2. RELACIONI MODE

Slede}e dve bitne karakteristike ~ine relacioni model najpopularnijim modelom baza podataka:

* Struktura modela je veoma jednostavna, prihvatljiva svakom korisniku, jer relaciona baza podataka predstavlja skup tabela. I same operacije, koje iz skupa datih tabela (baze podataka) generi{u izlaz (tako|e tabelu), su jednostavne i lako prihvatljive.

* Mogu}a je formalno-matemati~ka interpretacija tabela. Tabela se mo`e definisati kao matemati~ka relacija i zatim iskoristiti bogata teorijska osnova odgovaraju}eg matemati~kog aparata.

2.1. Struktura relracionog modela

Kartezijanski (Dekartov) proizvod. Neka je data kolekcija skupova D1, D2, . . . , Dn (ne neophodno razli~itih). Kartezijanski proizvod ovih n skupova

D1 x D2 x ... x Dn

je skup svi mogu}ih ure|enih n-torki

<d1, d2, ..., dn>, tako da je d1 $\in$ D1, d2 $\in$ D2, ..., dn $\in$ Dn.

Primer 1: A = {1, 2, 3, 4}, B = {4, 6, 8}

A x B = {<1,4>, <1,6>, <1,8>, <2,4>, <2,6>, <2,8>, <3,4>, <3,6>, <3,8>, <4,4>, <4,6>, <4,8>}.

Relacija. Relacija definisana na n skupova je podskup Dekartovog proizvoda tih n skupova.

R $\subset$ D1 x D2 x ... x Dn

Podskup sadr`i one n-torke Dekatrovog proizvoda koje zadovoljavaju zadatu relaciju.

Primer 2: Neka je na skupovima A i B iz Primera 1 zadata relacija R: A x B={<a,b> | a=b/2}

R = {<2,4>, <3,6>, <4,8>}

Domen relacije. Skupovi D1, D2, ..., Dn se nazivaju domenima relacije R.

Stepen relacije. Broj domena na kojima je definisana neka relacija se naziva stepen relacije. (Razlikujemo unarne (na jednom domenu), binarne (na dva domena) i n-arne relacije).

Kardinalnost relacije je broj n-torki u relaciji.

Kako je Dekartov proizvod skup ure|enih n-torki, redosled elemenata u jednoj n-torki je bitan. Na primer, ako na domenima

I1 = {001, 007, 035}, C1 = {Milo{, Ana}, I2 = {19, 22}

defini{emo relaciju R

R $\subset$ I1 x C1 x I2 = {<001, Milo{, 19>, {007, Ana, 19>, <035, Ana, 22>},

u njoj je bitno da prvi elemenat trojke uzima vrednost iz prvog, drugi iz drugog, a tre}i iz tre}eg skupa.

Me|utim, ako vrednostima elemenata u n-torkama pridru`imo imena domena (semanti~ka, da bismo interpetirali odgovaraju}e podatke, daju}i sa istim ciljem i ime relaciji), mo`emo u~initi redosled elemenata u n-torkama bezzna~ajnim.

STUDENT $\subseteq$ BRIND x IME x STAROST = {<BRIND:001, IME:Milo{, STAROST:19>,

<IME: Ana, BRIND: 007, STAROST:19>, <BRIND: 035, STAROST: 22, IME: Ana>}

Atribut relacije. Imenovani domen, sa imenom koje defini{e ulogu domena u relaciji se naziva atribut relacije. Atributi relacije STUDENT su BRIND, IME i STAROST.

Koncept atributa omogu}uje predstavljanje relacije kao tabele. Relacija STUDENT se mo`e predstaviti sa:

STUDENT	BRIND	IME	STAROST
	001	Milo{	19
	007	Ana	19
	035	Ana	22

Po{to je relacija skup, a svaka tabela nije, defini{u se slede}i uslovi koje tabela mora da zadovolji da bi bila relacija:

(1) Ne postoje duplikati vrsta tabele;
(2) Redosled vrsta nije zna~ajan;
(3) Redosled kolona nije zna~ajan.

Pored toga, da bi se mogao definisati jednostavan skup operacija nad relacijama, defini{e se slede}i dodatni uslov:

(4) Sve vrednosti atributa u relacijama su atomske, ili drugim re~ima, nisu dozvoljeni atributi ili grupe atributa "sa ponavljanjem" (kao {to su dozvoljena polja i grupe polja "sa ponavljanjem u zapisu neke datoteke - OCCURS naredba u COBOL-u), odnosno nije dozvoljeno da vrednosti nekih atributa u relaciji budu relacije (nisu dozvoljene "tabele u tabeli").

Ako relacija zadovoljava uslov (4) tada je ona u Prvoj normalnoj formi. Svaka relacija u relacionom modelu mora biti u prvoj normalnoj formi. Termin "normalizovana relacija" se koristi za relacije u prvoj normalnoj formi. (Za ostale normalne forme mora se precizirati o kojoj normalnoj formi se radi.)

Primer 3. Relacija (tabela) STUDISP, zapisana u slede}em obliku, nije normalizovana, jer su NAZIVPRED i OCENA "grupa sa ponavljanjem"

BRIND	IME	NAZIVPRED	OCENA
001	Milo{	Baze podataka	9
		Matematika	7
		Fizika	10
007	Ana	Fizika	7
		Matematika	9

Relacija STUDISP se mo`e dovesti u prvu normalnu formu na slede}i na~in (uvode}i redundansu podataka):

BRIND	IME	NAZIVPRED	OCENA
001	Milo{	Baze podataka	9
001	Milo{	Matematika	7
001	Milo{	Fizika	10
007	Ana	Fizika	7
007	Ana	Matematika	9

Klju~evi. ^injenica da su sve n-torke u relaciji razli~ite, govori da postoji jedan atribut ili vi{e atributa zajedno (u krajnjem slu~aju svi zajedno) ~ije vrednosti jedinstveno identifikuju jednu n-torku u relaciji (jednu vrstu u tabeli). Taj atribut ili ta grupa atributa se nazivaju klju~em relacije (jedan atribut - prost klju~, grupa atributa - slo`en klju~). Na primer, u relaciji STUDENT, klju~ je BRIND (prost klju~), dok je u relaciji STUDISP klju~ grupa BRIND, NAZIVPRED (slo`en klju~).

Klju~ relacije se stro`ije defini{e na slede}i na~in:

Klju~ relacije R je takva kolekcija K njenih atributa koja zadovoljava slede}a dva uslova:

- Osobina jedinstvenosti. Ne postoje bilo koje dve n-torke sa istom vredno{}u K.
- Osobina neredundantnosti. Ako se bilo koji atribut izostavi iz K, gubi se osobina jedinstvenosti.

Ona kolekcija atributa K koja zadovoljava samo osobinu jedinstvenosti naziva se nadklju~ relacije.

Mo`e, u jednoj relaciji postojati vi{e razli~itih kolekcija K atributa koje zadovoljavaju definiciju klju~a. (Na primer, ako u relaciju STUDENT dodamo atribut MLB - mati~ni li~ni broj, i on }e, pored BRIND, zadovoljavati definiciju klju~a). Sve takve kolekcije se nazivaju kandidati za klju~. Jedan od kandidata koji se izabere da prakti~no slu`i za identifikaciju n-torke relacije se tada naziva primarni klju~. Ostali (neizabrani) kandidati se tada nazivaju alternativnim klju~evima.

Atributi koji u~estvuju u klju~evima (koji su deo kandidata za klju~) nazivaju se klju~nim atributima. Ostali atributi u realciji su neklju~ni (ili sporedni) atributi.

Spoljni klju~ je atribut (ili grupa atributa) u relaciji R1 ~ija se vrednost koristi za povezivanje sa vredno{}u primarnog klju~a u nekoj relaciji R2. Spoljnji klju~ i njemu odgovaraju}i primarni klju~ moraju biti definisani nad istim domenom. Spoljni klju~evi slu`e da uspostave veze izme|u relacija u relacionoj bazi podataka. Na primer, u relaciji STUDISP, BRIND je spoljnji klju~, preko koga se ova relacija mo`e povezati sa relacijom STUDENT.

Relaciona baza podataka je kolekcija vremenski promenljivih relacija.

Izvedena relacija (pogled) je relacija koja se mo`e izvesti iz skupa datih baznih i izvedenih relacija, preko operacija koje se defini{u nad relacijama.

Bazna relacija je relacija koja se ne mo`e izvesti iz ostalih relacija u relacionoj bazi podataka.

Relacija se mo`e predstaviti kao tabela. U mnogim relacionim bazama podataka koristi se terminologija vezana za tabele, pa zbog toga, a i da bi se, preko analogije sa pojmovima u klasi~noj obradi podataka, jo{ jednom objasnili uvedeni pojmovi, dajemo slede}u ekvivalenciju relacione, "tabelarne" i terminologije klasi~ne obrade podataka.

RELACIONA	TABELARNA	KLASI^NA OBRADA PODATAKA
DOMEN	SKUP VREDNO5TI	TIP
ATRIBUT	KOLONA	POLJE
N-TORKA	VRSTA	REKORD
PRIMARNI KLJU^	IDENTIFIKATOR VRSTE	JEDINSTVENI KLJU^
RELACIJA (normalizovana)	TABELA	RAVNA DATOTEKA
STEPEN	BROJ KOLONA	BROJ POLJA U ZAPISU
KARDINALNOST	BROJ VRSTA	BROJ REKORDA U DATOTECI.

Ekstenzija relacije je skup svih n-torki date relacije, odnosno predstavljanje tabele navo|enjem svih vrsta. Tabela STUDENT i normalizovana tabela STUDISP predstavljaju ekstenzije odgovaraju}ih relacija.

Intenzija relacije je generalizacija ekstenzije, ona je vremenski nepromenljiva i ozna~ava se na slede}i na~in:

STUDENT (BRIND, IME, STAROST)

STUDISP (BRIND, NAZIVPRED, IME, OCENA)

(Ispred zagrade je naziv relacije, u zagradi su navedeni atributi, a primarni klju~ obele`en je masnim otiskom).

[ema relacione baze podataka je predstavljanje strukture relacione baze kao skupa intenzija relacija.

Primer 3. Primer {eme relacione baze podataka.

[EMA BAZE PODATAKA

RADNIK (RADN#, MLB, IME, STAROST, MESTO_RO\, ORGJED#)

ORGANIZACIJA (ORGJED#, NAZIVORG, MESTO_ORG)

PROJEKAT (PROJ#, NAZIVPR, [RUK, MESTO)

ZADATAK (PROJ#, BRZAD, OPISZAD)

RASPORED (RADN#, PROJ#, BRZAD, DANARADA)

DAKT I LOGRAF(RADN#, KLASA)

(Atribut [RUK je definisan nad istim domenom kao i RADN#, ili drugim re~ima u relaciji PROJEKAT atribut RADN# imao bi ulogu da defini{e {ifru rukovodioca projekta, pa zato dobija ime [RUK).

BAZA PODATAKA

RADNIK

RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
r1	1312040..	Ana	24	Beograd	oj1
r2	0505945..	Milan	22	Ni{	oj1
r3	1010943..	Pera	44	Sarajevo	oj2
r4	1111937..	Persa	52	Beograd	oj1
r5	0101970..	Zoran	24	Beograd	oj3
r6	0202950..	Sima	24	Zagreb	oj2
r7	0203955..	Mira	27	Novi Sad	oj3
r8	0404944..	Milo{	24	Beograd	oj1
r9	0707947..	Aca	25	Zagreb	oj2
r10	0909899..	Pera	72	Beograd	oj1
r11	0908952..	?	24	Beograd	oj1

ORGANIZACIJA

ORGJED#	NAZIVORG	MESTO_ORG
oj1	Prodaja	Zagreb
oj2	Razvoj	Beograd
oj3	Skladi{te	Skopje

PROJEKAT	PROJ#	NAZIVPR	[RUK	MESTO
	pr1	Stanovi	r6	Beograd
	pr2	Fabrika	r4	Zagreb

ZADATAK	PROJ#	BRZAD	OPISZAD
	pr1	z1	Idejni projekat
	pr1	z2	Glavni projekat
	pr1	z3	Izgradnja
	pr2	z1	Nabavka opreme
	pr2	z2	Monta`a

RASPORED	RADN#	PROJ#	BRZAD	DANARADA
	r1	pr1	z1	20
	r1	pr1	z2	18
	r1	pr2	z1	10
	r2	pr1	z3	45
	r3	pr2	z1	18
	r3	pr2	z2	25
	r5	pr1	z3	30
	r7	pr1	z1	12
	r7	pr2	z1	33
	r8	pr1	z2	23
	r8	pr2	z1	20
	r8	pr2	z2	20
	r9	pr2	z2	45
	r10	pr1	z1	13
	r10	pr1	z2	7
	r10	pr1	z3	15
	r10	pr2	z1	20
	r10	pr2	z2	25

DAKTILOGRAF	RADN#	KLASA
	r7	B
	r10	A

Nula vrednosti. Termin "nula vrednost" (koji }emo obele`avati sa ?) se koristi da ozna~i "jo{ nepoznatu vrednost" za neki atribut ili "neprimenljivo svojstvo" za neki objekat ili vezu koje predstavlja tabela. Na primer, u relaciji RADNIK radnik sa RADN# = r11 je za atribut IME dobio "nula vrednost" (?) jer je njegovo ime "jo{ nepoznato" i odgovaraju}i podatak }e biti unet u bazu kada se sazna. Pretpostavimo da smo, umesto da imamo posebnu relaciju DAKTILOGRAF, pro{irili relaciju RADNIK za atribut KLASA. Tada bi svi radnici, osim radnika r7 i r10 imali za vrednost atributa KLASA nula vrednost, jer je taj atribut "neprimenljivo svojstvo" za sve druge radnike, zato {to oni nisu daktilografi. Pojava nula vrednosti koje su "neprimenljivo svojstvo" u nekoj relacionoj bazi podataka ukazuje na to da takva relaciona baza nije dobro projektovana. (Zbog toga su, u navedenom primeru, daktilografi, kao poseban podskup radnika, izdvojeni u posebnu relaciju). Me|utim, nula vrednosti kao "jo{ nepoznate vrednosti" mogu se pojavljivati u bazi podataka i o na~inu izvo|enja operacija sa ovim nula vrednostima bi}e kasnije vi{e re~i.

2.2. Ograni~enja u relacionom modelu

U relacionom modelu je neophodno definisati i ograni~enja na vrednosti nekih atributa u pojedinim relacijama, da bi se korektnije opisao realni sistem koga baza podataka modelira. Specifikacija domena atributa ve} sama po sebi predstavlja ograni~enje na vrednosti atributa. (Na primer, domen atributa STAROST je skup prirodnih brojeva od 15 do 65). Mogu se davati i slo`enija ograni~enja koja defini{u granice nekih izvedenih promenljivih, ili koja defini{u granice vrednosti jednog atributa u zavisnosti od vrednosti nekih drugih atributa. (Na primer, prose~ni LD svih radnika mora biti manji od 2M, ili u~e{}e DAKTILOGRAFA na ZADATKU z2 PROJEKTA pr2 ne mo`e biti ve}e od 30 DANARADA). Ovakva ograni~enja zavise od konkretnog sistema i moraju u bazi podataka da budu uvek zadovoljena da bi se o~uvao integritet baze podataka.

Postoje i op{ta ograni~enja koja va`e za bilo koji relacioni model, koja proizilaze iz na~ina opisa realnog sistema u relacionom modelu i koja se nazivaju pravilima integriteta relacionog modela. Defini{u se dva op{ta pravila integriteta relacionog modela:

Pravilo integriteta 1 - Integritet entiteta. Ni jedan atribut koji je primarni klju~ ili deo primarnog klju~a neke bazne relacije nemo`e da uzme nula vrednost.

Svaka bazna relacija u nekom relacionom modelu predstavlja bilo objekat bilo vezu izme|u objekata u realnom sistemu. U na{em primeru relacije RADNIK, DAKTILOGRAF, ORGANIZACIJA, PROJEKAT i ZADATAK predstavljaju objekte realnog sistema, a relacija RASPORED vezu ovih objekata, koja defini{e koji radnici rade na nekom zadatku, nekog projekta. Primarni klju~evi u baznim relacijama identifikuju jedan objekat u skupu objekata datog tipa (jednog radnika u skupu RADNIKA) ili jednu vezu u skupu datih veza, pa je o~igledno da zbog takve njihove uloge ne mogu dobijati nula vrednosti.

Pravilo integriteta 2 - Referencijalni integritet. Ako neka bazna relacija (recimo R2) poseduje spoljni klju~ (recimo SK) koji ovu relaciju povezuje sa nekom drugom baznom relacijom (recimo R1), preko primarnog klju~a (recimo PK), tada svaka vrednost SK mora biti bilo jednaka nekoj vrednosti PK, ili biti nula vrednost. Relacije R1 i R2 ne moraju biti razli~ite.

Na primer, u relaciji RADNIK spoljni klju~ je ORGJED# i on ne mo`e uzeti neku vrednost, ako se ta ista vrednost ne pojavljuje kao vrednost primarnog klju~a ORGJED# u relaciji ORGANIZACIJA. Ili, u relaciji ZADATAK, atribut PROJ# je spoljni klju~ i on ne mo`e uzeti neku vrednost, ako se ta vrednost nije pojavila kao vrednost primarnog klju~a PROJ# u relaciji PROJEKAT.

Referencijalni integritet obezbe|uje korektno povezivanje objekata koji su predstavljeni u relacionom modelu i neformalno se mo`e iskazati i na slede}i na~in: Ne mo`e objekat koji nije predstavljen u odgovaraju}em skupu objekata u bazi podataka da u~estvuje u nekoj od veza predstavljenih u bazi podataka.

2.3. Operacije relacionog modela

Postoje dva op{ta na~ina iskazivanja operacija relacionog modela:

(a) Relaciona algebra, u kojoj se defini{e skup operacija pomo}u kojih je mogu}e, na proceduralan na~in, dobiti `eljenu relaciju (tabelu) iz skupa datih relacija (tabela).

(b) Relacioni ra~un, koji je neproceduralni na~in iskazivanja operacija, gde se, pomo}u konstrukcija predikatskog ra~una prvog reda u kome su promenljive bilo n-torke relacija (relacioni ra~un n-torki) ili domeni relacija (relacioni ra~un domena), defini{u osobine relacije koja se `eli dobiti.

2.3.1. Relaciona algebra

Relacija se defini{e kao podskup Dekartovog proizvoda i mo`e se tretirati kao skup n-torki. Zbog toga su osnovne skupovne operacije i operacije relacione algebre. Me|utim, o~igledno je da se operacije unije, preseka i diferencije ne mogu primeniti, (ili preciznije, nemaju semanti~ki zna~aj) na bilo koje dve relacije. (Na primer, unija relacija ORGANIZACIJA i PROJEKAT formalno treba da dâ relaciju ~ije su n-torke elementi bilo relacije ORGANIZACIJA bilo relacije PROJEKAT, a tako dobijen rezultat nije relacija u smislu u kome su relacije ovde definisane). Zbog toga se defini{e slede}i uslov kompatibilnosti relacija R1 i R2 za izvo|enje operacija unije, preseka i diferencije:

> Relacije R1 i R2 moraju imati isti broj atributa (isti stepen), a odgovaraju}i atributi moraju biti definisani nad istim domenima.

U literaturi se ovakve relacije nazivaju relacije kompatibilne za uniju (union compatible relations).

1. Unija. Date su relacije R1 i R2 koje zadovoljavaju navedeni uslov kompatibilnosti. Rezultat operacije unije

$$R3 = R1 \cup R2$$

je relacija R3 koja sadr`i sve n-torke koje se pojavljuju bilo u R1 bilo u R2.

Primer: Defini{imo relaciju

NOVAORGAN	ORGJED#	NAZIVORG	MESTO_ORG
	oj4	Komercijala	Beograd
	oj6	Skladi{te 2	Zagreb

RR = ORGANIZACIJA $\cup$ NOVAORGAN

ORGJED#	NAZIVORG	MESTO_ORG
oj1	Prodaja	Zagreb
oj2	Razvoj	Beograd
oj3	Skladi{te	Skopje
oj4	Komercijala	Beograd
oj6	Skladi{te 2	Zagreb

2. Diferencija. Date su relacije R1 i R2 koje zadovoljavaju navedeni uslov kompatibilnosti. Rezultat operacije diferencije

$$R3 = R1 - R2$$

su n-torke relacije R1 koje nisu istovremeno i n-torke relacije R2.

Primer:

RRR = RR - ORGANIZACIJA

ORGJED#	NAZIVORG	MESTO_ORG
oj4	Komercijala	Beograd
oj6	Skladi{te 2	Zagreb

3. Presek. Date su relacije R1 i R2 koje zadovoljavaju navedeni uslov kompatibilnosti. Rezultat operacije preseka

$$R3 = R1 \cap R2$$

je relacija R3 koja sadr`i n-torke koje se pojavljuju u obe relacije R1 i R2.

Operacije unije i diferencije su operacije pogodne za a`uriranje baze podataka. Operacijom unije mogu se u neku relaciju dodati nove n-torke, a operacijom diferencije mogu se iz neke relacije izbaciti ne`eljene n-torke. Izmena vrednosti pojedinih atributa u nekoj relaciji vr{i se uzastopnom primenom operacije diferencije, pomo}u koje se izbaci cela n-torka u kojoj se nalazi posmatrani atribut, a zatim se operacijom unije "vra}a" izba~ena n-torka sa promenjenom vredno{}u atributa.

4. Dekartov proizvod. Dekartov proizvod se mo`e primeniti na bilo koje dve relacije. Rezultat ove operacije

R3 = R1 x R2

je relacija R3 ~ije su n-torke svi "parovi" koje ~ine jedna n-torka relacije R1 i jedna n-torka relacije R2.

Primer:

RRR = PRDJEKAT x DAKTILOGRAF

PROJ#	NAZIVPR	[RUK	MESTO	RADN#	KLASA
pr1	Stanovi	r6	Beograd	r7	B
pr1	Stanovi	r6	Beograd	r10	A
pr2	Fabrika	r4	Zagreb	r7	B
pr2	Fabrika	r4	Zagreb	r10	A

Pored skupovnih operacija u relacionom modelu se defini{u i slede}e specifi~ne operacije:

5. Projekcija. Operacija projekcije je unarna operacija koja iz neke relacije selektuje skup navedenih atributa, odnosno "vadi" vertikalni podskup iz odgovaraju}e tabele. Formalno se mo`e definisati na slede}i nadin:

Neka je R(A1, A2, ..., An) relacija, a X podskup njenih atributa. Ozna~imo sa Y komplement {A1, A2, ..., An} - X. Rezultat operacije projekcije relacije R po atributima X je

P[X] = {x | tako da postoji y da je <x,y> ∈ R}.

Primer: RR = P[STAROST,MESTO_RO\]RADNIK

STAROST	MESTO_RO\
24	Beograd
22	Ni{
44	Sarajevo
52	Beograd
24	Zagreb
27	Novi Sad
25	Zagreb
72	Beograd

(Kada se iz neke relacije selektuje podskup atributa preko operacije projekcije u rezultatu se mogu pojaviti duplikati n-torki. Operacija projekcije podrazumeva da se ovi duplikati elimini{u, kako je to u navedenom primeru i ura|eno).

6. Selekcija (Restrikcija). Selekcija je tako|e unarna operacija koja iz date relacije selektuje n-torke koje zadovoljavaju zadati uslov ("vadi" horizontalni podskup tabele). Formalno se defini{e na slede}i na~in:

Data je relacija R(A1, A2, ..., An) i predikat P definisan nad njenim atributima. Rezultat operacije selekcije

S[P]R = {x | x ∈ R i P(x)}

Primer: Prika`i radnike koji su stariji od 24 godine, a ro|eni su u Beogradu.

RR = S[STAROST > 24 AND MESTO_RO\ = 'Beograd']RADNIK

RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
r4	1111937..	Persa	52	Beograd	oj1
r10	0909899..	Pera	72	Beograd	oj1

7. Spajanje (Join). Spajanje je binarna operacija koja spaja dve relacije na taj na~in da se u rezultatu pojavljuju oni parovi n-torki jedne i druge relacije koji zadovoljavaju uslov zadat nad njihovim atributima. Formalno se defini{e na slede}i na~in: Date su relacije R1(A1, A2, ..., An) i R2(B1,B2, ..., Bm) i predikat Θ definisan nad njihovim atributima. Obele`imo sa X i Y skupove atributa relacija R1 i R2, respektivno. Rezultat operacije spajanja ovih relacija (tzv. teta spajanje) je

R1[xΘ]R2 = {<x,y> | x ∈ R1 AND y ∈ R2 AND Θ(x,y)}.

Oznaka xΘ za operaciju spajanja ukazuje na ~injenicu, o~iglednu iz definicije teta spajanja, da ova operacija nije primitivna operacija relacione algebre, ve} se mo`e izvesti uzastopnom primenom operacije Dekartovog proizvoda (x) i selekcije po predikatu Θ iz tako dobijene relacije.

Ako je predikat Θ definisan sa Ak = Bj, s tim da su i atributi Ak i Bj definisani nad istim domenima, tada se takvo spajanje naziva ekvispajanje.

Primer ekvispajanja:

RR = RADNIK[RADNIK.RADN# = DAKTILOGRAF.RADN#]DAKTILOGRAF

RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#	RADN#	KLASA
r7	0203955..	Mira	27	Novi Sad	oj3	r7	B
r10	0909899..	Pera	72	Beograd	oj1	r10	A

O~igledno je da se u rezultatu ekvispajanja uvek pojavljuju dve iste kolone. Ako se jedna od te dve kolone izbace, takvo spajanje se naziva prirodno spajanje. Uobi~ajeno je da se ozna~ava sa *

RR = RADNIK[RADNIK.RADN# * DAKTILOGRAF.RADN#]DAKTILOGRAF

RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#	KLASA
r7	0203955..	Mira	27	Novi Sad	oj3	B
r10	0909899..	Pera	72	Beograd	oj1	A

8. Deljenje. Deljenje je operacija pogodna za upite u kojima se javlja re~ “svi”. Formalno se defini{e na slede}i na~in:

Neka su A(X,Y) i B(Z) relacije gde su X, Y i Z skupovi atributa takvi da su Y i Z jednakobrojni, a odgovaraju}i domeni su im jednaki. Rezultat operacije deljenja

A[Y ÷ Z]B = R(X)

gde n-torka x uzima vrednosti iz A.X, a par <x,y> postoji u A za sve vrednosti y koje se pojavljuju u B(Z).

Primer: Izlistaj {ifre radnika koji rade na svim projektima. Da bi se izvr{io ovaj upit postupa se po slede}em algoritmu:

1. R1 = P[PROJ#]PROJEKAT

R1	PROJ#
	pr1
	pr2

2. R2 = P[RADN#, PROJ#]RASPORED

R2

RADN#	PROJ#
r1	pr1
r1	pr2
r2	pr1
r3	pr2
r5	pr1
r7	pr1
r8	pr1
r8	pr2
r9	pr2
r10	pr1
r10	pr2

3. R# = R2[PROJ# ÷ PROJ#]R1

R3

RADN#
r1
r8
r10

Operacija deljenja nije primitivna operacija relacione algebre ve} se mo`e izvesti na slede}i na~in:

A(X, Y) [Y ÷ Z]B(Z) = P[X]A - P[X] ((P[X]A x B) - A)

Obja{njenje:

P[X]A daje sve n-torke koje mogu da u~estvuju u rezultatu

P[X]A x B daje relaciju u kojoj se za svaku vrednost z iz B pojavljuju parovi <x,z> sa svim vrednostima x.

(P[X]A x B) - A ne sadr`i ni u jednom paru <x,z> one vrednosti x za koje u relaciji A, kao vrednosti y, postoje sve vrednosti z.

Kompletan, izraz prema tome, je relacija koja sadr`i one n-torke x za koje postoje u paru <x,y>, kao vrednosti y, sve vrednosti z.

Relaciona algebra je proceduralni jezik. Pomo}u operacija relacione algebre sa~injava se procedura koja dovodi do odgovora na postavljeni upit, kao {to je to pokazano u prethodnom primeru. I ako je proceduralni jezik, relaciona algebra je znatno mo}nija od klasi~nih programskih jezika koji su tako|e proceduralni. Razlog za to je {to je operand relacione algebre relacija (cela tabela), a operand operacija sa datotekama u klasi~nim jezicima je rekord (vrsta tabele).

Mo`e se pokazati da se bilo koja relacija (bilo koji upit), izvodljiva iz skupa datih relacija, mo`e dobiti procedurom (algoritmom) od tri koraka:

(1) Dekartov proizvod svih relacija koje se koriste.
(2) Selekcija n-torki za koje se predikat selekcije sra~unava u T.
(3) Projekcija rezultata po atributima koji se prikazuju.

Ova procedura se mo`e iskazati jednim op{tim izrazom relacione algebre

P[A1, A2, ..., An] (S[P] (R1 x R2 x ... x Rm)).

Me|utim, ovakvo izvo|enje operacija bilo bi veoma "skupo" jer bi rezultat Dekartovog proizvoda relacija R1, R2, ..., Rm bio relacija sa k1 x k2 x ... x km n-torki, gde su sa ki ozna~ene kardinalnosti odgovaraju}ih relacija. Zbog toga se i defini{e operacija spajanja koja istovremeno obavlja Dekartov proizvod i selekciju, smanjuju}i na taj na~in broj n-torki koji se pretra`uje. Pored toga o~igledno je da su algoritmi sa vi{e koraka, u kojima bi se prvo vr{ile sve selekcije koje se mogu izvesti na

pojedina~nim relacijama, zatim projekcije, pa tek onda spajanja, znatno efikasniji. O optimizaciji izvr{enja upita bi}e kasnije vi{e re~i. Gornji op{ti izraz za realizaciju upita u relacionoj algebri dat je da ilustruje mo} ovog jezika i da omogu}i pore|enje relacione algebre sa relacionim ra~unom i nekim komercijalnim upitnim jezicima.

Operacije sa nula vrednostima

Navedene operacije relacione algebre nisu uzimale u obzir nula vrednosti koje mogu postojati u relacijama. Podrazumevalo se da se vrednosti predikata sra~unavaju na osnovu standardnih, dvovrednosnih tablica istinitosti. Isto tako, bez dvoumljenja je bilo mogu}e da se odrede duplikati n-torki, odnosno kardinalnost pojedinih skupova. Pojavljivanje nula vrednosti u relacionoj bazi podataka zahteva da se pro{iri skup definisanih operacija koje bi na neki na~in uklju~ile i nula vrednosti. Osnovne postavke za operacije sa nula vrednostima su slede}e:

(1) Tablice istinitosti trovrednosne logike:

AND	T	?	F
T	T	?	F
?	?	?	F
F	F	F	F

OR	T	?	F
T	T	T	T
?	T	?	?
F	T	?	F

NOT	
T	F
?	?
F	T

(? predstavlja nula vrednost, a mo`e se unutar tablica istinitosti ~itati i kao "mo`da").

(2) Sra~unavanje aritmeti~kih formula. Neka α ozna~ava neki od aritmeti~kih operatora (+, -, *, /) i neka su x i y dve numeri~ke vrednosti. Vrednost aritmeti~kog izraza "x α y" je po definiciji nula vrednost ako bilo x, bilo y, bilo oba dobiju nula vrednost.

(3) Skupovi sa nula vrednostima. Posmatrajmo kolekciju {1, 2, 3, ?}. Da li je ova kolekcija skup? Ako ? uzme vrednost 1, 2 ili 3 onda nije, ili je mo`emo tretirati kao skup sa kardinalno{}u 3. Ako ? nije ni 1, ni 2, ni 3 onda gornja kolekcija predstavlja skup sa kardinalno{}u 4. To pokazuje da postoje razli~ite nula vrednosti, nula vrednost koja nije 1, nula vrednost koja nije 2, zatim nula vrednost koja nije ni 1 ni 2 i tako dalje. Operacije koje bi uzele u obzir razli~itost nula vrednosti bile bi veoma slo`ene. Zbog toga se operacije defini{u sa jednom vrstom nula vrednosti, a ostale nula vrednosti se tretiraju kao duplikati.

Imaju}i u vidu ove op{te postavke nave{}emo primere nekih ranije definisanih operacija na relacijama koje poseduju nula vrednosti:

UNIJA: $R3 = R1 \cup R2$

R1	A	B
	a	b
	a	?
	?	b
	?	?

R2	A	B
	x	y
	?	b
	?	?

R3	A	B
	a	b
	a	?
	?	b
	?	?
	x	y

DIFERENCIJA: $R4 = R1 - R2$

R4	A	B
	a	b
	a	?

DEKARTOV PROIZVOD. Dekartov proizvod ostaje neizmenjen

SELEKCIJA. Operacija selekcije ostaje neizmenjena, selektuju se one n-torke za koje se odgovaraju}i predikat sra~unava u T na osnovu trovrednosnih tablica istinitosti. Ova operacija se ~esto zove i "TRUE SELECTION" (istinita selekcija).

S[A = a]R1

A	B
a	b
a	?

PROJEKCIJA. Uzima se u obzir da su nula vrednosti duplikati (jedna vrsta nula vrednosti).

P[A]R1

A
a
?

SPAJANJE. Operacija spajanja ostaje neizmenjena, jer operacije Dekartovog proizvoda i Selekcije ostaju neizmenjene. U rezultatu se pojavljuju one n-torke za koje se predikat spajanja sra~unava u T, na osnovu trovrednosnih tablica istinitosti (TRUE TETA JOIN - istinito teta spajanje).

DODATNE OPERACIJE. Zbog postojanja nula vrednosti u bazi podataka, neophodno je da se defini{e logi~ka funkcija "IS NULL" ^iji argument mo`e da bude neki skalarni izraz, a koja se sra~unava u T ako je vrednost tog argumenta nula vrednost, a ina~e uzima vrednost F.

MO@DA_SELEKCIJA (MAYBE_SELECT). Selektuju se one n-torke relacije za koje se predikat selekcije, na osnovu trovrednosnih tablica istinitosti, sra~unava u nula vrednost.

MAYBE_SELECT[A]R3

A	B
?	b
?	?

MO@DA_SPAJANJE (MAYBE_JOIN). U rezultatu spajanja se pojavljuju one n-torke za koje se predikat spajanja sra~unava u nula vrednost, na osnovu trovrednosnih tablica istinitosti.

Ra = Rb[MAYBE_JOIN A = D]Rc

Rb

A	B
a1	b1
a2	b2
?	b3

Rc

C	D	E
c1	?	e1
c2	d2	e2
c3	?	e3

Ra

A	B	C	D	E
a1	b1	c1	?	e1
a1	b1	c3	?	e3
a2	b2	c1	?	e1
a2	b2	c3	?	e3
?	b3	c1	?	e1
?	b3	c2	d2	e2
?	b3	c3	?	e3

SPOLJNO_SPAJANJE (OUTER_JOIN). Neka su date relacije R1(A,B) i R2(C,D). Pretpostavimo da se vr{i operacija ekvispajanja ovih relacija, R1[A = C]R2. Ako je P[A]R1 ≠ P[C]R2 (projekcije relacija po atributima spajanja su razli~ite), tada }e se u rezultatu spajanja "izgubiti" neke n-torke relacija R1 i R2. Ako se takvo gubljenje informacija ne `eli, SPOLJNO_SPAJANJE ih mo`e sa~uvati na taj na~in {to se u rezultat dodaju i ove n-torke i to tako {to za takve n-torke relacije R1 atributi C i D uzimaju nula vrednosti, a za takve n-torke relacije R2 atributi A i B uzimaju nula vrednosti.

Primer: R1[OUTER_JOIN A = C]R2

R1	A	B
	1	2
	2	1
	4	?

R2	C	D
	3	4
	2	2

R3	A	B	C	D
	2	1	2	2
	1	2	?	?
	?	?	3	4
	4	?	?	?

SPOLJNA_UNIJA (OUTER_UNION). Kao {to je re~eno, operacija unije se mo`e izvesti samo nad relacijama koje zadovoljavaju kriterijume kompatibilnosti (da su istog stepena i da su im odgovaraju}i atributi definisani nad istim domenima). Dodavanjem novih atributa i postavljanjem njihovih vrednosti na nula vrednosti mogu se uvek dve nekompatibilne relacije u~initi kompatibilnim. Spoljna unija podrazumeva da su, na ovaj na~in, dve nekompatibilne relacije u~injene kompatibilnim, pa je tada izvr{ena operacija unije.

Primer: R3 = R1 OUTER_UNION R2

R1	A	B	C
	a1	b1	c1
	a2	b1	c2

R2	A	D
	a1	4
	a1	7
	a2	5

R3	A	B	C	D
	a1	b1	c1	?
	a2	b1	c2	?
	a1	?	?	4
	a1	?	?	7
	a2	?	?	5

2.3.2. Relacioni ra~un

Dok je relaciona algebra proceduralni, relacioni ra~un je neproceduralni jezik. Umesto da programer sastavlja proceduru kojom }e se dobiti `eljeni rezultat, relacionim ra~unom se samo formalno specificira `eljeni rezultat. Postoje dva oblika relacionog ra~una: (1) Relacioni ra~un n-torki i (2) Relacioni ra~un domena.

2.3.2.1. Relacioni ra~un n-torki

Relacioni ra~un n-torki je predikatski ra~un prvog reda u kome su domeni promenljivih n-torke relacija date baze podataka.

Osnovni pojmovi predikatskog ra~una su:

(1) Afirmativna re~enica, koja ima smisla i koja je istinita ili neistinita naziva se sud.
(2) Afirmativna re~enica koja ima smisla i koja sadr`i jedan ili vi{e promenljivih parametara i koja postaje sud uvek kada parametri iz re~enice dobiju konkretnu vrednost naziva se predikat. Broj parametara u predikatu se naziva du`ina predikata. (Primer predikata je $x^2 + y^2 \leq 1$).
(3) Predikatski ili kvantifikatorski ra~un je matemati~ka teorija ~iji su objekti formule koje predstavljaju predikate. Simboli koji se koriste da ozna~e neki sud nazivaju se atomskim formulama ili atomima. Atomi u relacionom ra~unu n-torki su:
   - $x \in R$ gde je x n-torka-promenljiva, a R relacija, odnosno promenljiva x uzima vrednosti iz skupa n-torki relacije R,
   - $x.A\ \Theta\ y.B$ gde su x i y promenljive (n-torke), A i B su atributi relacija R1 i R2 iz ~ijih n-torki, respektivno, promenljive x i y uzimaju vrednosti ($x \in R1$, $y \in R2$), a Θ je operacija pore|enja definisana nad domenom atributa A i B (A i B moraju biti definisani nad istim domenom) i

- x.A Θ c gde su x, A i Θ kao i u prethodnom stavu, a c je konstanta koja ima isti domen kao i A.

Formule se formiraju od atoma preko slede}ih pravila:

- Atom je formula;
- Ako je P1 formula, tada su formule i NOT P1 i (P1);
- Ako su P1 i P2 formule tada su formule i P1 AND P2 i P1 OR P2;
- Ako je P1(s) formula koja sadr`i neku slobodnu promenljivu s tada su i ∃ s (P1(s)) i ∀ s (P1(s)) tako|e formule (∃ - "postoji", egzistencijalni kvantifikator, ∀ - "za svako", univerzalni kvantifikator).

Jedna promenljiva u nekoj formuli se mo`e pojaviti vi{e puta. Promenljive mogu biti "slobodne" i "vezane". Vezana promenljiva u formuli je neka vrsta "prividne" (dummy) promenljive koja ima ulogu da pove`e promenljivu iza kvantifikatora sa promenljivima u zoni dejstva kvantifikatora. Drugim re~ima vezana promenljiva u je (∃ u), (∀ u) ili (∃ u) A ili (∀ u) A, gde je A formula u kojoj se pojavljuje u. Sve promenljive koje u nekoj formuli nisu vezane, slobodne su u toj formuli. Na primer, u formuli

∃ x (x > 3)

x je vezana promenljiva, pa je gornja formula ekvivalentna sa

∃ y (y > 3).

U formuli

∃ x (x > 3) AND x < 0

prvo pojavljivanje promenljive x je vezano, a drugo slobodno, pa je ova formula ekvivalentna sa

∃ y (y > 3) AND x < 0

Neka su R1, R2, ..., Rn relacije u nekoj bazi podataka. Neka su A, B, ..., C atributi ovih relacija, respektivno i neka je f formula. Op{ti izraz relacionog ra~una n-torki je tada:

t ∈ R1, u ∈ R2, . . . , v ∈ Rn
t.A, u.B, . . . , v.C GDE_JE f

(Prikazuju se vrednosti atributa A relacije R1, atribut B relacije R2, ... i atribut C relacije Rn, za one n-torke koje zadovoljavaju uslov definisan formulom f).

Primeri:

(a) Prika`i {ifre i imena radnika koji su stariji od 25 godina i koji su ro|eni u Beogradu.

x ∈ RADNIK
x.RADN#, x.IME GDE_JE x.STAROST > 25 AND
x.MESTO_RO\ = 'BEOGRAD'

(b) Prika`i imena radnika koji rade u organizacionoj jedinici “Razvoj”.

x ∈ RADNIK,
y ∈ ORGANIZACIJA
x.IME GDE_JE ∃ y (y.ORGJED# = x.ORGJED# AND
y.NAZIVORG = 'Razvoj')

(c) Prika`i imena radnika koji rade na svim projektima

x ∈ RADNIK,
y ∈ RASPORED
z ∈ PROJEKAT

x.IME GDE_JE ∀ z (∃ y (y.RADN# = x.RADN# AND
y. PROJ# = z. PROJ#))

2.3.2.2. Relacioni ra~un domena

U relacionom ra~unu domena promenljive uzimaju vrednosti iz nekih domena definisane relacione baze podataka. Ovde se, pored navedenih, defini{e jo{ jedna atomska formula, tzv. "uslov ~lanstva" (membership condition). Uslov ~lanstva ima slede}i oblik:

R(term, term, ...)

gde je R ime neke relacije a svaki term ima oblik A: v, gde je A neki atribut relacije R a v je ili promenljiva ili konstanta. Uslov ~lanstva se sra~unava u T (TRUE) ako postoji n-torka u relaciji R koja ima zadate vrednosti navedenih atributa. Na primer, uslov ~lanstva

RADNIK (RADN# = 'r3', ORGJED# = 'oj2')

se sra~unava u T samo ako postoji n-torka u relaciji RADNIK sa navedenim vrednostima datih atributa.

Op{ti izraz relacionog ra~una domena je:

x, y, ..., z GDE_JE f

gde su x, ..., z promenljive a f je formula koja uklju~uje i uslov ~lanstva.

Primeri:

(a) Prika`i {ifre i imena radnika koji su stariji od 25 godina i ro|eni su u Beogradu.
x, y GDE_JE ∃ z > 25 AND RADNIK (RADN#: x, IME:y, STAROST: z,
MESTO_RO\:'Beograd')

(b) Prika`i imena radnika koji rade u organizacionoj jedinici “Razvoj”.
x GDE_JE ∃ y (RADNIK(IME: x, ORGJED#: y) AND
ORGANIZACIJA(ORGJED#: y, NAZIVORG:'Razvoj'))

(c) Prika`i imena radnika koji rade na svim projektima.
x GDE_JE ∀ z (∃ y (RADNIK(RADN#: y, IME: x) AND
PROJEKAT(PROJ#: z) AND RASPORED(RADN#: y, PROJ#: z))

Relaciona algebra i relacioni ra~un (i ra~un n-torki i ra~un domena) su fundamentalno ekvivalentni jedno drugom. E.F Codd je pokazao da se svaki izraz relacionog ra~una mo`e svesti na semanti~ki ekvivalentan izraz u relacionoj algebri, a Ullman, da se svaki izraz u relacionoj algebri mo`e svesti na izraz relacionih ra~una, pa se, na osnovu toga, mo`e zaklju~iti da su ova dva formalizma logi~ki ekvivalentna.

2.4. Relacioni upitni jezici

Komercijalni SUBP, mada se zasnivaju na relacionoj algebri i/ili relacionom ra~unu, poseduju upitne jezike sa konstrukcijama koje su mnogo bli`e korisniku, mnogo bli`e prirodnom jeziku, nego {to su konstrukcije same relacione algebre, odnosno ra~una. Najpoznatiji relacioni upitni jezici su: SQL koji se bazira na kombinaciji relacione algebre i realcionog ra~una, Quel, koji predstavlja implementaciju relacionog ra~una n-torki i QBE, koji predstavlja implementaciju relacionog ra~una domena.

2.4.1. SQL (Structured Query Language)

SQL je 1986 godine postao ameri~ki nacionalni standard (ANSI X3:135 - 1986), sa izgledima da postane i me|unarodni (ISO) standard. Po~eo je da se razvija u IBM-u, ranih sedamdesetih godina, pod imenom SEQUEL. (Structured English Query Language) i mnogi jo{ i danas skra}enicu SQL izgovaraju kao "sequel". Zbog toga {to je postao standard, danas je SQL jezik koji se koristi u velikoj ve}ini relacionih baza podataka. Orginalno se koristi u relacionim sistemima IBM-a (SQL/DS i DB2) i ORACLE, a naknadno je dodat i mnogim drugim koji ga nisu orginalno koristili.

SQL je homogeno relacioni jezik, {to zna~i da mo`e da se koristi i interaktivno, sa terminala i u okviru standardnih programskih jezika (COBOL, PL/I i drugih), za komunikaciju sa bazom podataka. Pored konstrukcija analognih relacionoj algebri ili relacionom ra~unu (koje su osnove jezika za rukovanje podacima) SQL sadr`i i konstrukcije za opisivanje baze podataka (jezik za opis podataka), operacije koje omogu}uju vezivanje SQL-a sa nekim standardnim jezikom (CURSOR operacije), kao i kontrolne konstrukcije za upravljanje konkurentnim obradama, oporavak baze podataka, definisanje za{tite podataka i integriteta baze podataka.

(1) OPIS BAZE PODATAKA (DATA DEFINITION)

Osnovni iskazi za opis baze podataka su:

CREATE SCHEMA AUTHORIZATION ime-kreatora-{eme> {[ema je deo baze podataka ~iji je vlasnik navedeni korisnik. Jedna kompletna baza podataka }e se sastojati od vi{e {ema}.

```
CREATE TABLE ime-tabele
      (definicija-atributa [,definicija-atributa]...);
gde "definicija atributa ima oblik
    ime-kolone tip-podatka [NOT NULL]
```

{Ovom operacijom kreira se "prazna" tabela sa datim imenom i imenima i tipom kolona. Podaci se kasnije mogu uneti sa INSERT naredbom. Na primer,

```
CREATE TABLE RADNIK
     (RADN# CHAR(7) NOT NULL,
     MLB CHAR(13),
     IME VARCHAR(20),
     STAROST SMALLINT,
     MESTO_RO\ VARCHAR(20),
     ORGJED# CHAR(4) );     }
```

SQL podr`ava slede}e tipove podataka:

INTEGER	ozna~eni binarni ceo broj (31 bit).
SMALLINT	ozna~eni binarni ceo broj (15 bitova).
DECIMAL	(p [,q]) ozna~eni pakovani decimalni broj sa p cifara, sa pretpostavljenom decimalnom ta~kom q cifara od desna ulevo (difolt vrednost za q je 0).
FLOAT	ozna~eni dvostruki (dve re~i) broj u teku}em zarezu
CHAR(n)	niz karaktera fiksne du`ine n.
VARCHAR(n)	niz karaktera promenljive du`ine, sa maksimalnom du`inom n.

```
ALTER TABLE ime-tabele
        ADD ime-kolone tip-podatka;
```

{ Dodavanje nove kolone, kao poslednje, u tabelu. Na primer,
ALTER TABLE ORGANIZACIJA
ADD VRSTA CHAR(5); }

DROP TABLE ime-tabele;

{ Data tabela, svi indeksi i pogledi definisani nad njom se izbacuju iz baze podataka}

CREATE [UNIQUE] INDEX naziv-indeksa
ON naziv-tabele (ime-kolone [redosled]
[, ime-kolone [redosled]] ...)

{ Nad jednom tabelom se mo`e definisati vi{e indeksa. Opcija UNIQUE specificira da dve n-torke u tabeli ne mogu imati istu vrednost indeksiranog atributa (ili indeksiranih atributa). U mnogim implementacijama SQL-a definisanje jedinstvenog indeksa nad jednim ili grupom atributa je jedini na~in da se specificira primarni klju~ relacije. Opcija za redosled mo`e da bude ASC (rastu}i) ili DSC (opadaju}i) - ASC je difolt.

Primeri:

CREATE UNIQUE INDEX IRAD ON RADNIK (RADN#)
CREATE INDEX ST ON RADNIK (STAROST DESC)
CREATE INDEX X ON RASPORED (PROJ#, RADN#, DANARADA DESC)

{Kreira se indeks (B-stablo) sa nazivom X nad slo`enim atributom PROJ#, RADN#, DANARADA koji ure|uje pristup po opadaju}im vrednostima DANARADA u okviru rastu}ih vrednosti RADN#, u okviru rastu}ih vrednosti PROJ#}.

Neki kreirani indeks se mo`e uni{titi sa naredbom

DROP INDEX ime-indeksa;

(2) OPERACIJE (DATA MANIPULATION)

(2.1) Operacije za pretra`ivanje (izve{tavanje)

Tipi~ni SQL upitni blok ima slede}i oblik:

SELECT A1, A2, ..., An
FROM R1, R2, ..., Rm
WHERE P;

- A1, A2, ..., An su atributi relacija koji se `ele u rezultatu. O~igledno je da SELECT odgovara operaciji projekcije u relacionoj algebri.
- R1, R2, ..., Rm su relacije koje se koriste u pretra`ivanju, pa je iskaz FROM ekvivalentan operaciji Dekartovog proizvoda u relacionoj algebri.
- P je predikat koga zadovoljavaju selektovane n-torke u rezultatu, pa je iskaz WHERE ekvivalentan operaciji selekcije u relacionoj algebri.

To zna~i da je tipi~ni upitni blok SQL jezika ekvivalentan ranije datom op{tem izrazu za realizaciju upita u relacionoj algebri:

P[A1, A2, ..., An] (S[P] (R1 x R2 x ... x Rm)).

Isto je tako o~igledno da se mo`e pokazati i ekvivalencija op{teg iskaza upita u relacionom ra~unu n torki sa tipi~nim SQL upitnim blokom,

t ∈ R1, u ∈ R2, ..., v ∈ Rn
t. A1, u. A2, ..., v. An GDE_JE P

~ime se posredno pokazuje i ekvivalentnost relacionog ra~una i relacione algebre. Ponovo napominjemo da ekvivalentnost SQL upitnog bloka sa op{tim izrazom za realizaciju upita u relacionoj algebri ne zna~i da se SQL upit implementira na taj na~in.

Tipi~nom SQL bloku mogu se dodati jo{ mnogi detalji. Umesto definisanja op{te sintakse ovoga bloka, prikaza}emo nekoliko karakteristi~nih primera.

Upiti nad jednom relacijom

(1) Prika`i {ifre radnika koji rade na projektima.

```
SELECT RADN#
FROM RASPORED;
```

U ovom upitu nema WHERE klauzule (selekcije), pa ovaj upit realizuje neku vrstu projekcije. Rezultat upita je

RADN#
r1
r1
r1
r2
r3
r3
r5
r7
r7
r8
r8
r8
r9
r10
r10
r10
r10
r10

Rezultat nije prava projekcija. Ako se `eli prava projekcija upit izgleda:

```
SELECT DISTINCT RADN#
FROM RASPORED;
```

RADN#
r1
r2
r3
r5
r7
r8
r9
r10

Prava projekcija, odnosno eliminisanje duplikata je skupa operacija pa se preporu~uje samo onda kada je to stvarno neophodno.

(2) Prika`i mati~ne brojeve i imena radnika starijih od 25 godina koji nisu ro|eni u Beogradu.

```
SELECT MLB, IME
FROM RADNIK
WHERE STAROST > 25 AND MESTO_RO\ ~ = ' Beograd' ;
```

MLB	IME
1010943..	Pera
0203955..	Mira

Ako se `ele prikazati svi atributi neke relacije, umesto njihovog navo|enja mo`e se napisati SELECT *. Isto tako, po{to u razli~itim relacijama postoje atributi sa istim imenom, ime atributa se mo`e (mora kada imena mogu da dovedu do zabune) kvlifikovati imenom relacije u obliku "ime_relacije ime_atributa" (na primer, RADNIK.RADN# ili RASPORED.RADN#). Mo`e se koristiti i iskaz ime_relacije.* (RADNIK.*)

(3) Redosled n-torki u rezultatu nije poznat. Ako se `eli odre|eni redosled u rezultatu upotrebljava se klauzula

```
ORDERED BY ime_kolone [redosled] [ ,ime_kolone [redosled]] ...
```

Prika`i {ifre i imena radnika iz Beograda starih izme|u 22 i 30 godina, po opadaju}em redosledu starosti.

```
SELECT RADNIK. RADN#, RADNIK. IME
FROM RADNIK
WHERE MESTO_RO\ = ' Beograd' AND
STAROST BETWEEN 22 AND 30
ORDERED BY STAROST DESC;
```

(4) Klauzula LIKE omogu}uje pretra`ivanje vrednosti atributa kao niza karaktera sa samo delimi~no zadatom vredno{}u. Pri tome se koriste oznake:

```
% predstavlja string od 0 ili vi{e karaktera i
_  predstavlja poziciju karaktera.
```

Prika`i imena radnika koja po~inju sa M.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE  'M%';
```

Prika`i imena radnika koja imaju tre}i karakter R.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE  '_R%';
```

(5) Testiranje nula vrednosti se vr{i sa klauzulama IS NULL i IS NOT NULL.

Prika`i {ifre radnika ~ija su imena nepoznata.

```
SELECT RADN#
FROM RADNIK
WHERE IME IS NULL;
```

Izrazi i ugra|ene funkcije

Pored prikazivanja vrednosti atributa koje su zapam}ene u bazi podataka, SQL omogu}uje sra~unavanja preko aritmeti~kih izraza i u njega su ugra|ene funkcije za dobijanje sumarnih informacija, neke aritmeti~ke funkcije, funkcije nad nizovima karaktera i NULL funkcija (koja privremeno menja nula vrednost sa nekom vredno{}u koju sami odredimo).

(6) Prika`i imena i trenutnu starost (u junu mesecu) u mesecima radnika iz Zagreba.

```
SELECT IME ,'Trenutna starost meseci', STAROS'T * 12 + 6
FROM RADNIK
WHERE MESTO_RO\ = ' Zagreb';
```

Rezultat ima oblik

IME		
Sima	Trenutna starost meseci	294
Aca	Trenutna starost meseci	306

Funkcije za dobijanje sumarnih informacija su:

AVG(atribut) - izra~unava srednju vrednost datog atributa,
SUM(atribut) - izra~unava sumu svih vrednosti atributa,
MIN(atribut) - nalazi minimalnu vrednost atributa,
MAX(atribut) - nalazi najve}u vrednost atributa,
COUNT(*) - nalazi broj n-torki u relaciji (grupi),
COUNT(atribut) - nalazi broj n-torki sa ne nula vrednostima atributa i
COUNT(DISTINCT atribut) - nalazi broj n-torki sa razli~itim vrednostima atributa.

(7) Prika`i ukupan i srednji broj dana rada radnika na projektu pr1.

```
SELECT SUM(DANARADA), AVG(DANARADA)
FROM RASPORED
WHERE PROJ# = 'pr1';
```

Rezultat ima oblik:

SUM(DANARADA)	AVG(DANARADA)
183	20.33

Aritmeti~ke funkcije (stepenovanje, kvadratni koren, apsolutna vrednost i druge) i funkcije nad nizovima karaktera (konkatenacija, odre|ivanje du`ine niza i druge) ne}e se ovde detaljnije prikazivati.

GROUP BY klauzula logi~ki deli relaciju na grupe n-torki tako da unutar jedne grupe sve n-torke imaju istu vrednost zadatog atributa. Time se omogu}uje da se funkcije za dobijanje sumarnih informacija primene na svaku ovakvu grupu posebno, umesto na celu relaciju.

(8) Prika`i srednju starost radnika po organizacionim jedinicama.

```
SELECT ORGJED#, AVG(STAROST)
FROM RADNIK
GROUP BY ORGJED#;
```

Rezultat je oblika

ORGJED#	AVG(STAROST)
oj1	38.8
oj2	31
oj3	25.5

Klauzula HAVING koristi se zajedno sa GROUP BY i za grupu n - torki ima isti efekat kao WHERE klauzula za pojedina~ne n-torke.

(9) Prikazati zadatke koje obavlja vi{e od dva radnika na jednom projektu.

```
SELECT PROJ#, BRZAD, COUNT (*)
FROM RASPORED
GROUP BY PROJ#, BRZAD
HAVING COUNT (*) > 2;
```

Izgled rezultata je:

PROJ#	BRZAD	COUNT(*)
pr1	z1	3
pr2	z1	5
pr1	z3	3
pr2	z2	4

Upiti nad vi{e relacija

Upiti nad vi{e relacija mogu se realizovati bilo tzv. "ulaganjem" jednog upita u drugi ili preko upita spajanja (join queries).

Ulaganje upita mo`e se primeniti kada se u pretra`ivanju koristi vi{e relacija, ali se prikazuju samo atributi jedne relacije. Ulo`eni upit se naziva podupit (subquery).

(10) Prika`i {ifre i imena radnika koji rade u organizacionoj jedinici Razvoj.

```
SELECT RADN#, IME
FROM RADNIK
WHERE ORGJED# =
        (SELECT ORGJED#
        FROM ORGANIZACIJA
        WHERE NAZIVORG = 'Razvoj');
```

{Podupit daje rezultat ORGJED# = oj2, koji se primenjuje u WHERE delu osnovnog upita}.

Ako je rezultat podupita skup vrednosti u WHERE delu osnovnog upita se primenjuje predikat IN.

(11) Prika`i nazive projekata u kojima rade radnici ro|eni u Beogradu.

```
SELECT NAZIVPR
FROM PROJEKAT
WHERE PROJ# IN
            (SELECT PROJ#
             FROM RASPORED
             WHERE RADN# IN
                        (SELECT RADN#
                         FROM RADNIK
```

```
                    WHERE MESTO_RO\ = 'Beograd'));
```

(12) Prika`i nazive projekata i imena radnika koji rade na njima, a ro|eni su u Beogradu. Mada u osnovi potpuno ekvivalentan sa prethodnim, ovaj upit, po{to zahteva prikazivanje atributa iz dve relacije ne mo`e se realizovati ulaganjem upita, ve} samo pomo}u upita spajanja:

```
SELECT RADNIK. IME, PROJEKAT. NAZIVPR
FROM RADNIK, PROJEKAT, RASPORED
WHERE RADNIK.RADN# = RASPORED.RADN#
AND PROJEKAT.PROJ# = RASPORED.PROJ#
AND RADNIK.MESTO_RO\ = 'Beograd' ;
```

Upiti koji se mogu realizovati ulaganjem upita, mogu se realizovati i pomo}u upita spajanja. U sistemima koji imaju dobar optimizator upita, vreme odgovora za upit napisan na jedan ili drugi na~in bi trebalo da bude isto, pa je na~in pisanja upita stvar izbora korisnika.

(13) Spajanje relacije sa samom sobom. Prika`i parove radnika iste starosti.

```
SELECT RAD1.IME, RAD2.IME, RAD1.STAROST
FROM RADNIK RAD1, RADNIK RAD2
WHERE RAD1.STAROST = RAD2.STAROST;
```

{Relacija RADNIK se pojavljuje dva puta u FROM delu upita i svakom od tih pojavljivanja se daje drugo ime (RAD1 i RAD2). Ovi sinonimi se koriste u delovima SELECT i WHERE upita, {to logi~ki omogu}uje da se upit postavlja kroz spajanje dve relacije}.

(2.2) Operacije za odr`avanje baze podataka

Operacije za odr`avanje su UPDATE, DELETE i INSERT. Op{ti oblik naredbe UPDATE je:

```
UPDATE relacija
SET atribut = izraz
      [ , atribut = izraz]
[WHERE predikat];
```

U svim n-torkama u relaciji koje zadovoljavaju predikat menja se vrednost navedenih atributa.

(14) Sve radnike starije od 25 godina rasporediti u organizacionu jedinicu oj3.

```
UPDATE RADNIK
SET ORGJED# = oj3
WHERE STAROST > 25;
```

Op{ti oblik naredbe DELETE je:

```
DELETE
FROM relacija
[WHERE predikat];
```

Izbacuju se sve n-torke koje zadovoljavaju dati predikat.

(15) Iz relacije RASPORED izbaci radnike koji na nekom zadatku rade manje od 15 dana.

```
DELETE
FROM RASPORED
WHERE DANARADA < 15;
```

(16) Izbaci sve radnike koji rade u organizacionim jedinicama iz Zagreba.

```
DELETE
FROM RADNIK
WHERE ORGJED# IN
            (SELECT ORGJED#
             FROM ORGANIZACIJA
             WHERE MESTO_ORG = ' Zagreb' );
```

Op{ti oblik naredbe INSERT je:

```
INSERT
INTO relacija [(atribut [ ,atribut] ...)]
VALUES (konstanta [ ,konstanta] ...);
```

ili

```
INSERT
INTO relacija [(atribut [ ,atribut] ...)]
SELECT ... FROM ... WHERE ... ;
```

(17) Ubaci novu organizacionu jedinicu oj4, Nabavka koja je locirana u Beogradu.

```
INSERT
INTO ORGANIZACIJA (ORGJED#, NAZIVORG, MESTO_ORG)
VALUES ('oj4', 'Nabavka', 'Beograd');
```

(2.3) Formiranje pogleda (Vier)

Pogled (vier) u SQL-u se defini{e kao izvedena relacija, izvedena iz skupa baznih tabela i drugih definisanih pogleda. Pogled je virtuelna relacija, ona fizi~ki ne postoji, ali je korisnik, po pravilu (izuzeci se odnose uglavnom na operacije odr`avanja baze podataka) mo`e koristiti kao bilo koju drugu relaciju u sistemu.

Op{ta sintaksa za kreiranje pogleda je:

```
CREATE VIEW ime-pogleda
[(ime-atributa [ ,ime-atributa] ...)]
  AS SELECT ... WHERE ... FROM ... ;
```

Pogled se uni{tava sa iskazom:

```
DROP VIEW ime-pogleda;
```

Primeri:

(18)
```
CREATE VIEW STARI_RAD
AS  SELECT *
      FROM RADNIK
      WHERE STAROST > 50;
```

{Ako se ne navedu imena atributa pogleda on nasle|uje ime atributa iz relacija od kojih se formira, na o~igledan na~in}.

(19) Podskup radnika, okarakterisan sa njihovim {iframa i imenima koji su anga`ovani na projektima ukupno vi{e od 50 dana, mo`emo nazvati DOBRI_RADN i kreirati odgovaraju}u tabelu - pogled:

```
CREATE VIEW DOBRI_RADN ([IFDR, IMEDR)
AS   SELECT RADN#, IME
     FROM RADNIK
     WHERE RADN# IN
               (SELECT RADN#
                FROM RASPORED
                GROUP BY RADN#
                HAVING SUM(DANARADA) > 50);
```

Naredbom CREATE VIEW ne izvr{ava se upit preko koga je pogled definisan, ve} se odgovaraju}i iskaz sme{ta u katalog baze podataka. Tek kada se nad pogledom izvr{ava neka operacija, tada se ta operacija izvodi zajedno sa operacijama preko kojih se pogled kreira, a koje su zapam}ene u katalogu.

Upit sa kojim se pogled kreira mo`e se formirati na isti na~in (sa istim klauzulama) kao i bilo koji upit. Izuzetak, o~igledno ~ini klauzula ORDERED BY, jer pogled defini{e relaciju, a u relaciji redosled n-torki nije bitan. Neki komercijalni sistemi obi~no dodaju jo{ neka ograni~enja.

Sa ta~ke gledi{ta pretra`ivanja (izve{tavanja), pogled se tretira kao bilo koja druga relacija i nad njim se mogu izvr{avati bilo kakvi upiti. Na primer,

(20) Izlistaj imena starijih dobrih radnika iz organizacione jedinice oj2.

```
SELECT IMEDR
FROM DOBRI_RADN
WHERE SIFDR IN
 (SELECT RADN#
  FROM STARI_RAD
  WHERE ORGJED# = 'oj2' );
```

O~igledno je da se, najop{tije gledano, nad pogledima ne mogu vr{iti operacije a`uriranja baze podataka, jer pogled fizi~ki ne postoji, a a`uriranje se mo`e preneti na relacije od kojih je pogled sa~injen samo u izuzetnim slu~ajevima (kada je pogled izveden iz jedne relacije, ili kada se a`uriranje vr{i samo nad jednom od relacija koje ~ine pogled i kada je prenos a`uriranja iz pogleda na baznu relaciju nedvosmislen). Neki komercijalni sistemi ne dozvoljavaju a`uriranje pogleda uop{te, a neki dozvoljavaju u ta~no navedenim posebnim slu~ajevima.

Upotreba i zna~aj pogleda u razvoju aplikacija nad relacionom bazom podataka je vi{estruka:

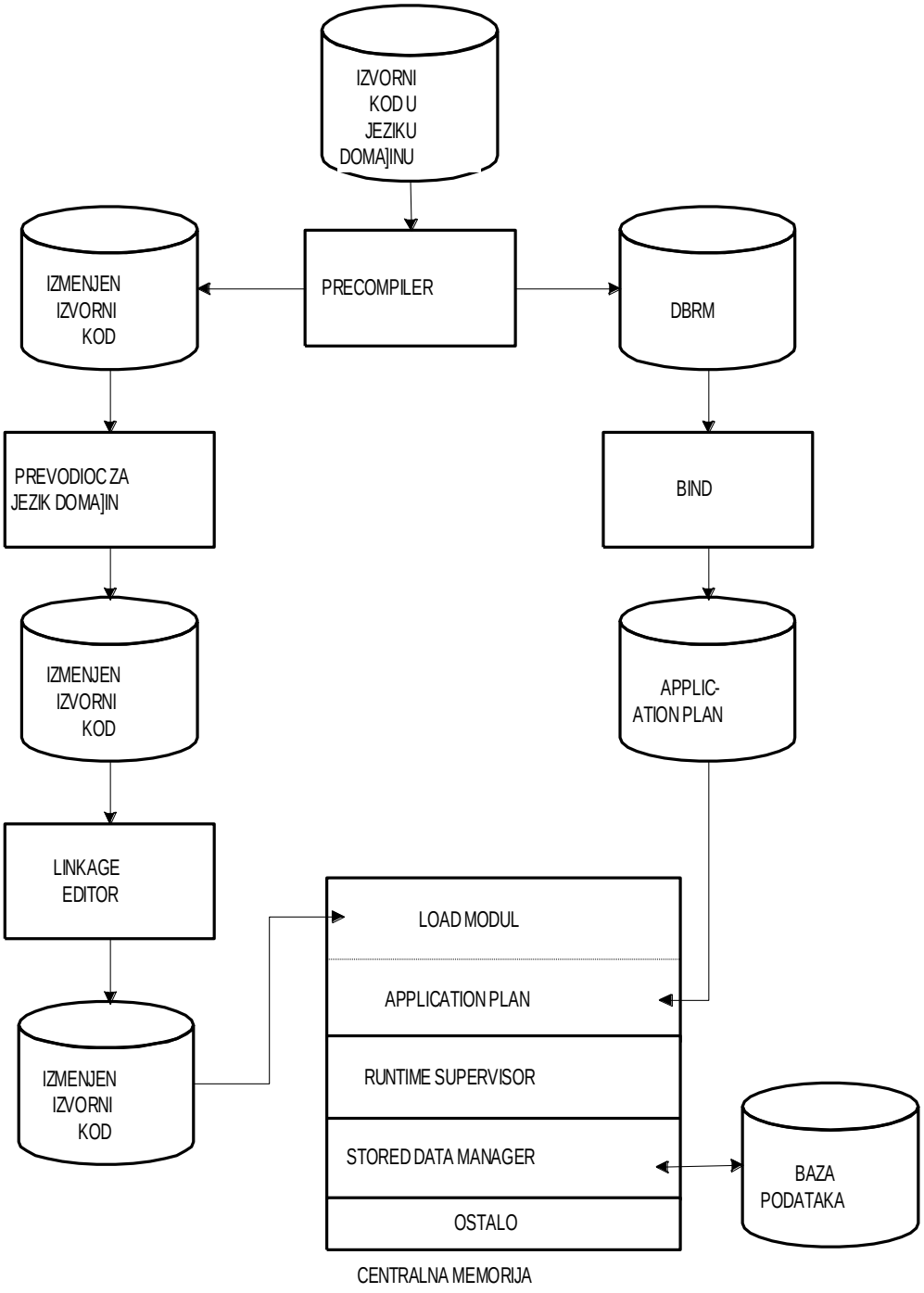
(a) Ostvarivanje logi~ke nezavisnosti programa i podataka. Da bi se ostvarila nezavisnost programa od logi~ke strukture baze podataka, programe treba razvijati nad pogledima, a ne nad baznim relacijama. Ovo je posebno zna~ajno u prototipskom razvoju, kada treba formirati poglede ~ak i potpuno ekvivalentne baznim relacijama. Na taj na~in aplikacioni programi se ne}e menjati, ~ak i ako se jedna bazna relacija "vertikalno" dekomponuje na dve, ili se izvr{i spajanje relacija, ili izmena naziva atributa i sli~no. Jedino {to je, tada, neophodno uraditi je redefinisanje odgovaraju}eg pogleda.

(b) Pogledi omogu}uju da se isti podaci, razli~itim korisnicima, prika`u na razli~ite na~ine, upravo onako kako oni `ele da ih vide. Time se zna~ajno mo`e pojednostaviti kori{}enje podataka iz baze podataka.

(c) Pogledi omogu}uju i jedan stepen za{tite podataka, jer se neki podaci u baznim tabelama, daju}i korisniku na raspolaganje samo poglede, mogu sakriti.

(2.4) SQL unutar standardnih programskih jezika (Embedded SQL)

Kao {to je re~eno SQL je homogeno relacioni jezik, jezik koji se mo`e koristiti i za interaktivnu komunikaciju korisnika sa BP i unutar standardnih programskih jezika (COBOL, PL/I i C naj~e{}e), za razvoj aplikativnih programa. U ovom drugom slu~aju, naredbe programskog jezika koji se koristi, tzv. "jezika doma}ina" i SQL su "pome{ane". [ematski prikaz procesiranja jednog ovakvog programa dat je na Slici 1 (za IBM-ov sistem DB2).



Slika 1. Izvr{enje aplikativnog programa u DB2

Osnovne komponente ovog sistema su:

Precompiler predprocesor koji iz aplikacionog programa "izvla~i" SQL naredbe i sme{ta ih u DBRM (Data Base Request) modul, zamenjuju}i ih sa pozivima Run Time Supervisor-u.

Bind komponenta prevodi DBRM u tzv. action plan, ma{inski kod za implementaciju SQL naredbi uklju~uju}i i poziv Stored Data Manager-u.

Run Time Supervisor nadgleda izvr{enje SQL naredbi, prenose}i poziv za izvr{enje neke naredbe od predprocesora u aplication plan, koji onda poziva Stored Data Manager da izvr{i odgovaraju}u funkciju.

Stored Data Manager, upavlja fizi~kom bazom podataka, memori{u}i i pretra`uju}i podatke pozivaju}i komponente ni`eg nivoa koje vr{e baferovanje, zaklju~avanje, sortiranje i sli~no.

Na~in pisanja aplikacionog programa sa SQL-om, objasni}emo na primeru COBOL-a kao jezika doma}ina.

Da bi predprocesor razlikovao SQL naredbe od naredbi jezika doma}ina, po~etak i kraj svake SQL naredbe ozna~en je na slede}i na~in:

```
EXEC  SQL  naredba      END-EXEC
```

Samo jedna SQL naredba mo`e stajati izme|u ovih klju~nih re~i.

SQL naredbe se koriste u deklarativnom (DATA DIVISION) i u proceduralnom (PROCEDURE DIVISION) delu programa. U deklarativnom delu programa defini{u se sve tabele (bazne i pogledi) koji se koriste u programu preko naredbi jezika doma}ina i na taj na~in se obezbe|uje komunikacija aplikacionog programa sa bazom podataka. Pored toga, u deklarativni deo programa prenose se i sistemski podaci, specijalne promenljive (registri) preko kojih se u program prenose statusi izvr{enja pojedinih SQL naredbi nad bazom podataka (korektno izvr{enje ili kod neke gre{ke).

Na primer,

```
. . . . . .
. . . . . .
DATA DIVISION.
FILE SECTION.
* Defini{u se datoteke van baze podataka koje se koriste u programu.

WORKING-STORAGE SECTION.
. . . . . .
. . . . . .
EXEC SQL BEGIN DECLARATION SECTION END-EXEC.
01 REKRADN.
      05 [RADN              PICTURE X(7).
      05 MATBR              PICTURE X(13).
      05 IMER               PICTURE X(20).
      05 STAR               PICTURE 99.
      05 MESTOR             PICTURE X(20).
      05 RADIU              PICTURE X(4).
EXEC SQL END DECLARE SECTION END-EXEC.
EXEC SQL INCLUDE SQLCA  END-EXEC.
```

U IBM-ovom relacionom sistemu DB2 SQLCA je skra}enica za SQL Communication Area u kojoj se nalaze promenljive sa statusima izvr{enja SQL naredbi. To su, na primer:

- SQLCODE koja ima vrednost nula ako je operacija nad bazom uspe{no izvr{ena, pozitivnu vrednost ako operacija nije izvr{ena, a pojavila se neka normalna situacija (kraj podataka) i negativnu vrednost ako je do{lo do neke gre{ke.
- SQLERRM sadr`i poruku ako je SQLCODE manji od nule. Naredba INCLUDE SQLCA stavlja aplikacionom programu na raspolaganje ove sistemske promenljive.

U proceduralnom delu programa osnovni problem u usagla{avanju jezika doma}ina sa SQL-om le`i u tome {to SQL operi{e nad tabelama, a jezik doma}in nad rekordima (vrstama tabele). U slu~ajevima kada je rezultat upita SELECT ... FROM ... WHERE ... samo jedna n-torka (vrsta), kao i u slu~ajevima naredbi UPDATE, DELETE (osim tzv. CURRENT oblika) i INSERT, komunikacija SQL i programskog jezika je direktna. Na primer,

```
EXEC SQL
      SELECT IME. STAROST
      INTO :IMER, :STAR
      FROM RADNIK
      WHERE RADN# = ' r3'
END-EXEC.

MOVE 'r4' TO [RADN.
MOVE 20 TO STAR.
MOVE ' Ub' TO MESTO.
EXEC SQL
UPDATE RADNIK
     SET STAROST = :STAR, MESTO_ RO\ = :MESTO
     WHERE RADN# = :[RADN
END-EXEC.
```

Da bi se razlikovale promenljive baze podataka i promenljive programa, u SQL upitima nazivima promenljivih programa prethode dve ta~ke (:).

U slu~ajevima kada upitni blok SELECT ... FROM ... WHERE ... daje vi{e n-torki kao odgovor, neophodno je definisati, za svaki takav upit koji se u programu koristi, jedan specijalni "bafer" koji se naziva CURSOR, u koga }e se smestiti rezultat takvog upita i iz koga }e biti mogu}e preuzimati n-torku po n-torku, kako to jezik doma}in zahteva. CURSOR se defini{e na slede}i na~in:

```
EXEC SQL DECLARE ime-kursora CURSOR
      FOR select-naredba

END-EXEC.
```

Na primer,

```
EXEC SQL
     DECLARE XYZ CURSOR
        FOR SELECT RADN#, IME, STAROST
        FROM RADNIK
        WHERE MESTO_RO\ = :MESTO
     END EXEC.
```

Deklaracija kursora nije izvr{na naredba, mada se pojavljuje u proceduralnom delu programa (u PROCEDURE DIVISION). Izvr{ne operacije nad kursorom su:

- EXEC SQL OPEN ime-kursora END-EXEC.

  Ova naredba "aktivira" kursor, odnosno izvr{ava upit preko koga je kursor definisan i sme{ta rezultat u odgovaraju}i "bafer". Kursor ima i ulogu pokaziva~a i posle OPEN naredbe pokazuje na polo`aj ispred prve n-torke u "baferu".

- EXEC SQL FETCH ime-kursora INTO ime-programske-prom END-EXEC.

  Ova naredba pomera kursor koji je "otvoren" za jedno mesto unapred i pridru`uje vrednosti atributa promenljivim programa. Kada kursor dospe do kraja "bafera" vrednost sistemske promenljive SQLCODE postaje +100.

- EXEC SQL. CLOSE ime-kursora END-EXEC.

  Ova naredba zatvara kursor koji se kasnije mo`e opet otvoriti, da bi se, na primer isti upit ponovo izvr{io sa novim teku}im vrednostima programskih promenljivih.

Primer:

```
PROCEDURE DIVISION.
. . . . . .
. . . . . .
. . . . . .

          EXEC SQL
            DECLARE XYZ CURSOR
                FOR SELECT RADN#, IME, STAROST
                FROM RADNIK
                WHERE MESTO_RO\ = : MESTO
            END EXEC.
. . . . . .
. . . . . .
          MOVE ' Beograd' TO MESTO.
          EXEC SQL OPEN XYZ END-XEC.
          EXEC SQL
                FETCH XYZ INTO   : [RADN,   : IMER,   : STAR
          END-EXEC.
. . . . . .
. . . . . .
. . . . . .
          EXEC SQL CLOSE XYZ END-EXEC.
          MOVE 'Zagreb' TO MESTO.
          EXEC SQL OPEN XYZ END-EXEC.
          EXEC SQL
                FETCH XYZ INTO   : [RADN,   : IMER,   : STAR
          END-EXEC.
```

Ovde su dati samo osnovni koncepti kori{}enja SQL-a u okviru jezika doma}ina za izradu aplikacionih programa. Mada su principi ovakve implementacije isti za sve sisteme i jezike doma}ine, ipak postoji i niz detalja vezanih za konkretan sistem i jezik. Zna~ajan aspekt u aplikacionim programima, kontrolne (upravlja~ke) naredbe SQL (CQMMIT i ROLLBACK), bi}e obja{njeni docnije, posle diskusije problema konkurentne obrade i oporavka baze podataka.

2.4.2. QUEL (Query Language)

QUEL je upitni jezik koji predstavlja direktnu implementaciju relacionog ra~una n-torki. QUEL je upitni jezik u okviru sistema za upravljanje bazom podataka INGRES, koji se, u po~etku, razvijao na univerzitetu Berkli u SAD, a kasnije u okviru firme Relational Technology. I QUEL je homogeno relacioni jezik. Umesto detaljnijeg prikaza, QUEL }e ovde biti ilustrovan sa nekoliko primera uz obja{njenja.

Naredbe za definisanje podataka su: CREATE (za kreiranje baznih tabela), INDEX (za kreiranje indeksa), DEFINE VIEW (za definisanje pogleda) DESTROY (za brisanje bazne tabele, pogleda ili indeksa) i MODIFY (za promenu strukture bazne tabele ili indeksa).

```
CREATE RADNIK
        (RADN# = TEXT(7)
         MLB = TEXT(13)
         IME = TEXT(20)
         STAROST = I2
         MESTO_RO\ = TEXT(20)
         ORGJED# = TEXT(4))
```

QUEL podr`ava slede}e tipove podataka: I1, I2, I4 (celi brojevi od 1, 2 i 4 bajta, respektivno); F4, F8 (brojevi u teku}em zarezu od 4 i 8 bajtova, respektivno); MONEY (dolari i centi); TEXT(n) (niz karaktera sa najvi{e n bajtova); DATE (datum).

Promenljive u naredbama za rukovanje sa podacima (Data Manipulation) se deklari{u sa:

```
RANGE OF naziv-promenljive IS naziv-relacije
```

~ime se iskazuje da neka promenljiva uzima za svoje vrednosti n-torke navedene relacije. Ako se RANGE klauzula ne koristi, QUEL pretpostavlja da je naziv promenljive jednak nazivu relacije.

Primeri upita:

(1) Prikazati {ifre radnika koji rade na projektima

```
RANGE OF x IS RADNIK
RETRIEVE UNIQUE (x. RADN#)
```

(3) Prika`i {ifre i imena radnika is Beograda starih izme|u 22 i 30 godina, po opadaju}em redosledu starosti.

```
RANGE OF x IS RADNIK
RETRIEVE (x.RADN#, x.IME)
WHERE x.STAROST >= 22 AND x.STAROST <= 30
AND x.MESTO_RO\ = 'Beograd'
SORTED BY STAROST:DECCENDING
```

(7) Prika`i srednji i ukupan broj dana rada radnika na projektu pr1.

```
RANGE OF y IS RASPORED
RETRIEVE (Y = SUM(y.DANARADA), Z = AVG(y.DANARADA))
WHERE y.PROJ# = 'pr1'
```

(12) Prika`i nazive projekata i imena radnika koji rade na njima a ro|eni su u Beogradu.

```
RANGE OF x IS RADNIK
RANGE OF y IS PROJEKAT
```

```
RANGE OF z IS RASPORED
RETRIEVE (z.NAZIVPR, x.IME)
WHERE x.RADN# = z.RADN#
AND y.PROJ# = z.PROJ
AND x.MESTO_RO\ = 'Beograd'
```

(13) Prika`i parove radnika iste starosti.

```
RANGE OF x IS RADNIK
RANGE OF y IS RADNIK
RETRIEVE (x.IME, y.IME, x.STAROST)
WHERE x.STAROST = y.STAROST
```

(14) Sve radnike starije od 25 godina rasporediti u oranizacionu jedinicu oj3.

```
REPLACE RADNIK (ORGJED# = oj3)
WHERE RADNIK. STAROST > 25
```

(16) Izbaci sve radnike koji rade u organizacionim jedinicama iz Zagreba.

```
DELETE RADNIK
WHERE RADNIK.ORGJED# = ORGANIZACIJA.ORGJED#
AND ORGANIZACIJA.MESTO_ORG ='Zagreb'
```

(17) Ubaci novu organizacionu jedinicu oj4, Nabavka, koja je locirana u Beogradu.

```
APEND TO ORGANIZACIJA (ORGJED# = 'oj4'
        NAZIVORG = 'Nabavka', MESTO_ORG = 'Beograd')
```

(18) Definisanje pogleda. Kreiraj pogled DOBRI_RADN sa atributima {ifra radnika i ime radnika za radnike koji su na projektima anga`ovani ukupno vi{e od 50 dana.

```
DEFINE VIEW DOBRI_RADN
   (RADNIK.RADN#, RADNIK.IME)
    WHERE SUM(RASPORED.DANARADA BY RASPORED, RADN#) > 50
```

QUEL uklju~en u jezik doma}in (COBOL, FORTRAN, BASIC, Pascal i C) koji se naziva EQUEL primenjuje se mnogo jednostavnije nego SQL. Razlog za to je {to se SQL naredbe u jeziku doma}inu prevode pre izvr{enja programa, a EQUEL naredbe tek u toku izvr{enja programa (interpretacija). Osim toga, EQUEL nema koncept analogan kursoru u SQL-u, ve} se uvek pretpostavlja da operacija RETRIEVE daje vi{e n-torki koje se obra|uju jedna po jedna u okviru koncepta nazvanog RETRIEVE loop (petlja). Primer jednog programa u FORTRANU sledi.

```
##   DECLARE
##   CHARACTER * 7       X
##   CHARACTER * 20      Y
##   INTEGER Z
##   CHARACTER * 20      V
     V = 'Beograd'
##   RETRIEVE (X=RADNIK. RADN#, Y=RADNIK. IME, Z=RADNIK. STAROST)
##   WHERE RADNIK.MESTO_RO\ = 'Beograd'
##   {
     Fortranske naredbe za obradu X, Y i Z
##   {
```

EQUEL naredbe se razlikuju od naredbi jezika doma}ina tako {to u prve dve kolone imaju znake ##. DECLARE naredba je potrebna da bi se definisale promenljive za vezu programskog jezika i baze podataka. Pretpostavlja se da se RETRIEVE naredba izvr{ava tako {to se kod izme|u velikih zagrada izvr{ava jednom za svaku n-torku koja je rezultat pretra`ivanja. Kada se sve n-torke obrade kontrola se prenosi na prvu naredbu iza velike zagrade. Za "prevremeno iskakanje" iz petlje postoji naredba ENDRETRIEVE.

Kako je EQUEL u osnovi interpreter, u slu~aju da se jedan upit izvr{ava u petlji, prevodio bi se i optimizirao, nepotrebno, onoliko puta koliko se puta petlja izvr{ava. Klauzula REPEAT uz upit informi{e optimizator da zapamti "query plan", koji bi se ponovo koristio kada se pojavi isti upit docnije.

```
##  REPEAT REPLACE ORGANIZACIJA (MESTO_ORG = @X)
##  WHERE ORGANIZACIJA.ORGJED# = @Y
```

Oznaka @ uz promenljive jezika doma}ina X i Y kazuje da ih treba tretirati kao parametre, odnosno svaki put kada se naredba izvr{ava one imaju "teku}u vrednost".

2.4.3. QBE (Query By Example)

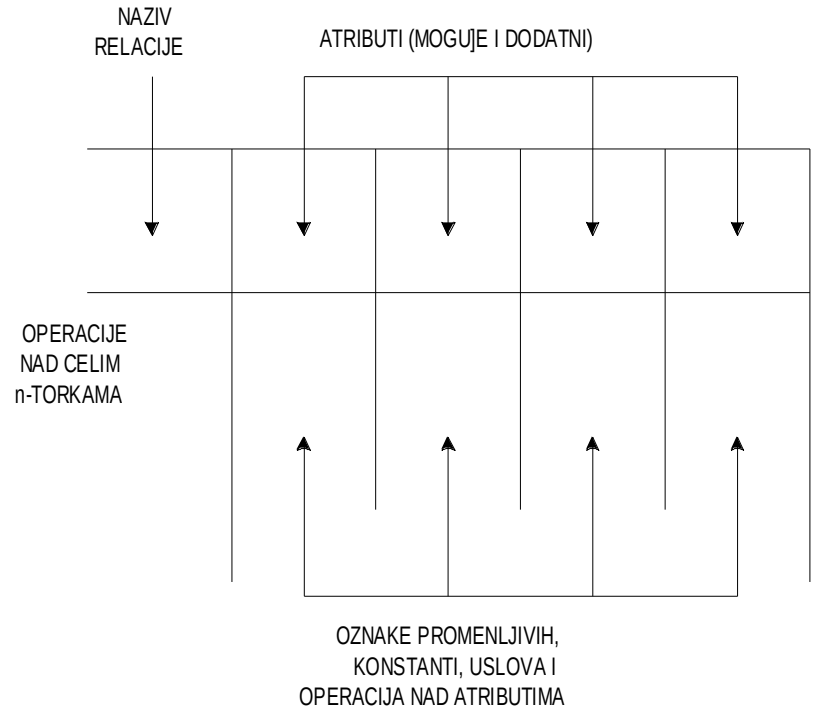
QBE je upitni jezik koji je razvio IBM i koji predstavlja implementaciju relacionog ra~una domena, preko specifi~ne "dvodimenzione" sintakse, koja je vrlo bliska korisniku, jer se preko nje, direktno, u skeletu tabele predstavljene na ekranu, zadaje "primer odgovora" koji korisnik `eli. Otuda i ime Query By Example (upit na osnovu primera).

Postupak rada je slede}i:

Pritiskom odre|ene tipke na tastaturi dobija se na ekranu skelet tabele predstavljene na Slici 2.

Ako se u rubriku za ime relacije unese "naziv-relacije P" u toj rubrici }e se pojaviti naziv relacije koji je unet, a u rubrikama za atribute nazivi atributa te relacije. Na primer, na Slici 3 je u rubriku za naziv relacije uneto RADNIK P., a dobijena su zaglavlja svih kolona. U kolonu ispod naziva relacije upisuju se operacije koje se odnose na celu n-torku. Na primer, na Slici 4 kod operacije P. (Print) ozna~ava prikazivanje svih atributa n-torki relacije RADNIK. Kodovi ostalih operacija su I. (ubacivanje), D. (izbacivanje) i U. (a`uriranje).

Ako se operacija ne odnosi na celu n-torku ve} samo na neke njene atribute, kôd odgovaraju}e operacije se unosi u kolonu datog atributa. U kolone atributa se, pored koda operacije unose i uslovi pretra`ivanja, promenljive i konstante.



Slika 2. Skelet tabela za QBE

Na Slici 3 prikazan je upit

(2) Prika`i mati~ni li~ni broj i ime radnika ro|enih u Beogradu i starijih od 25 godina.

Iskaz ovoga upita u relacionom ra~unu domena je:

x, y GDE_JE ∃ z > 25 AND RADNIK (MLB: x, IME: y, STAROST: z,
MESTO_RO\: 'Beograd')

O~igledna je analogija ovako iskazanog upita i QBE sintakse. U QBE egzistencijalni kvantifikator je implicitan. Uslovi iskazani u jednom redu tabele implicitno su povezani sa AND. Uslovi koje treba da zadovolje pojedini atributi upisuju se u odgovaraju}e kolone. Sa operatorom P. nazna~avaju se atributi koji se prikazuju.

RADNIK	RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
		P.	P.	> 25	Beograd	

Slika 3. Upit QBE 2

Slika 4 predstavlja QBE realizaciju upita

(3) Prika`i sve atribute radnika koji su bilo ro|eni u Zagrebu ili su stari 40 ili manje godina.

Uslovi koji su povezani sa OR upisuju se u razli~itim redovima tabele.

RADNIK	RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
P.				<= 40		

P.					Zagreb	

Slika 4. Upit QBE 3

Drugi na~in iskazivanja uslova selekcije je u posebnom bloku koji se naziva "condition box", gde se mo`e iskazati uslov (predikat) bilo koje slo`enosti, kako je to na Slici 5 i prikazano. Za ovakvo iskazivanje potrebno je uvesti i koncept promenljive. Promenljive se ozna~avaju sa prvim znakom "_" (na primer _ST ili _MR) i uzimaju vrednost iz domena odgovaraju}ih atributa.

RADNIK	RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
		P.	P.	_ST	_MR	

CONDITIONS
ST>25 AND MR=Beograd

Slika 5. Druga realizacija upita QBE 1

Preko promenljivih se vr{i i povezivanje tabela za upite nad vi{e relacija (upite spajanja). Na Slici 6 prikazana je implementacija upita.

RADNIK	RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
	_RAD		P.		Beograd	

PROJEKAT	PROJ#	NAZIVPR	MESTO
	_PR	P.	

RASPORED	RADN#	PROJ#	BRZAD	DANARADA
	_RAD	_PROJ		

Slika 6. Upit QBE 12

(12) Prika`i nazive projekata i imena radnika koji rade na njima, a ro|eni su u Beogradu.

Na slici 7 prikazan je i na~in ubacivanja nove n-torke u relaciju. O~igledno je kako bi se formulisalo izbacivanje n-torki i a`uriranje vrednosti atributa.

ORGANIZACIJA	ORGJED#	NAZIVORG	MESTO_ORG
1.	oj4	Prodaja	Beograd

Slika 7. Dodavanje n-torke

Na slici 8 prikazan je na~in kreiranja relacija u QBE. Oznakom KEY (Y ili N) ozna~ava se da li je atribut deo klju~a relacije ili ne. Tipovi podataka mogu da budu: CHAR (niz karaktera promenljive du`ine), CHAR(n) (niz karaktera fiksne du`ine n), FLOAT (realni broj) i FIXED (ceo broj). Klju~na re~ DOMAIN defini{e domen atributa. Uslov spajanja po atributima se ne}e izvr{iti ako atributi nisu definisani nad istim domenima. Klju~na re~ INVERSON (Y ili N) ozna~ava da li je nad atributom dignut indeks ili ne.

Na Slici 9 prikazan je na~in formiranja pogleda STARI_RAD, skup radnika starijih od 50 godina. Kako se u pogled prenose svi atributi bazne relacije, svi atributi su promenljive. (Promenljive su _123, _45, _ZORAN i ostale).

I. ORGANIZACIJA I.		ORGJED#	NAZIVORG	MESTO_ORG
KEY	I.	Y	N	N
TYPE	I.	CHAR	CHAR	CHAR
DOMAIN	I.	SIFREOJ	NAZIVIOJ	MESTA
INVERSION	I.	Y	N	N

Slika 8. Kreiranje relacije u QBE

I.VIEW STARI_RAD I.	RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
I.	_123	_456	_ZORAN	_54	_Ub	_oj

RADNIK	RADN#	MLB	IME	STAROST	MESTO_RO\	ORGJED#
	_123	_456	_ZORAN	_54 > 50	_Ub	_oj

Slika 9. Kreiranje pogleda STARI_RAD

2.5. Oporavak baze podataka (RECOVERY)

Oporavak baze podataka je vra}anje baze podataka u stanje za koje se zna da je korektno, posle nekog softverskog ili hardverskog otkaza sistema. Postoje mnogi razli~iti uzroci otkaza sistema, od gre{ke u programiranju, preko gre{ki u operativnom sistemu i samom SUBP-u, padanja glava diska, nestanka napajanja, do sabota`e.

Osnovni princip oporavka baze podataka je redundansa podataka. Postoje mnogi na~ini redundantnog pam}enja podataka za potrebe oporavka baze podataka. Tehnike redundantnog pam}enja podataka i oporavka baze su veoma kompleksne. Jednostavne procedure koje bi se bazirale samo na, na primer, periodi~nom kopiranju baze podataka u neku arhivsku memoriju i svih transakcija koje su se u me|uvremenu desile ili jednostavno dupliciranje baze podataka, imaju mnogo nedostataka.

Da bi se objasnili osnovni koncepti, proces oporavka baze podataka upro{}eno }emo opisati na slede}i na~in:

1. Periodi~no se (jednom dnevno, na primer) cela baza podataka kopira (dump) na neku arhivsku memoriju (traku).
2. Za svaku promenu u bazi, u tzv. log (ili `urnal) zapisuje se stara (before image) i nova vrednost (after image) rekorda baze podataka (bloka).
3. Posle otkaza sistema:
   (a) Ako je sama baza podataka o{te}ena, ona se rekonstrui{e (oporavlja) na osnovu svoje poslednje arhivske kopije koriste}i log za ponovno izvr{enje svih promena (redo) koje su izvr{ene od trenutka uzimanja arhivske kopije, do trenutka otkaza.
   (b) Sama baza podataka nije o{te}ena, ve} je samo dovedena u neko nekonzistentno stanje. Tada se, pomo}u loga, poni{te sve nekorektne promene (undo), a same transakcije koje su ih proizvele, ponove.

Transakcije. Osnovni cilj baze podataka je da omogu}i efikasnu obradu transakcija. Transakcija je jedno izvr{enje neke "logi~ke jedinice rada" korisnika baze podataka, jedno izvr{enje neke logi~ke

celine jednog programa, ili jedno izvr{enje celog programa. Vi{e izvr{enja jednog istog programa mogu se u sistemu odvijati kokurentno, svako od njih predstavlja transakciju.

Skup aktivnosti nad bazom podataka koje se izvr{avaju po principu "sve ili ni{ta" - ili su sve uspe{no obavljene ili je baza podataka ostala nepromenjena, je atomski skup aktivnosti. O~igledno je da "logi~ka jedinica rada" mora predstavljati atomski skup aktivnosti, "atomsku transakciju".

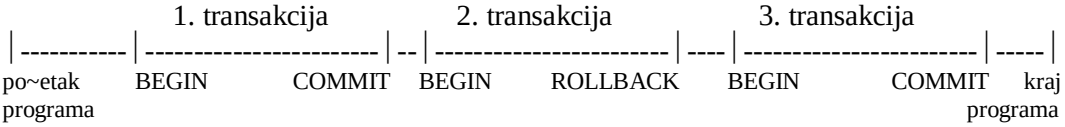
Imaju}i u vidu ovakvu definiciju transakcije, mo`e se tako|e re}i da ona predstavlja izvr{enje jednog skupa operacija nekog programa, koji (skup) po~inje sa specijalnom operacijom BEGIN TRANSACTION, a zavr{ava se bilo sa operacijom COMMIT (ako je ceo skup operacija uspe{no izvr{en) ili sa ROLLBACK tako ceo skup operacija nije uspe{no zavr{en). Jedan program mo`e predstavljati sekvencu transakcija, mada u praksi, ~esto, jedan program predstavlja jednu transakciju. Transakcije ne mogu biti "ugnje`dene", odnosno u jednom programu operacija BEGIN TRANSACTION mo`e se izvr{iti (zadati) samo ako taj program nema ni jednu drugu transakciju u izvr{enju, odnosno naredbe COMMIT i ROLLBACK se mogu izvr{iti samo ako je neka transakcija u izvr{enju.

Primer: Data je relaciona {ema
DOBAVLJ(SIFDOB, ...)
ISPORUKA(SIFDOB,SIFPRO, ...)

Sa~initi program za izmenu SIFDOB

```
EXEC SQL. WHENEVER SQLERROR GO TO UNDO;
GET LIST (SX,SY); /* korisnik daje vrednosti SX i SY*/
EXEC SQL UPDATE DOBAVLJ
                SET SIFDOB = SY
                WHERE SIFDOB = SX;
EXEC SQL UPDATE ISPORUKA
                SET SIFDOB = SY
                WHERE SIFDOB = SX;
EXEC SQL. COMMIT;
GO TO KRAJ
UNDO: EXEC SQL ROLLBACK;
KRAJ: RETURN;
END;
```

----------> vreme



(BEGIN mo`e biti implicitno, po~etak programa otpo~inje novu transakciju, ili to ~ine COMMIT ili ROLLBACK).

U izvr{enju transakcije nije samo problem o~uvanje integriteta baze podataka nego i neke poruke korisniku baze, odnosno tzv. "stvarni izlazi" mogu postati nekorektni. Na primer, vlasnik ra~una u banci komunicira sa svojim zapisom u bazi podataka preko terminala za izdavanje novca. U tom slu~aju transakcija mora da bude atomska i u slede}em smislu: ili je pla}anje izvr{eno i a`uriranje odgovaraju}eg zapisa u~injeno ili ni jedno ni drugo. Upravljanje porukama }emo ilustrovati na slede}em primeru (program za prebacivanje novca sa ra~una na ra~un). Program je napisan u jednoj vrsti pseudokoda, koji se lako prilago|ava bilo kom SUBP i bilo kom jeziku doma}inu (Date).

PRENOS: PROC;

```
        GET(RA^1, RA^2, IZNOS); /*ulazna poruka*"/
        FIND UNIQUE (BROJRA^ WHERE BROJRA^ = RA^1);
        ASSIGN (STANJE - IZNOS) TO STANJE;
        IF STANJE < 0
        THEN
                DO;
                    PUT ('NEMA DOSTA LOVE'); /*izlazna poruka*/
                    ROLLBACK;
                END;
        ELSE
                DO;
                    FIND UNIQUE (BROJRA^ WHERE BROJRA^ = RA^2);
                    ASSIGN (STANJE + IZNOS) TO STANJE;
                    PUT ('LOVA PRENE[ENA'); /*izlazna poruka*/
                    COMMIT;
                END;
END: /*PRENOS*/
```

O~igledno je da se izlazne poruke ne smeju prenositi direktno, ve} tek nakon planiranog zavr{etka transakcije (sa COMMIT ili ROLLBACK). Postupak upravljanja porukama u oporavku baze podataka (za {ta je zadu`en tzv. Data Communication Manager) je slede}i: Poruke se ne prihvataju direktno ve} se ulazne poruke stavljaju u tzv. ulazni red, a izlazne poruke u izlazni red. Operacija GET uzima ulaznu poruku iz ulaznog reda, a operacija PUT stavlja izlaznu poruku u izlazni red. Pri planiranom zavr{etku transakcije (COMMIT ili eksplicitni ROLLBACK) izvr{ava se:

(a) upisuju se na log izlazne poruke,
(b) stvarno se prenose izlazne poruke i
(c) izbacuju se ulazne poruke iz ulaznog reda.

Pri neplaniranom ROLLBACK (koji se izvr{ava automatski ako do|e do neke gre{ke u programu tipa "overflow", deljenje sa nulom i sli~no), prouzrokuje izbacivanje izlaznih poruka iz izlaznog reda.

2.5.1. Vrste otkaza

(1) Naru{avanje integriteta u nekoj transakciji koje program sam otkriva, odnosno planirani ROLLBACK (na primer 'NEMA DOSTA LOVE').
(2) Lokalna gre{ka u nekoj transakciji koja nije eksplicitno planirana u programu (na primer aritmeti~ki overflow).
(3) Sistemski otkaz koji uti~e na sve transakcije u obradi, ali ne o{te}uje bazu podataka (na primer nestanak napajanja).
(4) Fizi~ko o{te}enje baze podataka.

Lokalni otkazi (otkazi u transakciji)

Za neplanirani zavr{etak transakcije (zavr{etak koji nije na naredbi COMMIT ili eksplicitnom ROLLBACK) neophodno je da se automatski izvr{i rollback. Rollback proceduru obavlja Recovery Manager, poni{tavaju}i sve promene unazad kroz log, dok se ne dostigne u logu rekord koji ozna~ava po~etak transakcije (koriste se before image). Za vreme ovog procesa poni{tavanja, tako|e je mogu}e da do|e do otkaza. U tom slu~aju se rollback procedura mora startovati ponovo i ona zbog toga treba da ima osobinu idempotencije (UNDO(UNDO ... (UNDO(X) = UNDO(X)). O~igledno je, da zbog rollback procedura transakcije treba da budu {to kra}e.

Sistemski otkazi bez o{te}enja baze podataka

Sistemski otkaz prouzrokuje prestanak rada celog sistema i njegovo ponovno pokretanje. O~igledno je da bi se oporavak baze podataka u tom slu~aju mogao da izvr{i pretra`ivanjem celog loga i identifikovanjem tansakcija koje su otpo~ele, a nisu zavr{ene i njihovim ponovnim izvr{enjem. Da se ne bi pretra`ivao ceo log uvode se tzv. checkpoints (ta~ke ispitivanja). To se radi na slede}i na~in:

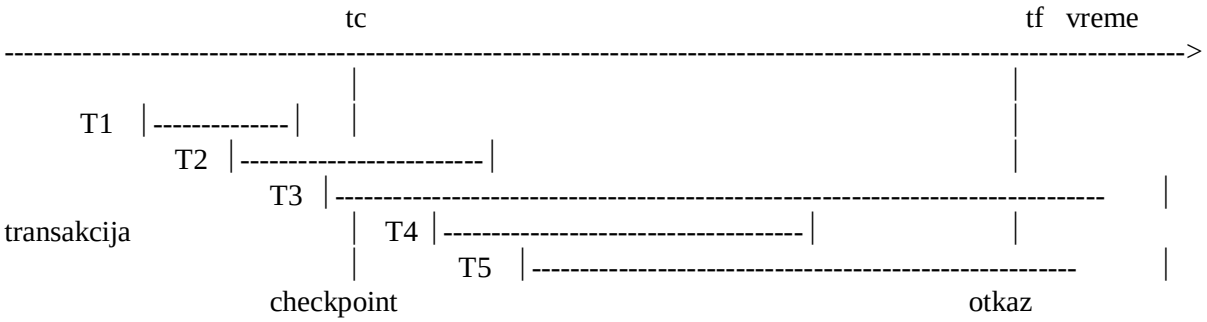
U zadatim intervalima vremena "uzimaju se ta~ke ispitivanja" na taj na~in {to se:

1. Upisuje sadr`aj bafera za log na log.
2. Upisuje "checkpoint record" na log.
3. Upisuje sadr`aj bafera baze podataka u bazu podataka.
4. Upisuje adresa "check point" rekorda na "restart file".

Ispisivanje sadr`aja bafera je forsirano, jer se njihov sadr`aj normalno prenosi tek kada se oni napune.

"Check point record" sadr`i listu svih transakcija koje su aktivne u trenutku uzimanja "checkpoint" i za svaku transakciju adresu rekorda u logu kojem je poslednjem pristupljeno.

Posmatrajmo slede}i primer:



O~igledno je da transakcije T3 i T5 moraju biti poni{tene (undone), a transakcije T2 i T4 ponovljene (redone), jer nije garantovano da nisu njihovi efekti registrovani u bazi podataka (mo`da su jo{ uvek u baferu). Transakcija T1 je obavljena u potpunosti pre uzimanja "checkpoint" i ne ulazi u "checkpoint record". Algoritam oporavka je slede}i:

Formiraju se dve liste - "undo-lista" u kojoj su sve transakcije koje su zapisane u "checkpoint record-u" i "redo-lista" koja je u po~etku prazna. Pretra`ivanje po logu po~inje od "checkpoint record-a" i:

- svaka transakcija za koju se prona|e BEGIN TRANSACTION, stavlja se u "undo-listu" i
- svaka transakcija za koju se prona|e COMMIT stavlja se u"redo-listu".

Zatim, Recovery Manager, unazad, izvr{ava "undo" transakcija, a onda unapred "redo" odgovaraju}ih transakcija.

"Log Write-Ahead". Svaka operacija nad bazom podataka zahteva upisivanje u samu bazu i na log. Kako u intervalu izme|u ova dva upisivanja mo`e da do|e do otkaza, da bi se obezbedio oporavak baze uvek se upisivanje prvo vr{i na log.

Otkaz sekundarne memorije

Ako otka`e neki deo sekundarne memorije, baza podataka se oporavlja na osnovu arhivske kopije i loga, obnavljaju}i sve transakcije od trenutka uzimanja kopije do otkaza. Arhivska kopija se uzima u trenutku kada ni jedna transakcija nije aktivna i vrlo je dugotrajan proces. Ponekad se vr{i tzv.

inkrementni dump, odnosno na kopiju se prenose samo oni zapisi koji su pretrpeli promene posle poslednjeg uzimanja kopije.

Dvofazni COMMIT (TWO-PHASE COMMIT) protokol

Na~in izvr{avanja COMMIT operacije, da se prvo promene upi{u na log, a tek potom u samu bazu podataka (log write-ahead) se komplikuje u slu~ajevima kada ceo sistem koristi ne samo DBMS nego i neke druge resurse (na primer dva DMBS-a ili u distribuiranim sistemima). Ako svaki resurs ima svoj nezavisni mehanizam oporavka, postavlja se pitanje kako se izvr{ava COMMIT operacija, jer obavljanje ove operacije na jednom resursu ne zna~i da }e se ona uspe{no obaviti i na drugom, {to mo`e da naru{i integritet baze podataka. U ovakvim slu~ajevima se uvodi nova sistemska komponenta koja se naziva koordinator.

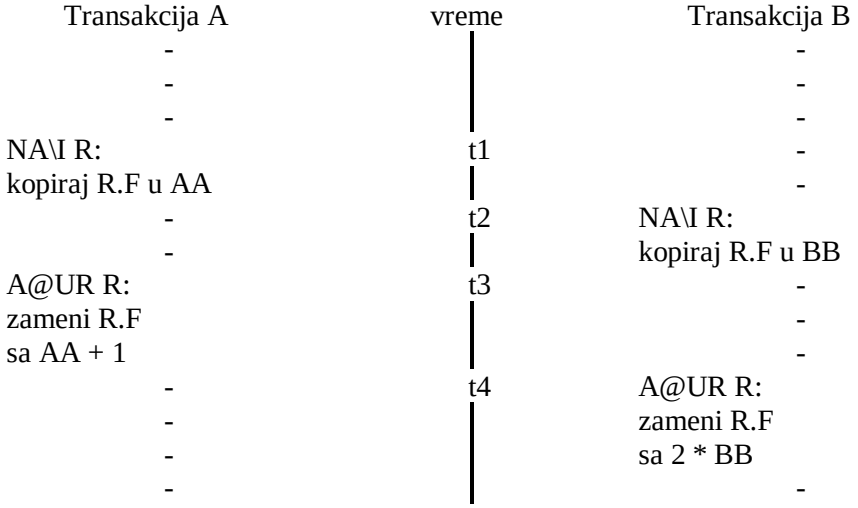
U ovoj vrsti protokola, transakcije prenose COMMIT koordinatoru a ovaj postupa na slede}i na~in:

Faza 1: Koordinator zahteva od svih participanata (resursa) da do|u u stanje u kome bilo da izvr{e COMMIT bilo ROLLBACK. To zna~i da participanti treba da upi{u sve odgovaraju}e zapise na svoj lokalni log. Ako su to uspe{no uradili, participanti {alju koordinatoru poruku "OK" (ili, obrnuto "NOT OK").

Faza 2: Ako je odgovor "OK" koordinator oda{ilje naredbu COMMIT svim participantima i svi oni zavr{avaju svoj lokalni COMMIT. U obrnutom slu~aju na isti na~in se postupa sa ROLLBACK. Na taj na~in transakcija ne mo`e biti potvr|ena (COMMIT) od strane nekog resursa, a poni{tena (ROLLBACK) od strane nekog drugog.

2.6. Konkurentna (uporedna) obrada transakcija

Transakcija se u sistemima zasnovanim na bazi podataka ne obavlja u izolaciji, ve} konkurentno (uporedno) sa drugim transakcijama u sistemu (vi{e transakcija mogu istovremeno zahtevati iste resurse, isti rekord baze podataka). Nekontrolisana me|usobna interferencija transakcija mo`e da dovede bazu u nekonzistentno stanje. Posmatrajmo izvr{enje slede}e dve transakcije



Serijsko izvr{enje transakcija je izr{enje u kome se transakcije ne prepli}u, ve} se prvo potpuno izvr{i jedna, pa onda druga transakcija (A pa B, ili P pa A). Rezultat serijskog izvr{enja transakcija T1, T2, ..., Tn u bilo kom redosledu }emo smatrati korektnim.

Ako je vrednost R.F bila 1 tada:

(a) Serijsko A,B proizvodi novu vrednost R.F = 4.
(b) Serijsko B,A proizvodi novu vrednost R.F = 3.
(c) Uporedno izvr{enje, prikazano na gornjoj slici proizvodi novu neta~nu vrednost R.F = 2, neta~nu zbog toga {to nije jednaka vrednosti koju proizvodi neko serijsko izvr{enje transakcija.

Da bi se spre~ilo ovakvo dinami~ko naru{avanje integriteta baze podataka mo`e se postupiti na jedan od slede}a tri na~ina:

(a) Zabranjuje se izvr{enje naredbe NA\I R u transakciji B zato {to je transakcija A ve} adresirala taj rekord (ekskluzivno zaklju~avanje - ekskluzivni lokot E-lokot).

(b) Zabranjuje se izvr{enje naredbe A@UR R u transakciji A zato {to je transakcija B ve} adresirala ("videla") rekord R (deljivi lokot - shared locks).

(c) Zabranjuje se transakciji B u trenutku t4 da a`urira rekord R zato {to ga je "mla|a" transakcija A ve} a`urirala (Vremensko ozna~avanje transakcija - timestamping. Transakcija A je "mla|a" je otpo~ela prva i dobila je manju vremensku oznaku)

Ekskluzivni lokoti (E-lokot)

Neka transakcija T mo`e da zaklju~a (da dobije lokot) rekord R prenose}i takav zahtev na komponentu sistema koja se naziva Lock Manager. Lokot sadr`i, izme|u ostalog, identifikaciju rekorda R i identifikaciju transakcije A.

Ako transakcija A postavi ekskluzivni lokot na neki objekt BP, tada ni jedna druga transakcija ne mo`e postaviti lokot na isti objekt, dok ga transakcija A ne oslobodi.

Slede}i PX protokol omogu}uje de se re{i navedeni problem a`uriranja:

PX protokol. Bilo koja transakcija koja `eli da a`urira neki rekord BP mora prvo izvr{iti ENA\I operaciju kojom se adresira i zaklju~ava dati rekord. Ako se zaklju~avanje ne mo`e da izvr{i, transakcija prelazi u stanje ~ekanja, dok se odgovaraju}i zapis ne oslobodi, kada se procesiranje nastavlja.

Da bi se izbegla situacija da neka transakcija ~eka zauvek, zbog toga {to druge preuzimaju lokot na odgovaraju}em objektu ("`ivi lokot") uvodi se neki redosled zaklju~avanja objekta (na primer FIFO).

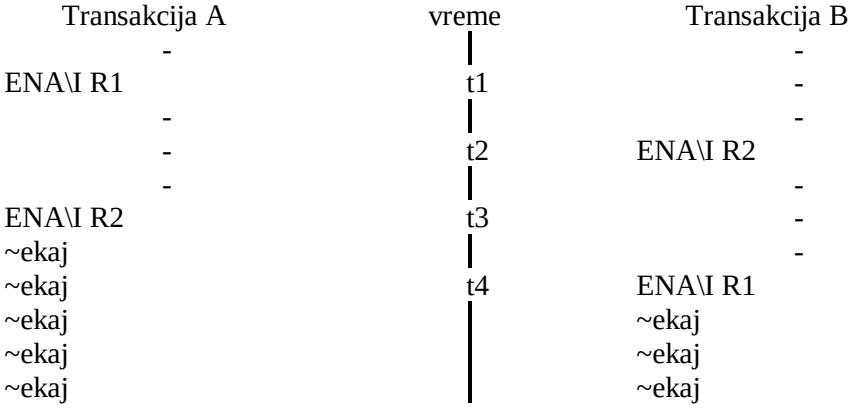
Transakcija A	vreme	Transakcija B
-	|	-
-	|	-
-	|	-
ENA\I R:	t1	-
kopiraj R.F u AA	|	-
-	t2	ENA\I R:
-	|	~ekaj
A@UR R:	t3	~ekaj
zameni R.F	|	~ekaj
sa AA + 1	|	~ekaj
[EOSLOB R]	t4	~ekaj
-	t5	(nastavi) ENA\I R:
-	|	kopiraj R.F u BB
-	t6	A@UR R:
	|	zameni R.F sa
		2 * BB
	t7	[EOSLOB R]

Operacija EOSLOB se ne zadaje eksplicitno ve} se izvodi obi~no zajedno sa operacijom COMMIT.

Serijabilnost ili linearnost izvo|enja skupa transakcija

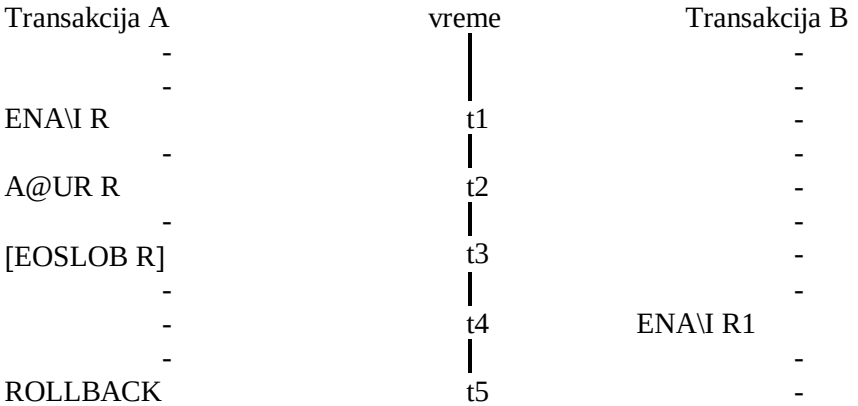
Uporedno izvr{avanje skupa transakcija je serijabilno (linearno) ako proizvodi isti rezultat kao i neko serijsko izvr{enje istog skupa transakcija. Usvaja se da je skup uporednih transakcija izvr{en korektno, ako i samo ako je taj skup serijabilan.

Mrtvi lokoti ("samrtni~ki zagrljaj"). Mrtvi lokot je ilustrovan slede}im na~inom izvr{enja dve transakcije:



Problemi "mrtvih lokota" se re{avaju bilo nametanjem protokola obrade transakcija koji treba da spre~e da do njih do|e, ili se dozvoljava da do njih do|e, pa se tada neka od transakcija koja ga je izazvala "ubije", njeni efekti na bazu podataka poni{te, a ona sama ponovo startuje, nadaju}i se da tada ne}e ponovo izazvati mrtvi lokot. Za otkrivanje mrtvih lokota u sistemu koristi se tzv. "Wait-For" graf ~iji su ~vorovi transakcije T u izvr{enju, a grana Ti - Tj postoji ako transakcija Ti zahteva neki objekat koji je transakcija Tj zaklju~ala. Ispitivanje mrtvih lokota se mo`e vr{iti bilo svaki put kada neka transakcija prelazi u stanje "~ekaj", bilo periodi~no, ili se mo`e smatrati da je mrtvi lokot nastao ako transakcija ni{ta nije uradila u nekom periodu vremena. Kriterijumi za izbor "`rtve" mogu biti razli~iti, na primer, transakcija koja je poslednja startovala ili transakcija koja zahteva najmanje lokota. "Ubijanje" transakcije i poni{tavanje njenih efekata (ROLLBACK) osloba|a i sve lokote koje je ona postavila.

Mogu}nost da se transakcija poni{ti (ROLLBACK) pre njenog uspe{nog zavr{etka, mo`e da dovede do slede}eg problema:



O~igledno je da transakcija B radi na poni{tenoj promeni rekorda R, na "prljavim" podacima. Jo{ je gori slu~aj ako bi transakcija B zahtevala ekskluzivni lokot na R (ENA\I R), pa ga zatim a`urirala (A@UR R), jer bi se time "prljavi" podatak trajno uneo u bazu podataka. Da bi se ovaj problem izbegao defini{e se PXC protokol koji na ranije definisani PX protokol dodaje jo{ jedno pravilo:

Ekskluzivni lokot se zadr`ava do kraja transakcije (do COMMIT ili ROLLBACK).

Me|utim PXC protokol je veoma strog protokol i mo`e, ponekad nepotrebno, izazvati ~ekanja transakcija. Zbog toga se uvodi koncept deljivih lokota (D-lokot).

Transakcija A postavlja deljivi lokot na neki rekord baze podataka, ako se dozvoljava da druga transakcija tako|e postavi deljivi lokot na isti rekord, ali se ne dozvoljava da druga transakcija postavi ekskluzivni lokot na taj objekat.

Pravila zaklju~avanja daju se kroz slede}u "matricu kompatibilnosti":

	E	D	-
E	N	N	Y
D	N	Y	Y
-	Y	Y	Y

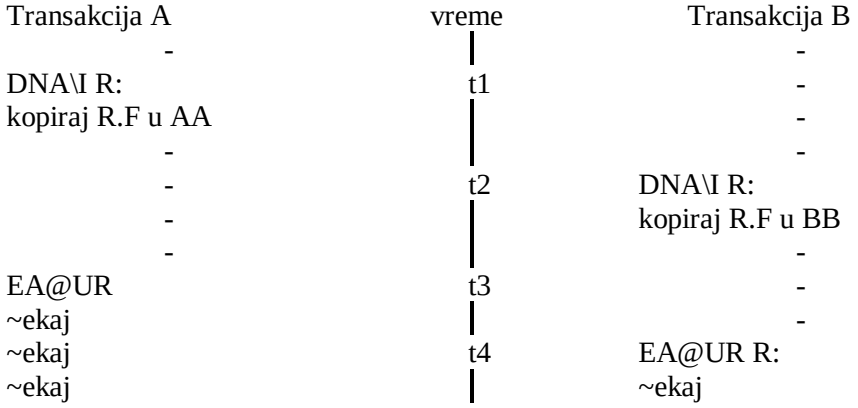
Sistemi koji podr`avaju D-lokote, umesto da zahtevaju E-lokote pri adresiranju rekorda koji }e se a`urirati (ENA\I operacija) zahtevaju samo postavljanje D-lokota (DNA\I-operacija), a D-lokot postaje E-lokot tek kada stvarno a`uriranje nastupi. (Mo`e se definisati operacija EA@UR da bi se ovaj efekat objasnio, mada je normalno postavljanje E-lokota sastavni deo operacija a`uriranja.)

Na bazi D-lokota mo`e se definisati i slede}i protokol zaklju~avanja:

PSX protokol. Svaka transakcija koja namerava da a`urira neki objekat baze podataka prvo izvr{ava operaciju DNA\I, kojom se adresira objekat i postavlja na njega D-lokot. Posle dobijanja D-lokota, svako stvarno a`uriranje }e ne samo a`urirati objekat, nego }e D-lokot prevesti u E-lokot. E-lokoti se osloba|aju na kraju transakcije (sa COMMIT ili ROLLBACK).

Iz ovog protokola sledi da se D-lokot mo`e osloboditi i pre zavr{etka transakcije, pa je u sistemima koji podr`avaju ovakav protokol data i eksplicitna naredba DOSLOB (za razliku od ranije uvedene naredbe EOSLOB koja je slu`ila samo za obja{njenja i koja eksplicitno ne postoji, ve} je njena semantika ugra|ena u COMMIT i ROLLBACK).

Me|utim, PSX protokol pove}ava broj mrtvih lokota u sistemu, {to se vidi iz slede}eg primera:



~ekaj | ~ekaj

Da bi se smanjio broj mrtvih lokota u nekim sistemima uvodi se i lokot a`uriranja (A-lokot). Ako transakcija namerava da a`urira neki rekord ona ga adresira i na njega postavlja A-lakot.

Protokol PUX. Svaka transakcija koja namerava da a`urira neki rekord, operacijom AA@UR ga adresira i na njega postavlja A-lokot. Posle dobijanja A-lokota, svako stvarno a`uriranje prevodi A-lokot u E-lokot. E-lokoti se osloba|aju po zavr{etku transakcije (sa COMMIT ili ROLLBACK).

Matrica kompatibilnosti E, D i S lokota je:

	E	U	S	-
E	N	N	N	Y
U	N	N	Y	Y
S	N	Y	Y	Y
-	Y	Y	Y	Y

O~igledno je da u prethodnom primeru protokol PUX ne}e dovesti do mrtvog lokota.

Razli~iti SUBP koriste razne vrste protokola. SYSTEM R (prete~a IBM-ovog relacionog sistema DB2) koristi PSC protokol, hijerarhijski SUBP IBM-a, IMS koristi PUC protokol. U DBTG sistemima programer mo`e da bira izme|u PSC i PXC protokola.

Dvofazno zaklju~avanje (2PL)

Protokoli PSC i PUC koji dozvoljavaju osloba|anje deljivih lokota i lokota a`uriranja pre zavr{etka transakcije ne garantuju serijabilnost transakcija. Zbog toga se defini{e Dvofazni protokol zaklju~avanja (2PL) koji garantuje serijabilnost.

Dvofazni protokol zaklju~avanja. Pre bilo koje operacije nad nekim objektom transakcija zahteva lokot nad tim objektom. Posle osloba|anja nekog lokota transakcija vi{e ne zahteva nijedan lokot.

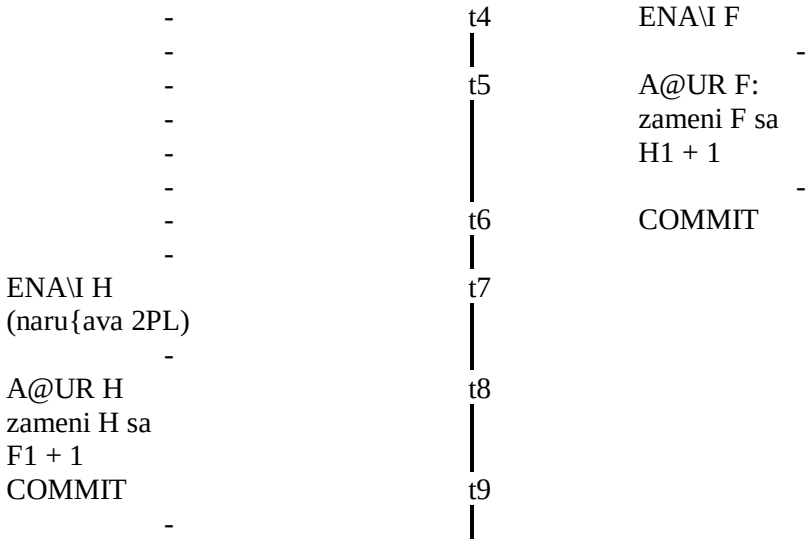
Posmatrajmo dve transakcije:

(A) H := F + 1; (B) F := H + 1;

Neka je u po~etku H = 0 i F = 0.

Serijsko izvr{enje "A pa B" }e rezultovati u H = 1, F = 2, a serijsko izvr{enje "B pa A" u H = 2, F = 1. Uporedno izvr{avanje ovih transakcija, na na~in koji je prikazan na slede}oj slici, koji ne po{tuje dvofazni protokol }e dati nekorektan rezultat F = 1 i H = 1, ~ime se pokazuje da takvo izvr{enje nije serijabilno.

Transakcija A	vreme	Transakcija B
-	|	-
DNA\I F:	t1	-
kopiraj F u F1	|	-
-	|	-
DOSLOB F	t2	-
-	|	-
-	t3	DNA\I H:
-	|	kopiraj H u H1
-	|	-



IZBEGAVANJE MRTVIH LOKOTA

Kao {to je re~eno DBMS mogu bilo dozvoljavati mrtve lokote, pa ih zatim razre{avati, ili ih pomo}u razli~itih tehnika obrade transakcija mogu izbegavati. U centralizovanim sistemima tehnike izbegavanja mrtvih lokota su skuplje nego tehnike njihovog razre{avanja, za razliku od distribuiranih sistema. Neki protokoli izbegavanja mrtvih lokota koje se primenjuju u operativnim sistemima ovde nisu prakti~no primenljive. Na primer, tehnika da se defini{e neki poredak svih objekata zaklju~avanja, pa da se ne dozvoli da transakcija T zaklju~a objekat A posle objekta B, ako je objekat A pre objekta B u nametnutom redosledu, ovde nije primenljiva zbog toga {to je skup objekata zaklju~avanja u BP veoma velik, oni se ne adresiraju samo po imenu, ve} i po sadr`aju.

Planiranje transakcija (Transaction Scheduling). Transakcije se planiraju tako da one transakcije koje zahtevaju iste podatke se ne izvr{avaju paralelno. To zahteva da se podaci kojima transakcije pristupaju znaju unapred, {to se mo`e ostvariti bilo eksplicitnim navo|enjem, bilo analizom programa za vreme prevo|enja. Me|utim, o~igledno je nemogu}e precizno definisati zahteve transakcija za podacima pre izvr{enja programa, pa se zbog toga "rezervi{u" ve}i skupovi podataka nego {to je neophodno i time smanjuje paralelizam obrade.

Odbacivanje zahteva. Odbacuje se svako zaklju~avanje koje bi odmah izazvalo mrtvi lokot, odnosno svaki zahtev za lokot koji se ne mo`e odmah da zadovolji. Odgovaraju}a transakcija "~eka malo" i ponavlja zahtev.

Ponovni poku{aj obrade (Transaction Aetry)

- Svaka transakcija dobija vremensku oznaku, tako da se mogu razlikovati "mlada", odnosno "starija" transakcija.
- Kada transakcija A zahteva lokot za neki rekord BP tada A ~eka ako je starija od B ili se poni{tava (ROLLBACK A). (U drugoj verziji, ako je A mla|a od B ona ~eka, a ako nije, ona prekida izvr{enje transakcije B i poni{tava je (ROLLBACK B)).

VREMENSKO OZNA^AVANJE TRANSAKCIJA (TIMESTAMPING)

Svakoj transakciji se dodeljuje jedinstveni identifikator koji se mo`e tretirati kao vreme njenog po~etka. Sva fizi~ka a`uriranja baze podataka se izvr{avaju tek sa COMMIT naredbom. Konflikti u izvr{enju transakcija nastaju kada neka transakcija zahteva da "vidi" neki rekord koga je ve} mla|a transakcija a`urirala, ili kada neka transakcija ho}e da a`urira neki rekord koga je mla|a transakcija ve}

a`urirala ili videla. Konflikti se razre{avaju ponovnim startovanjem starije transakcije. U tom cilju, za rekord baze podataka R, sistem treba da odr`ava dve sinhronizacione ta~ke:

- NMAX, identifikator transakcije koja je poslednja izvr{ila NA\I R operaciju i
- AMAX, identifikator transakcije koja je poslednja izvr{ila operaciju A@UR R (transakcija je zavr{ena po{to se a`uriranje radi sa naredbom COMMIT).

Algoritam izvr{enja transakcije T ~iji je identifikator (vremenska oznaka) t je:

```
NA\I R
  IF t >= AMAX
      THEN /* operacija se prihvata*/
          NMAX := MAX(t, NMAX);
      ELSE /* konflikt*/
          RESTART T;

A@UR R /* u stvari COMMIT*/
  IF t >= AMAX AND t >= NMAX
      THEN /*prihvata se operacjia*/
              UMAX := t;
      ELSE /*konflikt*/
              RESTART T;
```

U ovoj tehnici nema zaklju~avanja objekata BP, integritet je o~uvan, ne pojavljuju se ni `ivi ni mrtvi lokoti.

INTEGRITET BAZE PODATAKA

Termin integritet baze podataka koristi se da ozna~i ta~nost, korektnost ili konzistentnost, odnosno da ozna~i problem za{tite baze podataka od pogre{nog a`uriranja (pogre{nih ulaznih podataka, gre{ki operatera i programera, sistemskih otkaza). Za{tita baze podataka obi~no se tretira sa dva aspekta: (1) integritet - za{tita od slu~ajnog pogre{nog a`uriranja i (2) sigurnost - za{tita od neovla{}enog a`uriranja i kori{}enja podataka.

U prethodnom poglavlju diskutovan je problem naru{avanja integriteta baze podataka do koga mo`e da do|e u konkurentnoj obradi vi{e transakcija. Ovde se, nadalje, pretpostavlja da je BP uspe{no za{ti}ena od ovakvog naru{avanja integriteta, pa se analizira samo problem o~uvanja integriteta pri izvr{avanju jedne transakcije. Savremeni SUBP zadovoljavaju}e podr`avaju o~uvanje integriteta u konkurentnoj obradi, uz pretpostavku da svaka transakcija posebno ne naru{ava integritet, ali pru`aju veoma malu podr{ku o~uvanju integriteta u okviru jedne transakcije.

Pravila integriteta defini{u koje uslove podaci u BP treba da zadovolje, kada se vr{i provera i koje akcije treba preduzeti kada definisani uslovi nisu zadovoljeni.

Pravila integriteta se defini{u za osnovne operacije a`uriranja (dodavanje n-torke, izbacivanje n-torke i izmena vrednosti atributa) i treba da budu podr`ana od samog SUBP-a (njegovog dela koji bi se mogao nazvati podsistem integriteta), odnosno definisana zajedno sa samim relacijama. Pravila integriteta bi trebalo da budu definisana pomo}u nekog jezika visokog nivoa, zatim prevedena i memorisana u re~niku podataka SUBP-a. Nova pravila bi se u svakom trenutku mogla dodati, ukoliko nisu u koliziji sa postoje}im stanjem BP. Retki su SUBP koji podr`avaju integritet na ovaj na~in. U ve}ini se podr{ka prebacuje u aplikacione programe, odnosno ostavlja programerima da, uz eventualnu pomo} specifi~nih mehanizama, koji su mu stavljeni na raspolaganje (generatori aplikacija), sami vode ra~una o integritetu BP.

Obi~no se pravila integriteta dele u dve klase: (1) pravila integriteta domena, koja se defini{u za vrednosti pojedinih atributa nezavisno od vrednosti ostalih atributa u BP i (2) pravila integriteta relacija, koja se odnose na relaciju kao celinu i, na primer, defini{u kada neka n-torka mo`e da se ubaci u relaciju ili kako n-torke jedne relacije zavise od n-torki druge relacije. Za prvu grupu karakteristi~an primer je ranije navedeni integritet entiteta, a za drugu referencijalni integritet.

Pravila integriteta domena se mogu uop{teno definisati tretiranjem domena kao apstraktnog tipa podatka:

domen: ime-domena
skup od tip-podatka
operacije:
proveri-domen-ime-domena
ulaz: vrednost-atributa
izlaz: Boolean (true, false)
post: if predikat then true else false;

(Apstraktni tip podatka se defini{e kao skup vrednosti i skup operacija nad njima. Mo`emo razlikovati specifikaciju i implementaciju apstraktnog tipa podatka. U specifikaciji tipa podatka defini{e se odgovaraju}i skup vrednosti i sintaksa i semantika njegovih operacija, a u implementaciji, struktura podataka i algoritmi preko kojih }e se on implementirati. Sintaksa operacija apstraktnog tipa se ovde defini{e nazivom operacije, ulaznim i izlaznim parametrima operacije, a semantika preko uslova koji moraju biti zadovoljeni pre (pre) i posle izvr{enja operacije)

Za apstraktni tip domen skupa vrednosti je skup odgovaraju}ih tipova podataka koje podr`avaju gotovo svi programski i upitni jezici (ceo broj, realni broj, niz karaktera i drugo). Jedina operacija koja se nad njim defini{e je proveri-domen koja proverava da li dati atribut zadovoljava ograni~enja definisana nad domenom (zadati predikat), daju}i odgovor true ili false.

Na primer, defini{imo domen STAR, nad kojim je definisan atribut STAROST relacije RADNIK, na slede}i na~in:

domen: STAR
skup od INTEGER
opreracije:
proveri-domen-STAR
ulaz: vrednost-atributa
izlaz: Boolean
post: if vrednost-atributa between 15 and 80
then izlaz = true else izlaz = false;

O~igledno je da se u nekom jeziku (sli~nom Pascalu) apstraktni tip domen mo`e implementirati kao funkcija:

```
PROCEDURE proveri-domen-STAR(s:integer):Boolean
BEGIN
   IF s >= 15 AND s <= 80
        THEN RETURN (true)
        ELSE RETURN(false)
   END;
END proveri-domen-STAR;
```

S obzirom da se nad svim domenima mo`e definisati samo jedna operacija, logi~ka funkcija koja se svodi na testiranje zadatog predikata, i da su skupovi vrednosti standardni tipovi podataka, mo`e se za definisanje domena kao apstraktnog tipa podatka koristiti specifi~na, jednostavnija sintaksa:

naziv-domena: tip-podatka (gde-je predikat);

Na primer, domeni za atribute relacije RADNIK i RASPORED mogu se definisati na slede}i na~in:

```
SIFRAD:  CHAR(7)  gde-je  SIFRAD po~inje sa  "r",  i nije nula vrednost;
MATB:  INT gde-je    broj koji ~ine prve dve cifre manji od 31,
                     broj koji ~ine druge dve cifre manji od 12;
IMENA: VARCHAR(20);
STAR:  SMALLINT  gde-je  STAR izme|u 15 i 80;
MESTA:  VARCHAR(20);
ORG:  CHAR(4)  gde-je ORG po~inje sa "o";
SIFPROJ:  CHAR(7)  gde-je  SIFPROJ po~inje sa "pr" i nije nula vrednost;
SIFZAD:  CHAR(4)  gde-je  SIFZAD po~inje sa "z" i nije nula vrednost;
DANI:  SMALLINT  gde-je DANI izme|u 1 i 50;
```

U odgovaraju}em jeziku za opis relacija, atributi bi bili, sada, definisani nad domenima, a ne nad jezi~kim tipovima podataka, kao u postoje}im jezicima za opis podataka. Na primer,

```
relacija:  RADNIK
     struktura:    (RADN# SIFRAD,
                    MLB MATB,
                    IME IMENA,
                    STAROST STAR,
                    MESTO_RO\ MESTA,
                    ORGJED#  ORG)
     primarni-klju~   RADN#;
```

Ovakav opis relacije podrazumeva da svaka operacija ubacivanja nove n-torke poziva proceduru proveri-domen za svaki atribut relacije, a svaka operacija izmene vrednosti nekog atributa, operaciju proveri-domen za odgovaraju}e atribute i da se ove operacije odbiju, uz poruku o gre{ci, ako je rezultat operacije proveri-domen "false".

Poslednji red u opisu relacije podrazumeva da je implicitno definisana operacija nad relacijom proveri-klju~ koja proverava da li neki klju~ ve} postoji u bazi posmatrane relacije i koja se poziva pri svakom ubacivanju nove n torke ili pri izmeni vrednosti primarnog klju~a relacije.

Najzna~ajnija pravila integriteta relacija su referncijalni integriteti. Za svaki spoljni klju~ u relaciji mora se definisati jedno pravilo integriteta koje se sastoji od uslova, trenutka (odnosno mesta u programu) kada se uslov ispituje i akcija koje se preduzimaju kada uslov nije zadovoljen. Uslov integriteta se mo`e iskazati pomo}u slede}e sintakse:

identifikator-integriteta:predikat;

Na primer, za relaciju RADNIK, uslov da svaki radnik radi u nekoj (postoje}oj) organizacionoj jedinici se mo`e iskazati na slede}i na~in:

RADI1: P[ORGJED#]RADNIK $\subseteq$ P[ORGJED#]ORGANIZACIJA

(Projekcija P[ORGJED#]RADNIK daje skup svih {ifara organizacionih jedinica u relaciji RADNIK, a projekcija P[ORGJED#]ORGANIZACIJA daje skup svih {ifara organizacionih jedinica)

Ako za relaciju ORGANIZACIJA postavimo uslov da ne mo`e postojati nijedna organizaciona jedinica u kojoj ne radi barem jedan radnik, tada }e takav uslov integriteta biti predstavljen sa:

ORGI1: P[ORGJED#]ORGANIZACIJA $\subseteq$ P[ORGJED#]RADNIK

O~igledno je da uslovi RADI1 i ORGI1 rezultuju u tzv."uzajamni integritet":

RADI1-ORGI1: P[ORGJED#]RADNIK = P[ORGJED#]ORGANIZACIJA

Uzajamni integritet dveju relacija ukazuje na to da se operacije odr`avanja ove dve relacije moraju obavljati zajedni~ki ("istovremeno"), u okviru jedne atomske transakcije, odnosno da se provera uslova integriteta mo`e izvr{iti samo na kraju transakcije, neposredno pre naredbe COMMIT. U drugim slu~ajevima. kada nije u pitanju uzajamni integritet, provera uslova integriteta se mo`e izvr{iti bilo pre ili posle izvr{enja operacije odr`avanja relacije.

Pored referencijalnog integriteta ponekad, u slu~ajevima redundantnog pam}enja izvodljivih podataka, moraju se definisati i druga pravila integriteta, tzv "statisti~ki integritet". Na primer ako bi u relaciji ZADATAK imali i atribut UKUPNO_DANARADA, koji bi bio jednak zbiru DANARADA iz relacije RASPORED za dati zadatak, tada bi se moralo definisati odgovaraju}e pravilo integriteta. Operacije odr`avanja relacije ZADATAK i RASPORED bi morale biti u jednoj atomskoj transakciji, a provera integriteta na kraju transakcije.

Mogu se definisati i druga pravila integiteta koja vezuju vrednosti atributa jedne ili vi{e relacija, ili, ~ak, vrednost atributa pre i posle operacije odr`avanja (tzv. "prelazno ograni~enje", na primer da nova vrednost atributa STAROST mora da bude ve}a od stare). Treba, me|utim, napomenuti, da dobro projektovana relaciona {ema, relaciona {ema koja predstavlja dobar model realnog sistema, smanjuje broj eksplicitno zadatih pravila integriteta, odnosno da se preprojektovanjem relacione {eme neka slo`ena pravila integriteta mogu svesti na pravila integriteta domena i referencijalni integritet.

Napomenimo jo{, da bi se svi uslovi integriteta mogli ispitivati na kraju transakcije. Me|utim, ovakvo odlo`eno ispitivanje uslova integriteta je "skuplje", jer n-torci koja je u pitanju treba dva puta pristupiti, jednom za izvo|enje operacije a`uriranja, a drugi put za ispitivanje uslova integriteta.

Pored uslova integriteta, mesta njegovog ispitivanja (pre ili posle operacije odr`avanja ili na kraju transakcije), u jednom pravilu integriteta treba definisati i akcije koje se preduzimaju ako uslov integriteta nije zadovoljen. Razlikujemo slede}e mogu}nosti:

RESTRICTED - Ako se uslov ispituje pre operacije odr`avanja i ako nije zadovoljen, operacija se odbija, uz odgovaraju}u poruku. Ako se uslov ispituje posle operacije odr`avanja, i ako nije zadovoljen vr{i se ROLLBACK.

NULLIFIES - Spoljni klju~ dobija nula vrednost, ako je nula vrednost dozvoljena.

DEFAULT - Spoljni klju~ dobija neku difolt vrednost koju treba unapred predvideti u definiciji odgovaraju}eg domena.

CASCADES - Operacija se prenosi na relaciju koju referi{e spoljni klju~, da bi se u njoj izvr{ile promene koje }e zadovoljiti uslov integriteta.

Pravila integriteta treba, kao {to je ranije re~eno, da budu definisana u okviru opisa baze podataka, a ne u okviru aplikacionih programa. Zato, u opisu svake relacije, za svaki spoljnji klju~, treba navesti odgovaraju}e pravilo integriteta, koriste}i, na primer, slede}u sintaksu:

spoljni klju~ (naziv-atributa [,naziv atributa . . .])
referi{e naziv-relacije. (naziv-atributa [,naziv atributa . . .]),
uslov:predikat,
operacija-odr`avanja [mesto-ispitivanja] akcija [,
operacija-odr`avanja [mesto-ispitivanja] akcija, . . .];

operacija-odr`avanja :: DODAJ naziv-relacije
| IZBACI naziv-relacije
| IZMENI naziv-relacije.primarni-klju~

mesto-ispitivanja : : PRE | POSLE

akcija : : RESTRICTED | NULLIFIES | DEFAULT | CASCADES (poruka: tekst)

Ako se za akciju izabere CASCADES, NULLIFIES ili DEFAULT tada se pri nekoj operaciji odr`avanja vr{e operacije nad obe relacije koje vezuje uslov integriteta (ili dve operacije nad istom relacijom, ako je uslov integriteta zadat nad jednom relacijom), u jednoj atomskoj transakciji, {to zna~i da se uslov integriteta proverava neposredno pre COMMIT naredbe. Zbog toga, mesto proveravanja uslova integriteta ima smisla zadavati samo za opciju RESTRICTED i tada mo`e biti PRE ili POSLE.

Na primer, potpuna definicija relacije RADNIK bi bila:

```
relacija: RADNIK
    struktura:   (RADN# SIFRAD,
                  MLB MATB,
                  IME IMENA,
                  STAROST STAR,
                  MESTO_RO\ MESTA,
                  ORGJED# ORG)
    primarni-klju~:RADN#
    spoljni-klju~:   (ORGJED#) referi{e (ORGJED#).ORGANIZACIJA
                     uslov: P[ORGJED#]RADNIK = P[ORGJED#]ORGANIZACIJA
                     IZBACI ORGANIZACIJA CASCADES
                     UBACI ORGANIZACIJA CASCADES
                     IZMENI ORGANIZACIJA.ORGJED# CASCADES;
```

Gornji iskaz pravila integriteta se mo`e interpretirati na slede}i na~in: '

- Ako se iz relacije ORGANIZACIJA izbaci neka n-torka (organizaciona jedinica), iz relacije RADNIK }e se automatski izbaciti sve n-torke sa istom vredno{}u ORGJED#. (svi radnici koji su u njoj radili).
- Po{to, na osnovu uslova integriteta, ne mo`e postojati organizaciona jedinica bez radnika u njoj, prilikom ubacivanja nove organizacione jedinice u relaciju ORGANIZACIJA, prenosi se i na relaciju RADNIK ubacivanje novih radnika ove organizacione jedinice.
- Promena {ifre neke organizacione jedinice prenosi se na promenu odgovaraju}e {ifre u relaciji RADNIK.

Relacija RASPORED bi mogla da se defini{e na slede}i na~in:

```
relacija: RASPORED
    struktura:   (RADN# SIFRAD,
                  PROJ# SIFPROJ,
                  BRZAD SIFZAD,
                  DANARADA DANI)
    primarni-klju~: RADN#, PROJ#, BRZAD
    spoljni-klju~:    (RADN#) referi{e RADNIK.(RADN#),
                      uslov: P[RADN#]RASPORED ⊆ P[RADN#]RADNIK,
      IZBACI RADNIK DEFAULT
      IZMENI RADNIK.RADN# PRE RESTRICTED
    spoljni-klju~:    (PROJ#,BRZAD) referi{e ZADATAK.(PROJ#,BRZAD)
                       uslov: P[PROJ#,BRZAD]RASPORED =
                                    P[PROJ#,BRZAD]ZADATAK,
                       IZBACI ZADATAK CASCADES
```

UBACI ZADATAK CASCADES
IZMENI ZADATAK.PROJ#,BRZAD CASCADES;

Ovakva definicija relacije RASPORED bi zna~ila slede}e:

- Kada se iz relacije RADNIK izbaci neka n-torka (neki radnik napusti organizaciju), automatski se u svim n-torkama relacije RASPORED {ifra odgovar}aju}eg radnika postavlja na difolt vrednost, {to, kasnije, u nekoj aplikacji mo`e da zna~i da odgovaraju}e zadatke treba poveriti drugim radnicima.

- Promena {ifre radnika u relaciji RADNIK se ne mo`e izvr{iti ako taj radnik radi na nekom zadatku nekog projekta (ako se ta vrednost RADN# pojavljuje u relaciji RASPORED), a provera se vr{i pre operacije IZMENI RADNIK.RADN#.

- Ubacivanje novog radnika u relaciju RADNIK nema uticaja na relaciju RASPORED, zbog toga se za operaciju UBACI RADNIK ne defini{u akcije zadovoljavanja integriteta u relaciji RASPORED.

- Operacija IZBACI ZADATAK i IZMENI ZADATAK.PROJ#, BRZAD se automatski prenose na odgovaraju}e operacije u relaciji RASPORED.

- Operacija UBACI ZADATAK se tako|e prenosi na relaciju RASPORED, jer, na osnovu uslova integriteta, ne mo`e postojati neki zadatak nekog projekta ako na njemu niko ne radi (ako ne postoji barem jedna n-torka sa odgovaraju}om vredno{}u PROJ#, BRZAD u relaciji RASPORED).

Ovde definisani jezik za opis relacija je jedan hipoteti~ki jezik, pomo}u koga bi se pravila integriteta definisala tamo gde im je su{tinsko mesto, uz opis relacije i zadovoljavala automatski. Me|utim, savremeni SUBP uglavnom ne podr`avaju jedan ovakav jezik.

SQL standard ne podr`ava definisanje domena na prikazani na~in, ve} samo standardnu proveru unapred zadatih tipova podataka. Podr`an je samo iskaz o nula vrednostima (klauzula NONULL). Specifikacija primarnog klju~a uvodi se posredno, definisanjem jedinstvenog indeksa nad odgovaraju}im atributom, {to nije zadovoljavaju}a podr{ka, pa se zbog toga neki upiti komplikuju, onemogu}avaju se neka mogu}a a`uriranja pogleda i definicija spoljnjeg klju~a. Definicija spoljnjeg klju~a tako|e nije podr`ana, pa samim tim ni referencijalni integritet.

QBE, kao {to je pokazano, delimi~no podr`ava definisanje domena. Podr`an je i koncept primarnog klju~a, ali ne i spoljnjeg klju~a.

INGRES ne podr`ava koncept domena, na na~in kako je to ovde definisano. Primarni klju~ se tako|e defini{e posredno, preko jedinstvenog indeksa. Me|utim integritet relacija se u jeziku QUEL mo`e specificirati na na~in prikazan u slede}im primerima:

```
DEFINE INTEGRITY ON RADNIK IS RADNIK.STAROST >= 15 AND RADNIK.STAROST
                                                <= 80

DEFINE INTEGRITY CONSTRAINT ON RADNIK IS RADNIK.ORGJED# = 'oj1' AND
                                        RADNIK.STAROST <= 45
```

Me|utim, ni u INGRESU se ne mo`e definisati referencijalni integritet koji obuhvata dve relacije, niti se mogu zadati akcije koje treba izvr{iti kada uslov nije zadovoljen, a ni mesto provere integriteta (provera se uvek vr{i posle operacije odr`avanja).

U sistemu CA UNIVERSE mo`e se zadati i referencijalni integritet, ilustracija je data na slede}em primeru, ali se ni ovde ne mogu specificirati akcije koje slede ako uslov nije zadovoljen, niti mesto provere integriteta (provera se vr{i uvek posle operacije odr`avanja).

DEFINE INTEGRITY ON RASPORED IS RASPORED.RADN# = RADNIK.RADN#

Zbog svega ovoga, pravila integriteta moraju se uklju~ivati u razvoj aplikacija. Me|utim i tada je neophodno po}i od napred zadate specifikacije relacija, sa svim zadatim pravilima integriteta, jer to omogu}uje efikasan, struktuiran i delimi~no automatizovan razvoj aplikacija.

2.8. Sigurnost podataka

Termin sigurnost podataka odnosi se na mehanizme za{tite baze podataka od neovla{}enog kori{}enja. Postoje mnogi aspekti sigurnosti podataka (fizi~ko obezbe|enje, elektronska za{tita, kriptografska za{tita, za{tita u prenosu), a ovde }emo se baviti samo za{titom od neovla{}enog kori{}enja koja pru`aju sistemi za upravljanje bazom podataka.

Op{ti model za{tite podataka treba da defini{e koji subjekat za{tite, mo`e nad kojim objektom za{tite da izvr{i neku operaciju i pod kojim uslovima i obi~no se predstavlja slede}om tabelom:

SUBJEKAT	OBJEKAT	OPERACIJA	USLOV
Zoran Milan	RADNIK RASPORED	Prikaz, Dodav Prikaz	STAROST>40 PROJ# = pr1 . . . .

Granularnost objekata za{tite (cela relacija ili pojedina~ni atribut) i na~in definisanja uslova odre|uju kvalitet softverske za{tite u razli~itim SUBP.

U SQL standardu se ovla{}enje korisnicima daje na slede}i na~in:

GRANT privilegije [ON objekat] TO subjekat [WITH GRANT OPTION]

Privilegije mogu biti:

SELECT,
UPDATE,
INSERT,
DELETE,
INDEX (za kreiranje indeksa nad relacijom),
EXPAND (dodavanje atributa relaciji),
ALL PRIVILEGIES (sve gore navedene)
RUN (ako je objekat program),
DBA (za administratora baze podataka kome su dozvoljene sve operacije nad njegovom bazom),
RESOURCE (za kreiranje nove relacije i svih operacija nad njom).

WITH GRANT OPTION zna~i da autorizovani subjekt mo`e da prenosi svoju autorizaciju i drugim subjektima.

Na primer:

GRANT SELECT, INSERT ON RADNIK TO Zoran
GRANT SELECT, INSERT, DELETE ON RASPORED TO Milan WITH GRANT OPTION

GRANT DBA TO Milo{

Kao {to se vidi, u SQL standardu nedostaje mogu}nost zadavanja uslova pod kojim se privilegije daju, ve} se uslovi defini{u kreiranjem pogleda, pa se privilegije daju na pogled. Na primer, u koliko bi hteli da realizujemo prvi red iz tabele op{teg modela za{tite, morali bi prvi definisati pogled,

```
CREATE VIEW STARI_RADN
 AS SELECT *
        FROM RADNIK
        WHERE STAROST > 40;
```

a zatim privilegije za Zorana nad tim pogledom

GRANT SELECT, INSERT ON STARI_RADN TO Zoran

Na ovaj na~in se svi problemi sa pogledima, posebno problem a`uriranja pogleda prenosi i na definisanje za{tite.

U INGES-u op{ta sintaksa za definisanje za{tite je:

```
DEFINE PERMIT operacije     ON relacija [(atributi)]
                            TO subjekat
                            [AT terminal]
                            [FROM vreme1 TO vreme2]
                            [ON dan1 TO dan2]
                            [WHERE predikat]
```

Na primer:

```
DEFINE PERMIT RETRIEVE, APPEND ON RADNIK TO Zoran > 40
                    WHERE STAROST > 40
DEFINE PERMIT RETRIEVE ON RASPORED TO Milan
                    WHERE PROJ# = 'pr1'
```

Prednost ovakvog zadavanja privilegija u odnosu na SQL je o~igledna.

Na sli~an na~in, u tabelarnoj sintaksi, mo`e se zadati autorizacija i u QBE.

2.9. Kriterijumi za ocenu relacionih SUBP-a

Veliki broj SUBP sa atributom "relacioni" koji se pojavio na tr`i{tu sedamdesetih godina, od trenutka kada je E.F. Codd dao prvu teorijsku definiciju ovoga modela, primorao je njegovog tvorca da 1981. godine defini{e, a kasnije i dopunjava, kriterijume koji neki sistem treba da zadovolji da bi se mogao zvati relacionim. Ovde }emo navesti poslednju verziju ovih kriterijuma (PC Magazine, May 1988).

1. Struktura baze podataka se na logi~kom nivou predstavlja samo tabelama.
2. Svaki podatak u relacionoj bazi podataka dostupan je preko kombinacije imena tabele (relacije), vrednosti primarnog klju~a i imena kolone (atributa). (Bez unapred zadatih pristupnih puteva i bez rekurzije ili iteracije).
3. Specifi~an indikator (razli~it od "blanka", "praznog" niza karaktera, nule ili bilo kog broja) koristi se za predstavljanje nula vrednosti, bez obzira na tip podatka. Sistem tako|e podr`ava i sve operacije sa nula vrednostima bez obzira na tip podataka.

4. Sistem poseduje katalog (re~nik) podataka koji se logi~ki predstavlja na isti na~in kao i sama baza podataka, tako da korisnik mo`e da koristi isti jezik da bi pristupio ovim meta podacima.
5. Bez obzira koliko jezika i na~ina kori{}enja terminala sistem podr`ava, mora posedovati barem jedan jezik ~ije se naredbe mogu izraziti kao niz karaktera sa dobro definisanom sintaksom i koji podr`ava: (1) definiciju podataka, (2) definiciju pogleda, (3) manipulaciju podatka, interaktivno i kroz programe, (4) definiciju pravila integriteta, (5) autorizaciju (sigurnost), (6) granice transakcija (BEGIN, COMMIT, ROLLBACK).
6. Sistem poseduje efikasan algoritam pomo}u koga mo`e da odredi, za svaki definisani pogled, u trenutku njegovog definisanja, da li se i koje operacije odr`avanja mogu da primene na taj pogled. Rezultat ovoga algoritma se sme{ta u katalog baze podataka.
7. I nad baznim relacijama i nad pogledima mogu se izvr{avati ne samo operacije pretra`ivanja, ve} i operacije odr`avanja BP.
8. Aplikacioni program i interaktivna komunikacija ostaju neizmenjeni kada se promeni fizi~ka organizacija baze ili fizi~ki metod pristupa.
9. Aplikacioni program i interaktivna komunikacija ostaju nepromenjene kada se bilo koje promene, koje ne menjaju odgovaraju}i sadr`aj tabele, unesu u baznu tabelu.
10. Pravila integriteta se defini{u u okviru definicije baze podataka i ~uvaju se u katalogu. (Ne implementiraju se kroz aplikacione programe).
11. Sve navedene karakteristike su nezavisne od distribucije baze podataka.
12. Ako relacioni sistem poseduje ili mo`e da radi sa nekim jezikom tre}e generacije u kome se obra|uje jedna n-torka u jednom trenutku vremena, kroz taj jezik se ne mogu zaobi}i pravila integriteta zadana preko samog relacionog jezika.

Ve}ina ~ak i najsavremenijih relacionih SUBP-a ne zadovoljava sve navedene kriterijume. Na primer Codd navodi da IBM-ov DB2 zadovoljava samo sedam, a neki kao Cullinet-ov IDMS/R ili CA Datacom, ~ak nijedan. Osnovi problem, kao {to se iz diskusije u prethodnim poglavljima vidi su kriterijumi koji se odnose na pravila integriteta, zatim odr`avanje pogleda, nula vrednosti, distribucija i drugo.