavanjem lokacija, odnosno identičnim kopijama klase).

- Svaka lokacija l održava, za svaku drugu lokaciju k u sistemu, dva reda zahteva koje su ispostavile transakcije sa lokacije k, i to u poretku vremenskih oznaka tih transakcija: red zahteva za čitanje i red zahteva za ažuriranje objekata smeštenih na lokaciji l.
- Svaka lokacija obrađuje transakcije svoje klase jednu po jednu, tj. serijski, u redosledu njihovih vremenskih oznaka.
- Kada lokacija l dobije zahtev od transakcije sa vremenskom oznakom t za izvršenje operacije čitanja njenog objekta x, tada l odlaže taj zahtev sve dok se na vrhu reda ažuriranja svake lokacije čiji skup ažuriranja sadrži x ne nađe zahtev sa vremenskom oznakom većom od t. (To znači, prvo, da su svi zahtevi za ažuriranjem koji mogu doći u konflikt sa zahtevom prispelim na lokaciju l "mlađi" od tog zahteva, pa se on može bezbedno zadovoljiti, i drugo, razmatraju se samo klase transakcija koje mogu doći u konflikt sa zahtevom za čitanje, a ne sve klase transakcija). Zatim se izvršava operacija čitanja. Ako je prispeli zahtev bio za ažuriranjem, izvršava se ono ažuriranje koje ima najmanju vremensku oznaku na lokaciji l.

17.3.2 Analiza grafa konflikta

Ideja tehnike analize grafa konflikta je u daljem smanjenju skupova transakcijâ za koje je potrebno obezbediti određeni redosled izvršavanja.

Primer 17.2 Neka transakcija T čita i ažurira slog o određenoj knjizi, a transakcija T_1 samo čita taj isti slog. Te dve transakcije su u konfliktu ali se mogu nesmetano izvršavati u proizvoljnom redosledu – svako njihovo izvršenje je linearizovano.

Analiza grafa konflikta treba da iskoristi, u opštem slučaju, mogućnosti ilustrovane prethodnim primerom.

Graf konflikta sastoji se od skupa čvorova i tri skupa grana – horizontalnih, vertikalnih i dijagonalnih, i gradi se na sledeći način.

Za svaku klasu transakcija, graf konflikta sadrži po jedan čvor čitanja i jedan čvor ažuriranja. Čvor čitanja jedne transakcije postavlja se iznad čvora ažuriranja te transakcije i povezuje se sa njim vertikalnom granom. Ako skup ažuriranja klase transakcija C_i ima neprazan presek sa skupom ažuriranja klase transakcija C_j, tada se čvor ažuriranja klase C_i povezuje sa čvorom ažuriranja klase C_j horizontalnom granom; ako skup čitanja klase transakcija C_i ima neprazan presek sa skupom ažuriranja klase transakcija C_j, tada se čvor čitanja klase C_i povezuje sa čvorom ažuriranja klase C_j dijagonalnom granom. Značenje horizontalnih i dijagonalnih grana je da transakcije dve klase (koje takva grana spaja) mogu, mada ne moraju, doći u odgovarajuću konfliktnu situaciju (upis–upis ili čitanje–upis). Ako takvih grana nema između čvorova dve klase, onda transakcije te dve klase ne mogu biti u konfliktu i njihovo konkurentno izvršenje ne proizvodi nijedan od

problema konkurentnosti. Cilj analize grafa konflikta je identifikovanje parova klasa transakcija čija su izvršenja linearizovana, mada su klase povezane horizontalnim (odnosno dijagonalnim) granama.

Primer 17.3 Neka su zadate sledeće tri klase transakcija svojim skupovima čitanja i ažuriranja:

C_1: UVEĆATI_TIRAŽ_NAJVEĆIM_IZDAVAČIMA

skup čitanja:

```
SELECT *
FROM   I, KI
WHERE  I.I_SIF = KI.I_SIF AND I.STATUS =
      (SELECT MAX(STATUS)
       FROM   I)
```

skup ažuriranja:

```
SELECT *
FROM   I, KI
WHERE  I.I_SIF = KI.I_SIF AND I.STATUS =
      (SELECT MAX(STATUS)
       FROM   I)
```

Primer transakcije ove klase je "Uvećati tiraž najvećim izdavačima iz Jugoslavije za 10%".

C_2: UVEĆATI_STATUS_NAJVEĆIM_IZDAVAČIMA

skup čitanja:

```
SELECT *
FROM   I
WHERE  STATUS = ( SELECT MAX(STATUS)
                  FROM   I)
```

skup ažuriranja:

```
SELECT *
FROM   I
WHERE  STATUS = ( SELECT MAX(STATUS)
                  FROM   I)
```

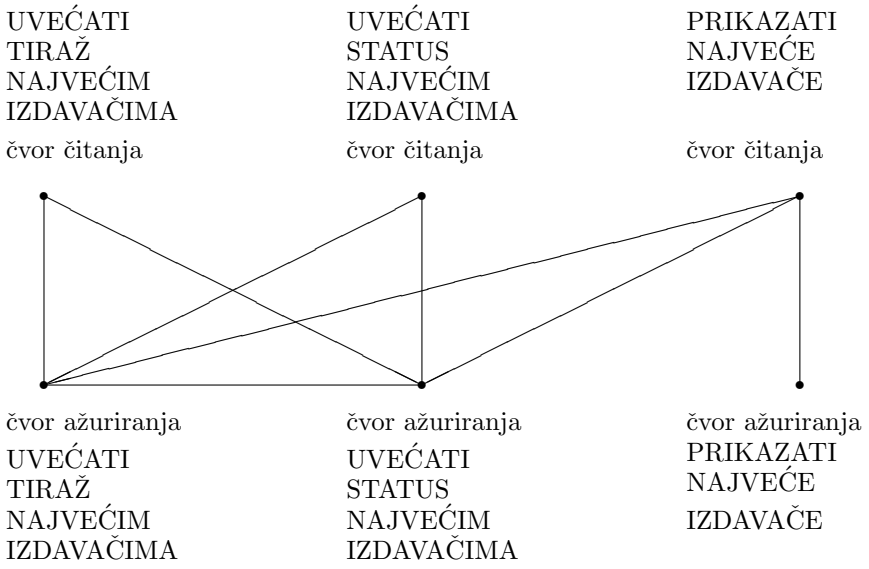
C_3: PRIKAZATIE_NAJVEĆE_IZDAVAČE

skup čitanja:

```
SELECT *
FROM   I
WHERE  STATUS = ( SELECT MAX(STATUS)
                  FROM   I)
```

skup ažuriranja: prazan skup.

Graf konflikta ove tri klase transakcija može se prikazati slikom 17.3.



Slika 17.3: Primer grafa konflikta

Transakcije klasâ C_1 i C_2 mogu doći u različite konfliktne situacije (upis–upis, čitanje–upis), pa se za njih mora obezbediti neki kompletni protokol kontrole konkurentnosti (npr. zaključavanjem ili vremenskim oznakama). Sa druge strane, proizvoljno konkurentno distribuirano izvršenje transakcije iz klase C_3 i transakcije iz klase C_1 je linearizovano ukoliko se obezbedi ispunjenost sledećeg uslova: redosled izvršavanja konfliktnih operacija čitanja (transakcije iz klase C_3) i ažuriranja (transakcije iz klase C_1) isti je na svim lokacijama; lokalna izvršenja takvih parova transakcija su bezuslovno linearizovana. Slično važi i za klase C_3 i C_2. Za kontrolu prethodnog uslova postoje efikasni mehanizmi, znatno jednostavniji od opštih metoda zaključavanja ili vremenskih oznaka ([8]).

Ova razlika u odnosu različitih klasa transakcija pri konkurentnom izvršavanju može se delimično "očitati" i iz grafa konflikta: čvorovi (čitanja i ažuriranja) klase transakcija C_3 imaju samo jednu (dijagonalnu) granu sa čvorovima klase C_1 i samo jednu granu sa čvorovima klase C_2, dok klase transakcija C_1 i C_2 imaju veći broj grana (tri grane) među svojim čvorovima.

Uopšte, može se dokazati sledeće: ako čvorovi jedne klase transakcija imaju najviše po jednu granu sa čvorovima druge klase transakcija, onda u slučaju horizontalne grane transakcije tih dveju klasa ne mogu biti u konfliktu i njihovo konkurentno izvršenje može se odvijati bez posebne kontrole; ako je ta grana dijagonalna, onda jednostavna sinhronizacija koja obezbeđuje prethodno navedeni uslov, dovodi do linearizovanog izvršenja. (Ova sinhronizacija u nekim slučajevima, kao što su transakcije koje se sastoje od jedne operacije čitanja odnosno upisa, postaje trivijalna. Takav je i primer 17.2). Ako je broj takvih grana veći (a najviše može biti

tri), onda su te dve klase transakcija *u ciklusu* i potrebno je kontrolisati konkurentno izvršenje njihovih transakcija nekim od protokola konkurentnosti, koji obezbeđuje isti redosled konfliktnih operacija po svim putanjama u ciklusu.

Klase transakcija definišu se statički, pri definisanju sheme baze podataka, pa se i analiza grafa konflikta sprovodi statički. Njen rezultat je obično tabela koja se distribuira svim lokacijama, i koja ukazuje na horizontalne odnosno dijagonalne grane koje zahtevaju kontrolu konkurentnosti.

17.4 Oporavak

Da bi se dobila atomičnost transakcije u distribuiranom okruženju, u smislu da se izvrše sve njene radnje ili nijedna, sistem za upravljanje transakcijama mora da obezbedi da se svi procesi te transakcije uspešno kompletiraju ili da se svi njihovi efekti poništе. Na primer, da bi se izvršilo ažuriranje logičkog objekta x od strane neke transakcije, potrebno je uspešno izvršiti ažuriranja svih primeraka tog objekta. Taj zadatak zahteva posebno razmatranje upravo zato što su primerci jednog objekta na raznim lokacijama koje, nezavisno jedna od druge, mogu da budu onemogućene da takvu radnju izvrše (npr. zbog pada sistema). Da bi se obezbedilo da ažuriranja svih primeraka svih objekata od strane jedne transakcije budu uspešno izvršena, ili da nijedno ne bude izvršeno, primenjuje se poseban protokol *dvofaznog kompletiranja transakcije*. Njemu se pristupa kada je koordinator transakcije obavešten da su svi procesi te transakcije završili sa radom (uspešno ili neuspešno), i sastoji se od sledeće dve faze:

- Za svaki logički objekat x koji transakcija T ažurira, i za svaki primerak x_i objekta x, koordinator te transakcije šalje poruku lokaciji i (njenom SUBP) da preliminarno ažurira objekat x_i. Ako je proces transakcije T na lokaciji i uspešno završen, lokacija i odgovara tako što izvrši preliminarno ažuriranje, tj. prepisuje vrednost x_i iz radnog prostora transakcije u svoju log datoteku, i obaveštava o tome koordinatora; u suprotnom, lokacija obaveštava koordinatora o neuspehu.

- Ako je koordinator obavešten da su sve radnje preliminarnog ažuriranja svih objekata koje transakcija T ažurira, na svim lokacijama, uspešno obavljene, on šalje poruke tim lokacijama da izvrše operaciju definitivnog ažuriranja svih svojih primeraka svih objekata koje T ažurira; lokacije odgovaraju tako što odgovarajuće vrednosti iz svojih log datoteka prepisuju u svoje lokalne baze i oslobađaju resurse koje je transakcija T držala. Kada se obave sva ta prepisivanja i o tome obavesti koordinator, izvršenje transakcije T je završeno. Ako je koordinator obavešten da bar jedna lokacija nije uspešno obavila prvu fazu, on šalje poruku svim lokacijama o poništavanju transakcije.

Ovakav protokol dvofaznog kompletiranja transakcije u implementacijama se poboljšava na razne načine, u cilju smanjenja broja poruka koje se prenose. Na primer, moguće je pretpostaviti da će kompletiranje biti uspešno i time eliminisati

slanje poruka iz prve faze. Ukoliko je kompletiranje zaista uspešno, broj poruka je znatno smanjen; ukoliko je kompletiranje neuspešno (a pretpostavlja se da je to ređi slučaj), broj poruka se uvećava, a zahteva se i poništavanje određenog broja radnji.

17.5 Pitanja i zadaci

1. Definisati sledeće pojmove:
   - lokalno izvršenje skupa transakcija
   - distribuirano izvršenje skupa transakcija
   - centralizacija upravljača zaključavanja
   - primarni primerak objekta
   - lokalni graf čekanja
   - globalni graf čekanja
   - protokol dvofaznog kompletiranja transakcije
   - klasa transakcija
   - skup čitanja
   - skup ažuriranja
   - graf konflikta
2. Objasniti specifičnosti dvofaznog zaključavanja u distribuiranom sistemu.
3. Opisati metodu vremenskih oznaka u konkurentnom izvršenju skupa transakcija.
4. Dokazati linearizovanost izvršenja skupa transakcija koje poštuje redosled vremenskih oznaka.
5. Opisati tehniku za konkurentno izvršavanje skupa transakcija zasnovanu na klasama transakcija.
6. Navesti primer analize grafa konflikta.

18

Klijent-server sistemi baza podataka

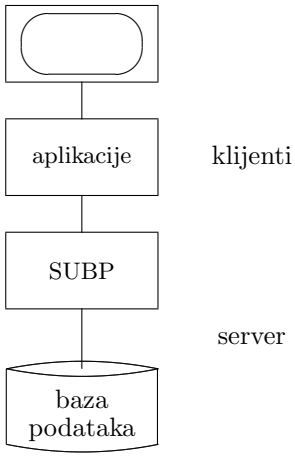
Klijent-server sistemi baza podataka zasnovani su na klijent-server arhitekturi, unekoliko različitoj od ANSI/SPARC arhitekture sistema baza podataka prikazane u glavi 1. Ova arhitektura se odnosi na jednostavnu logičku podelu odgovornosti unutar sistema na dva dela. Jedan deo (prvobitno u engleskoj terminologiji backend, *pozadinski*), server, u stvari je SUBP sa bazama podataka, i odgovoran je za definisanje podataka, manipulisanje podacima, bezbednost, integritet, oporavak, itd. Drugi deo (prvobitno je korišćen engleski termin frontend, *prednji sistem*), klijent, odgovoran je za rad aplikacija, kako korisničkih tako i sistemskih (na primer, procesor upitnog jezika, generator aplikacija, generator izveštaja, itd). Ovakva podela posla ne zahteva razdvajanje odgovornosti na razne računare (slika 18.1).

Mada se termin "klijent-server" danas najčešće odnosi na slučaj kada su klijent i server na odvojenim računarima, jer se mogućnost izdvajanja funkcijâ na odvojene računare pokazala veoma privlačnom i korisnom, termin se upotrebljava i za opisivanje okruženja u kome neki programi – "serveri" – pružaju određene usluge drugim programima – "klijentima", bez obzira da li se oni nalaze na različitim računarima ili ne.

Osnovni klijent-server sistemi, dakle, podrazumevaju veći broj klijenata koji dele isti server (slika 18.2). Moguće je, međutim, da jedan klijent pristupi većem broju servera, pri čemu je svaki pojedinačni zahtev (ili transakcija) upućen tačno jednom serveru.

Postoji više načina na koje aplikativni program sa klijenta može da pristupi bazi podataka na serveru, tj. postoji više metoda za ugradnju blokova za komunikaciju sa SUBP u aplikativni program:

- Ugnježdenje statičkog i dinamičkog SQL-a u aplikativni program.



Slika 18.1: Klijent-server arhitektura

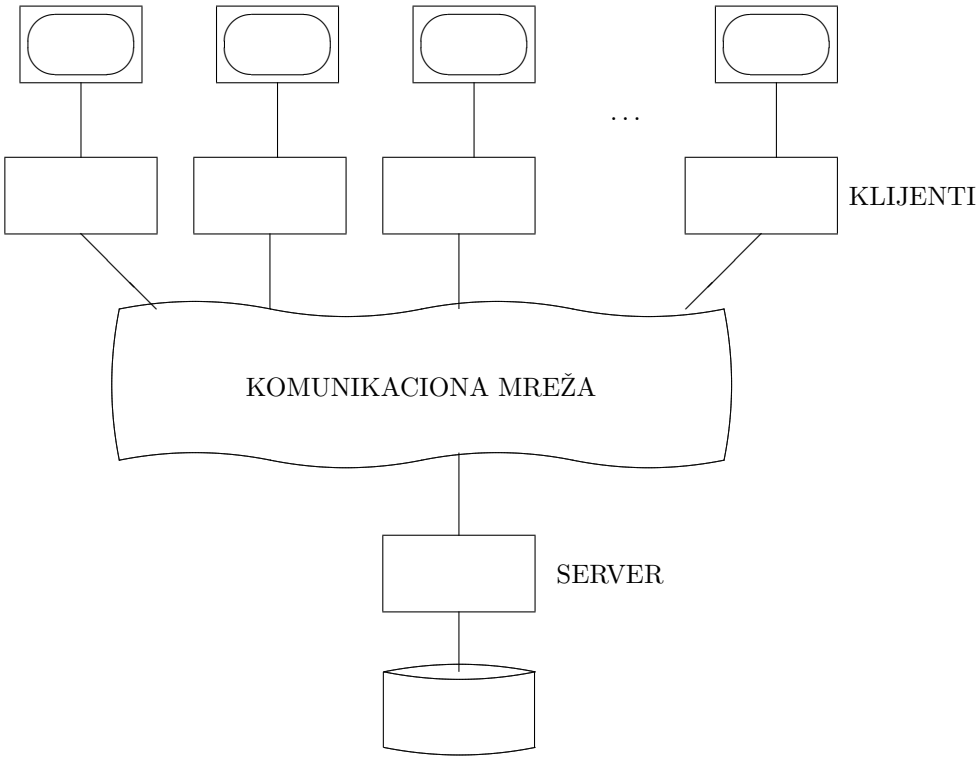
- Direktni pozivi, iz aplikativnog programa, funkcijâ odgovarajućeg SUBP (engl. Call Level Interface – CLI) za rad sa dinamičkim SQL iskazima. Pri korišćenju CLI sistema, aplikacija predaje dinamičke SQL iskaze kao argumente funkcijâ, upravljaču bazama podataka na obradu. Tako je CLI sistem alternativa ugnježdenom dinamičkom SQL-u.

- Zapamćene procedure na serveru. Zapamćena procedura (engl. stored procedure) je programski blok koji se poziva iz aplikacije na klijentu a izvršava se na serveru baza podataka. Piše se u odgovarajućim proširenjima SQL-a, kao što je, na primer, Oracle PL/SQL ili C-Java/SQL u DB2, kompilira i pamti u biblioteci odgovarajućeg SUBP. Tako se primenom zapamćenih procedura deo logike aplikacije prenosi sa klijenta na server, i povećava funkcionalnost servera.

  Najčešći razlog za korišćenje zapamćenih procedura je intenzivna obrada podataka iz baze podataka, koja proizvodi malu količinu rezultujućih podataka, ili činjenica da je skup operacija (koje se izdvajaju u zapamćenu proceduru) zajednički za više aplikacija. Naime, obrada pojedinačnog SQL iskaza (iz aplikacije na klijentu) nad bazom na serveru uključuje dve poruke – zahtev i prijem (v. **??**). U jednom aplikativnom programu može biti mnogo SQL iskaza, što usložnjava mrežni saobraćaj. Za korišćenje zapamćene procedure od strane klijenta potrebne su samo dve poruke za ceo proces.

  Zapamćene procedure ostvaruju mnoge prednosti:

  1. koriste prednosti moćnih servera;
  2. donose poboljšanja performansi statičkom (kompiliranom) SQL-u;



Slika 18.2: Klijent-server arhitektura: jedan server – više klijenata

3. smanjuju mrežni saobraćaj;
4. povećavaju bezbednost podataka učaurenjem podataka i operacija, tj. ograničavanjem pristupa objektima od strane aplikacije: pristup je moguć samo procedurama a ne i podacima;
5. poboljšavaju integritet podataka dopuštanjem raznim aplikacijama da pristupe istom programskom kodu.

Klijent-server arhitektura podseća na centralizovane (engl. mainframe) računarske sisteme sa skupovima terminala. Međutim, kod klijent-server sistema ispunjen je prvi uslov distribuiranih sistema, a to je u ovom slučaju autonomija klijenata. Oni vrše samostalno sve operacije upotrebe podataka, dok se sa serverom komunicira samo zbog potrebe za pretraživanjem ili ažuriranjem podataka. Uloga servera je usko i precizno definisana. Osim sa serverom baza podataka, koji upravlja bazama podataka, klijenti mogu da komuniciraju i sa drugim mrežnim serverima koji im pružaju usluge u obavljanju drugih operacija (nevezanih za bazu), npr. sa Web serverom.

18.1 Jedna realizacija klijent-server RSUBP

Jedan primer klijent-server relacionog sistema za upravljanje bazama podataka je IBM DB2 Universal Database (DB2 UDB) pod operativnim sistemima OS/2, Windows/NT ili UNIX. Ovaj sistem pripada novoj generaciji komercijalnih proizvoda razvijenih iz prototipskog RSUBP System R (IBM istraživačka laboratorija u San Hozeu). Prethodne generacije ovog proizvoda firme IBM za različite operativne sisteme nosile su različite nazive – DB2 za MVS, SQL/DS za VM, DB2/2 za OS/2, DB2/6000 za AIX/6000, DB2/400 za OS/400.

Pored servera baza podataka (IBM DB2 UDB), postoje i drugi IBM (softverski) serveri namenjeni raznim poslovima. Neki od tih servera su:

- IBM server za komunikaciju
- IBM server za direktorijume i bezbednost (engl. directory, security)
- IBM serveri za povezivanje sa Internetom

Sâm DB2 UDB uključuje grafičke alate koji omogućuju prilagođavanje i optimizovanje performansi, upravljanje svim serverima sa jednog mesta, razvoj aplikacija i obradu SQL upita.

18.1.1 DB2 server

IBM DB2 UDB server omogućuje lokalnim i udaljenim klijentima i aplikacijama da kreiraju, ažuriraju, kontrolišu i upravljaju relacionim bazama podataka korišćenjem SQL-a, ODBC-a (Open DataBase Connectivity) ili DB2 CLI (Call Level Interface) [**?**].

DB2 UDB server može biti pod operativnim sistemom OS/2 Warp, Windows NT ili UNIX.

Moguće je instalirati dva DB2 proizvoda: DB2 Enterprise Edition, ili DB2 Workgroup Edition. Jedna razlika među ovim proizvodima je što DB2 Enterprise Edition podržava i povezivanje sa centralizovanim (matičnim, host) sistemima, što dalje omogućuje korisnicima pristup bazama podataka na tim sistemima pod MVS/ESA, OS/390, OS/400, VM, VSE operativnim sistemima; druga razlika je u politici licenciranja klijenata (v. [**?**]).

Kontrolni centar Posebno značajan grafički alat DB2 UDB jeste *kontrolni centar* koji omogućuje izvršenje administrativnih zadataka na serveru, kao što su konfigurisanje, proizvođenje rezervnih kopija (engl. backup) i oporavak podataka, upravljanje direktorijumima, raspodelu poslova i upravljanje medijima. Kontrolni centar je novi grafički alat u ovoj verziji sistema DB2.

Kontrolni centar prikazuje objekte baze podataka kao što su same baze i tabele i njihove međusobne veze. Korišćenjem kontrolnog centra može se, sa jednog mesta, upravljati lokalnim ili udaljenim serverom baza podataka i njihovim objektima.

Kontrolni centar se može instalirati i na klijentu, i to na OS/2, Windows NT ili Windows 95 radnoj stanici.

Jedna od glavnih komponenti kontrolnog centra, *komandni centar*, koristi se za pristup i manipulisanje bazama podataka iz grafičkog interfejsa. Iz komandnog centra mogu se izdavati DB2 komande ili postavljati SQL upiti u interaktivnom prozoru i pratiti rezultati izvršavanja u rezultujućem prozoru, ili pamtiti u izlaznoj datoteci.

Pored mogućnosti prihvatanja zahteva sa klijenata, DB2 UDB (server baza podataka) ima ugrađen distribuirani (DRDA – Distributed Relational Database Architecture) aplikacioni server. On prihvata i obrađuje zahteve sa MVS, OS/400, VM i drugih DRDA klijenata. Distribuirana (DRDA) arhitektura gradi se na formalnim komunikacionim protokolima i funkcijama klijent-server komponenti aplikacije. Jedna transakcija u DB2 distribuiranom okruženju može da pristupi većem broju aplikacionih servera, pod uslovom da svim objektima jednog SQL iskaza upravlja jedan aplikacioni server.

18.1.2 DB2 klijent

Klijenti DB2 mogu biti pod operativnim sistemom OS/2, Windows NT, Windows 95, Windows 3.1x, na UNIX platformama (AIX, HP-UX, Solaris, SINIX, Silicon Graphics IRIX, SCO OpenServer), Macintosh, DOS.

Na svakoj klijent radnoj stanici, bez obzira na platformu, instaliran je paket CAE (Client Application Enabler), koji omogućuje pristup DB2 serveru i povezivanje na bilo koji DB2 UDB proizvod. Ovaj paket uključuje sve neophodne komponente DB2 sistema, zatim ODBC podršku korišćenju DB2 baze podataka od strane drugih (ne-DB2) programa za obradu podataka, opsežnu dokumentaciju i DB2 komandni interfejs (Command Line Processor). Za svaki operativni sistem postoji odgovarajući CAE proizvod, ali sve platforme ne podržavaju i sve klijentske alate (v. [**?**]).

Na klijentu mogu da se instaliraju i komponente za udaljeno administriranje servera.

18.1.3 Ostali DB2 proizvodi

Ostali DB2 proizvodi uključuju:

- DB2 Application Developer's Kit sadrži kolekciju DB2 UDB proizvoda, DB2 proizvoda za povezivanje, Software Developer's Kits – alata za razvoj aplikacija za sve podržane platforme.
- DB2 Universal Database Personal Edition, personalno izdanje sistema DB2 UDB, omogućuje kreiranje i korišćenje lokalnih baza i pristup udaljenim bazama podataka ako su na raspolaganju.
- DB2 Connect Enterprise Edition / DB2 Connect Personal Edition (ranije DDCS multi-user gateway / DDCS single user) obezbeđuje klijentima u mreži

/ na pojedinačnoj radnoj stanici pristup DB2 bazama podataka pod operativnim sistemima MVS/ESA, OS/390, OS/400, VM, i VSE.

- DB2 Universal Database Extended Edition (ranije DB2 Parallel Edition) omogućuje particioniranje baze podataka na većem broju samostalnih računara sa istim operativnim sistemom, svaki sa sopstvenim procesorom i memorijom. Korisniku i aplikaciji u razvoju baza izgleda kao da je jedinstvena. SQL operacije mogu paralelno da se izvršavaju nad particijama baze podataka, smanjujući vreme izvršavanja upita.

Alati za razvoj aplikacija Posebno značajan DB2 proizvod koji se uobičajeno instalira i na serveru i na klijentima jeste Software Deloper's Kit (DB2 SDK) – alat koji omogućuje razvoj aplikacija, tj. kolekcija alata koja uključuje i biblioteke, datoteke zaglavlja, dokumentovane sumeđe za programiranje aplikacija (API – Application Programming Interface), i primere programa za građenje tekstuelnih, multimedijalnih ili objektno-orijentisanih aplikacija.

Ako se instalira na serveru, SDK omogućuje pristup i udaljenim i lokalnim bazama podataka; ako se instalira na klijentu, SDK omogućuje pristup samo udaljenim bazama podataka.

Kao i Client Application Enabler, i SDK se gradi za svaku specifičnu platformu – operativni sistem.

DB2 SDK omogućuje razvoj aplikacija koje koriste razne metode ugradnje baze podataka u aplikativni program:

- ugnježđeni SQL
- CLI (Call Level Interface) razvojnu okolinu (koja je kompatibilna sa Microsoft ODBC)
- Java Database Connectivity (JDBC)
- sumeđe za programiranje aplikacija – API
- zapamćene procedure

Prve dve mogućnosti već su opisane u delu 3.4 – Aplikativni SQL. DB2 SDK podržava nekoliko programskih jezika (uključujući COBOL, C, C++) za razvoj aplikacija, i obezbeđuje prekompilatore za te jezike.

Treća metoda ugradnje baze podataka u aplikativni program, JDBC, predstavlja posebno zanimljivu mogućnost u savremenom razvoju RSUBP – to je pristup bazama podataka preko Interneta. Popularnost Interneta i World Wide Web-a postavlja zadatak pristupa podacima sa Web-a kao obavezu svakog RSUBP-a.

Pozivi DB2 API funkcija koriste se za izvršavanje administrativnih funkcija kao što su pravljenje rezervnih kopija – **sqlubkp**, ili restaurisanje baze podataka – **sqlurst**. API funkcije su isprogramirane u programskom jeziku, npr. C-u, i

mogu se pozivati eksplicitno ili implicitno; one se pozivaju implicitno pri prekompiliranju aplikativnog programa sa ugnježdenim SQL-om kada se SQL iskazi zamenjuju pozivima API funkcija u sintaksi odgovarajućeg jezika (v. sledeću tačku – Priprema i izvršavanje aplikativnih programa).

Proces razvoja aplikacije u klijent-server okruženju podrazumeva da su ispunjene sledeće pretpostavke:

- na serveru je instaliran upravljač bazama podataka (RSUBP);
- na klijent ili server radnoj stanici na kojoj se razvija aplikacija instaliran je DB2 SDK softver;
- odgovarajući jezički procesor je instaliran i konfigurisan;
- instaliran je i konfigurisan zajednički komunikacioni protokol za server i klijent;
- postoji mogućnost povezivanja na bazu podataka iz komandne linije (command line processor).

Aplikativni program može se razvijati na serveru ili na bilo kom klijentu na kome je instaliran DB2 SDK softver. Aplikacija se može izvršavati ili na serveru ili na bilo kom klijentu na kome je instaliran CAE (Client Application Enabler).

Da bi aplikativni program mogao da se izvršava, neophodno je da prethodno uspostavi vezu – poveže se – sa ciljnom bazom podataka. Uspostavljanjem veze identifikuje se korisnik (svojom korisničkom identifikacijom) koji izvršava program, i server (svojim imenom) nad kojim se program izvršava. U opštem slučaju aplikativni program može u jednom trenutku da bude povezan samo sa jednim serverom baza podataka; taj server se naziva *tekućim serverom*. U slučaju distribuirane transakcije, aplikativni program može se povezati sa većim brojem servera baza podataka, pri čemu je u svakom trenutku samo jedan od tih servera – tekući server.

Veza programa sa serverom baza podataka uspostavlja se eksplicitno iskazom CONNECT koji treba da bude prvi izvršni iskaz, ili implicitno vezom na podrazumevani server baza podataka (pri startovanju aplikativnog programa, ako je omogućeno implicitno povezivanje). Program zatim može da izvršava proizvoljni broj iskaza za manipulisanje podacima, definisanje i održavanje objekata baze podataka, kontrolnih operacija kao što je dodela prava korisnicima, potvrđivanja ili poništavanja promena. Veza sa bazom podataka prekida se izvršavanjem iskaza CONNECT RESET ili DISCONNECT.

18.1.4 Priprema i izvršavanje aplikativnih porograma

U ovoj tački biće izložen postupak pripreme aplikativnog programa za izvršenje u klijent-server okruženju.

Izvorni program P na matičnom jeziku (npr. C-u) u koji je ugnježden statički SQL, kreira se, korišćenjem editora, u standardnoj ASCII datoteci, i ima odgovarajuću ekstenziju (npr. sqc za SQL ugnježden u C).

Prvi korak u pripremi programa za izvršenje je njegovo *prekompiliranje* tj. obrada programa korišćenjem odgovarajućeg *prekompilatora*.

Aplikativni (.sqc) program se prekompilira izvršavanjem PREP komande i tom prilikom obavljaju se sledeće radnje:

1. SQL iskazi u aplikativnom programu stavljaju se pod komentar i generišu se pozivi DB2 API funkcija za te iskaze, koje matični (C) kompilator "razume"; tako modifikovani izvorni program P' je program na matičnom (C) jeziku i on se smešta u datoteku sa odgovarajućom ekstenzijom (za C jezik – .c).

2. Kreira se *vezna datoteka*[1] (engl. bind file) sa ekstenzijom .bnd koja sadrži izdvojene (iz aplikativnog programa P) SQL iskaze i informacije o njima; vezna datoteka se koristi u trećem koraku – povezivanju (engl. bind) aplikacije i baza podataka. Umesto kreiranja vezne datoteke, u koraku prekompiliranja može se izvršiti i povezivanje aplikacije sa bazom podataka (v. treći korak – povezivanje).

3. Generišu se poruke o procesu prekompiliranja koje se mogu usmeriti u datoteku poruka.

Pre prekompiliranja aplikacija mora da se poveže (CONNECT komandom) sa serverom, tj. sa specifičnom bazom podataka, bilo eksplicitno bilo implicitno. Mada se aplikativni program prekompilira na klijent radnoj stanici a i modifikovani izvorni program i poruke generišu na klijentu, veza sa serverom je potrebna zbog izvesnih provera i kontrole.

Drugi korak je klasična priprema modifikovanog izvornog programa P' za izvršenje – kompiliranje korišćenjem C kompilatora (kompiliraju se eventualno i drugi izvorni C programi koji ne sadrže SQL iskaze) i povezivanje povezivačem (linkerom) matičnog jezika (povezuju se svi korisnički moduli kao i biblioteka matičnog jezika i biblioteka API SUBP-a). Tako se dobijaju prevedeni programi i izvršni program u datotekama sa ekstenzijama .obj odnosno .exe.

Treći korak u pripremi aplikativnog programa za izvršenje je *povezivanje* (engl. binding) aplikacije i baze podataka. Izvršava ga proces *veznik* (engl. bind) koji kreira *pristupni paket* [2] (ako povezivanje nije već izvršeno u koraku prekompilacije) na osnovu sadržaja vezne datoteke, koji će RSUBP koristiti da bi pristupio bazi podataka u vreme izvršavanja aplikacije. Pristupni paket smešta se u bazu podataka. Veznik je kompilator baze podataka i "prevodi" sadržaj vezne datoteke gradeći pri tom pristupni paket kao optimizovanu upravljačku strukturu, nivoa

[1] U terminologiji ranijih verzija sistema DB2 vezna datoteka se nazivala "moduo zahteva nad bazom podataka" (engl. DBRM – Data Base Request Modul).

[2] U terminologiji ranijih verzija sistema DB2 paket se nazivao "plan aplikacije" (engl. application plan).

bliskog mašinskom; veznik pri tom proverava prava korisnika (koji ga poziva) za izvršenje pojedinih radnji.

Ako se aplikacija obraća većem broju baza podataka, za svaku se mora kreirati pristupni paket. Takoďe, ako aplikacija uključuje veći broj posebno prekompiliranih izvornih modula, za svaki moduo se kreira po jedan pristupni paket.

Ukoliko se struktura pristupnih puteva promeni (na primer, doda se novi indeks ili ukloni postojeći), da bi aplikacija mogla da koristi nove mogućnosti pristupa, veznik mora ponovo da se pozove i da se kreira novi pristupni paket. Ponovno pozivanje veznika može biti automatsko – implicitno (na primer u slučaju uklanjanja indeksa koji je korišćen u pristupnom paketu) ili eksplicitno (na primer, u slučaju dodavanja novog indeksa koji korisnik želi da iskoristi).

Aplikacija se izvršava na klijentu; kada se izvršavanjem programa doďe do poziva DB2 API funkcije (zahteva za operacijom nad bazom), upravljanje preuzima server koji izvršava odgovarajuću operaciju nad bazom, i vraća rezultat i kontrolu upravljanja programom klijentu.

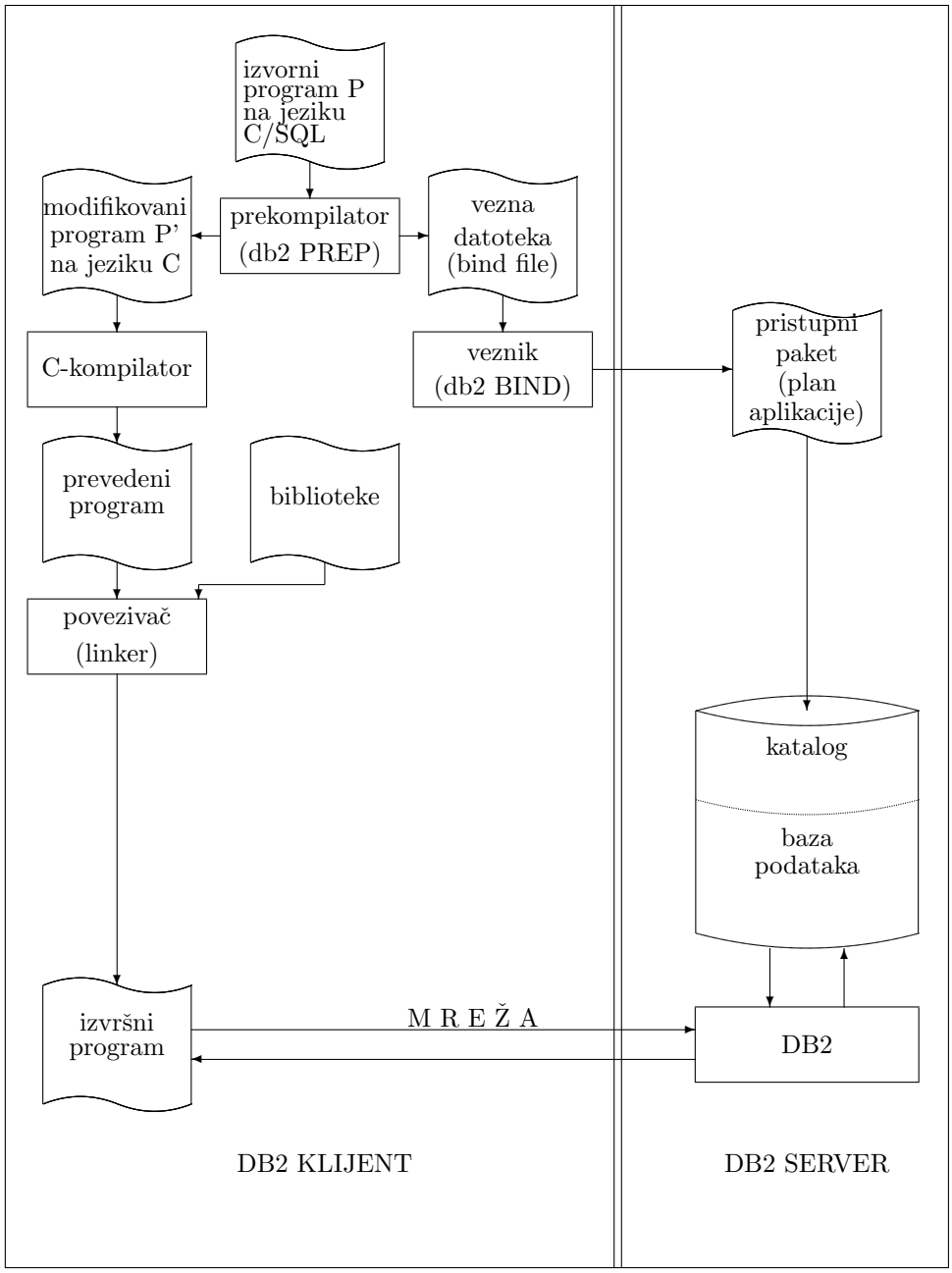
Postupak pripreme i izvršenja aplikativnog programa u klijent-server okruženju može se prikazati shematski kao na slici 18.3. Matični jezik je programski jezik C. Pravougaonicima su predstavljeni sistemski programi koji se koriste u pripremi i izvršenju aplikacije, zakrivljenim kvadratima prikazuju se datoteke koje se koriste i proizvode u procesu pripreme, dok je valjkom predstavljena baza podataka, uključujući i katalog kao njen sistemski deo.

Prethodno razmatranje odnosi se na aplikativne programe sa statičkim SQL-om. Programi sa dinamičkom pripremom i izvršavanjem SQL iskaza (korišćenjem PREPARE i EXECUTE ili EXECUTE IMMEDIATE) ne prekompiliraju se već ceo postupak obrade prolaze u vreme izvršenja.

U aplikativnom programu može da se koristi DB2 CLI sistem koji je baziran na Microsoft Open Database Connectivity (ODBC) specifikaciji, i na ISO CLI standardu. Ovaj sistem koristi dinamičke SQL iskaze kao argumente funkcijâ koji se prosleďuju sistemu DB2 na obradu. U tom slučaju koraci prekompiliranja i povezivanja aplikacije sa bazom (bind) nisu potrebni jer se koriste javni pristupni paketi koje obezbeďuje DB2. Dovoljno je samo kompilirati i linkovati aplikaciju.

Iz DB2 CLI aplikacije mogu da se pozivaju i zapamćene procedure. Tada se aplikacija izvršava delom na klijentu a delom na serveru (zapamćene procedure izvršavaju se na serveru), smanjujući tako mrežni saobraćaj.

Sistem DB2 je kompilatorski sistem, za razliku od gotovo svih nerelacionih sistema koji su interpretatorski. Ovo važi i za interaktivni DB2 i za aplikativni DB2, tj. prevode se ne samo aplikativni programi već i interaktivno zadati SQL iskazi. Razlog za prednost koja se daje prevoďenju je upravo u sistemskoj optimizaciji koja karakteriše relacione sisteme, i koja značajno povećava efikasnost izvršavanja kako aplikativnih programa tako i interaktivnih SQL iskaza.



Slika 18.3: Priprema i izvršenje aplikativnog programa u DB2

18.1.5 Fizička organizacija podataka u sistemu DB2

Za predstavljanje fizičkih podataka sistem DB2 koristi nekoliko vrsta objekata:

- *Fizički slog* koji odgovara vrsti bazne tabele; sastoji se od prefiksa sloga koji ukazuje na tabelu čijoj fizičkoj reprezentaciji slog pripada, i od fizičkih polja koja odgovaraju vrednostima atributa, ali osim njih sadrži i dodatne informacije kao što je dužina polja. Fizički slogovi su grupisani u *stranice* veličine 4KB (v. glavu 11).

- *Prostor tabela* (engl. tablespace); sadrži tabele, indekse, kompleksne objekte i dugačke (LONG) tipove podataka. Prostor tabela je osnovna fizička struktura u sistemu DB2 i može biti jednog od sledeća dva tipa:

  1. prostor tabela kojim upravlja upravljač bazama podataka (engl. DMS – Database Managed Space)
  2. prostor tabela kojim upravlja operativni sistem (engl. SMS – System Managed Space).

  Svaki prostor tabela sastoji se od *skladišta* (uređaja ili direktorijuma) u kojima su smešteni objekti (npr. tabele) unutar tog prostora tabela. Podaci pročitani iz pojedinih skladišta prostora tabela smeštaju se u prostor unutrašnje memorije koji se naziva *red bafera* (engl. buffer pool). Red bafera pridružen je prostoru tabela. U slučaju paralelne MPP hardverske konfiguracije (engl. massive parallel processing), korišćenjem UDB proizvoda DB2 UDB Extended Edition (v. 18.1.3 – Ostali DB2 proizvodi), baza podataka može biti *particionirana* na više čvorova (klijenata i servera); tada se prostoru tabela može dodeliti *grupa čvorova* na kojima se nalaze delovi tog prostora tabela.

- *Baza podataka* – skup prostora tabela kojima se zajedno upravlja; skup svih podataka podeljen je na veći broj disjunktnih korisničkih i sistemskih baza podataka; sistemski katalog je deo jedne od sistemskih baza podataka;

Dijalekt SQL-a koji podržava DB2 uključuje, pored iskaza za kreiranje, izmenu i uklanjanje baznih tabela i indeksa, i iskaze za kreiranje, uklanjanje i izmenu prostora tabela i reda bafera. Pored SQL iskaza za ove operacije, sistem DB2 uključuje komandu za kreiranje i uklanjanje baze podataka.

- DB2 komanda za kreiranje baze podataka je oblika

  CREATE DATABASE ime-baze-podataka
  [AT NODE] [ON oznaka-uređaja]
  [CATALOG TABLESPACE definicija-prostora-tabela]
  [USER TABLESPACE definicija-prostora-tabela]

[TEMPORARY TABLESPACE definicija-prostora-tabela]

Ova komanda može da se izvrši samo sa servera. Njenim izvršenjem inicijalizuje se nova baza podataka, kreiraju tri početna prostora tabela i sistemske tabele, i alocira log datoteka za oporavak. U paralelnoj hardverskoj konfiguraciji (MPP) ova komanda se odnosi na sve čvorove navedene u odgovarajućoj datoteci (db2nodes.cfg). Baza podataka tada postaje *particionirana*, a čvor sa kog sa ova komanda izvršava postaje kataloški čvor nove baze podataka.

AT NODE opcija (u MPP konfiguraciji) precizira da se baza podataka kreira samo na čvoru sa koga se komanda izdaje.

ON oznaka-uređaja precizira slovo pridruženo uređaju na kome se kreira nova baza podataka (npr. C:).

Ovom komandom se uvek, eksplicitno ili implicitno, kreira po jedan CATALOG, USER i TEMPORARY prostor tabela za katalog, korisnike i privremene podatke (SYSCATSPACE, USERSPACE1, TEMPSPACE1, redom). Ako se odgovarajuće opcije ne navedu kreiraju se podrazumevani SMS prostori tabela sa odgovarajućim brojem direktorijuma.

Definicija prostora tabela uključuje opcije vezane za tu vrstu objekta, kao što su tip prostora tabela (SYSTEM / DATABASE), direktorijum ili uređaj i broj stranica, itd. (v. iskaz CREATE TABLESPACE).

- SQL iskaz kreiranja prostora tabela je kompleksan, i u grubim crtama ima oblik

$$\text{CREATE} \left\{ \begin{array}{l} [\text{REGULAR}] \\ [\text{LONG}] \\ [\text{TEMPORARY}] \end{array} \right\} \text{TABLESPACE ime-prostora-tabela}$$

[IN NODEGROUP ime-grupe-čvorova]

$$[\text{MANAGED BY} \left\{ \begin{array}{l} \text{SYSTEM sistemska-skladišta} \\ \text{DATABASE skladišta baze podataka} \end{array} \right\}]$$

[BUFFERPOOL ime-reda-bafera]

[ostali-parametri].

Ovim iskazom kreira se novi prostor tabela u bazi podataka (sa kojom je prethodno izvršeno povezivanje), dodeljuju mu se skladišta i definicija novog prostora tabela beleži u katalogu.

Prostor tabela, s obzirom na podatke koji se u njega mogu upisati, može biti REGULAR za sve podatke osim privremenih, LONG za LOB (Large Object) kolone tabela i TEMPORARY za privremene tabele.

Opcija MANAGED BY opisuje da li prostorom tabela upravlja operativni sistem (SYSTEM) ili upravljač bazama podataka (DATABASE); pri tom se navode i skladišta za prostor tabela – imena direktorijuma u slučaju SMS

ili imena datoteka ili uređaja, sa navedenim brojem stranica veličine 4KB, u slučaju DMS prostora tabela.

Opcija IN NODEGROUP (u MPP konfiguraciji) precizira na kojoj se grupi čvorova particionirane baze podataka kreira novi prostor tabela. U tom slučaju postoji i mogućnost navođenja specifičnih čvorova na kojima se prostor tabela nalazi u okviru date grupe.

- SQL iskaz kreiranja baznih tabela, CREATE TABLE, pored definicije kolona i primarnog i stranih ključeva (v. tačke 1.4.1, 1.4.2) i uslova ograničenja , ima i mogućnosti navođenja prostora tabela za tu tabelu (tipa REGULAR), ključa particioniranja (u slučaju particionirane baze podataka), prostora tabela za indekse i dugačke (LONG) objekte, oblika

  [IN ime-prostora-tabela

  [INDEX IN ime-prostora-tabela]

  [LONG IN ime-prostora-tabela]]

  [PARTITIONING KEY (lista-kolona)]

- SQL iskazom za kreiranje reda bafera, CREATE BUFFERPOOL kreira se novi red bafera u tekućoj bazi podataka, i precizira se veličina u stranicama; u slučju particionirane baze podataka, u iskazu se mogu navesti i čvorovi na kojima će se ovaj red bafera kreirati.

Ukoliko se pri kreiranju bilo kog od prethodnih objekata neki od parametara fizičke strukture ne navede, uzimaju se podrazumevane sistemske vrednosti.

- Iskaz DROP $\left\{ \begin{array}{l} \text{[TABLESPACE ime-prostora-tabela]} \\ \text{[TABLE ime-tabele]} \\ \text{[INDEX ime-indeksa]} \\ \text{[BUFFERPOOL ime-reda-bafera]} \end{array} \right\}$

  uklanja odgovarajući fizički objekat, kao i sve objekte koji su posredno ili neposredno zavisni od njega; iz kataloga se uklanja opis tog objekta a pristupni paketi koji od njega zavise označavaju se nevažećim.

  DROP iskaz može da se primeni i na paket, shemu, pogled, triger, proceduru i druge kreirane objekte.

  Pored SQL DROP iskaza, postoji i DB2 komanda za uklanjanje baze podataka,

  DROP DATABASE ime-baze-podataka

  koja se može primeniti i samo na specifični čvor u particioniranoj bazi (MPP konfiguraciji).

- Iskaz izmene (ALTER) može se primeniti na izmenu definicije tabele, prostora tabela ili reda bafera, i odnosi se na fizičke parametre ovih objekata, odnosno na dodavanje kolona i definisanje ili uklanjanje primarnog i stranog ključa i uslova ograničenja u slučaju bazne tabele.

Detaljniji opis fizičke organizacije sistema DB2 može se naći u [24], [27]. Detalji sintakse i semantike SQL iskaza koji podržavaju fizičku strukturu sistema opisani su u [37], [**?**].

18.2 Pitanja i zadaci

1. Definisati sledeće pojmove:

klijent-server arhitektura	zapamćena procedura
CLI sistem	pristupni paket

2. Koje su razlike između distribuiranih i klijent-server SUBP?
3. Kako aplikativni progran može da pristupi bazi podataka na serveru?
4. Opisati arhitekturu DB2 UDB.
5. Opisati postupak pripreme aplikativnog programa za izvršenje u sistemu DB2.
6. Opisati funkciju prekompilatora.
7. Šta je veznik i koja mu je funkcija?
8. Objasniti vrste objekata koje DB2 koristi za predstavljanje fizičkih podataka.
9. Navesti DB2 (SQL) iskaze za kreiranje i uklanjanje baza podataka, prostora tabela i baznih tabela.

Dodatak A

Objektno orijentisane baze podataka

Relacioni sistemi za upravljanje bazama podataka predstavljali su značajni napredak u tehnologiji upravljanja bazama podataka u odnosu na prethodne generacije hijerarhijskih i mrežnih sistema. Njihove prednosti, za veliki broj korisnika i aplikacija, ogledale su se pre svega u

- neproceduralnim upitnim jezicima i
- značajnoj meri nezavisnosti podataka.

Mada veoma uspešni u obradi poslovnih podataka i aplikacija, relacioni sistemi se, s obzirom na oskudan repertoar tipova podataka i zahtev za normalizovanošću relacija, pokazuju neadekvatnim za široke klase aplikacija koje manipulišu podacima kompleksnije strukture. Kao takve aplikacije obično se vide aplikacije nad geometrijskim, tekstuelnim i programskim podacima, odnosno aplikacije projektovanja uz pomoć računara (Computer Aided Design – CAD), zatim geografski informacioni sistemi (GIS), hipertekstuelne aplikacije, čuvanje i obrada dokumenata, kao i aplikacije razvoja softvera uz pomoć računara (Computer Aided Software Engineering – CASE).

Zbog uočenih nedostataka, proizvodjači relacionih sistema proširuju svoje proizvode novom funkcionalnošću, kao što je:

- podrška bogatijem skupu tipova i pravila,
- nasledjivanje atributa u odnosima tipa generalizacija/specijalizacija,
- učaurenje podataka i operacija (funkcija, procedura), pri čemu je način pristupa podacima ograničen definisanim operacijama (funkcijama, procedurama).

Ovakvim proširenjima relacioni sistemi izlaze iz okvira relacionog modela, i prerastaju u sisteme nove generacije ([63]).

Sa druge strane, neke od pomenutih funkcija kojima se proširuju relacioni sistemi (kompleksni objekti, nasledjivanje, učaurenje) predstavljaju sastavni deo programske paradigme poznate kao objektno orijentisana[1] paradigma. Ova programska paradigma privukla je posebnu pažnju krajem osamdesetih godina, i predstavlja važan istraživački pravac u većem broju računarskih disciplina, kao što su programski jezici, reprezentacija znanja, baze podataka, arhitektura računara ([41]). Zato jedan pravac razvoja oblasti baza podataka jeste definicija novog modela podataka, *objektno orijentisanog modela*, i implementacija sistemâ za upravljanje bazama podataka nad tim modelom.

Baze podataka izgradjene nad objektno orijentisanim modelom su *objektno orijentisane baze podataka*, a sistem za upravljanje takvim bazama zove se, po analogiji se relacionim i distribuiranim sistemima, *objektno orijentisani sistem za upravljanje bazama podataka*, u oznaci OOSUBP. Objektno orijentisane baze podataka, zajedno sa OOSUBP, obrazuju *sistem objektno orijentisanih baza podataka*. Za obe vrste sistema koristi se i kraći termin, *objektni sistem*, kada je iz konteksta jasno na koju vrstu sistema se termin odnosi.

Pored podrške novim svojstvima vezanim za objektno orijentisani model, pred OOSUBP se koncepcijski postavljaju ista pitanja i zadaci kao i pred RSUBP (upitni jezik, integritetni deo, upravljanje transakcijama, fizička organizacija, indeksi i grupisanje objekata, konkurentnost i autorizacija); odgovori i rešenja, medjutim, sada zahtevaju sasvim drugačija razmatranja (v. odeljak A.2).

Mada objektno orijentisani pristup programiranju i projektovanju kompleksnih programskih sistema doživljava veliku popularnost u raznim disciplinama, ne postoji potpuna saglasnost o tome šta objektna orijentisanost uopšte znači i, posebno, šta su to objektno orijentisane baze podataka. Za razliku od precizne definicije relacionog modela, ne postoji jedinstveni objektno orijentisani model podataka; zato su objektni sistemi, koncepti na kojima su izgradjeni, kao i njihova funkcionalnost – dosta šaroliki. Bez obzira na odsustvo jedinstva u konceptima i standarda u modelu, arhitekturi i jezicima, postoji veći broj prototipskih i komercijalnih OOSUBP, npr. GemStone ([10]), ObjectStore ([45]), Iris ([69]), ORION ([41]), O2 ([28]), itd. Kako se funkcionalnost proširenih relacionih sistema preklapa (a u nekim sistemima i poklapa) sa objektnim sistemima, i oni prvi se mogu smatrati objektnim sistemima (npr. POSTGRES, [60], Starburst [47]).

U sledećim odeljcima biće prikazani koncepti objektno orijentisane paradigme koji su značajni za oblast baza podataka, i to objektno orijentisani model podataka (jezgro i proširenja), dugotrajne transakcije i objektno orijentisani upitni jezici. Poslednji odeljak posvećen je sistemima baza podataka nove generacije.

[1] Termini "objektno orijentisani" i "objektni" obično se upotrebljavaju u sinonimnom značenju; tako će biti i u ovom tekstu. Prvi termin je stariji i njegova skraćenica (OO) postala je sastavni deo niza drugih imena i skraćenica. U punim nazivima, drugi termin ima tendenciju potiskivanja prvog.

A.1 Objektno orijentisani model podataka

Model podataka predstavlja logičku reprezentaciju objekata realnog sveta i njihovih odnosa, mogućnosti manipulisanja objektima i zadavanja pravila integriteta. Osnovna ideja objektno orijentisanog modela je da podigne nivo apstrakcije podataka, tako da se, umesto bitovima, bajtovima, slogovima, poljima, n-torkama, manipuliše prirodnijim entitetima iz realnog sveta, objektima. Kao što je već rečeno, jedinstveni objektno orijentisani model ne postoji. U prvom Manifestu sistemâ objektno orijentisanih baza podataka, ideolozi ovih sistema odustaju od namere za definisanjem apstraktnog objektno orijentisanog modela na kome bi se bazirale implementacije OOSUBP, uz sledeći stav: "U odnosu na specifikaciju (OO) sistema primenjujemo darvinistički pristup: nadamo se da će se, iz niza eksperimentalnih prototipova koji su u izgradnji, iskristalisati jedinstveni objektno orijentisani model" ([5]).

U odsustvu jedinstvenog objektno orijentisanog modela podataka, pogodno je identifikovati minimalni skup objektnih koncepata koje podržava (na jedan ili drugi način) svaki objektni sistem (*jezgro* OO *modela podataka*, [42]), i prepoznati moguće koncepte nadgradnje tog minimalnog skupa, prisutne u implementaciji različitih sistema.

A.1.1 Jezgro objektno orijentisanog modela podataka

Jezgro OO modela podataka, sa aspekta baza podataka, predstavljaju sledeći koncepti:

- objekat
- klasa
- poruka i metoda
- učaurenje
- nasledjivanje

Objekat. U OO modelu podataka objekat je svaki entitet realnog sveta, tj. objekat je sve. Primitivna vrednost je objekat (npr. ceo broj 2 ili niska 'aaa'), ali objekat je i kompleksni entitet tipa OSOBA, kao i čitav programski sistem. Svaki objekat ima pridružen jedinstveni (u sistemu) *identifikator*. Tako se nekoj odredjenoj osobi tipa OSOBA može pridružiti identifikator o_1. Postojanje identifikatora objekta vodi poreklo iz objektno orijentisanih programskih jezika, gde je podrazumevano prisustvo objekta u unutrašnjoj memoriji i gde se pretraživanje objekata po svojstvu, tako karakteristično za baze podataka, nije primenjivalo.

Jedinstveni identifikator ima i svoju specifičnu ulogu u objektno orijentisanim bazama. Naime, svaki objekat ima *stanje* i *ponašanje*. Stanje objekta opisuje se

vrednostima njegovih *instancnih promenljivih* (ili *atributa*). Uloga jedinstvenog identifikatora objekta u objektno orijentisanim bazama je da zameni objekat kada se on pojavi kao vrednost neke instancne promenljive nekog drugog objekta. Tako objekat sa identifikatorom o_1 može imati instancne promenljive sa sledećim vrednostima:

ime:	Mika Mikić
datum_rodjenja:	22.03.1955.
poslodavac:	p_1

gde je p_1 jedinstveni identifikator objekta – poslodavca osobe o_1. Objekat sa identifikatorom p_1 može imati kompleksnu strukturu sa instancnim promenljivim, na primer, naziv i adresa, pa je kao vrednost instancne promenljive poslodavac osobe sa identifikatorom o_1 mnogo jednostavnije navesti identifikator p_1 tog objekta nego čitav objekat.

Poruka, metoda, učaurenje. Ponašanje objekta opisuje se skupom *poruka* na koje je objekat sposoban da odgovori. Za objekat o_1 te poruke mogu biti, izmedju ostalih, **ime**, **godine_starosti**, **naziv_poslodavca**. Objekat na primljenu poruku odgovara izvršavanjem *metode*, tj. implementirane procedure koja je zadužena za "izvršavanje" poruke. Svaki objekat je *učauren*, što znači da je njegova interna struktura nevidljiva za korisnika tog objekta. Korisniku je omogućen pristup objektu samo preko skupa poruka definisanih za taj objekat. Na primer, korisnik objekta o_1 ne zna njegove instancne promenljive, kao ni način implementacije njegovih metoda, pa tako nema mogućnost pristupa podatku datum_rodjenja. Ova instancna promenljiva može se koristiti u implementaciji metode **godine_starosti**, da bi se, u odnosu na tekući datum, uz odgovarajuće zaokrugljivanje ili odsecanje, izračunala starost (u godinama) osobe o_1.

Za razliku od ovakvog, "čistog" objektno orijentisanog modela, veći broj objektnih sistema gradi se na objektnom modelu koji omogućuje definisanje "javnih" instancnih promenljivih, vidljivih svakom korisniku objekta, koje onda daju korisniku mogućnost kako čitanja vrednosti, tako i promene vrednosti instancne promenljive.

Klasa. Svaki objekat pripada nekoj *klasi*, kojom se definiše struktura objekata te klase (instancne promenljive) i skup operacija i funkcija koje se nad objektima te klase mogu izvršavati (poruke). Objekat se naziva *primerkom* klase kojoj pripada. Dakle, instancne promenljive i poruke definišu se na nivou klase, a primenjuju se na primerke klase. Metode kojima se implementiraju poruke – operacije i funkcije, implementiraju se na nivou klase. (Na primer, objekat o_1 može biti primerak klase OSOBA, čije su instancne promenljive ime, datum-rodjenja i poslodavac, a poruke, tj. metode – one koje se primenjuju na objekat o_1). Svaka klasa sadrži i metodu **novi** za kreiranje primeraka klase, koja traba da se primenjuje na samu klasu, a ne na primerak klase. Da bi se ujednačio način primene porukâ i izvršavanja metoda, potrebno je da se poruke uvek primenjuju na objekte, a ne na klase. Zato za svaku klasu postoji jedan specifični objekat, tzv. *objekat klase*, koji sadrži definiciju klase. Poruka, tj. metoda **novi**, primenjuje se na objekat klase.

Vrednost instancne promenljive a objekta iz klase X je objekat ili identifikator objekta iz neke klase Y, a može biti i skup ili lista objekata (ili identifikatora objekata) iz klase Y. Klasa Y je *domen* instancne promenljive a klase X, i može imati proizvoljno kompleksnu strukturu. Tako se dolazi do ugnježdene definicije klase X kao usmerenog grafa klasâ sa korenom u klasi X. Ovaj usmereni graf se zove *hijerarhija kompozicije klasâ*, ili još i *horizontalna hijerarhija klasâ*. On ne mora predstavljati striktnu hijerarhiju (može sadržati i cikluse) i odgovara odnosu agregacije izmedju klase X i klasâ – domena instancnih promenljivih (atributa) klase X.

Primer A.1 Klasa objekata KNJIGA može imati strukturu sledećeg oblika (uz instancne promenljive navedeni su njihovi domeni; domen u vitičastim zagradama označava da odgovarajuća instancna promenljiva ima skupovnu vrednost):

klasa KNJIGA

naziv:	NISKA_ZNAKOVA
autori:	{OSOBA}
izdavač:	PRAVNO_LICE
god_izdanja:	CEO_BROJ
sadržina:	{POGLAVLJE}

Klase NISKA_ZANAKOVA i CEO_BROJ su jednostavne (primitivne) klase, sa atomičnim vrednostima. Klase OSOBA, PRAVNO_LICE i POGLAVLJE, sa druge strane, imaju sopstvenu strukturu (i ponašanje), koja se mora zadati definicijom tih klasa. Na primer, struktura klasâ OSOBA i POGLAVLJE može biti sledećeg oblika:

klasa OSOBA

ime:	NISKA_ZNAKOVA
prezime:	NISKA_ZNAKOVA
adresa:	ADRESA

klasa POGLAVLJE

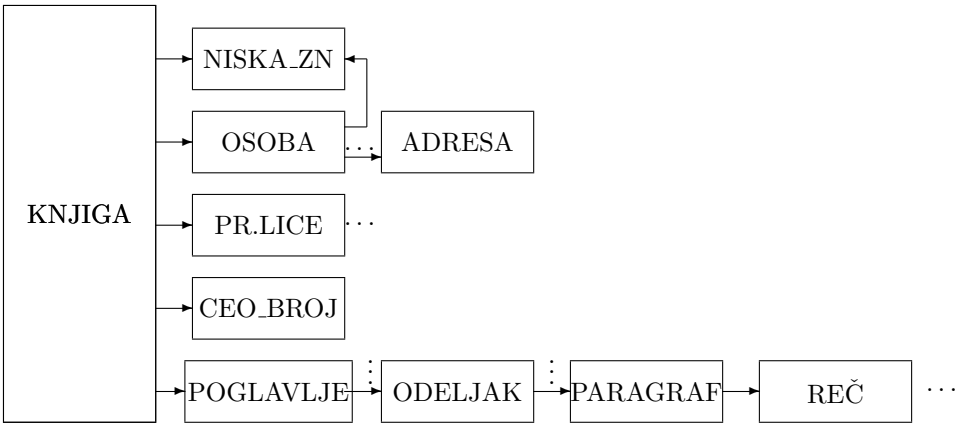
naslov:	NISKA_ZNAKOVA
sadržina:	{ODELJAK}

Slično, mora se zadati i definicija klasâ ADRESA i ODELJAK. Na primer, struktura klase ODELJAK može biti:

klasa ODELJAK

naslov:	NISKA_ZNAKOVA
sadržina:	{PARAGRAF}

Dalje, klasa PARAGRAF može imati instancnu promenljivu definisanu skupovno nad klasom REČ (paragraf se sastoji od reči), dok klasa REČ može imati niz instancnih promenljivih koje karakterišu reč kao osnovnu jedinicu strukture i značenja teksta. Tako se dobija hijerarhija kompozicije klase KNJIGA, tj. usmereni graf sa korenom u klasi KNJIGA. Jedan deo tog grafa predstavljen je na slici A.1.



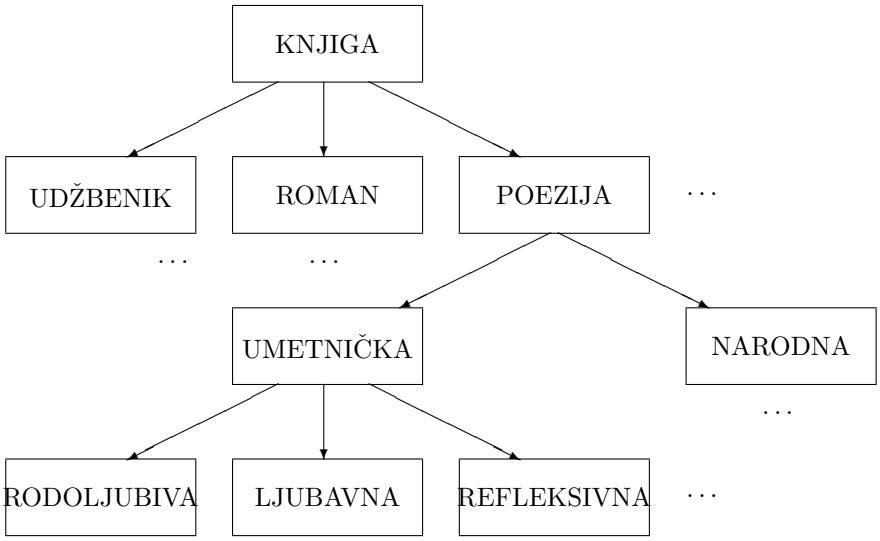
Slika A.1: Horizontalna hijerarhija klase KNJIGA

Nasledjivanje. Objektni sistemi omogućuju korisniku da izvede novu klasu iz postojeće klase. Nova klasa, koja se zove *podklasa* postojeće klase, *nasledjuje* sve instancne promenljive (*strukturno nasledjivanje*) i metode (*nasledjivanje ponašanja*) od postojeće klase, koja se naziva *nadklasa* nove klase. Svaki objekat – primerak podklase je ujedno i primerak nadklase, pa se može koristiti gdegod se koristi primerak nadklase. Ovo svojstvo odgovara odnosu generalizacije medju tipovima entiteta (klasama) i doprinosi mogućnosti ponovnog korišćenja već napisanih programa kojima se implementira model (engl. reusability). Podklasa obično sadrži i dodatne instancne promenljive i metode, koje definiše korisnik – kreator podklase. Metode nasledjene od nadklase mogu u podklasi biti *redefinisane*. Mogućnost primene iste metode (ili različitih metoda sa istim imenom) na različite klase, zove se *polimorfizam*.

Na primer, moguće je definisati klasu UDŽBENIK kao podklasu klase KNJIGA, sa dodatanim instancnim promenljivim OBLAST, PROFILI (iz koje je oblasti udžbenik i kojim je profilima namenjen) i novim metodama **profil-u** i **profil-iz** za dodavanje odnosno izuzimanje odredjenog profila iz skupa kome je udžbenik namenjen.

Jedna klasa može imati proizvoljan broj podklasa. U slučaju sistema sa *jednostrukim nasledjivanjem*, jedna klasa može imati najviše jednu nadklasu, pa takve klase čine *hijerarhiju klasâ*; u slučaju sistema sa *višestrukim nasledjivanjem*, klasa može imati veći broj nadklasa, pa takve klase grade opštiju strukturu usmerenog grafa bez ciklusa koja se naziva i *rešetkom* klasâ (engl. lattice, [42]). Skup klasâ koje su u odnosu podklasa/ nadklasa zove se i *vertikalna hijerarhija klasâ*.

Na primer, klasa KNJIGA, pored podklase UDŽBENIK, može imati i podklase ROMAN i POEZIJA, koje pak mogu imati svoje podklase. Tako bi hijerarhija klasâ ovog primera mogla grafički da se predstavi kao na slici A.2.



Slika A.2: Vertikalna hijerarhija klase KNJIGA

Struktura klasâ zajedno sa hijerarhijom kompozicije klasâ i hijerarhijom klasâ definiše *objektno orijentisanu shemu*.

Implementacija jezgra OO modela podataka u kontekstu OO baza podataka mora da uključi perzistentnu (trajnu) memoriju za objekte i shemu, kao i korisničke sumedje – jezik definisanja objekata i klasâ i jezik manipulisanja objektima i klasama. Postoje dva osnovna pristupa izgradnji sumedja za objektno orijentisane baze podataka. Prvi je izgradnja integrisanog jezika koji se koristi i za operacije nad bazom podataka i za kontrolne operacije nevezane za baze. Ovaj jezik može nastati proširenjem objektno orijentisanog programskog jezika konstrukcijama vezanim za baze podataka, ili definisanjem novog integrisanog jezika iste namene.

Drugi pristup je tradicionalan u bazama podataka: to je definisanje jezika baze podataka i njegovo ugnježdenje u matični programski jezik.

Najčešće korišćeni OO jezici koji se proširuju mehanizmima za rad sa bazama podataka su OPAL ([10]) i C++ ([61]). Prednost integrisanih jezika baza podataka je u tome što se njima prevazilazi razlika izmedju slogovnog nivoa podataka u programskom jeziku i skupovnog nivoa podataka upitnih jezika (npr. SQL-a). Nedostatak je, medjutim, što se ova razlika prevazilazi snižavanjem nivoa apstrakcije podataka, odnosno zamenom skupovnog nivoa upitnog jezika slogovnim nivoom programskog jezika. Drugi nedostatak integrisanog jezika je potreba da se iz različitih programskih jezika pristupa objektno orijentisanoj bazi, što je onda moguće samo izgradnjom većeg broja integrisanih jezika.

Drugi pristup ima osnovni nedostatak u nepodudarnosti modela podataka u bazi i modela podataka (tipova i struktura) u matičnom programskom jeziku. Zato

se, najčešće, sumedja OO baze podataka realizuje kroz dva različita jezika – jedan deklarativni, neproceduralni jezik za interaktivne (*ad hoc*) upite (npr. OSQL, [2]) i drugi aplikativni, proceduralni integrisani jezik baza podataka. Napomenimo ovde, bez detaljnijeg razmatranja problema, da interaktivni upitni jezik nad OO bazom, kao koncept, narušava princip učaurenja, jer omogućuje pristup objektima i njihovim svojstvima mimo definisanih metoda. Detaljnije o ovom problemu i uslovima pod kojima interaktivni upitni jezik ne narušava učaurenje objekata biće reči u tački A.2.2 (v. i [27]).

A.1.2 Ostali koncepti objektno orijentisanih modela podataka

Koncepti jezgra OO modela pokazuju se nedovoljnim (ili nedovoljno efikasnim) u modeliranju i implementaciji aplikacija nad objektima složene strukture. Specifične klase aplikacija zato zahtevaju nadgradnju jezgra specifičnim konceptima semantičkog modeliranja, proizvodeći različite OO modele i sisteme.

Najčešći koncepti kojima se nadgradjuje jezgro OO modela jesu kompozitni objekti, izmenljivost objektne sheme i podrška verzijama. Ako sistem za upravljanje objektno orijentisanim bazama podataka podržava ove koncepte, onda aplikacije ne moraju da se bave njihovom kontrolom.

Kompozitni objekti. Kompozitni objekti su jedno od semantičkih proširenja jezgra objektno orijentisanog modela. Implementacija koncepta kompozitnog objekta u objektnom sistemu omogućuje sistemsku kontrolu kreiranja, brisanja, izmene i pretraživanja kompozitnog objekta; ona takodje obezbedjuje sistemsku podršku uslovima integriteta, autorizaciji, konkurentnosti i drugim funkcijama koje se realizuju nad kompozitnim objektom kao celinom.

Kompozitni objekti zasnivaju se na specifičnoj semantici odnosa nekih klasa u horizontalnoj hijerarhiji klasâ. To je odnos tipa "deo-od" (engl. part-of). Tako, u primeru A.1 semantika odnosa izmedju klasa KNJIGA i OSOBA (preko atributa autori) nije ista kao semantika odnosa izmedju klasa KNJIGA i POGLAVLJE (preko atributa sadržina); u prvom slučaju atribut autori, tj. skup vrednosti iz domena OSOBA, jeste *svojstvo* objekta iz klase KNJIGA, dok u drugom slučaju vrednost atributa sadržina, tj. skup vrednosti iz domena POGLAVLJE, predstavlja *deo* objekta iz klase KNJIGA. Zato objekat iz klase KNJIGA, zajedno sa pripadnim objektima iz klasâ koje obrazuju horizontalnu hijerarhiju, a koje su u odnosu deo-od, jeste jedan kompozitni objekat. Naglasimo da u ovaj kompozitni objekat ne "ulaze" objekti – primerci klasa koje su u odnosu *svojstvo* u toj horiznotalnoj hijerarhiji, npr. objekti iz klase OSOBA.

Objasnimo pojam kompozitnog objekta nešto sistematičnije. Kao što je već naglašeno, stanje objekta je skup vrednosti iz domenâ atributâ klase kojoj objekat pripada. Domeni mogu biti jednostavne klase (brojeva, niski) i tada se vrednosti atributa predstavljaju baš tim jednostavnim objektima; domeni mogu biti i složene

klase, kada se vrednosti atributa predstavljaju identifikatorima objekata – primeraka tih složenih klasa. Ovi identifikatori zovu se *obraćanja* (reference) objektima koje identifikuju. Ako obraćanje nosi semantiku deo-od, onda se ono zove *kompozitno obraćanje* ([41]) i pripada semantičkom proširenju objektnog modela, inače je obraćanje *slabo* i sastavni je deo semantike jezgra objektnog modela. Kompozitno obraćanje može biti *ekskluzivno* ili *deljeno* (kada je jedan objekat deo-od samo jednog objekta odnosno kada jedan objekat može biti deo-od većeg broja objekata), *zavisno* ili *nezavisno* (kada je postojanje objekta na koji postoji kompozitno obraćanje uslovljeno postojanjem objekta koji vrši to obraćanje, odnosno kada nije time uslovljeno).

Kompozitni objekat je heterogena kolekcija objekata koja se sastoji od objekta koji ima atribute – kompozitna obraćanja, i od svih objekata (*komponentnih* objekata) na koje se ta obraćanja odnose. Atribut sa vrednostima – kompozitnim obraćanjima zove se *kompozitni atribut*. Komponentni objekti pripadaju različitim klasama, a te klase su organizovane u hijerarhiju koja se zove *hijerarhija kompozitnog atributa*. Svaka klasa ove hijerarhije zove se *komponentna klasa*, a sama hijerarhija je deo (podskup) hijerarhije kompozicije klasâ. Tako, u primeru A.1 atribut *sadržina* jeste kompozitni atribut, a klase POGLAVLJE, ODELJAK, PARAGRAF i REČ su komponentne klase hijerarhije kompozitnog atributa *sadržina* klase KNJIGA. Kompozitni objekat se u tom primeru sastoji od odgovarajućih primeraka klasa KNJIGA, POGLAVLJE, ODELJAK, PARAGRAF i REČ.

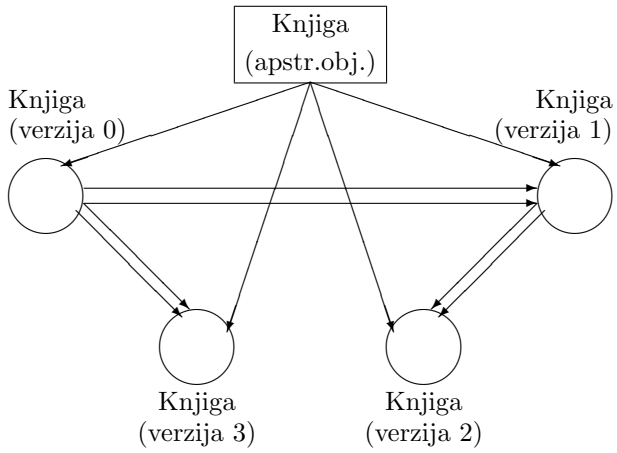
Verzije. Neki OOSUBP podržavaju koncept *verzijâ* objekta kao vremenskih podataka, koji se pokazuje potrebnim u nizu aplikacija. Takve aplikacije su razvoj softvera, projektovanje hardvera, kreiranje dokumenata. Ova podrška odnosi se na mogućnosti:

- kreiranja nove verzije objekta, koja se daje specifičnom korisniku na korišćenje i eventualno izmenu;
- proglašavanja kreirane (i izmenjene) verzije objekta u tekuću verziju;
- spajanja različitih verzija;
- brisanja zastarele verzije;
- pretraživanja istorije verzija datog objekta.

Verzije stvaraju nove vrste odnosa izmedju objekata. Jedan od tih odnosa je *izvedena-od*, izmedju nove i stare verzije objekta. Druga vrsta odnosa je *verzija-od*, izmedju svake verzije objekta i apstraktnog objekta čije su to verzije.

Verzije objekta, zajedno sa definisanim odnosima, obrazuju usmereni graf. Veći broj verzija može biti u odnosu izvedena-od sa jednom verzijom objekta, i svaka verzija objekta može biti u tom odnosu sa većim brojem verzija objekta (tj. iz jedne verzije objekta može se izvesti veći broj novih verzija, a i nova verzija jednog objekta može biti izvedena iz većeg broja starih verzija). Ipak, većina modela koji uključuju

semantički koncept verzija ograničava ovaj graf na hijerarhijsku strukturu koja se zove *hijerarhija verzija*. Slika A.3 predstavlja jednu hijerarhiju verzija objekta iz klase KNJIGA. Jednostruke grane označavaju odnos *verzija-od*, a dvostruke – odnos *izvedena-od*.



Slika A.3: Graf verzija

Modifikacija sheme. Osnovne prednosti objektnog pristupa projektovanju i razvoju programskih sistema su posledica koncepta nasledjivanja, i ogledaju se u mogućnosti jednostavnog proširenja sistema, kao i u mogućnosti ponovnog korišćenja fragmenata sistema (npr. jednom napisana metoda može se primeniti na sve podklase, a može se upotrebiti i u drugom programskom sistemu). U kontekstu sistema za upravljanje bazama podataka, prisustvo većeg broja korisnika uslovljava potrebu za različitim pogledima na shemu baze podataka, što podrazumeva i različite poglede na relaciju i mehanizam nasledjivanja. Različiti pogledi podrazumevaju, zato, modifikaciju (fizičku ili virtuelnu) sheme. Fizička modifikacija sheme predstavlja aspekt logičkog projektovanja objektne baze podataka.

Model modifikacije sheme podrazumeva definisanje smislenih promena koje su moguće nad shemom, kao i definisanje semantike tih promena. Sam objektni model podataka, s obzirom na medjuzavisnost klasa i njihovu tesnu povezanost odnosima generalizacije i agregacije, nameće znatno širi spektar mogućih i često potrebnih modifikacija sheme od, na primer, relacionog modela.

S obzirom na dve vrste odnosa modeliranih objektnim modelom podataka, i modifikacije sheme objektno orijentisanih baza podataka su u osnovi dvojake. To su modifikacija definicije klase (modifikacija hijerarhije kompozicije klasa) i modifikacija odnosa nasledjivanja (modifikacija hijerarhije klasa). Modifikacija definicije klase uključuje modifikaciju atributa, domena i metoda, dok se modifikacija

hijerarhije klasa odnosi na promenu odnosa nadklasa/podklasa para klasâ, ili na dodavanje ili uklanjanje klase.

Jedan od malog broja OOSUBP čiji model sadrži modifikaciju sheme je ORION ([41]), pa će ovde biti nabrojane mogućnosti modifikacije sheme koje on nudi. Ove mogućnosti su direktno zavisne od specifičnosti celokupnog modela, i u slučaju sistema ORION odražavaju višestrukost nasledjivanja sadržanu u modelu ovog sistema.

- Modifikacije pojedinačne klase:
    1. Modifikacije atributa
        - dodati novi atribut klasi;
        - ukloniti postojeći atribut iz klase;
        - promeniti ime atributa klase;
        - promeniti domen atributa klase;
        - promeniti podrazumevanu vrednost atributa;
    2. Modifikacije metoda
        - dodati novu metodu klasi;
        - ukloniti postojeću metodu iz klase;
        - promeniti ime metode klase;
        - promeniti metodu (blok implementirane procedure) klase;
- Modifikacije odnosa nadklasa/podklasa
    1. za dve postojeće klase proglasiti jednu za nadklasu druge;
    2. ukloniti jednu klasu iz liste nadklasâ druge klase;
    3. promeniti poredak nadklasâ date klase;
    4. dodati novu klasu;
    5. ukloniti postojeću klasu;
    6. dodati novu klasu kao nadklasu n postojećih klasa;
    7. razdeliti jednu klasu u n novih klasa;
    8. udružiti n klasa u jednu novu klasu.

Realizacija navedenih modifikacija sheme vodjena je pravilima koja se odnose na atribute, domene, metode, nasledjivanje atributa i metoda od jedne ili većeg broja nadklasa, na razrešavanje konflikata zbog istoimenih atributa ili metoda i, pre svega, na očuvanje hijerarhije klasâ kao usmerenog grafa sa korenom u sistemskoj klasi KLASA.

Modifikacija sheme koja uključuje kompozitne objekte uzrokuje specifične probleme koji su izvan okvira ovog razmatranja (v. [41]).

A.2 Objektno orijentisani sistemi baza podataka

Objektno orijentisani sistem baza podataka uključuju objektne baze podataka i odgovarajući sistem za upravljanje tim bazama, OOSUBP. OOSUBP, pored podrške konceptima objektnog modela, mora da podrži i sve funkcije koje su karakteristične za sistem za upravljanje bazom podataka uopšte. To, preciznije, znači da OOSUBP mora da poseduje odgovarajuće korisničke sumedje, tj. jezike za rad sa bazom podataka, programe za obradu i optimizaciju upita, kontrolu integriteta, autorizaciju, kao i oporavak od pada. OOSUBP mora da podrži i fizičku organizaciju i upravljanje objektima u fizičkoj memoriji, kao i upravljanje transakcijama. Mada su same funkcije iste kao i kod relacionih sistema, njihova realizacija u objektnom okruženju postavlja specifične probleme i zahteva specifična rešenja.

Osnovna razlika u realizaciji funkcijâ objektno orijentisanog sistema baza podataka, u odnosu na druge sisteme, nastaje kao posledica dvostruke hijerarhijske organizacije podataka (tj. klasâ). Tako je moguće pristup objektima iz neke klase (u cilju pretraživanja, unošenja, brisanja, ažuriranja) ograničiti samo na tu klasu, a moguće je pod pristupom objektima iz neke klase podrazumevati i pristup objektima iz svih podklasa te klase (s obzirom da je svaki primerak podklase neke klase istovremeno i primerak te klase). Kod fizičkog smeštanja objekata, moguće je grupisati objekte prema odnosu podklasa/nadklasa, a moguće je grupisati ih prema odnosu agregacije, tj. prema hijerarhiji kompozicije klasâ.

Kao objekat zaključavanja i autorizacije, u objektnom modelu može se pojaviti pojedinačni primerak klase, kompozitni objekat, cela klasa, cela hijerarhija kompozicije klasâ, ili deo hijerarhije klasa sa korenom u datoj klasi. S obzirom na moguću složenost objekata, i koncept transakcije se znatno menja. Transakcije mogu vremenski biti znatno duže, mogu sadržavati ugnježdene transakcije (npr. nad komponentnim objektima), pa se arhitektura same transakcije menja, a konkurentno izvršenje skupa transakcija zahteva drugačije protokole u odnosu na one koji se sreću kod relacionih sistema.

Poseban značaj u funkcionisanju SUBP (bilo koje vrste) imaju upitni jezici i upravljanje transakcijama. Kako ovi aspekti sistema imaju izražene specifičnosti u arhitekturi sistema za upravljanje objektno orijentisanim bazama podataka, u preostalom tekstu ovog odeljka biće razmotrene upravo ove dve komponente OOSUBP.

A.2.1 Dugotrajne transakcije

Fundamentalna pretpostavka u osnovi tradicionalnog modela transakcija u relacionim sistemima jeste kratko trajanje transakcija. Za objektno orijentisani sistem baza podataka, koji operiše kompleksnim objektima, transakcija može da traje veoma dugo, satima ili danima. Transakcija je u objektnom sistemu, kao i u drugim sistemima, jedinica kontrole konkurentnosti i oporavka. Zato za transakcije koje traju dugo implementacija tradicionalnog modela transakcija može, s jedne strane, da izazove neprihvatljivo dugo čekanje na dobijanje katanca i pristup objektu. Sa druge strane, ako se dugotrajna transakcija poništava zbog zahteva korisnika, pada

transakcije ili sistema, količina izgubljenog posla može da bude takodje neprihvatljivo velika.

Cilj modela transakcijâ je da, za definisani kriterijum konzistentnosti baze podataka, pod pretpostavkom konkurentnog izvršenja skupa transakcija, kao i pod pretpostavkom mogućeg pada sistema, obezbedi osnovu za automatsko nametanje tog kriterijuma. Kriterijum konzistentnosti u tradicionalnim transakcijama je linearizovanost. Ovaj kriterijum se nameće odgovarajućim postupkom dobijanja katanaca u slučaju konkurentnosti, odnosno korišćenjem dnevnika ažuriranja i oporavka u slučaju pada sistema.

Dugotrajne transakcije ne moraju prihvatiti linearizovanost kao kriterijum konzistentnosti. Problem sa dugotrajnim transakcijama je što odgovarajućeg, opšteg kriterijuma konzistentnosti još nema. Bez obzira na to, postoje pokušaji u pojedinim sistemima da se reši problem dugog čekanja na katance i skupog poništenja transakcije.

Dugotrajne transakcije imaju jedan od sledeća dva oblika. *Ravne* dugotrajne transakcije su one koje u svom sastavu nemaju druge transakcije (podtransakcije) i mogu da se posmatraju kao nizovi kratkotrajnih transakcija. *Ugnježdene* transakcije predstavljaju hijerarhijsku organizaciju transakcija.

Za kontrolu konkurentnosti ravnih dugotrajnih transakcija može se koristiti neka od metoda koje se ne zasnivaju na katancima i zaključavanju, npr. kontrola konkurentnosti pomoću više verzija ([27]), ako sistem uopšte podržava verzije. Objekat može da se "iskopira" u radni prostor transakcije koja zahteva čitanje ili upis u taj objekat. Ako transakcija samo čita svoju kopiju, onda ona ne proizvodi efekte na istoriju verzija tog objekta. Ako transakcija menja svoju kopiju, onda se ta kopija, po završetku transakcije, prijavljuje kao nova verzija objekta.

Zaključavanje objekta u slučaju verzijâ zahteva se samo za vreme kopiranja objekta u radni prostor transakcije, pa je potreba za dugotrajnim čekanjem na katance eliminisana. Mehanizam verzija takodje omogućuje rad (najčešće eksperimentalne prirode) sa raznim varijantama jednog objekta, i obezbedjuje informacije o istoriji verzija (istorijske podatke o objektu). Medjutim, održavanje istorije verzija značajno uvećava prostor i količinu obrade podataka. Zbog toga se za kontrolu konkurentnosti transakcija nad nekim objektima, pored mehanizma verzija, moraju obezbediti i alternativni mehanizmi. Jedan od njih je mehanizam *mekog katanca*. Naime, za kratkotrajne transakcije sistem postavlja kratkotrajne, *tvrde* katance, koji se ne mogu osloboditi pre kompletiranja ili poništenja transakcije. Za dugotrajne transakcije korisnik (a ne sistem) eksplicitno postavlja *meke* katance, koje može eksplicitno da oslobodi i u toku izvršavanja transakcije, ili koji se (sistemski) oslobadjaju na kraju transakcije. Meki katanac se eksplicitno oslobadja obično na zahtev druge transakcije koja je u konfliktu sa tim katancem.

Za oporavak od pada sistema, ravne dugotrajne transakcije obično primenjuju koncept *tačaka pamćenja*. Tačke pamćenja omogućuju parcijalno poništavanje transakcija. Deo dugotrajne transakcije izmedju dve tačke pamćenja može se tretirati kao kratkotrajna transakcija, koja je u tom slučaju jedinica oporavka. Ponekad

je potrebno poništiti ili ponovo izvršiti više od jedne kratkotrajne transakcije (striktno unazad), što predstavlja problem s obzirom da su katanci prethodnih kratkotrajnih transakcija (koje su izvršile COMMIT operaciju) oslobodjeni. Tada se može primeniti tehnika vremenskih oznaka koje se dodeljuju COMMIT slogovima log datoteke i objektima koji se ažuriraju. Radnja poništenja (ili ponovnog izvršenja) ažuriranja objekta dopušta se samo ako je vremenska oznaka COMMIT sloga tekuće kratkotrajne transakcije veća od vremenske oznake objekta, i ako objekat već nije zaključan od strane druge kratkotrajne transakcije.

Ugnježdene transakcije su uopštenje tačaka pamćenja i omogućuju, umesto serijske, hijerarhijsku organizaciju transakcije, a u slučaju potrebe, i hijerarhijsko poništavanje. Zato je iskaz početka transakcije, npr. BEGIN TRANSACTION, proširen tako da označava i početak podtransakcije; COMMIT i ROLLBACK operacije su takodje proširene tako da se odnose i na podtransakciju, ali samo u opsegu spoljašnje transakcije a ne i čitavog sistema.

A.2.2 Upitni jezici

Većina OOSUBP koristi dva različita jezika za definisanje i manipulisanje podacima: jedan za aplikativno programiranje i drugi za interaktivne upite. Za izgradnju aplikativnog jezika ovi sistemi najčešće ne primenjuju pristup ugnježdenja upitnog jezika u matični jezik, koji je najzastupljeniji kod relacionih sistema. Umesto toga, objektni sistemi primenjuju koncept *integrisanog jezika* ([27]) koji se koristi i za operacije nad objektnom bazom i za upravljačke operacije i operacije obrade tipične za programske jezike. Time se prevazilaze problemi nepodudarnosti tipova i nivoa semantike (deklaratvna/proceduralna) matičnog i upitnog jezika, ali, na žalost, najčešće po cenu spuštanja nivoa operacija nad bazom na proceduralni nivo operacija programskog jezika. Najzastupljeniji medju integrisanim jezicima objektnih sistema su OPAL (dijalekt Smalltalk-a, jezik OOSUBP GemStone, [10]) i C++ ([61]). Posledica korišćenja integrisanog jezika je i nepoštovanje principa dualnosti za interaktivni i aplikativni jezik.

Interaktivni upitni jezik za realizaciju *ad hoc* upita obično je neka verzija objektno proširenog SQL jezika; proširenje može biti i takvo da dobijeni jezik malo liči na sâm SQL. Prikažimo u osnovnim crtama upitni jezik OSQL sistema Iris ([69], [34]). Iskazi jezika OSQL mogu se klasifikovati na sledeći način:

- iskazi otvaranja baze podataka i povezivanja sa njom, START i CONNECT;
- iskazi definisanja

    1. Definicija klase (u terminologiji ovog sistema to je kreiranje tipa) uključuje ime klase, atribute i njihove domene. Atributi (u ovoj terminologiji – "funkcije svojstava") mogu imati opciju REQUIRED, što odgovara opciji NOT NULL kod relacionih sistema. U definiciji klase može se navesti i njena nadklasa (onda klasa nasledjuje sve atribute nadklase). Atribut klase može uzeti, kao svoju vrednost, i skup vrednosti

iz odgovarajućeg domena, što se označava opcijom MANY. Na primer, sledećim iskazima se kreiraju klase Osoba, Autor, Recenzent, Rad:

```
CREATE TYPE Osoba
      (ime Charstring REQUIRED,
       adresa Charstring,
       telefon Charstring);

CREATE TYPE Autor SUBTYPE Osoba
      (br_radova Integer);

CREATE TYPE Recenzent SUBTYPE Osoba
      (domen_istrazivanja Oblast MANY);
CREATE TYPE Rad
      (naslov Charstring REQUIRED,
       autori Autor REQUIRED MANY,
       oblast Oblast,
       recenzenti Recenzent MANY);
```

2. Definicija primerka klase uključuje ime promenljive (čija je vrednost taj primerak klase, tj. objekat) i vrednosti nekih (bar REQUIRED) atributa pripadne klase. Na primer, sledećim iskazima se kreiraju primerci klasa Recenzent, Autor i Rad:

```
CREATE Recenzent (ime, domen_istrazivanja)
INSTANCES  r_1 ('Mara Maric', (racunarstvo, ekonomija));

CREATE Autor (ime)
INSTANCES  a_1 ('Pera Peric');

CREATE Rad (naslov, autori)
INSTANCES rd_1 ('O objektnom modelu',  a_1);
```

3. Definicija funkcije uključuje ime funkcije i preslikavanje skupa klasa u klasu kojoj pripada objekat – vrednost funkcije. Na primer, sledeći iskaz definiše funkciju *ocena* koja vraća ocenu kojom je jedan recenzent ocenio jedan rad (ovo je primer tzv. *zapamćene* funkcije):

```
CREATE FUNCTION  ocena (Rad, Recenzent) →  Integer;
```

- iskazi dodele

1. Dodela klase objektu označava mogućnost pripadnosti jednog objekta većem broju klasa. Na primer, objektu a_1 (primerak klase Autor) može se dodeliti i klasa Recenzent, tj. autor a_1 biće istovremeno i recenzent:

```
ADD TYPE Recenzent to  a_1;
```

2. Dodela vrednosti funkciji (funkciji svojstva, tj. atributu, ili zapamćenoj funkciji) izvršava se iskazom SET. Tako, atributu recenzenti objekta rd_1 može se dodeliti vrednost r_1, a funkciji *ocena* sa argumentima (rd_1, r_1) može se dodeliti vrednost 5 pomoću sledećih iskaza:

```
SET recenzenti (rd_1)  = r_1;

SET ocena (rd_1, r_1) = 5;
```

- iskazi pretraživanja

  Ovi iskazi su oblika SELECT – FOR EACH – WHERE, pri čemu linija FOR_EACH odgovara FROM liniji SQL-a ali, za razliku od nje, nije obavezna. Na primer, upiti "naći naslov rada rd_1" i "naći naslove svih radova autora a_1" mogu se izraziti sledećim iskazima:

```
SELECT   naslov(rd_1);

SELECT   naslov
FOR EACH Rad  rd
WHERE    a_1 = autor(rd);
```

Interaktivni upitni jezici, mada široko zastupljeni u objektnim sistemima za upravljanje bazama podataka, narušavaju princip učaurenja podataka i operacija, osim ako su objekti projektovani prema jednom specifičnom, strogom kriterijumu.

Naime, princip učaurenja podrazumeva da se objektima može pristupiti samo preko definisanih poruka, dok interaktivni upitni jezik omogućuje pristup svim atributima (instancnim promenljivim) objekta. Dovoljan uslov za učaurenost objekta i pri upotrebi *ad hoc* upita je da skup metoda, tj. poruka koje definišu ponašanje objekata iz jedne klase, sadrži jednu kompletnu moguću reprezentaciju objekta iz te klase. Ilustrujmo značenje zahteva za kompletnošću, kao i opcije "moguće" reprezentacije.

Primer A.2 Neka su za klasu TAČKA definisane metode za dobijanje njenih Dekartovih koordinata x i y, a za klasu KRUG – metoda za dobijanja tačke – centra kruga, i metoda za dobijanje broja – dužine poluprečnika kruga. Ovaj skup metoda obrazuje jednu kompletnu reprezentaciju objekata iz klasa TAČKA i KRUG, jer su njima "pokrivena" sva svojstva tih objekata. Ukoliko su u upitu dopušteni i skalarni izrazi nad definisanim metodama, proizvoljne složenosti, onda upitni jezik zaista omogućuje, bez narušavanja učaurenosti objekta, postavljanje proizvoljnih, neplaniranih upita nad klasama TAČKA i KRUG (na primer, naći sve krugove sa centrom u tački (0,0); ili, naći koordinate centara svih krugova sa poluprečnikom 1).

Reprezentacija kruga centrom i poluprečnikom (tj. metodama koje vraćaju te podatke) je jedna moguća reprezentacija kruga. Druga reprezentacija objekta iz klase KRUG može da se sastoji od metoda za dobijanje koordinata krajnjih tačaka jednog prečnika; da bi reprezentacija bila precizno odredjena, može se posmatrati prečnik paralelan x-osi. Tako bi skup metoda nad klasom KRUG sada sadržao metode za dobijanje x_1-koordinate, x_2-koordinate i (zajedničke) y-koordinate krajnjih tačaka prečnika paralelnog x-osi. I ova reprezentacija je kompletna, a upitni jezik, kao i u prethodnom slučaju, omogućuje postavljanje svih upita.

Za razliku od prethodnih, logičkih reprezentacija, fizička reprezentacija objekta iz klase KRUG, odnosno stvarne instancne promenljive te klase mogu biti, na primer, polarne koordinate krajnjih tačaka prečnika paralelnog x-osi. Korisnik ne može da pristupi instancnim promenljivim i one nisu od interesa za upitni jezik (koji raspolaže samo odredjenim skupom metoda). One su od interesa i koriste se u implementaciji metoda, jer metode, prema postavkama objektnog modela, imaju pristup instancnim promenljivim sopstvene klase.

Skup metoda jedne kompletne moguće reprezentacije objekata iz date klase dovoljan je za postavljanje upita nad tom klasom. Skup svih metoda te klase obično je znatno širi; on sadrži (bar) još i metode za dodeljivanje vrednosti objektima, kao i metode za kreiranje novog objekta iz klase, odnosno za uništenje postojećeg objekta iz klase.

A.3 Sistemi baza podataka nove generacije

Razvoj sistema baza podataka u postrelacionom periodu kretao se u dva pravca: nadgradnja objektno orijentisanih programskih jezika upravljačem trajnih (perzistentnih) objekata (perzistentni OO sistemi), i izgradnja objektno orijentisanih baza podataka sa punom podrškom ANSI SQL jeziku.

OOSUBP prve vrste obično obezbedjuju razne funkcije standardnog SUBP – jednostavne upitne jezike, pristupne metode kao što je direktni pristup ili grupisanje, upravljanje transakcijama i kontrolu konkurentnosti i oporavka. Medjutim, oni su nekompatibilni sa RSUBP i ne poseduju čitav niz svojstava utemeljenih i razvijenih u kontekstu RSUBP, kao što su mehanizam pogleda, autorizacija, potpuno neproceduralni upitni jezici, upravljanje katalozima, itd. Zbog toga se drugi pravac smatra pravcem razvoja sistema baza podataka nove generacije ([43]).

Integrisani relaciono/objektni sistemi su proširenja relacionih sistema konceptima jezgra OO modela, kao i nekim specifičnim OO konceptima. Ovakvim integracijama pristupaju i proizvodjači perzistentnih OO sistema, i proizvodjači relacionih proizvoda. Prvi planiraju proširenja svojih sistema upitnim jezikom kompatibilnim sa ANSI SQL, dok drugi proširuju ANSI SQL2 konceptima objektnog modela (takvo proširenje je i najavljeni ANSI SQL3 standard).

I pored ovakve načelne saglasnosti u pogledu razvoja nove generacije sistema baza podataka, postoje i značajne razlike medju zainteresovanim istraživačkim i

proizvodjačkim stranama. Tako se i dogodilo da su se u periodu od 1990 – 1995. godine u relevantnoj literaturi pojavila tri manifesta sistema baza podataka nove generacije, koje su sastavili vrhunski istraživači i eksperti u oblasti OO i relacione tehnologije. Prvi od tih manifesta ([5]) zastupa koncepte objektnog modela sa upravljačem perzistentnih objekata, smatrajući ga sledbenikom relacionog modela u evoluciji modela podataka, u istom smislu odbacivanja i negiranja koncepata u kome je relacioni model sledbenik mrežnog i hijerarhijskog.

Drugi manifest ([63]) je delo Komiteta za unapredjenje funkcija SUBP, sastavljenog od najeminentnijih istraživača u oblasti relacione tehnologije, sa američkih univerziteta i iz industrije. On polazi od činjenice da RSUBP ne samo da ne podržava aplikacije sa kompleksnim podacima, već da ne podržava adekvatno ni poslovne aplikacije kojima je prvenstveno namenjen; adekvatna podrška poslovnim aplikacijama podrazumeva i podršku kompleksnim dokumentima, kao i multimedijalnim podacima – slici, rukopisu, zvuku, fotografiji. Osnovne postavke ovog manifesta su sledeće.

- Pored tradicionalnog upravljanja podacima, nova generacija SUBP treba da podrži bogatiju strukturu objekata i pravila. Poželjna svojstva su: sistem apstraktnih tipova podataka, raznovrsni konstruktori tipova podataka (niz, slog, skup, unija), rekurzivna kompozicija konstruktora, nasledjivanje, učaurenje, deklarativna ograničenja i proceduralni trigeri.

- Nova generacija SUBP mora da nasledi sva svojstva relacione generacije SUBP, pre svega neproceduralni pristup podacima, nezavisnost podataka i mehanizam pogleda.

- Nova generacija SUBP mora biti otvorena prema drugim podsistemima, tj. treba da omogući lak pristup iz drugih sistema kao što su sistemi raširenih tabela, programski jezici, alati za podršku odlučivanju, grafički paketi; ona takodje treba da omogući rad u klijent/server arhitekturi, kao i u distribuiranom okruženju. Kao sumedju sistema nove generacije treba razvijati perzistentne jezike (koji omogućuju rad sa bazom) za raznovrsne programske jezike. Za interaktivne upite treba koristiti SQL, tj. njegove nadgradnje, jer je uprkos svojim brojnim nedostacima taj jezik osvojio tržište.

Treći u ovom nizu manifesta ([20]) polazi od relacionog modela u njegovom primarnom obliku, a ne od relacionih sistema ili relacionih jezika, i pogotovu ne od SQL-a, s obzirom na njegova značajna odstupanja od samog relacionog modela. Relacioni model se ne proširuje, već se predlažu sistemi koji, osim dobrih svojstava relacionog modela, poseduju i dobra ortogonalna svojstva (druga dobra svojstva koja relacioni model ne poznaje); ta ortogonalna svojstva su uglavnom dobra svojstva objektno orijentisanog modela. Takodje, navode se svojstva postojećih modela koja se ocenjuju kao loša, i koja ovi sistemi treba da isključe.

Za razliku od niza implementacija integrisanih relaciono/objektnih sistema, koji integraciju baziraju na izjednačavanju relacionog koncepta relacije i objektnog kon-

cepta klase ([60], [69]), pristup koji se predlaže ovim manifestom polazi od izjednačavanja koncepta klase sa relacionim konceptom domena. Time se prevazilazi niz teškoća kao što su izražavanje *ad hoc* upita nad objektnom bazom, realizacija učaurenja strukture i ponašanja objekta, karakterizacija tipa, preneto ažuriranje objekata, održavanje pravila integriteta, semantika operacijâ relacione algebre, itd. ([27]).

Medju dobrim svojstvima relacionog modela, koja treba podržati u sistemima baza podataka nove generacije, ističu se sledeća:

- domen kao imenovani skup vrednosti proizvoljne složenosti; vrednosti iz domena može se pristupiti samo preko operacija definisanih nad domenom (vrednost domena, a ne n-torka relacije, odgovara objektu u objektno orijentisanom modelu, pa je vrednost domena učaurena);
- skup n-arnih operacija za svaku uredjenu listu od n domena (ne obavezno različitih), sa naznakom domena rezultata operacije;
- fizička reprezentacija različitih vrednosti jednog domena može biti različita (npr. dva prirodnojezička dokumenta mogu biti priredjena različitim procesorima teksta);
- za odredjenu strukturu n-torke (atribute i domene), moguće je definisati dva nova domena – jedan čije su vrednosti n-torke sa zadatom strukturom, i drugi čije su vrednosti relacije sa zadatom strukturom; dakle, "n-torka" i "relacija" su konstruktori domena;
- uslov integriteta može biti opšteg oblika formule relacionog računa, i može se odnositi na domen, atribut, relaciju ili bazu podataka u celini.

Loša svojstva relacionih sistema, koja se ne smeju podržati sistemom nove generacije (prema ovom manifestu), odnose se, pre svega, na svojstva SQL jezika kojima on odstupa od relacionog modela (npr. pozivanje na redosled atributa relacije, neimenovani atributi, dupliranje n-torki, prisustvo fizičke reprezentacije i pristupnih puteva na logičkom nivou, slogovno orijentisan pristup, brisanje i ažuriranje korišćenjem kursora, itd).

Medju novim svojstvima koja sistemi nove generacije treba da poseduju ističu se provera tipova u vreme kompilacije, precizni model nasledjivanja (jednostrukog ili višestrukog), ugnježdene transakcije. Kao loša svojstva objektnih sistema prepoznaju se izjednačavanje relacije i klase objekata, postojanje identifikatora objekata, koncept javnih i zaštićenih instancnih promenljivih, itd. Zato sistemi baza podataka nove generacije ne treba da podrže ova svojstva.

A.4 Pitanja i zadaci

1. Definisati sledeće pojmove:

objektno orijentisani model podataka	hijerarhija kompozicije klasâ
objekat	polimorfizam
klasa	kompozitni objekti
objekat klase	hijerarhija kompozitnog atributa
poruka i metoda	verzije
učaurenje	modifikacija sheme
nasledjivanje	OOSUBP
jednostruko nasledjivanje	dugotrajne transakcije
višestruko nasledjivanje	ugnježdene transakcije
objektna shema	meki katanac
hijerarhija klasâ	sistem baza podataka nove generacije

2. Koji su pristupi realizaciji sumedje kod objektno orijentisanih baza podataka?
3. Opisati uobičajene operacije nad verzijama.
4. Koje su dopuštene modifikacije sheme?
5. Kako se klasifikuju iskazi OSQL-a? Navesti primere.
6. Opisati probleme koji se javljaju pri realizaciji upitnog jezika nad objektno orijentisanim bazama podataka (OOBP).
7. Šta donosi predlog standarda SQL3 u odnosu na objektni aspekt SUBP?
8. Projektovati OOBP o izdavaštvu. Koliko ima različitih mogućnosti za to projektovanje? Koja je najbolja? Objasniti.
9. Koje biste metode definisali za OOBP iz prethodnog zadatka?
10. Mehanizam zaključavanja pokazuje se neefikasnim u kontroli konkurentnosti OOBP. Kako biste organizovali kontrolu konkurentnosti korišćenjem višestrukih verzija?
11. Koji elementi modifikacije sheme imaju svoje analogone u relacionom sistemu? Koji su to analogoni?

Bibliografija

[1] Alagić,S. *Relacione baze podataka*, Svjetlost, Sarajevo, 1984.

[2] Annevelink,J. i ost. "Object SQL – A language for the design and implementation of object databases", u [43].

[3] Armstrong,W.W. "Dependency structures of data base relationships", *Information Processing*, 74, North-Holland Publ. Co., Amsterdam, 1974.

[4] Astrahan,M. i ost. "System R: Relational approach to database management", *ACM ToDS*, 1(2), 1976.

[5] Atkinson,M.i ost. "The object-oriented database system manifesto", *Deductive and Object-Oriented Databases*, Elsevier Science, 1990.

[6] ANSI/X3/SPARC Study Group on Database Management Systems. "Interim report", *ACM SIGMOD bulletin*, 7(2), 1975.

[7] Bernstein,P.A. "Synthesizing third normal form relations from functional dependencies", *ACM ToDS*, 1(4), 1976.

[8] Bernstein,P.A. i ost. "Concurrency control in a System for Distributed Databases (SDD-1)", *ACM ToDS*, 5(1), 1980.

[9] Boyce,R. i ost. "Specifying queries as relational expressions: SQUARE", *IBM San Jose Research Reports*, RJ-1291, 1973.

[10] Butterworth,P. i ost. "The Gemstone object database management system", *Comm.ACM*, 34(10), 1991.

[11] Chamberlin,D., Boyce,R. "SEQUEL: A structured English query language", *ACM-SIGFIDET Workshop on Data Description, Access and Control*, Ann Arbor, MI, 1974.

[12] Chamberlin,D. i ost. "SEQUEL2: A unified approach to data definition, manipulation, and control", *IBM Journal of Research and Development*, 20(6), 1976.

[13] Chen,P.P.-S. "The Entity Relationship Model – toward a unified view of data", *ACM ToDS*, 1(1), 1976.

[14] Codd,E. "A relational model of data for large shared data banks", *Comm.ACM*, 13(6), 1970.

[15] Codd,E. "A data base sublanguage founded on the relational calculus", *ACM-SIGFIDET Workshop on Data Description, Access and Control*, San Diego, CA, 1971.

[16] Codd,E. "Further normalization of the data base relational model", *Data Base Systems* (Rustin,R. ured.), Prentice-Hall, 1972.

[17] Codd,E. "Relational completeness of data base sublanguages", *Data Base Systems* (Rustin,R. ured.), Prentice-Hall, 1972.

[18] Codd,E. "Extending the database relational model to capture more meaning", *ACM ToDS*, 4(4), 1979.

[19] Codd,E. *The Relational Model for Database Management – Version 2*, Addison Wesley Publ. Inc., 1990.

[20] Darwen,H., Date,C.J. "The Third Manifesto", *SIGMOD RECORD*, 24(1), 1995.

[21] Date,C.J. *An Introduction to Database Systems, vol. II*, Addison Wesley Publ. Inc., 1983.

[22] Date,C.J. *An Introduction to Database Systems vol. I*, 4. izdanje, Addison Wesley Publ. Inc., 1986.

[23] Date,C.J. *A Guide to INGRES*, Addison Wesley Publ. Inc., 1987.

[24] Date,C.J., White,C.J. *A Guide to DB2*, Addison Wesley Publ. Inc., 1989.

[25] Date,C.J. (ured.) *Relational Database Writings 1985–1989*, Addison Wesley Publ. Inc., 1990.

[26] Date,C.J., Darwen,H. (ured.) *Relational Database Writings 1989–1991*, Addison Wesley Publ. Inc., 1992.

[27] Date,C.J. *An Introduction ta Database Systems*, 6. izdanje, Addison Wesley Publ. Inc., 1995.

[28] Deux,O. i ost. "The O_2 system", kao [45].

[29] Djordjević,S. (ured.) *Oksfordski rečnik računarstva*, prevod sa engleskog, Nolit, Beograd, 1990.

[30] Eswaran,K.P. i ost. "The notions of consistency and predicate locks in a database system", *Comm.ACM*, 19(11), 1976.

[31] Fagin,R. "Multivalued dependencies and a new normal form for relational databases", *ACM ToDS*, 2(3), 1977.

[32] Fagin,R. "Normal forms and relational database operators", *ACM SIGMOD Conf.*, 1979.

[33] Fagin,R. "A normal form for relational databases that is based on domains and keys", *ACM ToDS*, 6(3), 1981.

[34] Fishman,D. i ost. "Iris: An object-oriented database management system", *ACM ToIS*, 5(1), 1987.

[35] Gray,J. i ost. "The recovery manager of the System R data manager", *ACM Computing Surveys*, 13(2), 1981.

[36] Hammer,M., McLeod,D. "Database description with SDM: A semantic database model", *ACM ToDS*, 6(3), 1981.

[37] IBM. *IBM DATABASE 2 SQL Reference*, IBM, 1992.

[38] IBM. *IBM DB2 Application programming and SQL Guide*, IBM, 1992.

[39] International Organization for Standardization (ISO). *Database Language SQL – Document ISO/IEC 9075*, 1992.

[40] Kim,W. "On optimizing an SQL-like nested query", *ACM ToDS*, 7(3), 1982.

[41] Kim,W. *Introduction to Object-Oriented Databases*, MIT Press, 1990.

[42] Kim,W. "Object-oriented databases: definition and research directions", *IEEE Trans. on Knowledge and Data Engineering*, 2(3), 1990.

[43] Kim,W. (ured.) *Modern Database Systems*, ACM Press, 1995.

[44] Knuth,D. *The Art of Computer Programming, vol. III, Sorting and Searching*, Addison Wesley Publ. Inc., 1968.

[45] Lamb,C. i ost. "The Objectstore database system", *Comm.ACM*, 34(10), 1991.

[46] Lipski,Jr.W. "On semantic issues connected with incomplete information databases", *ACM ToDS*, 4(3), 1979.

[47] Lohman,G.M. i ost. "Extensions to Starburst: objects, types, functions, and rules", *Comm.ACM*, 34(10), 1991.

[48] Maier,D. *The Theory of Relational Databases*, Computer Science Press, 1983.

[49] Obermarck,R. "Global deadlock detection algorithm", *IBM Research Reports*, RJ2845, 1980.

[50] IBM. *Query Management Facility: Planning and Administration Guide for MVS*, IBM, SC26–4235, 1992.

[51] Relational Technology Inc. *EQUEL C User's Guide, Version 2.0 VAX/UNIX*, Berkeley, CA, 1983.

[52] Rolland,F.D. *Relational Database Management with ORACLE*, Addison Wesley Publ. Inc., 1989.

[53] Rothnie,J.B. i ost. "Introduction to a System for Distributed Databases (SDD-1)", *ACM ToDS*, 5(1), 1980.

[54] Schmid,H.A., Swenson,J.R. "On the semantics of the relational data model", *ACM SIGMOD Conf.*, 1975.

[55] Sedgewick,R. *Algorithms in C*, Addison Wesley Publ. Inc., 1988.

[56] Selinger,P.G. i ost. "Access path selection in a relational database system", *ACM SIGMOD Conf.*, 1979.

[57] Smith,J.M., Chang,P.Y.T. "Optimizing the performance of a relational algebra database interface", *Comm.ACM*, 18(10), 1975.

[58] Smith,J.M., Smith,D.C. "Database abstractions: aggregation and generalization", *ACM ToDS*, 1(1), 1976.

[59] Stonebraker,M. (ured.) *The INGRES papers: Anatomy of a Relational Database System*, Addison Wesley Publ. Inc., 1986.

[60] Stonebraker,M., Rowe,L. "The design of POSTGRES", *ACM SIGMOD Conf.*, 1986.

[61] Stroustrup,B. *The C++ Programming Language*, Addison Wesley Publ. Inc., 1986.

[62] Teorey,T. *Database Modeling and Design: The Entity-Relationship Approach*, Morgan Kauffman, 1990.

[63] The Committee for Advanced DBMS Function. "Third generation database system manifesto", *SIGMOD RECORD*, 19(3), 1990.

[64] Ullman,J.D. *Principles of Database Systems*, Computer Science Press, 1980.

[65] Ullman,J.D. *Database and Knowledge-Base Systems, vol. I*, Computer Science Press, 1988.

[66] Ullman,J.D. *Database and Knowledge-Base Systems, vol. II*, Computer Science Press, 1989.

[67] Wiederhold,G. *Database Design*, McGraw-Hill Book Co., 1977.

[68] Williams,R. i ost. "R*: An overview of the architecture", *IBM Research Reports*, RJ3325, 1981.

[69] Wilkinson,K. i ost. "The Iris architecture and implementation", *IEEE Trans. on Knowledge and Data Engineering*, 20(1), 1990.

[70] Youssefi,K., Wong,E. "Query processing in a relational database management system", u [59].