BAZE PODATAKA

Uvod

BAZE PODATAKA – UVOD (1/3)

- Baza podataka se struktuirana kolekcija podataka koja postoji relativno dugo i koju najopštije može definisati kao dobro koristi i odžava, po pravilu, više korisnika, odnosno programa (aplikacija).

- Izučavanju baza podataka može se pristupiti sa dva različita, međusobno povezana aspekta u kojima se one tretiraju bilo kao:
  - Sistemi za upravljanje bazom podataka SUBP(Data Base Management System-DBMS)
  - Modeli podataka – intelektualni alata za projektovanje i razvoj BP

BAZE PODATAKA – UVOD (2/3)

- **Sistemi za upravljanje bazom podataka** (Database Management Systems), specifična tehnologija obrade podataka, odnosno softverski sistem koji obezbeđuje osnovne funkcije obrade velike količine podataka: jednostavno pretraživanje i održavanje podataka, višestruko paralelno (konkurentno) korišćenje istog skupa podataka, pouzdananost i sigurnost),
- **Modeli podataka**, odnosno specifične teorije pomoću kojih se specifikuje, projektuje i implementira neka konkretna baza podataka ili informacioni sistem, uopšte.

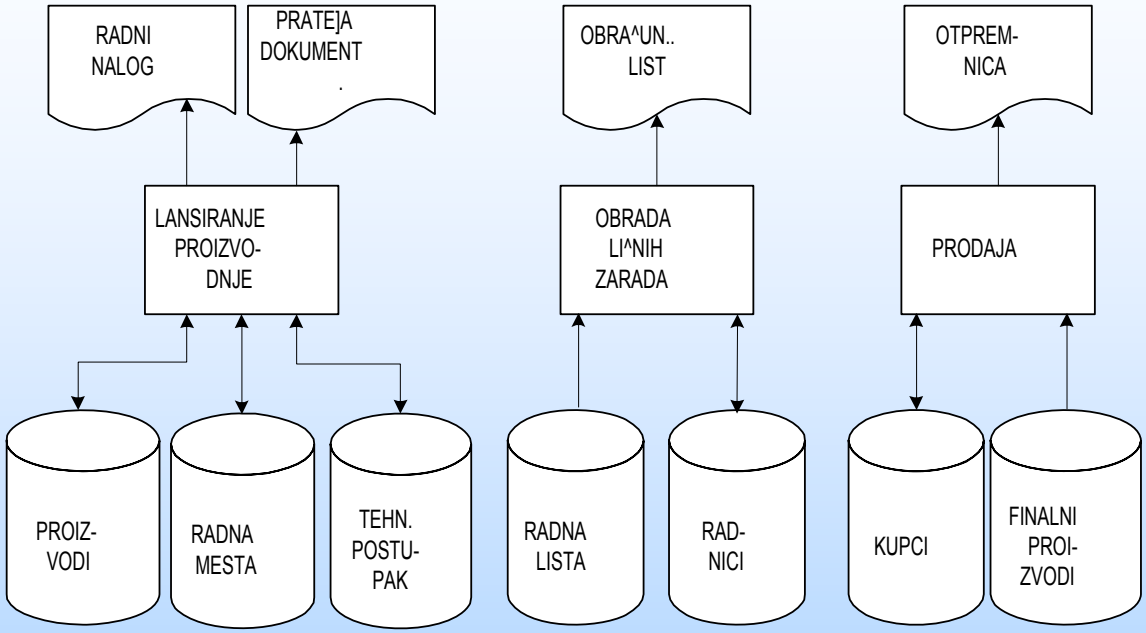
Sistemi za upravljanje bazom podataka obično se zasnivaju na nekom modelu podataka.

BAZE PODATAKA – UVOD (3/3)

- MODELI PODATAKA - PROJEKTOVANJE BAZA PODATAKA
- SISTEMI ZA UPRAVLJANJE BAZOM PODATAKA- RAZOJ SOFTVERA U OKRUŽENJU BAZA PODATAKA

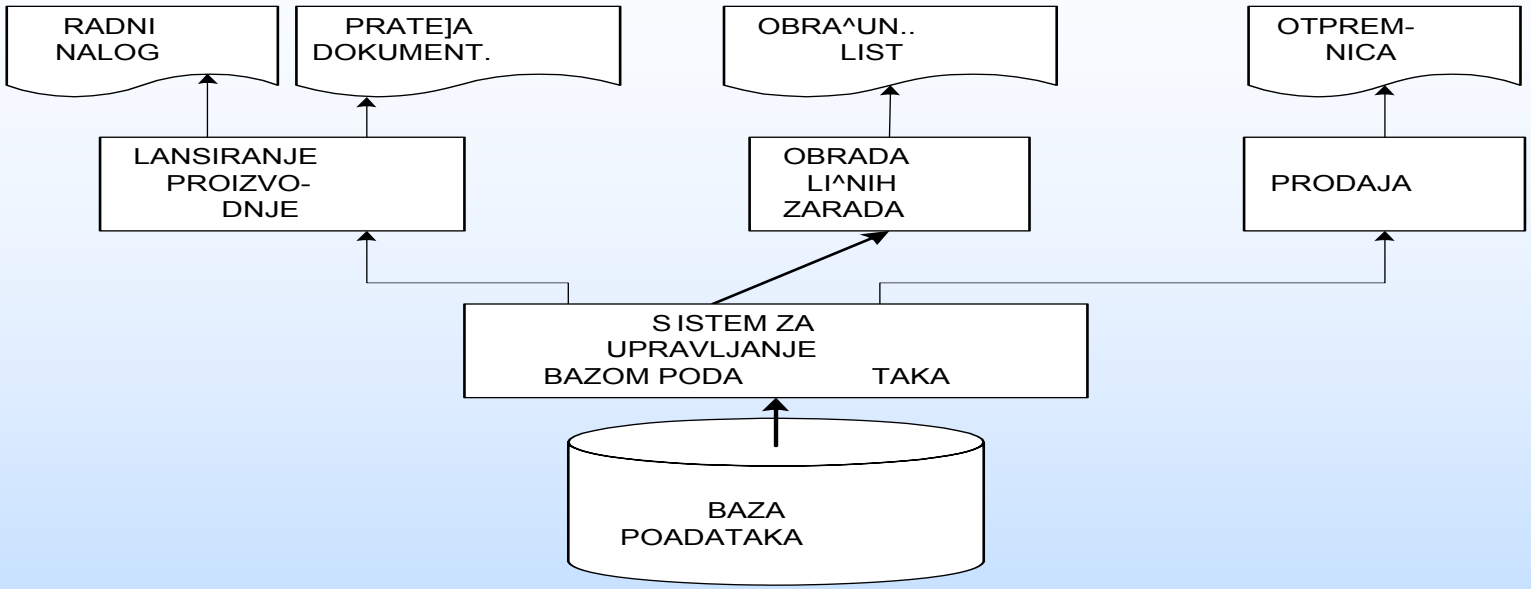
BAZE PODATAKA -SISTEMI ZA UPRAVLJANJE BAZOM PODATAKA (SUBP)

KONVENCIONALNA OBRADA - SISTEM DATOTEKA



- Redudansa podataka
- Logička i fizička zavisnost podataka i programa
- Niska produktivnost u razvoju sistema
- Nezadovoljavajuće pouzdan (otkazi sistema)
- Ne obezbedjuje tačnost i konzistentnost pri paralelnom radu

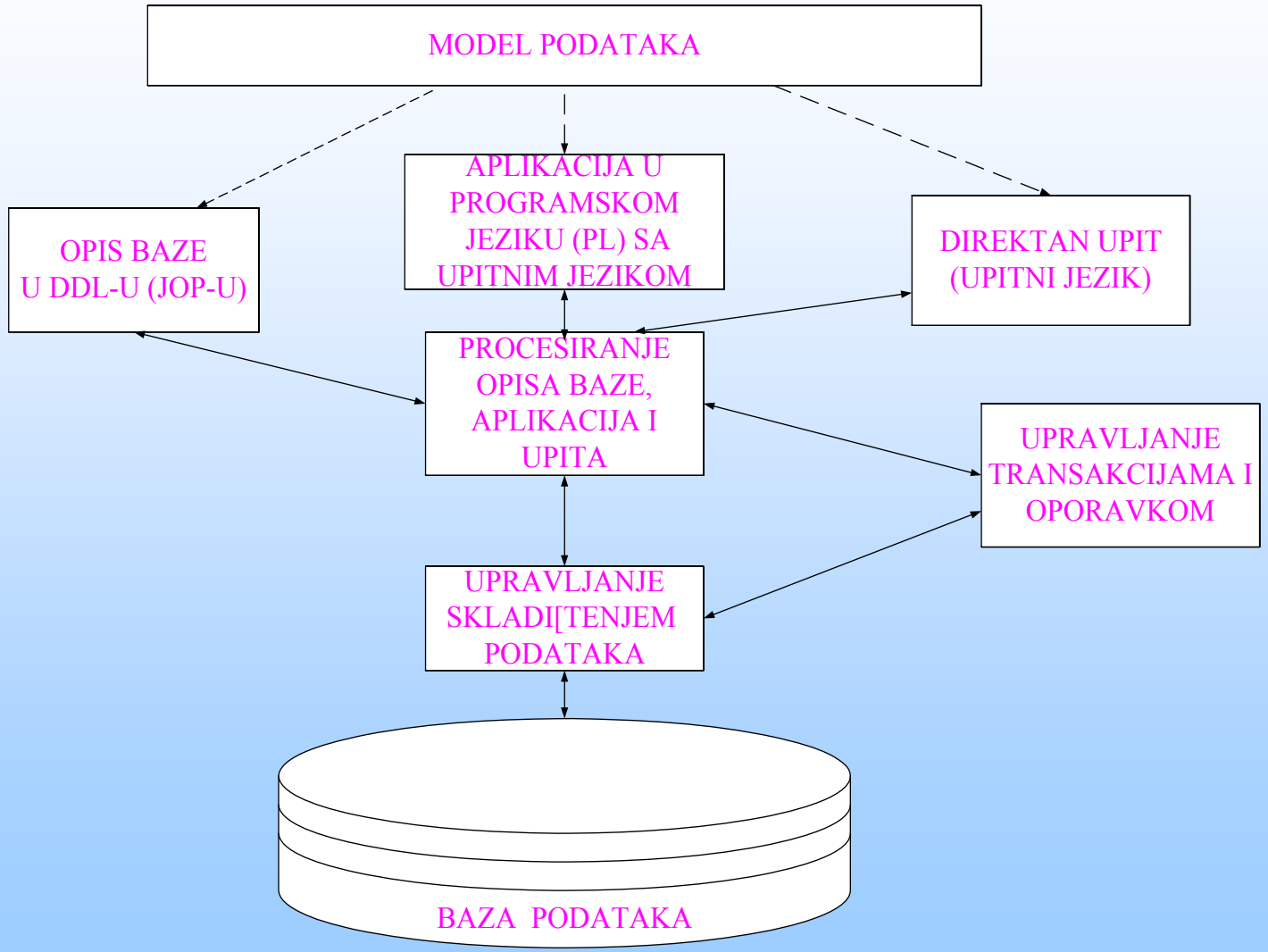
SISTEM ZA UPRAVLJANJE BAZOM PODATAKA –SUBP



Skladištenje podataka sa minimumom redundanse;

- Pouzdanost podataka i pri mogućim hardverskim i softverskim otkazima;
- Pouzdano paralelno korišćenje zajedničkih podataka od strane više ovlašćenih korisnika;
- Logičku i fizičku nezavisnost programa od podataka.
- Jednostavnu komuniciranju sa bazom podataka preko jezika bliskih korisniku, tzv “upitnih jezika”

KOMPONENTE SUBP-a



BAZA PODATAKA (1/2)

- Velike baze podataka pored diskova (sekundarne memorije) zahtevaju i tzv "tercijalnu" memoriju. Jedinice tercijalne memorije imaju kapacitet reda terabajta (1000 gigabajta, odnosno 10^{12} bajta). Na primer, sistem kompakt diskova sa "robotom" za izbor konkretnog diska. Očigledno je da je pristup podacima na tercijalnoj memoriji znatno sporiji (nekoliko sekundi) od pristupa podacima na diskovima (10 -20) milisekundi.

- Sistem za upravljanje skladištenjem podataka mora da obezbedi jednoobrazan pristup podacima na svim vrstama memorije.

BAZA PODATAKA (2/2)

- Baza podataka, pored podataka, sadrži i ***metapodatke***, odnosno tzv "***Recnik podataka***" (Data Dictionary, Data Directory, Catalog). ***Rečnik baze podataka*** opisuje posmatranu bazu podataka (strukturu baze, pravila očuvanja integriteta podataka, prava korišćenja i slično). Rečnik podataka je "baza podataka o bazi podataka", pa se taj deo baze podataka naziva i ***metabaza*** podataka.

- SUBP održava i ***bazu indeksa***. ***Index***, najopštije, predstavlja strukturu podataka koja omogućava brz pristup "indeksiranim" podacima baze. Najčešća struktura indeksa je B-stablo. Recnik podataka sadrži i opis indeksa posmatrane baze.

SISTEM ZA UPRAVLJANJE SKLADIŠTENJEM PODATAKA

- Sadrži dve osnovne komponente, ***Upravljanje baferima*** (Buffer Manager) i ***Upravljanje datotekama*** (File Manager).

- **Upravljanje datotekama** vodi računa o lokaciji datoteka i o pristupima blokovima (4.000 do 16.000 bajta) podataka na zahtev Upravljanja baferima.

- **Upravljanje baferima** prihvata blok podataka sa diska, dodeljuje mu izabranu stranicu centralne memorije, zadržava ga izvesno vreme, u skladu sa ugrađenim algoritmom upravljanja baferima, a zatim vraća na disk oslobađajući dodeljenu mu stranicu.

ODRŽAVANJE ŠEME BAZE PODATAKA

- Održavanje šeme (opisa) baze podataka. Šema baze podataka opisuje strukturu baze podataka, pravila integriteta i prava korišćenja . Održavanje šeme baze podataka podrazumeva prvo kreiranje, a zatim i modifikovanje ovog opisa koji se čuva u Rečniku podataka

- **Data Definition Language - DDL** (Jezik za opis podataka - JOP) koji se koristi za održavanje šeme baze podataka - naziv za konvencionalne baze

- Object definition language - ODL za objektne baze

UPITI I UPITNI JEZICI (1/2)

- **Upitni jezici -Neproceduralni jezici** sadrže konstrukcije preko kojih se samo specifikuju uslovi koje treba da zadovolji željeni rezultat, a ne i procedura pomoću koje se dobija taj rezultat.

- Osnovni zadatak **Procesora upita (Query Processor)** je da transformiše neproceduralni iskaz u sekvencu zahteva koje treba da realizuje **Sistem za upravljanje skladištenjem podataka**

- **Optimizacija upita** je nalaženje najpogodnije procedure za realizaciju neproceduralnog iskaza. Optimizacija upita koristi podatke iz Rečnika podataka: opis strukture, pravila integriteta, prava pristupa i definiciju indeksa.

UPITI I UPITNI JEZICI (2/2)

- **SQL:1999 (Structured Query Language)** - standardni upitni jezik za relacione baze podataka (i objekno-relacione)
- **OQL (Object Query Language)** - standardni upitni jezik za objektne baze podataka
- **SQL:1999 = SQL3**
- **SQL92 = SQL2** prethodni standard

DATA MANIPULATION LANGUAGE - DML

- ***Jezik za manipulaciju podataka*** - JMP (Data Manipulation Language - DML) je opšte ime za jezik preko koga se pristupa podacima u bazi i oni čitaju i menjaju.
- U relacionim bazama podataka DML je (kao i DDL) SQL.
- U starijim bazama (mrežnom i hijerarhijskom) postojali su specifični DML koji su praktično bili ugrađeni u programski jezik ("jezik domaćin").
- Ista strategija se koristi i u objektnim bazama, s tim što je ovde i ODL DML

OKRUŽENJE ZA RAZVOJ APLIKACIJA (1/2)

I. **Jezici IV generacije - generatori** aplikacija (Relacioni model). Generišu se aplikacije na osnovu sličnosti struktura dela baze(tabela) i korisničkog interfejsa. Dvoslojna softverska arhitektura - čvrsta veza baze i korisničkog interfejsa. Problemi koji odatle proizilaze su značajni.

II. **Aplikacija** se razvija u nekom programskom jeziku ("jeziku domaćinu") u koga se na neki način ugrađuje DML.

OKRUŽENJE ZA RAZVOJ APLIKACIJA (2/2)

- Bitno se razlikuju načini ugrađivanja DML-a u relacionim i objektnim bazama.
- U relacionim bazama postoji tzv" impedance mismatch" jezika domaćina i SQL-a. Cilj je da se ostvari nezavisnost jezika domaćina i upitnog jezika.
- U objektnim bazama DML se potpuno prilagođava programskom jeziku. Zato postoje C++ Binding, Java Binding, Smalltalk Binding.

DISTRIBUIRANE BAZE I RAZNE VRSTE KLIJENT-SERVER OKRUŽENJA

- Distribuirane baze – ostvarivanje transparentnosti distribucije
- Klijent-server okruženja:
  - Ostvarivanje konekcije sa BP
  - Call-Level Interface (CLI)
    - ODBC
    - JDBC
  - Različite vrste "middleware" – složenih komponenti za komunikaciju sa distribuiranim objektima (CORBA, .NET, EJB)

TRANSAKCIJA (1/4)

- ***Transakcija*** je niz operacija nad bazom podataka koji odgovara jednoj logičkoj jedinici posla u realnom sistemu.

  *Učitaj **iznosp** za prenos;*

  *Nađi račun R1 sa koga se **iznosp** skida;*

  *Upiši **iznosR1 - iznosp** na račun R1;*

  *Nađi račun R2 na koga se **iznosp** stavlja;*

  *Upiši **iznosR2 + iznosp** na račun R2.*

- **Transakcija** u izvršenju mora da ima tzv. **ACID osobine** (po početnim slovima sledećih engleskih reči)**:**

TRANSAKCIJA (2/4)

- ***Atomnost*** (Atomicity)**.** Zahteva se da se bilo sve operacije nad bazom podataka uspešno obave ili ne obavi nijedna. Da bi se ostvarila atomnost transakcije definišu se dve specifične operacije nad bazom podatataka:

- COMMIT koja označava uspešan kraj transakcije i koja "potvrđuje" sve promene u bazi koje je posmatrana transakcija proizvela;
- ROLLBACK kojom se poništavaju efekti svih prethodnih operacija nad bazom podataka u jednoj transakciji, ako ona, zbog predviđene ili nepredviđene greške (otkaza sistema) može da dovede bazu podataka u nekonzistentno stanje.

TRANSAKCIJA (3/4)

- ***Konzistentnost*** (Consistency). Očigledno je da se transakcija može definisati i kao "jedinica konzistentnosti" baze podataka: pre početka i posle okončanja transakcije stanje baze podataka mora da zadovolji uslove konzistentnosti. Za vreme obavljanja transakcije konzistentnost baze podataka može da bude narušena.

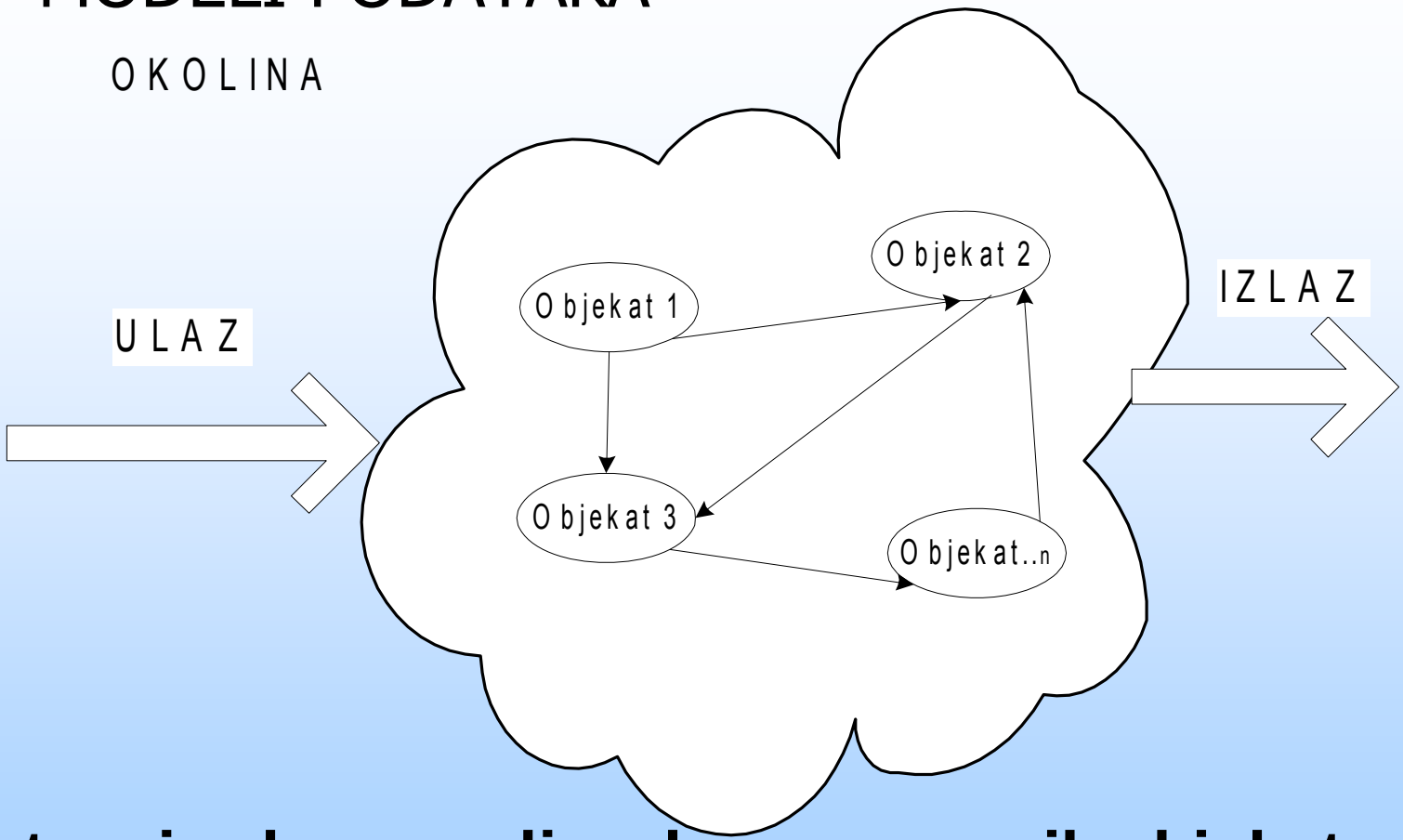
TRANSAKCIJA (4/4)

- ***Izolacija*** (Isolation). Kada se dve ili više transakcija izvršavaju istovremeno, njihovi efekti moraju biti međusobno izolovani. Drugim rečima efekti koje izazovu transakcije koje se obavljaju istovremeno moraju biti jednaki efektima nekog njihovog serijskog (jedna posle druge) izvršenja.

- ***Trajnost*** (Durability). Kada se transakcija završi njeni efekti ne mogu biti izgubljeni, čak i ako se neposredno po njenom okončanju desi neki ozbiljan otkaz sistema.

MODELI PODATAKA

MODELI PODATAKA



Sistem je skup medjusobno povezanih objekata. **Ulazi** u sistem opisuju dejstvo okoline na sistem, a **izlazi** odgovarajući odziv sistema, odnosno dejstvo sistema na okolinu.

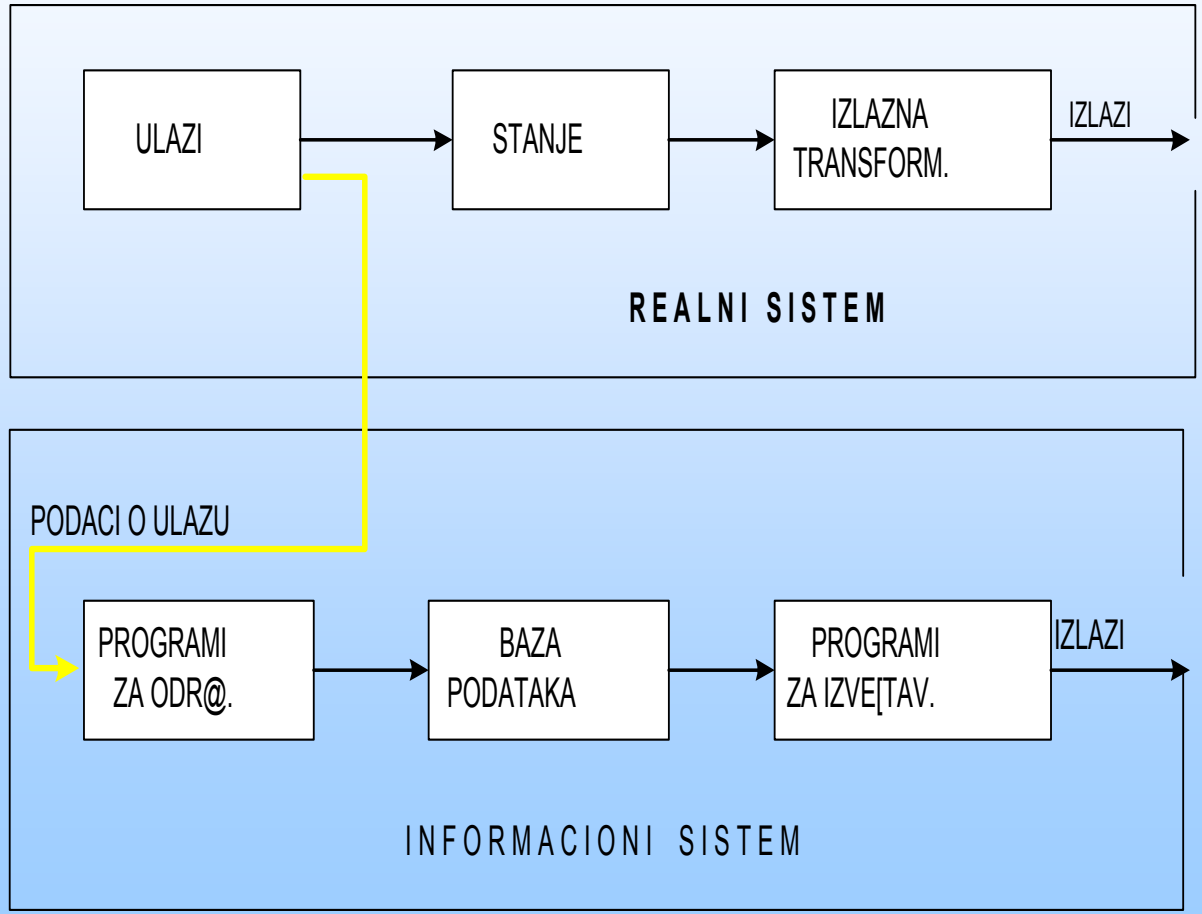
MODELI PODATAKA

- Sistem, odnosno objekti imaju svoja stanja koja se menjaju pod dejstvom ulaza, a čija se promena reflektuje na izlaz.

- **Stanje sistema** pretstavlja skup vrednosti atributa njegovih objekata i skup veza između objekata u datom trenutku vremena.

MODEL PODATAKA

INFORMACIONI SISTEM KAO MODEL REALNOG SISTEMA



MODEL PODATAKA - STANJE SISTEMA

- Ulazi u sistem (dejstvo okoline na sistem) menjaju stanja sistema
- Stanje sistema se definiše kao skup informacija o prošlosti i sadašnjosti sistema koji je potreban da bi se, pod dejstvom budućih poznatih ulaza, mogli odrediti budući izlazi. U stanju sistema skoncentrisana je celokupna istorija realnog sistema.
- Očigledno je da stanje sistema opisuje fundamentalne karakteristike sistema. U jednom trenutku vremena ono predstavlja skup objekata sistema, skup njihovih međusobnih veza i skup vrednosti atributa objekata u tom trenutku vremena.

MODEL PODATAKA

- Stanje sistema ne mora pretstavljati neku "fizičku veličinu", odnosno neki poznati koncept realnog sistema - ono je prosto "skup informacija ....".

- Ponašanje sistema se u realnom sistemu sagledava ("meri"), odnosno sistemom se upravlja na bazi njegovih stvarnih izlaza. Izlazna transformacija definiše odnos između stanja sistema i izlaza, odnosno predstavlja način merenja ili posmatranja dinamičkog ponašanja realnog sistema.

MODEL PODATAKA - STANJE SISTEMA

- BAZA PODATAKA PREDSTAVLJA MODEL STANJA SISTEMA.
- **Model podataka** je intelektualni alat za definisanje modela sistema, za prikazivanje objekata sistema, njihovih atributa i njihovih dozvoljenih vrednosti, medjusobnih veza objekata i dinamike sistema.
- **Model podataka je** specifičan teorijski okvir pomoću koga se specifikuje, projektuje i implementira neka konkretna baza podataka ili informacioni sistem, uopšte.
- **Model podataka** je osnova za razvoj Sistema za upravljanje bazom podataka (SUBP)

MODEL PODATAKA

- Podatak je neka kodirana činjenica iz realnog sistema, on je nosilac informacije. Informacija je protumačeni (interpretirani) podatak. Interpretacija podataka se vrši na osnovu strukture podataka, semantičkih ograničenja na njihove vrednosti i preko operacija koje se nad njima mogu izvršiti. Imajući sve ovo u vidu, svaki model podataka treba da čine sledeće osnovne komponente:

1. **Struktura modela.**
2. **Ograničenja** - semantička ograničenja na vrednosti podataka koja se ne mogu predstaviti samom strukturom modela.
3. **Operacije** nad konceptima strukture, preko kojih je moguće prikazati i menjati vrednosti podataka u modelu;

MODEL PODATAKA

- Pri analizi različitih modela podataka analizirаće se način specifikacije i implementacije *strukture, ograničenja, operacija i dinamičkih pravila integriteta* za svaki konkretan model.
- Osnovni problem u modeliranju je složenost sistema, odnosno mnoštvo objekata, atributa objekata i veza između objekata, koje postoje u jednom realnom sistemu. Opšti metodološki pristup za savladavanje ove složenosti u opisu sistema je **apstrakcija**. **Apstrakcija** je kontrolisano i postepeno uključivanje detalja u opis sistema, "sakrivanje" datalja u opisivanju sistema, odnosno izvlačenje i prikazivanje opštih, a odlaganje opisivanja detaljnih osobina nekog sistema.
- *Koje apstrakcije i kako pojedini modeli koriste?*

MODEL PODATAKA

POSTOJI VIŠE KRITERIJUMA ZA KLASIFIKACIJU MODELA PODATAKA:

- NAČIN OPISIVANJE DINAMIKE SISTEMA
- NAČIN OSTVARIVANJA OSNOVNIH CILJEVA SUBP-a
- DA LI SE MODEL KORISTI SAMO ZA PROJEKTOVANJE BP, SAMO KAO OSNOVA ZA NEKI SUBP ILI I JEDNO I DRUGO
- NAČIN KAKO PRETSTAVLJAJU OBJEKTE I VEZE

MODEL PODATAKA

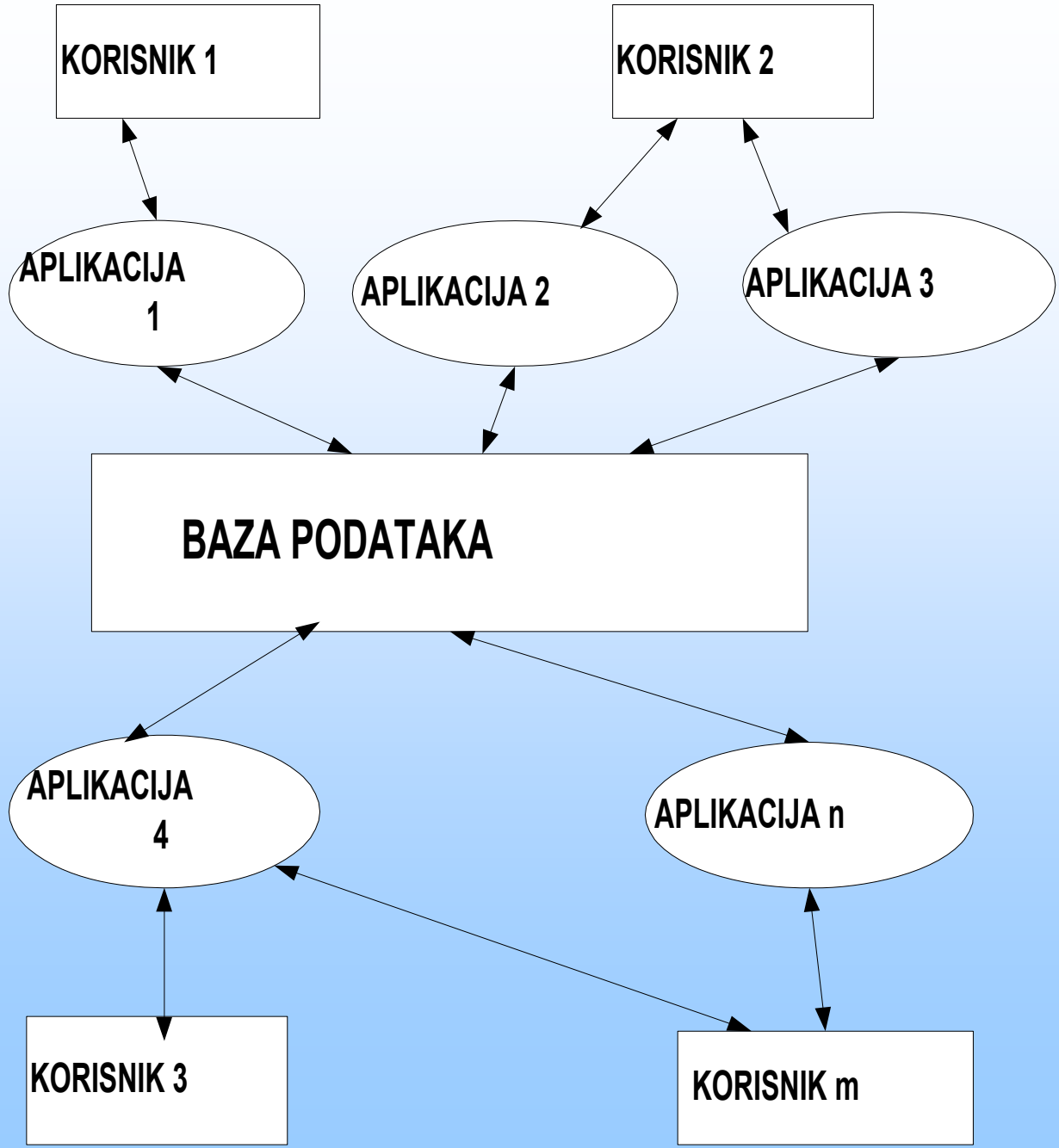
POREĐENJE - KRITERIJUM:
OPISIVANJE DINAMIKE SISTEMA
MOGUĆA PODELA:

- KONVENCIONALNI (HIJERARHIJSKI, MREZNI, RELACIONI, MODEL OBJEKTI-VEZE)
- OBJEKTNI
- AKTIVNE BAZE PODATAKA

KONVENCIONALNI:

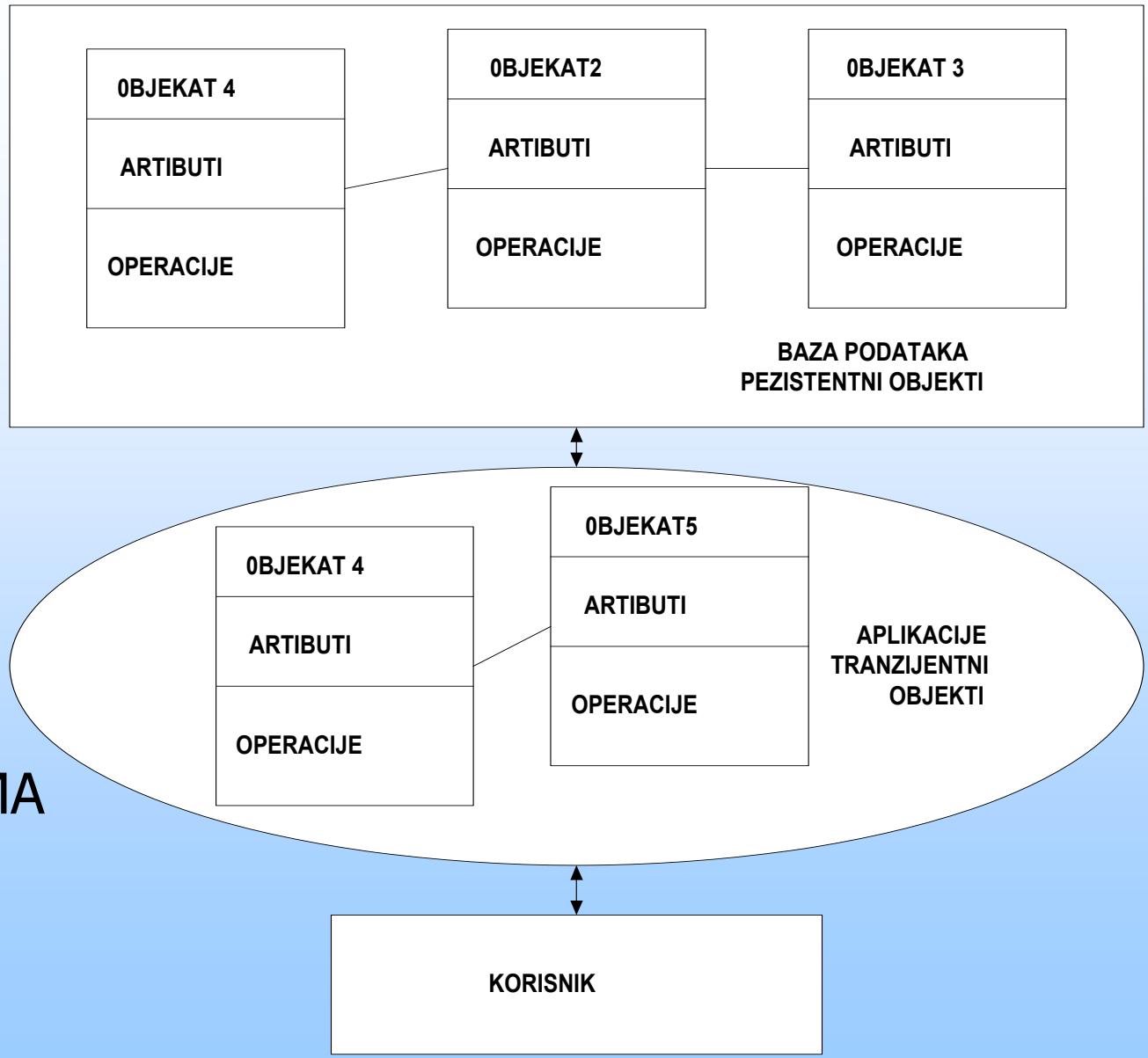
BAZA PODATAKA JE POTPUNO STATIČKI KONCEPT SA IZUZETKOM JEDNOSTAVNIH DINAMIČKIH PRAVILA INTEGRITETA

SVA DINAMIKA JE U APLIKACIJAMA



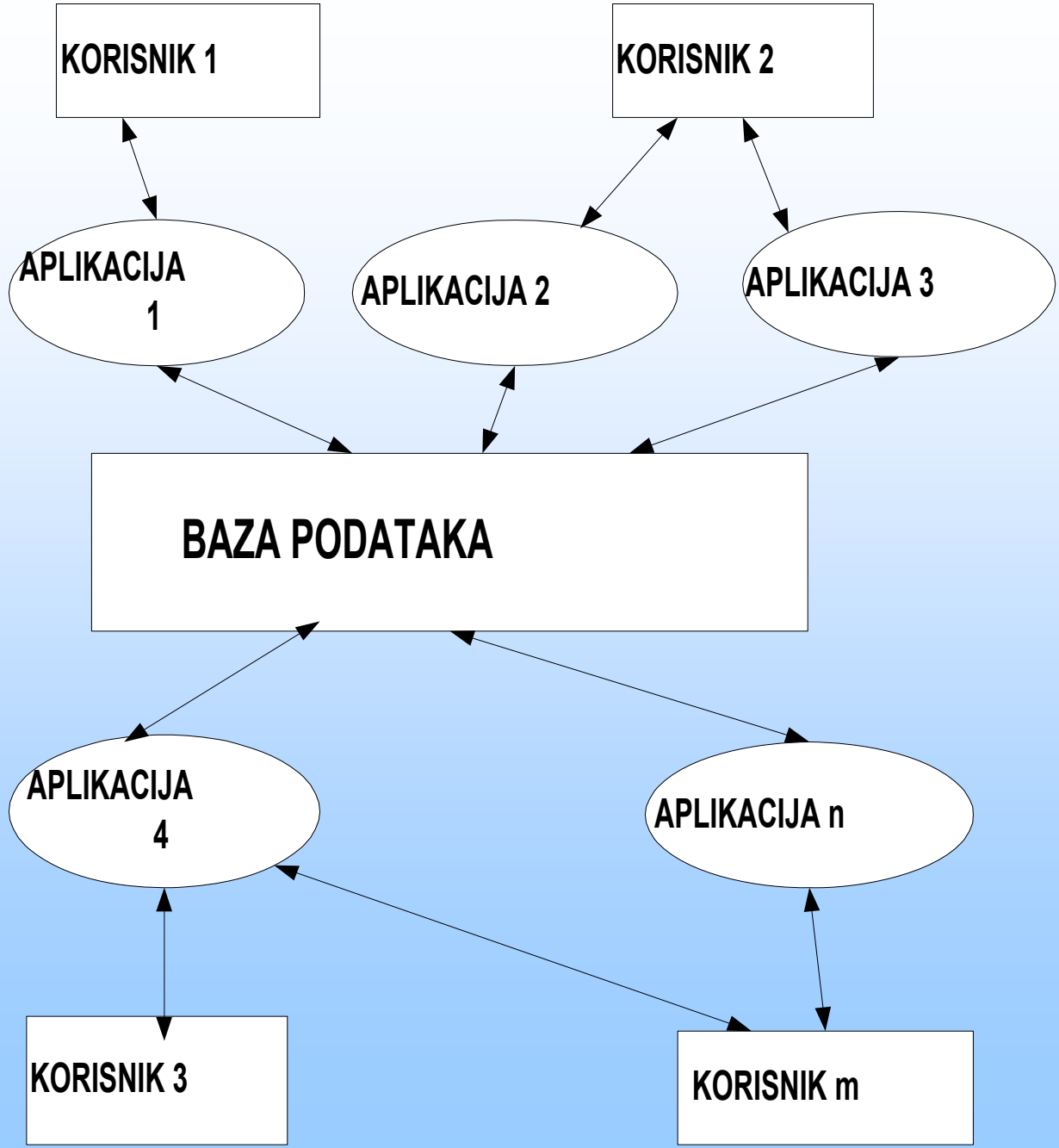
OBJEKTNI MODEL

DINAMIKA SE OBUHVATA NA ISTI NAČIN I U BAZI PODATAKA I U APLIKACIJAMA



AKTIVNE BAZE:

PREKO KONCEPTA PRAVILA KOJA SE ISKAZUJU KAO KOMBINACIJA <USLOV, AKCIJA>, ZNATNO VEĆA "KOLIČINA" DINAMIKE SISTEMA SE NALAZI U BAZI PODATAKA



MODEL PODATAKA

POREĐENJE - KRITERIJUM:
OSTVARIVANJE OSNOVNIH CILJEVA SUBP-A

OSNOVNI CILJEVI BP:

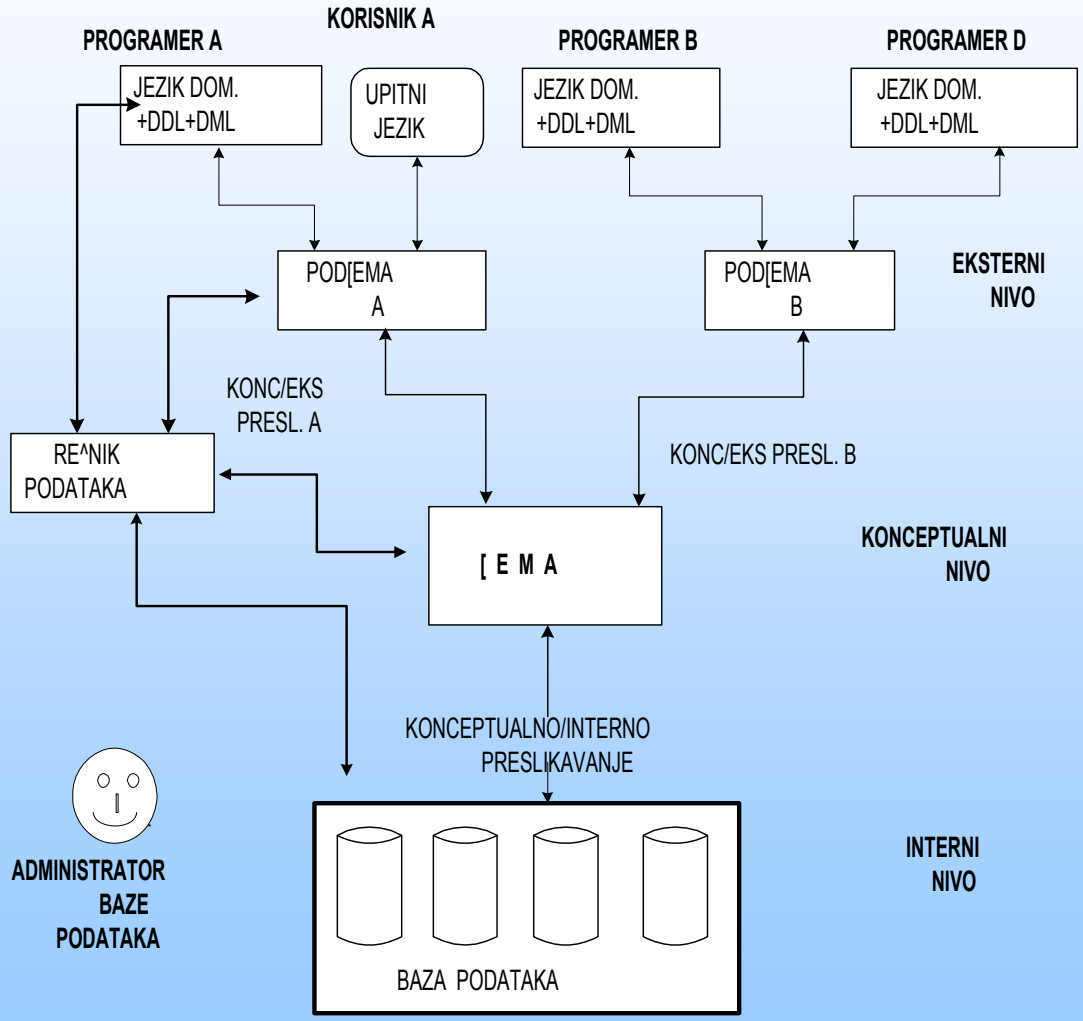
(1) Neredundatno pamćenje podataka

(2)Višestruko paralelno (konkurentno) korišćenje podataka

(3) Ostvarivanje nezavisnosti programa i logičke i fizičke strukture baze podataka)

- KONVENCIONALNE BP
- OBJEKTNE BP

KONVENCIONALNE BAZE-
ANSI/SPARC STANDARDNA ARHITEKTURA



OSNOVNAI CILJEVI BP - OBJEKTNE BAZE

- Objektne baze prva dva cilja ispunjavaju na sličan način kao i konvencionalne: (1) Neredeundanost podataka se ostvaruje dobrim projektovanjem BP, (2) Višestruko paralelno (konkurentno) korišćenje podataka, odnosno upravljanje obradom transakcija, ostvaruje se sličnim mehanizmima "zaključavanja"
- Nezavisnosti programa i logičke i fizičke strukture baze podataka ostvaruje se preko **koncepta nezavisnosti specifikacije tipa objekta od njegove implementacije.**

NEZAVISNOST PROGRAMA I STRUKTURE BAZE PODATAKA - SPECIFIKACIJA I IMPLEMENTACIJA OBJEKATA

- Objekti koji imaju isti skup stanja (isti skup atributa i veza) i isto ponašanje (isti skup operacija) mogu se pretstaviti opštim **tipom objekta.**

- Svaki tip se može posmatrati na dva načina: kao specifikacija - **interfejs tipa** koji definiše "spoljne", vidljive karakteristike tipa i kao jedna ili više **klasa. Klasa** je specifikacija implementacije tipa i pretstaljena je strukturom podataka i skupom metoda koje pretstavljaju implementaciju operacija definisanih u interfejsu tipa. Jedan tip može imati više različitih klasa kao svoje implementacije.

NEZAVISNOST PROGRAMA I STRUKTURE BAZE PODATAKA - SPECIFIKACIJA I IMPLEMENTACIJA OBJEKATA

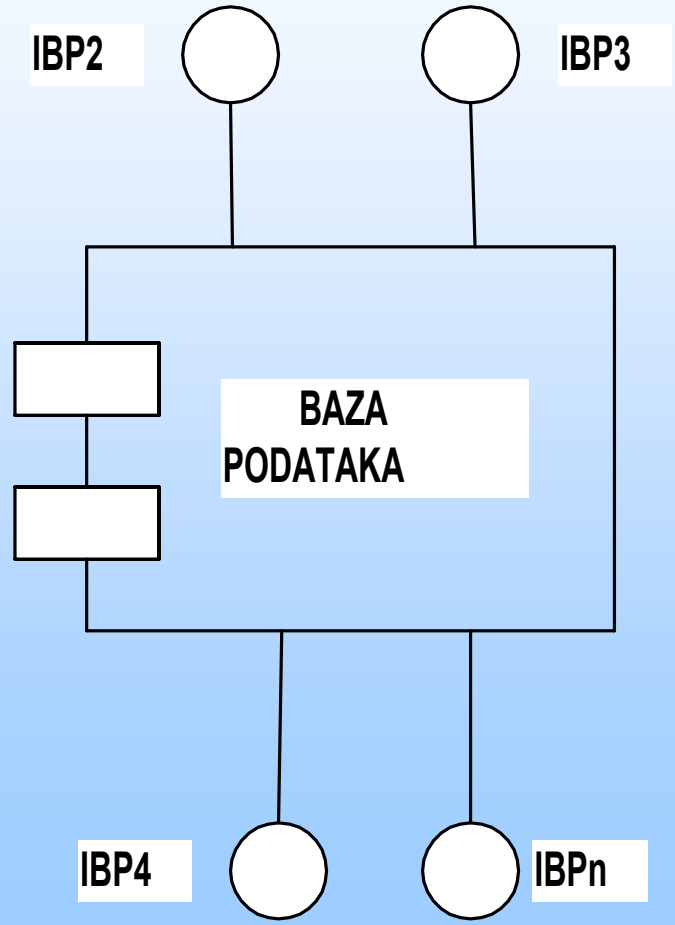
- Interfejsi tipova objekata, odnosno sve vidljive karateristike tipova objekata baze podataka čine konceptualni model baze podataka.
- Još je pogodnije objektnu bazu podataka posmatrati kao jedinstvenu softversku komponentu.
- KOMPONENTA je fizički, izmenljivi deo sistema, realizacija skupa interfejsa

NEZAVISNOST PROGRAMA I STRUKTURE BAZE PODATAKA - SPECIFIKACIJA I IMPLEMENTACIJA OBJEKATA

KOMPONENTA - FIZIČKA BAZA PODATAKA

UNIJA SVIH INTERFEJSA - KONCEPTUALNI MODEL

SPECIFIČNI INTERFEJSI - KORISNIČKI PODMODELI



MODEL PODATAKA

- ZA PROJEKTOVANJE: MODEL OBJEKTI VEZE, OBJEKTNI MODEL, RELACIONI MODEL

- KAO OSNOVA SUBP-a (IMPLEMENTACIJA): HIJERARHIJSKI, MREŽNI, RELACIONI, OBJEKTNI

Najčešća kombinacija za razvoj softvera danas:

Objektni pristup i jezici za razvoj aplikacija i relacioni SUBP.

MODEL PODATAKA

POREĐENJE - KRITERIJUM:
NAČIN KAKO PRETSTAVLJAJU OBJEKTE I VEZE:

- VREDNOSNO ORJENTISANI: vrednosti atributa se koriste i za identifikaciju objekata i za pretstavljanje veza: Relacioni model
- OJEKTNO ORJENTISANI: Objekti se identifikuju prilikom kreiranja, veze se uspostavljaju preko "pokazivača": Hijerarhijski, Mrežni,Objektni,

MODEL PODATAKA

	Proste strukture	Složene strukture
Upitni jezik	Proste strukture i upitni jezik (konvencionalni relacioni SUBP)	Složene strukture i upitni jezik (Relaciono- objektni SUBP
Bez upitnog jezika	Jednostavne strukture bez upitnog jezika (Mrežni i hijerarhijski model)	Složene strukture bez upitnog jezika (Objektno-orjentisani modeli

JEDAN MOGUĆI PRISTUP POREĐENJU RAZLIČITIH MODELA

MODEL PODATAKA -STANDARDI

- MREŽNI MODEL -CODASYL STANDARD
- KONVENCIONALNI RELACIONI MODEL - STANDARD SQL2 (SQL92)
- OBJEKTNI MODEL - ODMG 2.0 (3.0 APRIL 2000.) STANDARD
- OBJEKTNO-RELACIONE BAZE (STANDARD SQL3-SQL:1999)

SADRŽAJ KURSA (1/2)

- I DEO : MODELI PODATAKA

1. Model objekti-veze
2. Relacioni model
3. Standardni upitni jezik SQL
4. Objektne baze podataka
5. Objektno-relacioni model
6. Aktivne baze podataka
7. XML kao model podataka

SADRŽAJ KURSA (2/2)

II DEO: FUNKCIJE SISTEMA ZA UPRAVLJANJE BAZOM PODATAKA

1. Fizička struktura baze podataka
2. Optimizacija upita
3. Upravljanje izvršenjem transakcija i oporavak baze podataka
4. Sigurnost baze podataka
5. Katalog baze podataka
6. Distribuirane arhitekture

III DEO: PROJEKTOVANJE BAZA PODATAKA

1. Analiza sistema i zahteva korisnika: SSA
2. Analiza sistema i specifikacija aplikacija: objektne metode
3. Konceptualno modelovanje
4. Logičko i fizičko projektovanje baza podataka

IV DEO-BAZE PODATAKA I RAZVOJ APLIKACIJA

1. SQL unutar klasičnih i objektnih programskih jezika (embedded SQL)
2. Pristup bazama podataka preko poziva funkcija
3. Procedure baze podataka - Persistent Stored Module (PSM)

REFERENCE (1/5): NAJNOVIJE

1. C.J. Date: *An Introduction to Database Systems*, 7th edition, Addison Weseley, 2000.
2. H.G.Molina, J.Ullman, J Widom, *Database Systems, The Comlete Book*, Prentice Hall, 2002
3. J.Ullman, J Widom, *A First Course in Database Systems,* Prentice Hall, 2002
4. R.A.Elmasri, B.S.Navate, *Fundamentals of Database Systems*, 3rd edition (August 1999) Addison-Wesley Pub Co
5. P.Atceni, S.Ceri, S.Parabichi. R.Torlone, *Database Systems*, McGrawHill, 1999

REFERENCE (2/5): STARE DOBRE

1. C.J.Date: *An Introduction to Databsase Systems,* 6th edition , Addison-Wesley,1995
2. J.Ullman, J.Widom: *A First Course in Database Systems*, Prentice Hall, 1997
3. J.D.Ullman: *Principles od Database and Konwledge-Base Systems,* Vol I, II, Computer Science Press, 1988.
4. B. Lazarević, *Baze podataka, materijal za studente*, Fon 1992.

REFERENCE (3/5)

RELACIONI MODEL – TEORIJSKE REF.

1. E.F. Codd, *The Relational model for Database management,* , Version 2,Addisov -Weseley, 1990.
2. D. Maier, *The Theory of Relational Databases*, Computer Sience Press, 1983.
3. Chao-Chih Yang, *Relational Databases*, Prentice Hall, 1986
4. S. Alagi}, *Relacione baze podataka*, Svjetlost, Sarajevo1984.
5. P.Mogin, I. Lukovi}, *Principi baza podataka,* Stylos, Novi Sad, 1996
6. P.Mogin, I.Lukovi}, M.Govedarica, *Principi projektovanja baza podataka,* Univerzitet u Novom Sadu, 2000.

REFERENCE (4/5)

OBJEKTNE BAZE - OSNOVNE REFERENCE

1. R.C.G.Cattell, *The Object Database Standard: ODMG-93*, Release 1.2., Morgan Kaufman Publishers, 1996.
2. R.C.G.Cattell, D.K.Barry, *The Object Database Standard: ODMG2.0., Morgan Kaufman Publishers*, 1997.

Sve novije knjige o bazama podataka

REFERENCE (5/5)

OBJEKTNE I OBJEKTNO-RELACIONE BAZE PODATAKA

1. M. Loomis, *ObjectDatabases: the Essential*, Addison Weseley, 1995.
2. M.Stonebraker *Object-Relational Databases*, Morgan Kaufmann Publ., 1996.
3. SQL3 Standard, ftp://jery.ece.umassed.edu/SCG32/WG3/
4. Sve novije knjige o bazama podataka

MODEL OBJEKTI - VEZE

KONCEPTI MODELA

METODOLOGIJA MODELIRANJA

MODELI PODATAKA

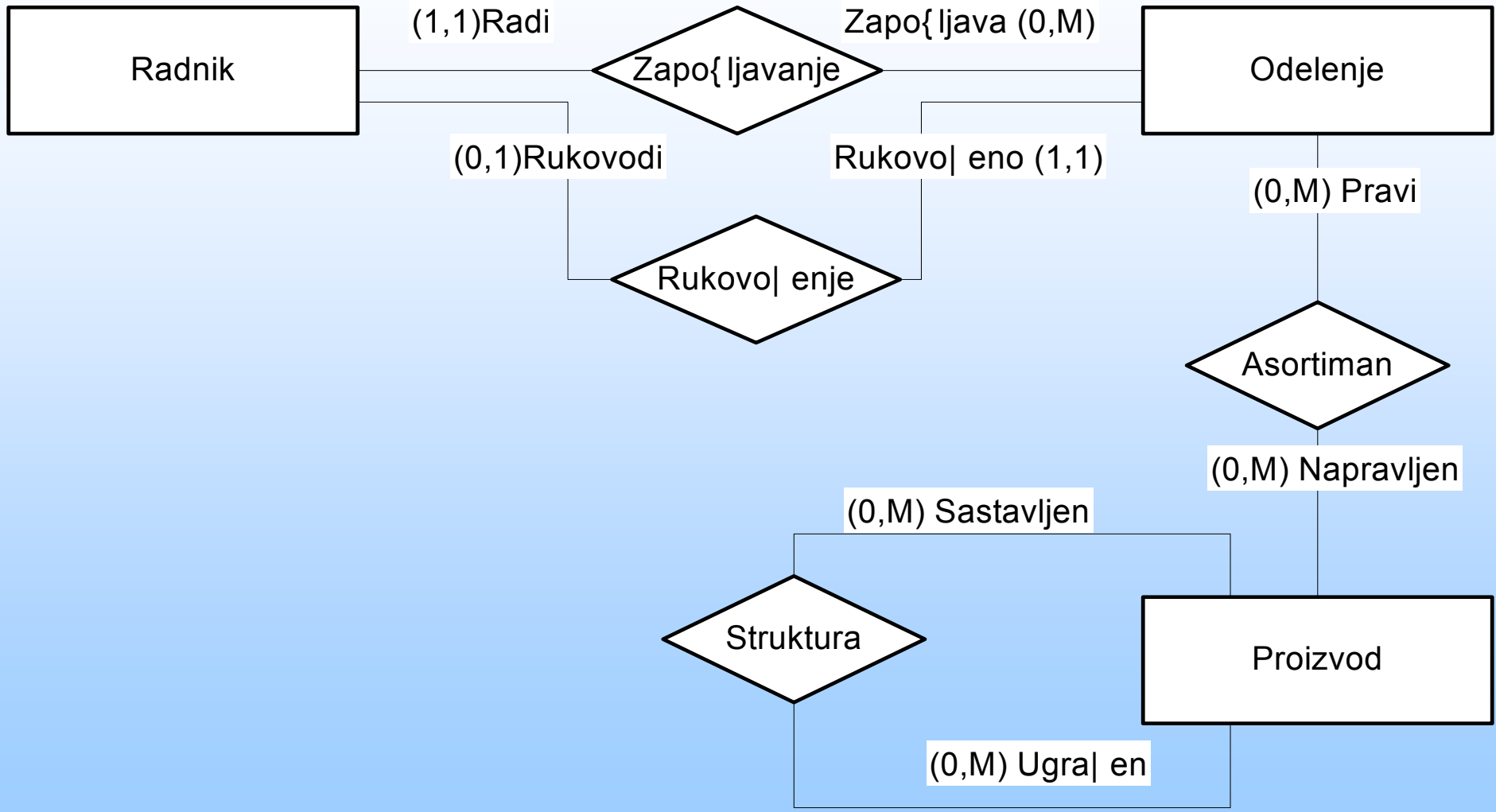
- Model objekti-veze
- Relacioni model
- Objektni model
- Objektno-relacioni model
- Aktivne baze podataka
- XML kao model podataka

MODEL OBJEKTI - VEZE

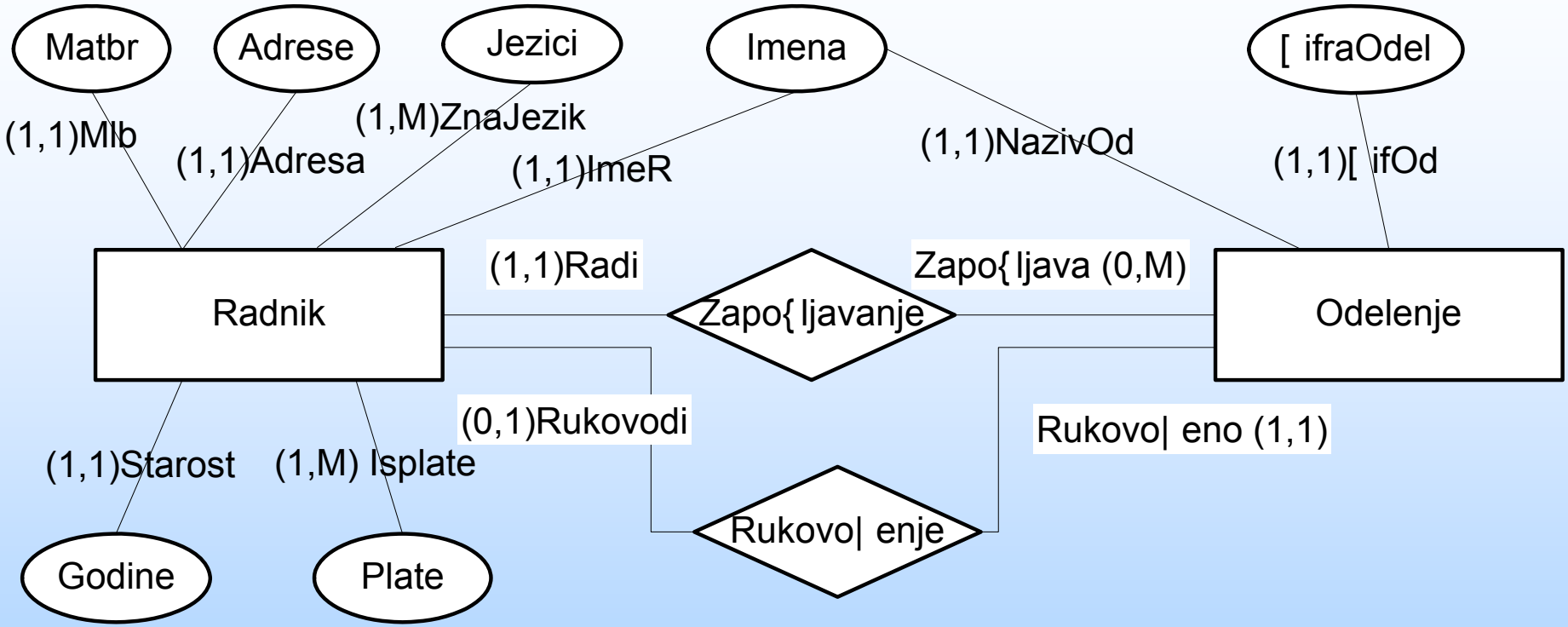
Model objekti-veze je najpopularniji i u praksi projektovanja BP najviše korišćeni konvencionalni model podatka. Postoji više različitih verzija ovog modela. Ovde se izlaže verzija koja polazi od originalne Chen-ove verzije i notacije.

Biće prikazana i notacija standarda IDEF1X, na kome se bazira CASE alat ERwin, veoma često korišćeni alat za modeliranje podataka preko Modela objekti veze.

OSNOVNI KONCEPTI PMOV



Definicija atributa



Mlb: Radnik ----> Matbr (1,1)
ImeR: Radnik ----> Imena (1,1)
ZnaJezik: Radnik ----> Jezici (1,M)

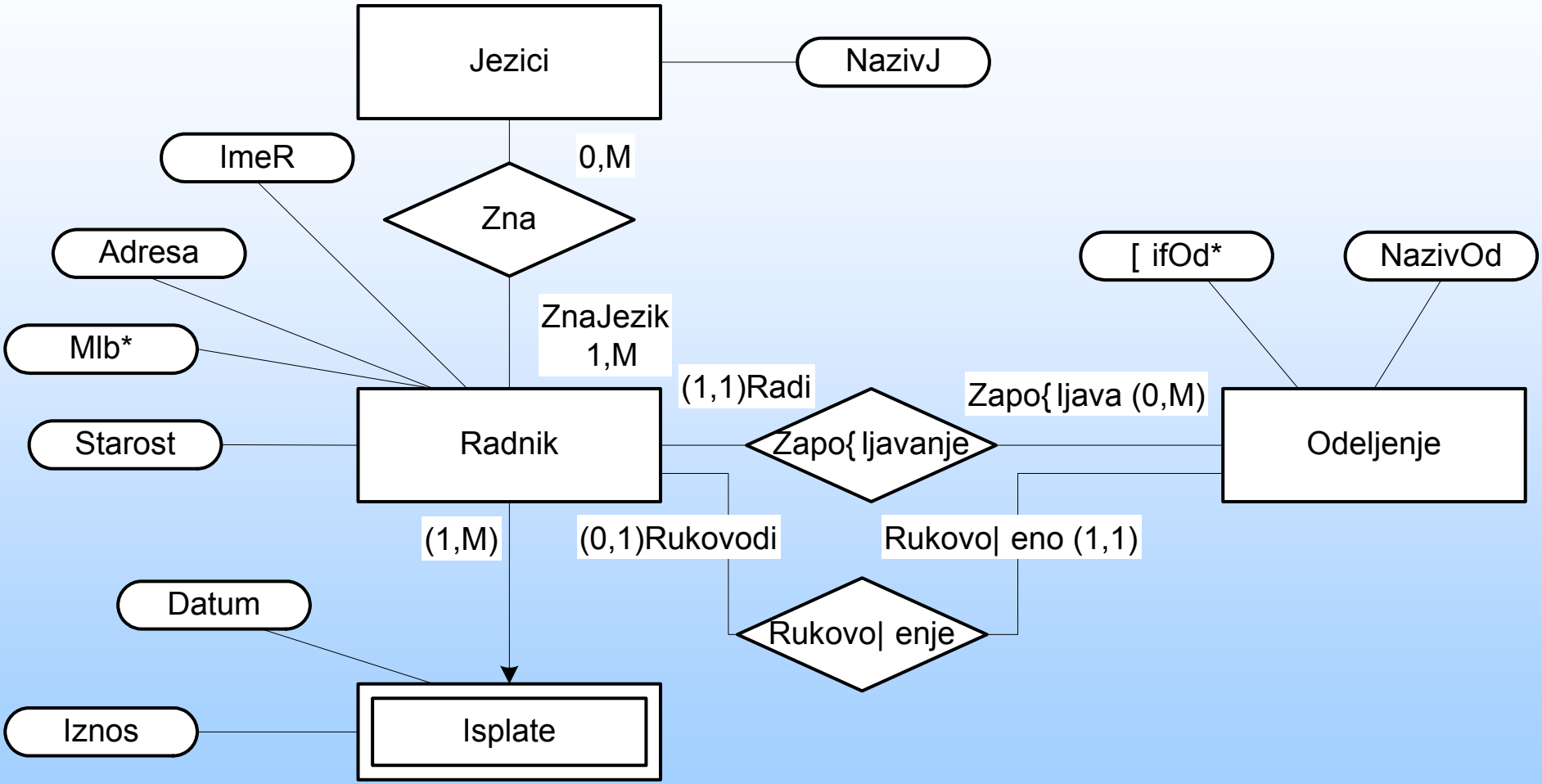
Definicija atributa - rekapitulacija

- U MOV se ne koriste višeznačni atributi

- Svi atributi moraju da budu primenljiva svojstva na sve objekte u odgovarajućoj klasi. Zbog toga je donja granica kardinalnosti atributa (preslikavanje KLASA--->DOMEN) uvek DG = 1. Kako se ne koriste višeznačni atributi, to je za ovo preslikavanje i GG = 1, pa se kardinalnosti atributa ne moraju predstavljati na DOV

Definicija atributa - rekapitulacija

- Atributi identifikatori objekata mogu se posebno označiti (na primer sa zvezdicom, ili podvlačenjem).
- U MOV se ne koriste semantički domeni, niti se može uspostaviti bilo kakva veza između atributa jedne ili više klasa.


Jezici
NazivJ
ImeR
0,M
Zna
Adresa
[ifOd*
NazivOd
ZnaJezik
1,M
Mlb*
(1,1)Radi
Zapo{ljava (0,M)
Starost
Radnik
Zapo{ljavanje
Odeljenje
(1,M)
(0,1)Rukovodi
Rukovo| eno (1,1)
Datum
Rukovo| enje
Iznos
Isplate

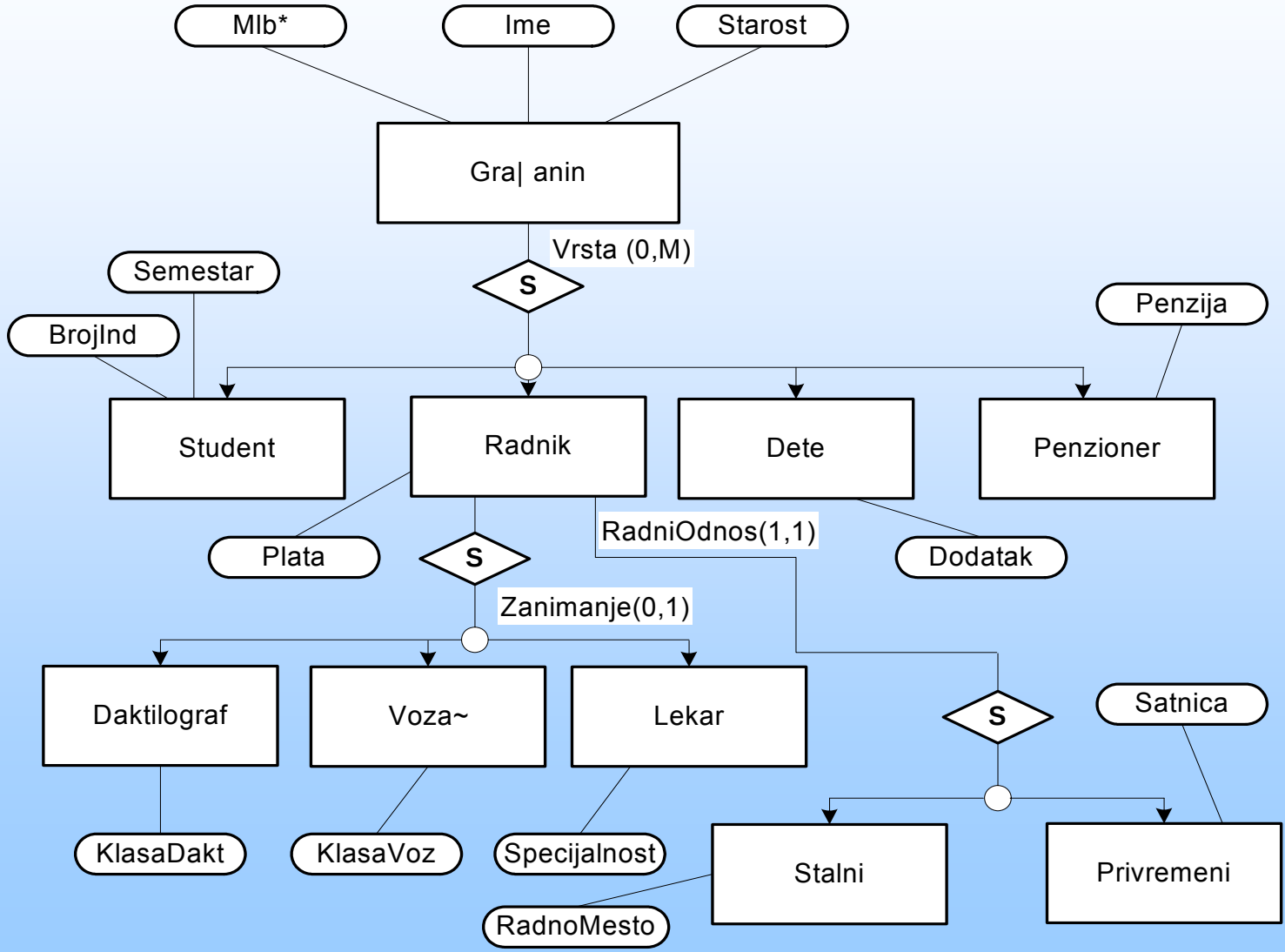
Pored Dijagrama objekti veze, neophodno je dati i definiciju atributa, odnosno njihovih domena.

U sledećoj tabeli dati su primeri definicija atributa sa ograničenjima:

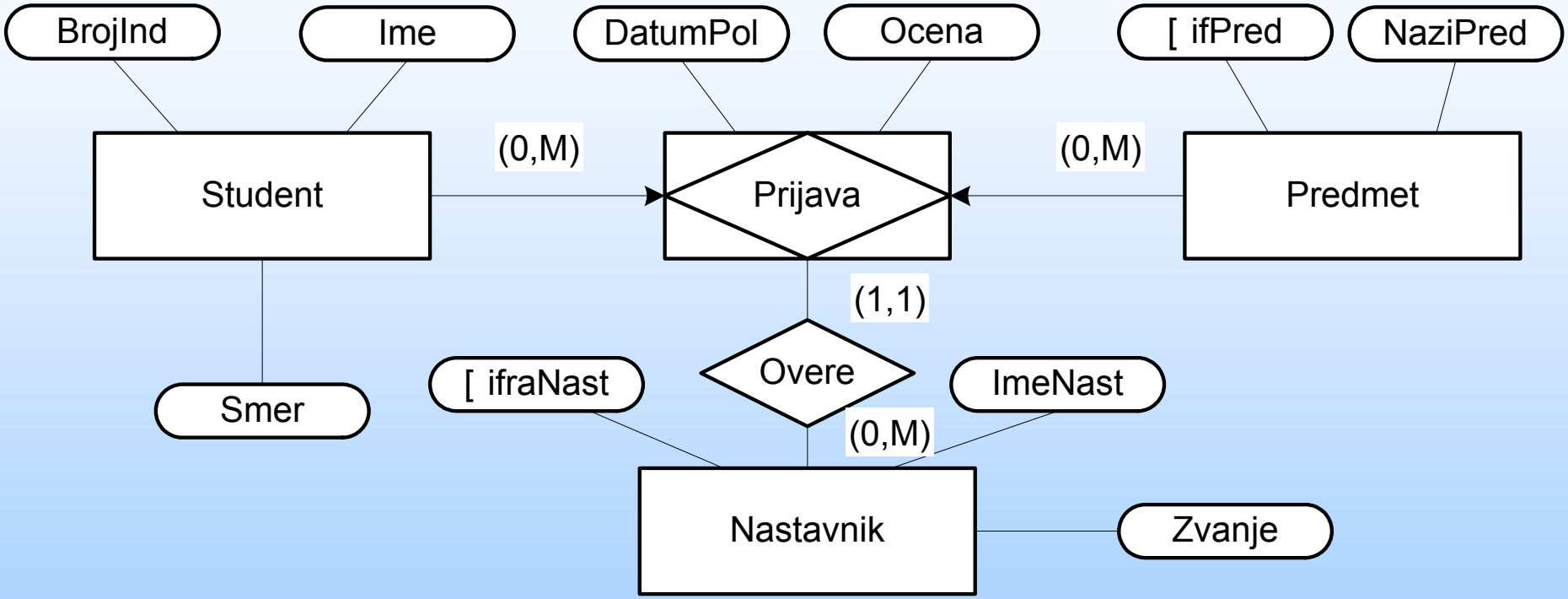
ATRIBUT	DOMEN	OGRANIČENJE
Mlb	long	NotNull And Substring(1,2) Between 1,31 And Substring(3,4) Between 1,12 ....
NazivJ	string	In(Srpski, Ruski, Engleski, Nemački)
Adresa	string	
Datum	date	
Starost	short	Between 15,65
ŠifOd	short	Moduo_11

Tabela 2.1. Definicija atributa sa ograničenjima

GENERALIZACIJA - SPECIJALIZACIJA

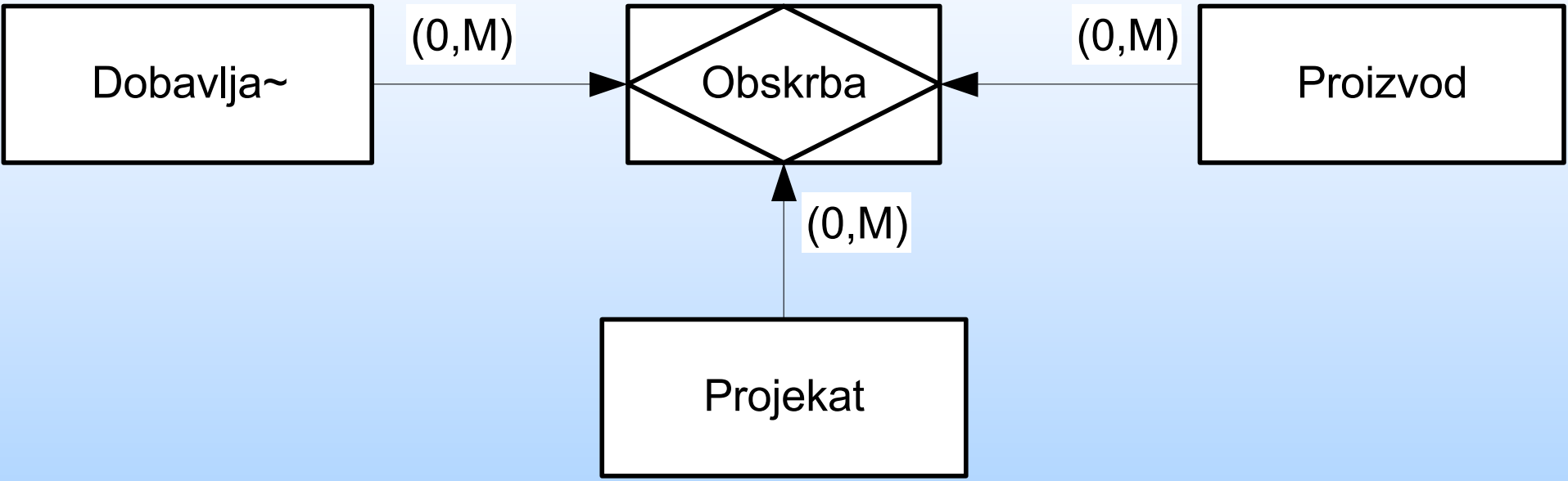


AGREGACIJA I DEKOMPOZICIJA

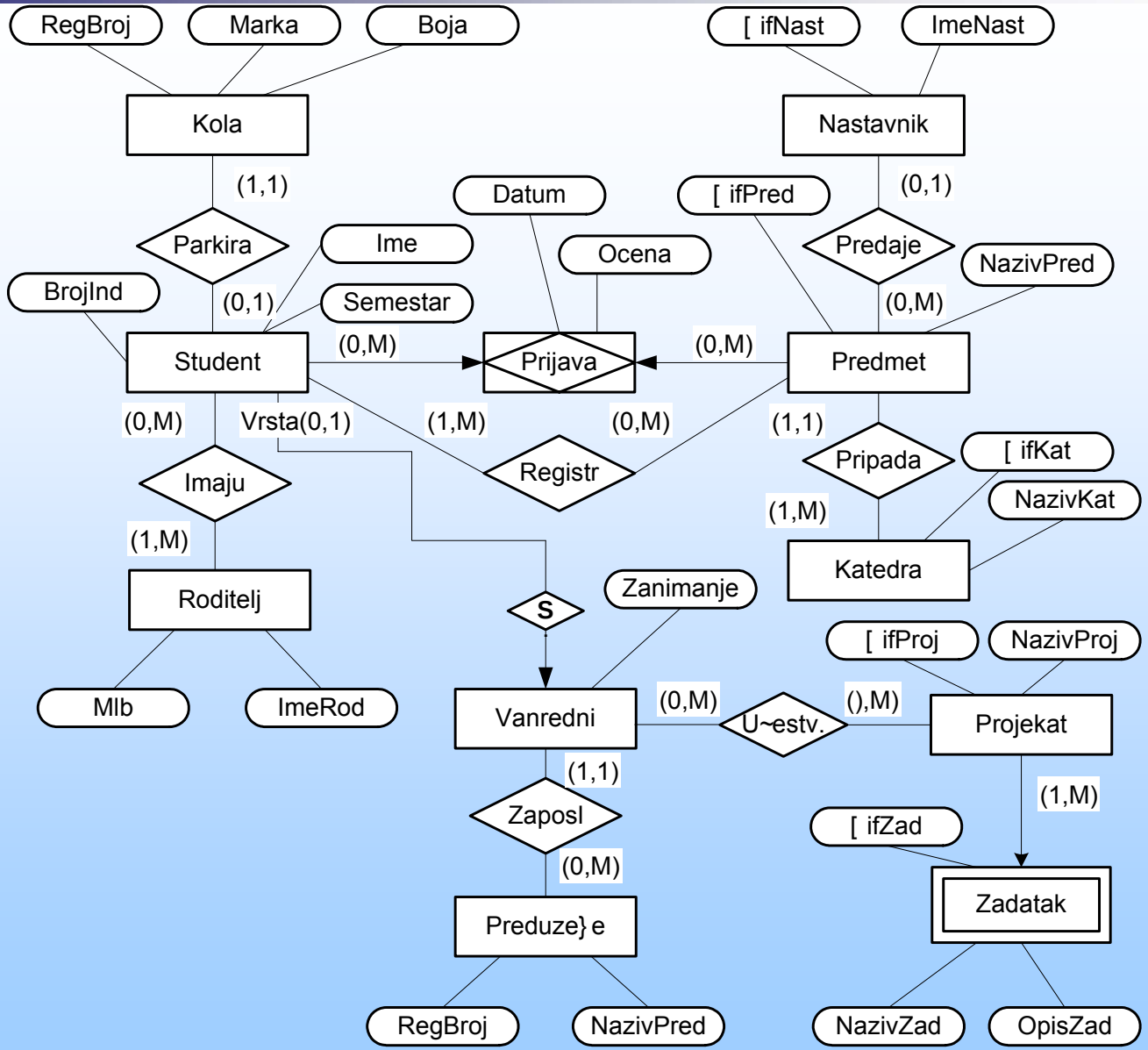


Slika 2.5. Agregacija (dekompozicija)

AGREGACIJA I DEKOMPOZICIJA



Slika 2.6. Višestruke veze između objekata – agregacija



DOV za fakultetski IS

OGRANIČENJA

- Strukturna ograničenja (ograničenja na preslikavanja), koja su prikazana na samom modelu;
- Vrednosna ograničenja (ograničenja na vrednosti atributa):
  - Prosta ograničenja na vrednosti pojedinačnih atributa
  - Složena ograničenja koja povezuju vrednosti više atributa istog i/ili različitih objekata.
  - Moguće je definisati formalni jezik za iskazivanje vrednosnih ograničenja.

Jezik za iskazivanje vrednosnih ograničenja

Za iskazivanje vrednosnih ograničenja koriste se formule tzv. ***Objektnog računa***. Pod ***Objektnim računom*** se ovde podrazumeva ***Predikatski račun prvog reda*** u kome promenljive uzimaju vrednosti iz definisanih klasa objekata u sistemu. Činjenica da promenljiva uzima vrednost iz neke klase objekta označava se na sledeći način:

x : O, gde je x pojavljivanje, a O klasa nekog objekta, odnosno promenljiva x uzima vrednosti iz skupa pojavljivanja objekta O.

x' označava vrednost promenljive pre operacije ažuriranja, dok x označava vrednost promenljive posle operacije ažuriranja. Obe vrednosti su neophodne da bi se iskazala dozvoljena promena stanja u modelu podataka.

Takozvana "dot notacija" se koristi da poveže promenljive iz dve klase objekata preko nekog preslikavanja definisanog u modelu.

Iskaz

- x : O.P, označava da x (odnosno x') uzima vrednosti iz klase koja je kodomen preslikavanja P klase O.

Atomske formule, odnosno simboli koji se koriste da označe neki atomski predikat su:

- x.A Θ y.B gde su x i y promenljive (objekti), A i B su atributi tipova objekata O_1 i O_2 iz čijih pojavljivanja, respektivno, promenljive x i y uzimaju vrednosti (x : O_1, y : O_2), a Θ je operacija poređenja (na primer <, >, =, Between,...), definisana nad domenom atributa A i B (A i B moraju biti definisani nad istim domenom).

- 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 objektnog računa (ddf) formiraju se preko sledećih produkcionih pravila (sintakse):

ddf:: = atom |
(ddf) |
Not ddf |
ddf **And** ddf |
ddf **O**r ddf |
Exists naziv-_promenljive (ddf) |
Foreach naziv_promenljive (ddf) |
If ddf **Then** ddf

Ograničenje se definiše na sledeći način:

definicija_promenljivih naziv_ograničenja
(lista_promenljivih) : = ddf;

Na primer:

x : Student
Ogr1 (x) := x.Semestar **Between** 1, 9;

Ako se usvoji konvencija da promenljiva dobije naziv klase iz koje uzima vrednost, tada se ona ne mora deklarisati. Po ovoj konvenciji, gornje ograničenje bi bilo:

Ogr1 (Student) : = Student.Semestar **Between** 1 i 9;

Primeri:

(1) Student petog ili višeg semestra mora da ima više od 10 položenih ispita.

Ogr2(Student,Prijava) : =
Foreach Student (**If** Student.Semestar >= 5
Then Card (Student.Prijava) >10);

Ovde je uvedena funkcija Card (S) koja daje broj elemenata skupa S

Primeri:

(2) Student trećeg semestra je položio Matematiku.

Ogr3(Student,Prijava ,Predmet) : = **Foreach** Student
(**If** Student.Semestar = 3
Then Exists Student.Prijava
(Student.Prijava.Predmet. NazivPr ='<Matematika'));

Primeri:

(3) Pretpostavimo da u modelu na Slici 2.7 Student ima i izvedeni atribut ProsOc (prosečna ocena). Tada očigledno važi sledeće ograničenje:

Ogr4(Student,Prijava) : =
Foreach Student (Student.ProsOc =
Suma (Student. Prijava.Ocena) / **Card** (Student. Prijava);

Primeri:

4) Ocena studenta iz nekog predmeta ne može opadati:

x:Prijava, x' : Prijava
Ogr5(x, x', Student, Predmet)=
Foreach x **Foreach** x'
(**If** x.Student.BrojInd = x'.Student. BrojInd **And**
x.Predmet.ŠifPred = x'. Predmet.ŠifPred
Then x'.Ocena x.Ocena)

OPERACIJE

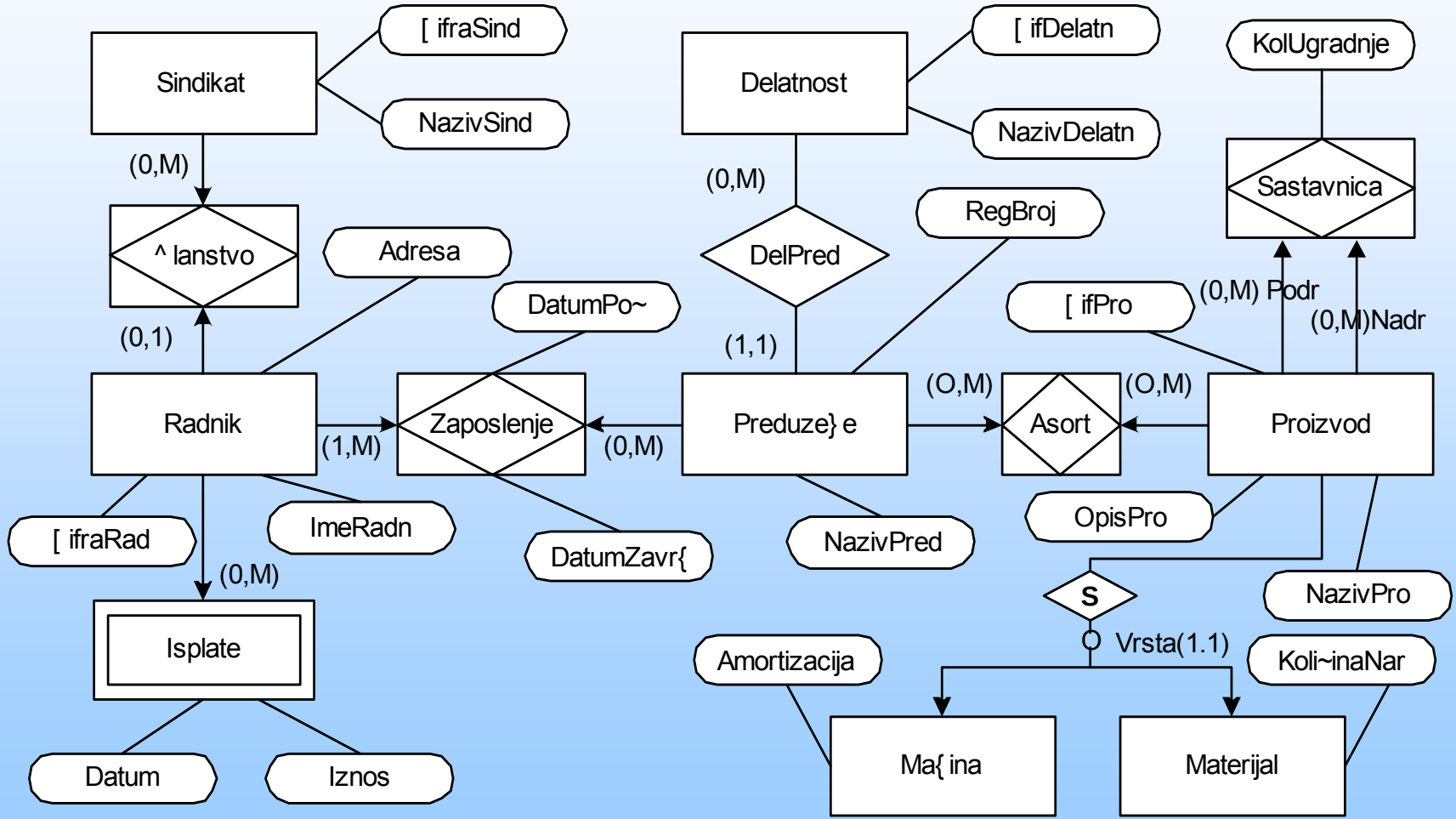
Očigledno je da se u MOV mogu definisati sledeće operacije održavanja baze podataka, analogne operacijama u Mrežnom modelu: ***Ubacivanje*** (**Insert**) novog pojavljivanja objekta u klasu, ***Izbacivanje*** (**Delete**) pojavljivanja objekta iz klase, ***Ažuriranje*** (**Update**) odnosno izmena vrednosti nekog atributa datog pojavljivanja objekta neke klase, ***Povezivanje*** (**Connect**) pojavljivanja O1 klase A sa pojavljivanjem O2 klase B, ***Razvezivanje*** (**Disconnect**) pojavljivanja O1 klase A od pojavljivanja O2 klase B i ***Prevezivanje*** (**Reconnect**) pojavljivanja O1 klase A od pojavljivanja O2 klase B.

DINAMIČKA PRAVILA INTEGRITETA

- Pravila integriteta definišu dozvoljena stanja i dozvoljene prelaze sistema iz stanja u stanje; čini ga trojka <Ograničenje, Operacija, Akcija> preko koje se iskazuje koje se akcija preduzima kada neka operacija naruši definisano ograničenje. Opšta sintaksna konstrukcija za iskazivanje pravila integriteta je:

```
CREATE  INTEGRITY RULE <naziv_pravila>
ograničenje | naziv-ograničenja
ON ATTEMPTED VIOLATION akcija
```

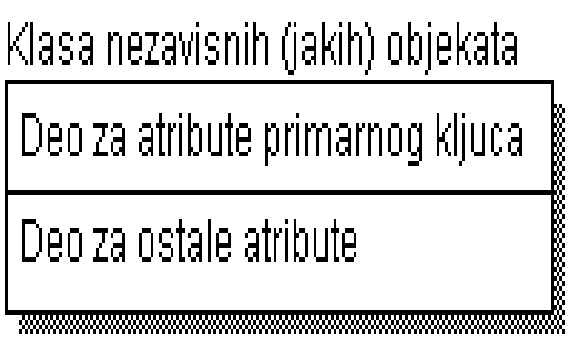
Strukturna dinamicka pravila integriteta i fizički MOV

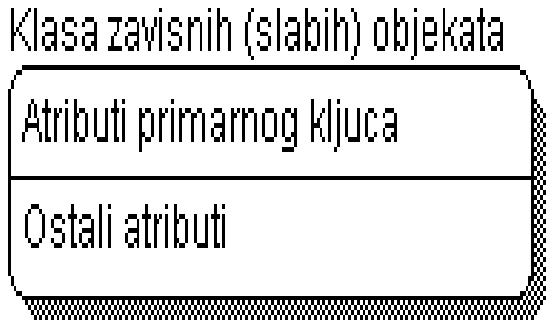


Strukturna dinamicka pravila integriteta i fizički MOV

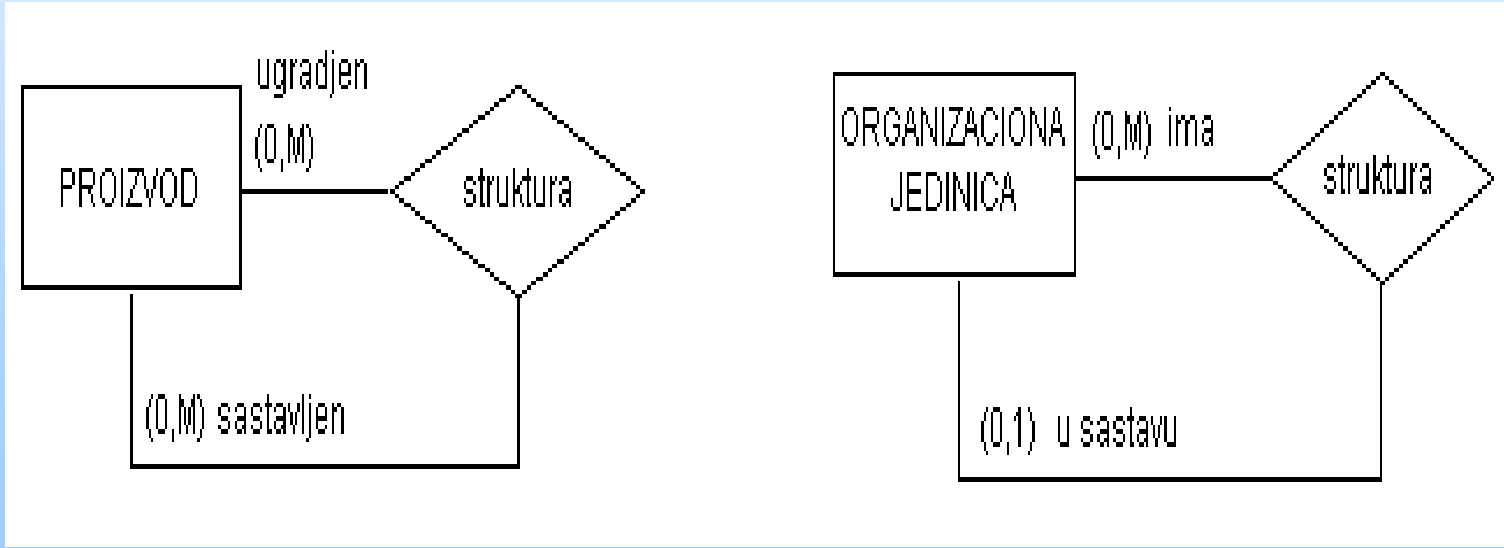
Operacija	Preslikavanje	Opcija
Insert Radnik	Radnik --> Zaposlenje	Cascades
	Radnik --> Članstvo	-
	Radnik --> Isplate	-
Delete Radnik	Radnik --> Zaposlenje	Cascades
	Radnik --> Članstvo	-
	Radnik --> Isplate	Cascades
Insert Članstvo	Članstvo --> Radnik	Restrict
	Članstvo --> Sindikat	SetNull
Delete Članstvo	Članstvo --> Radnik	-
	Članstvo --> Sindikat	-
Inset Zaposlenje	Zaposlenje --> Radnik	Restrict
	Zaposlenje --> Preduzeće	Restrict
Delete Zaposlenje	Zaposlenje --> Radnik	Cascade

VERZIJE MOV-a: IDEF1x standard

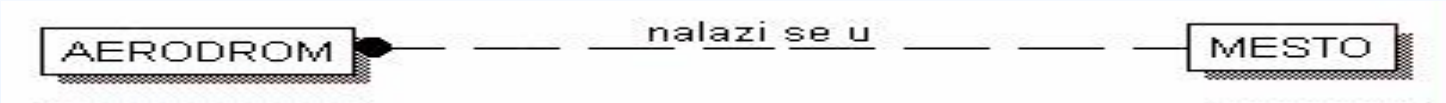


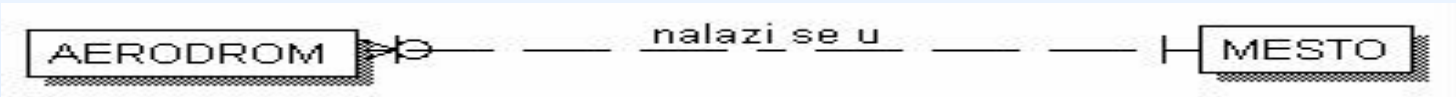


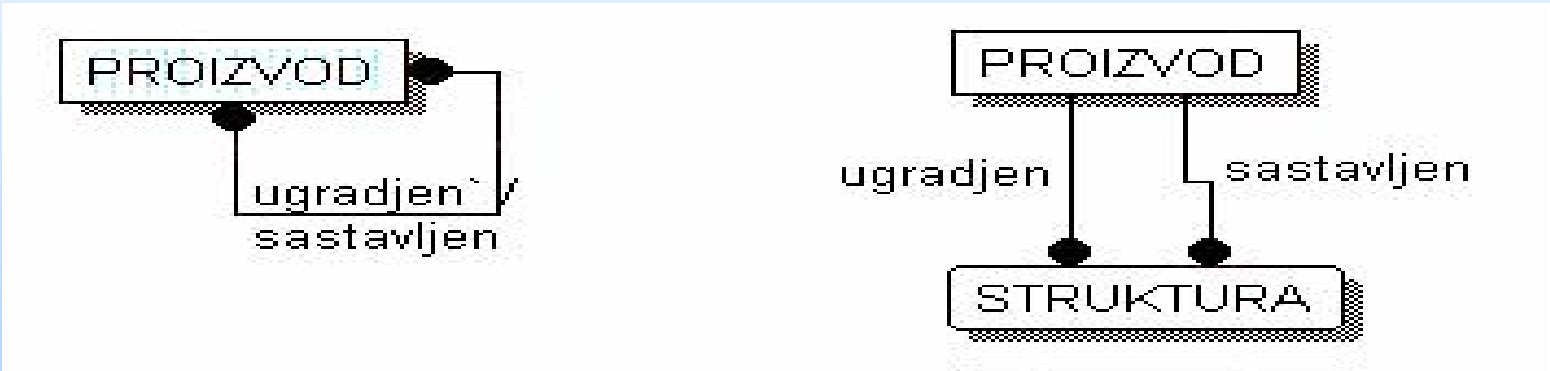
Veze po PMOV sintaksi

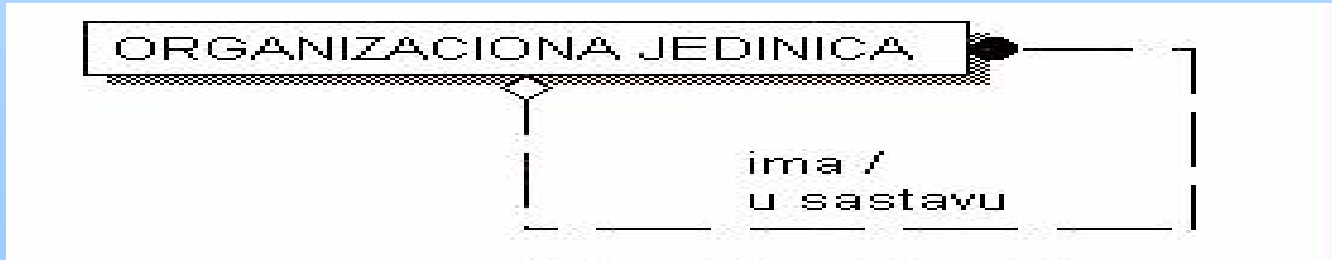


Veze po IDEF1x i IE standardu

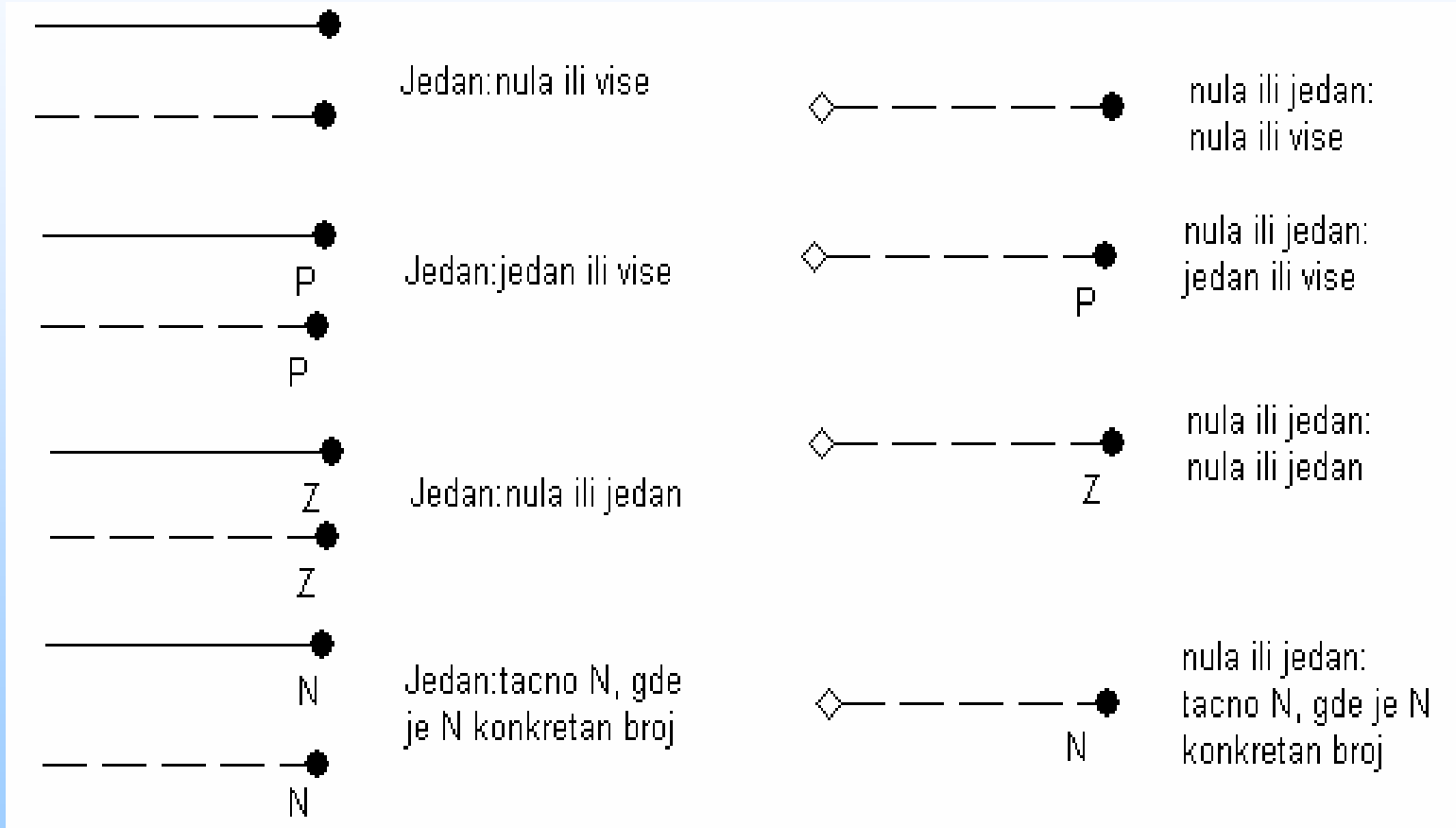




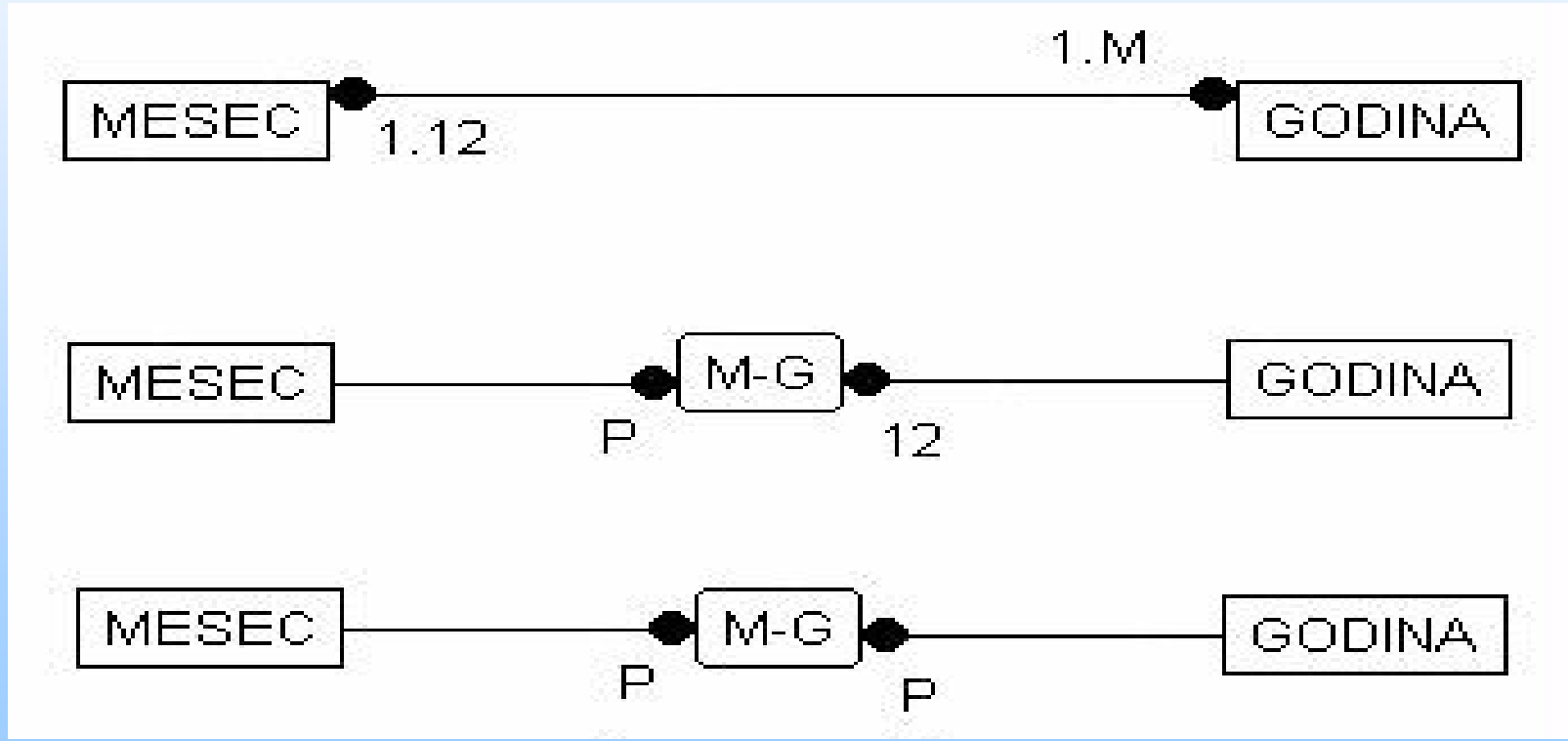




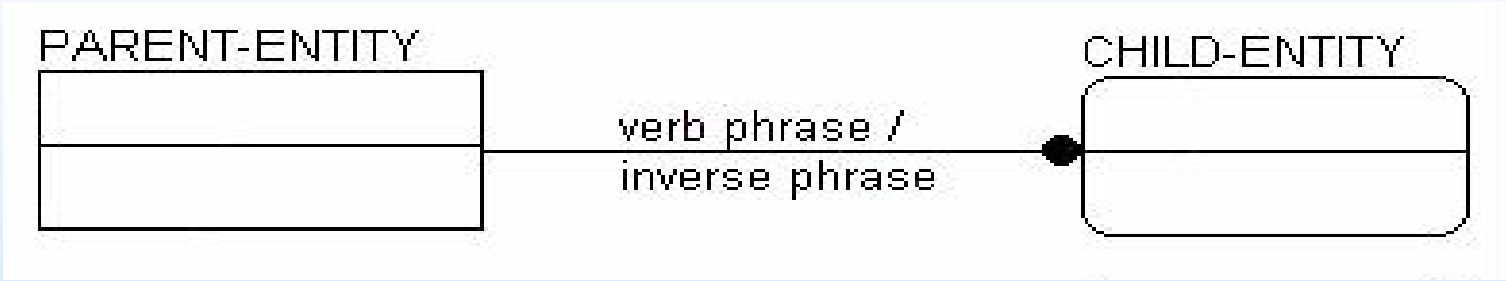
VERZIJE MOV-a: IDEF1x standard

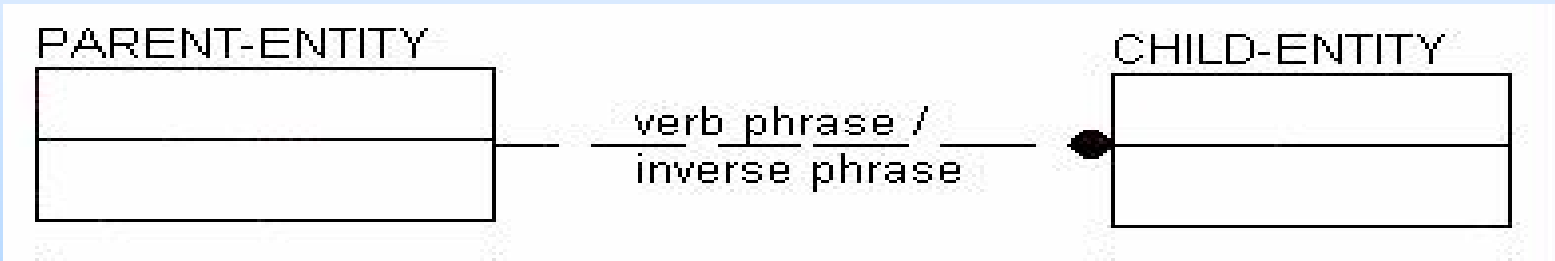


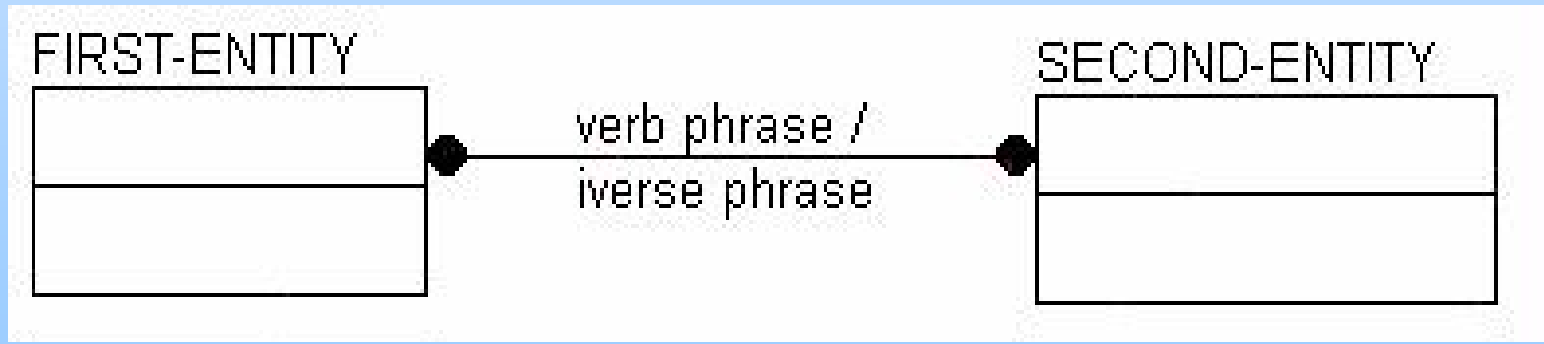
VERZIJE MOV-a: IDEF1x standard



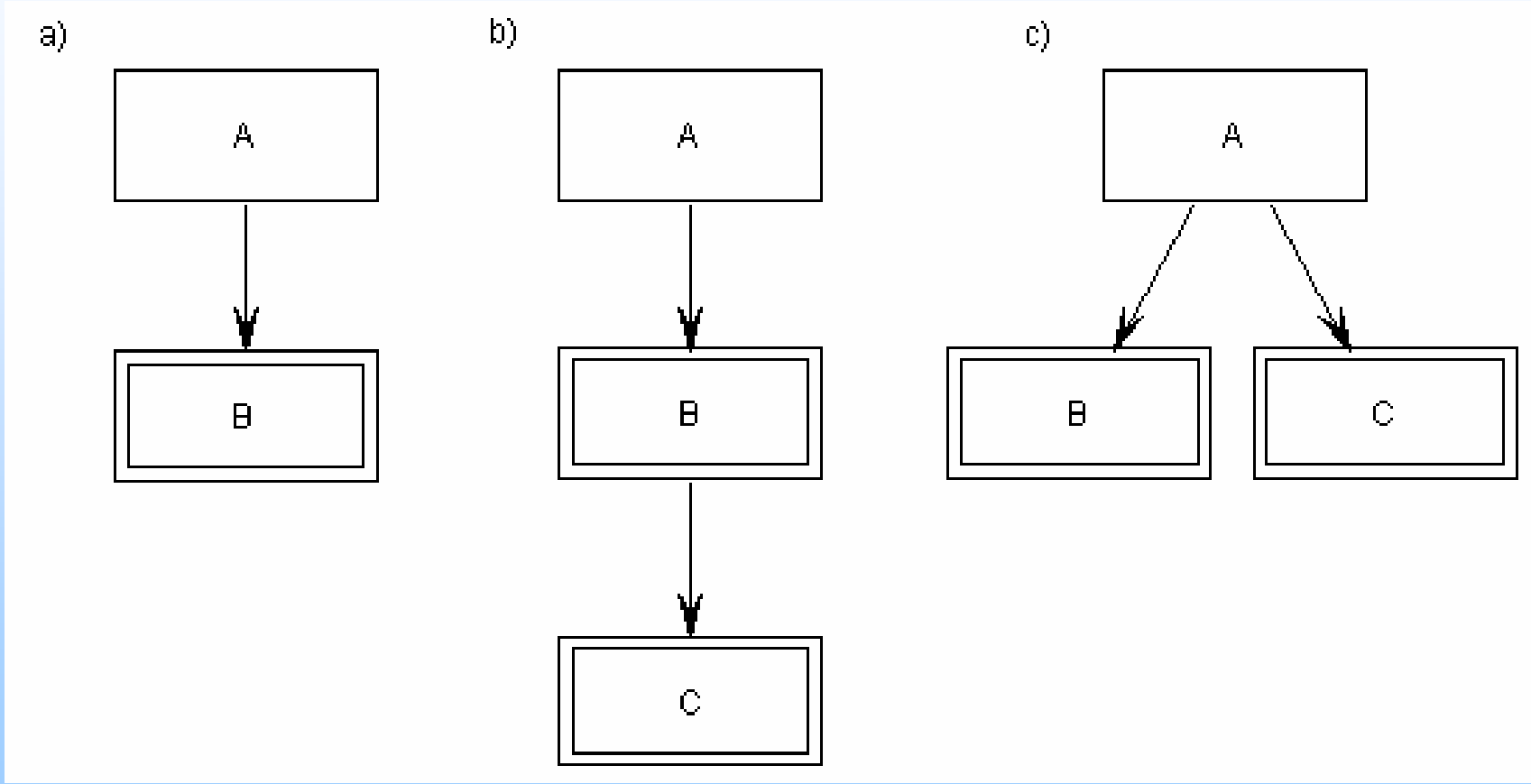
a) jedan:vise identifikujuća veza
b) jedan:vise neidentifikujuća veza
c) nespecificirana veza



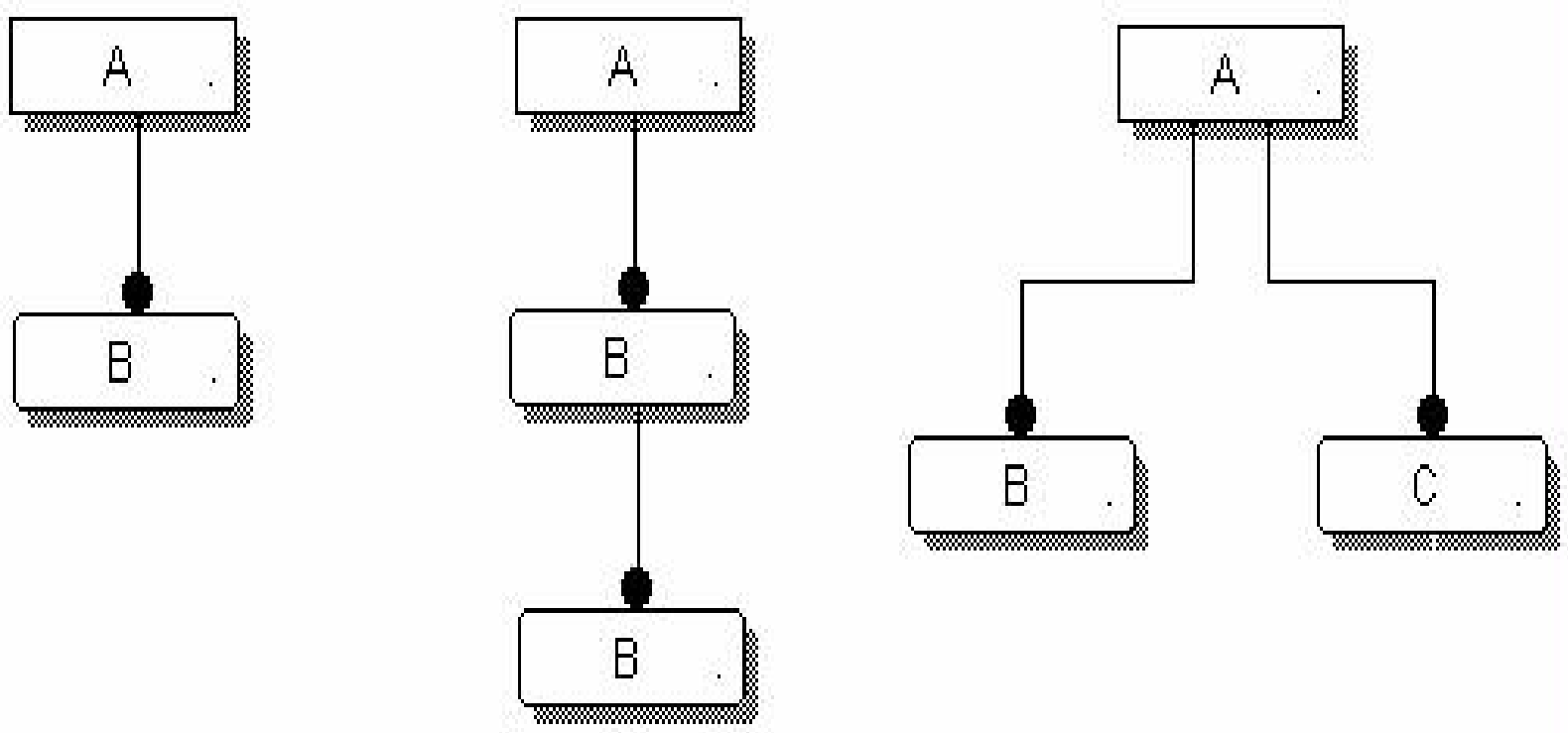




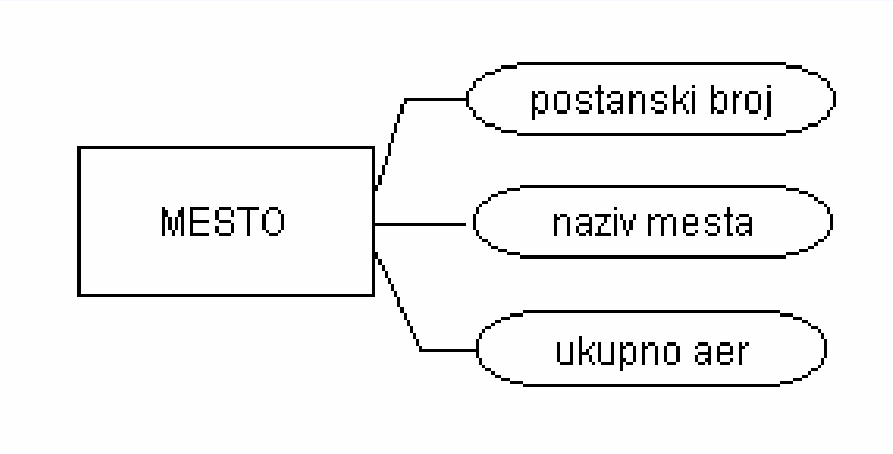
Objekti po PMOV sintaksi

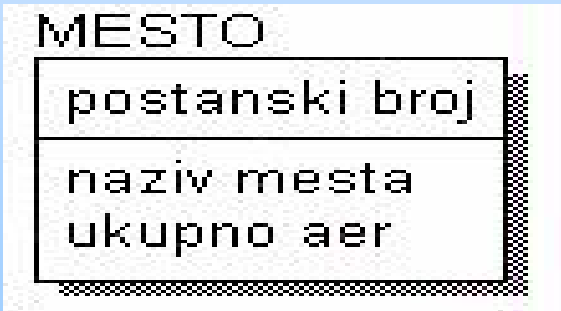


Objekti po IDEF1x standardu

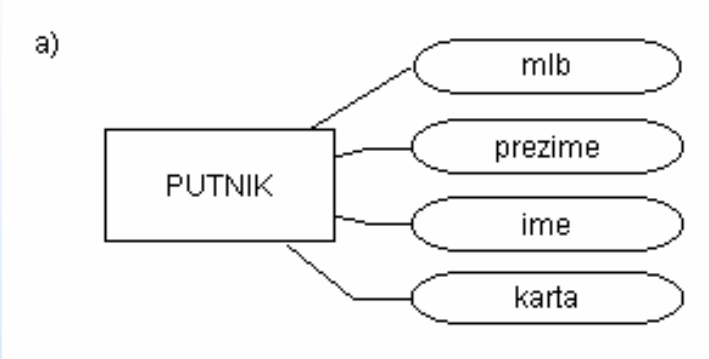


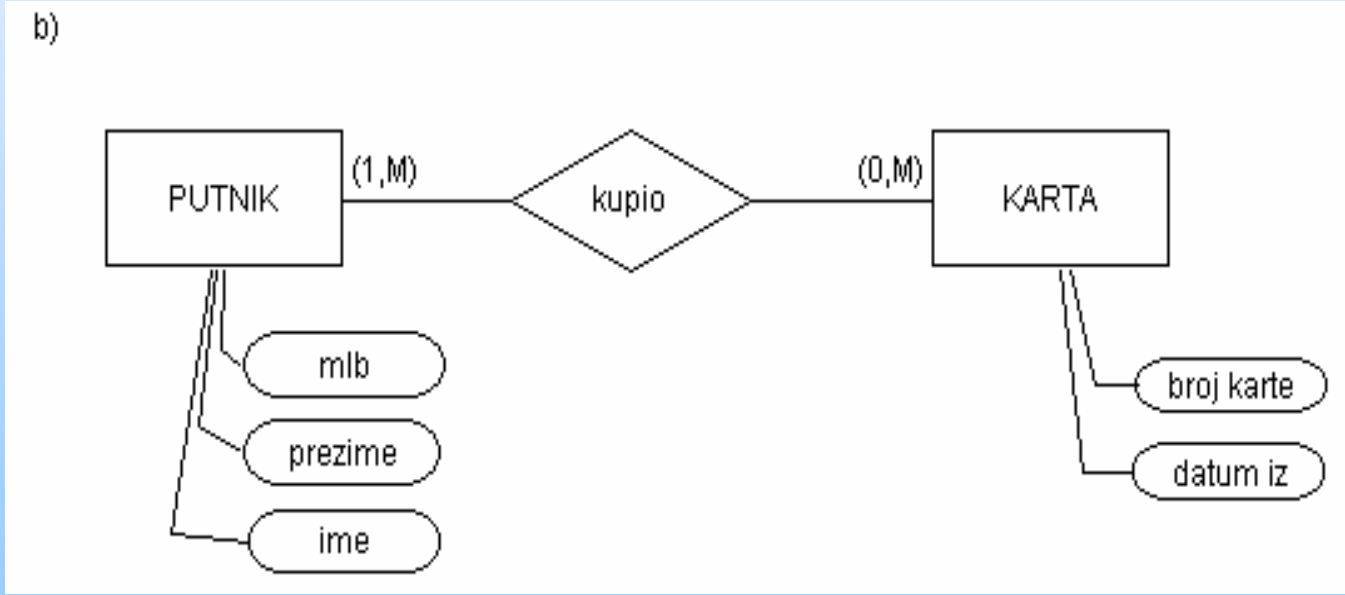
Atributi i domeni



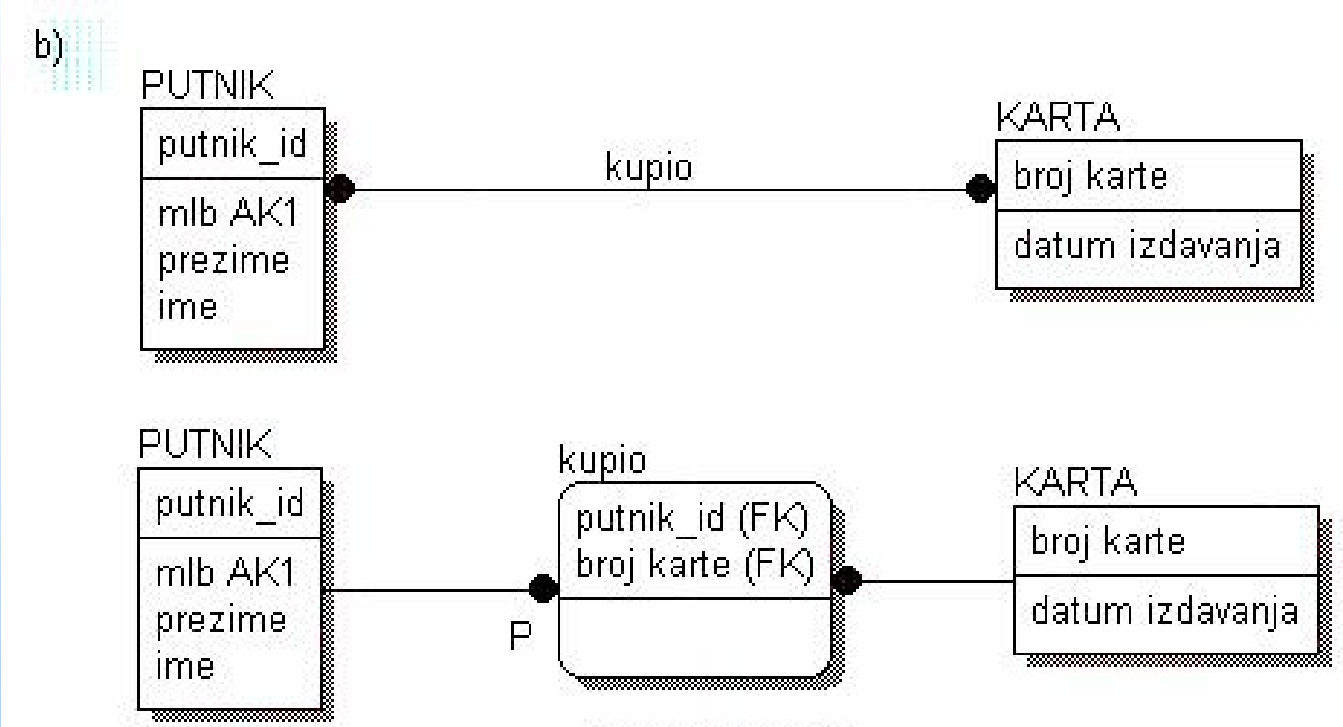


Viseznačni atributi

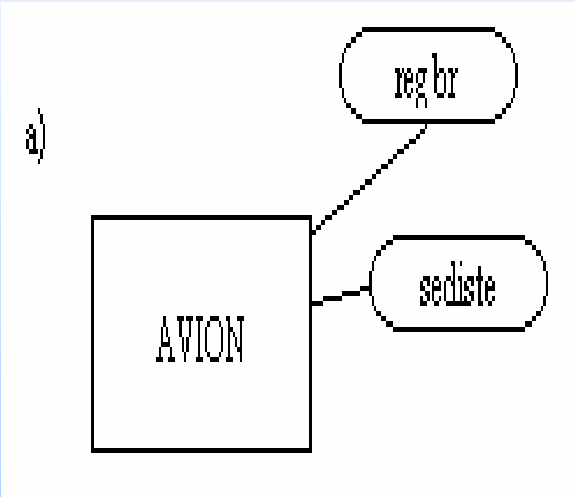


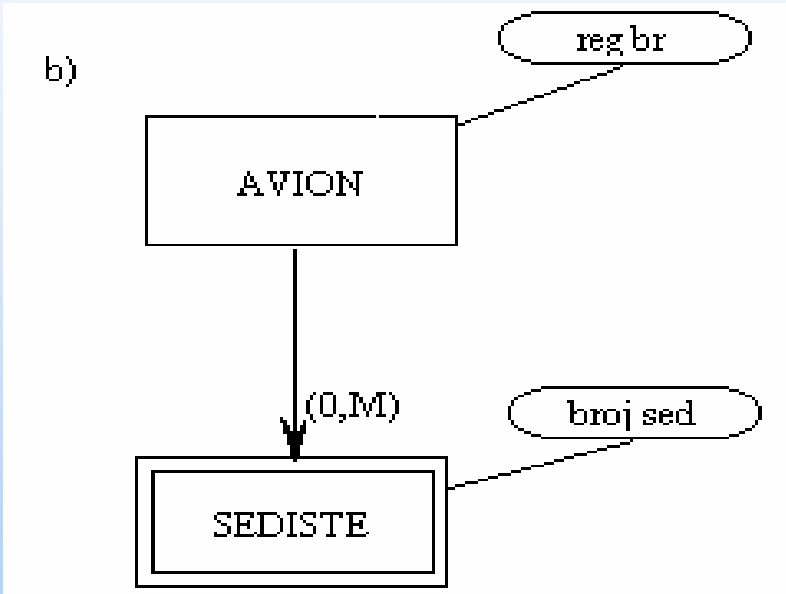


Viseznačni atributi

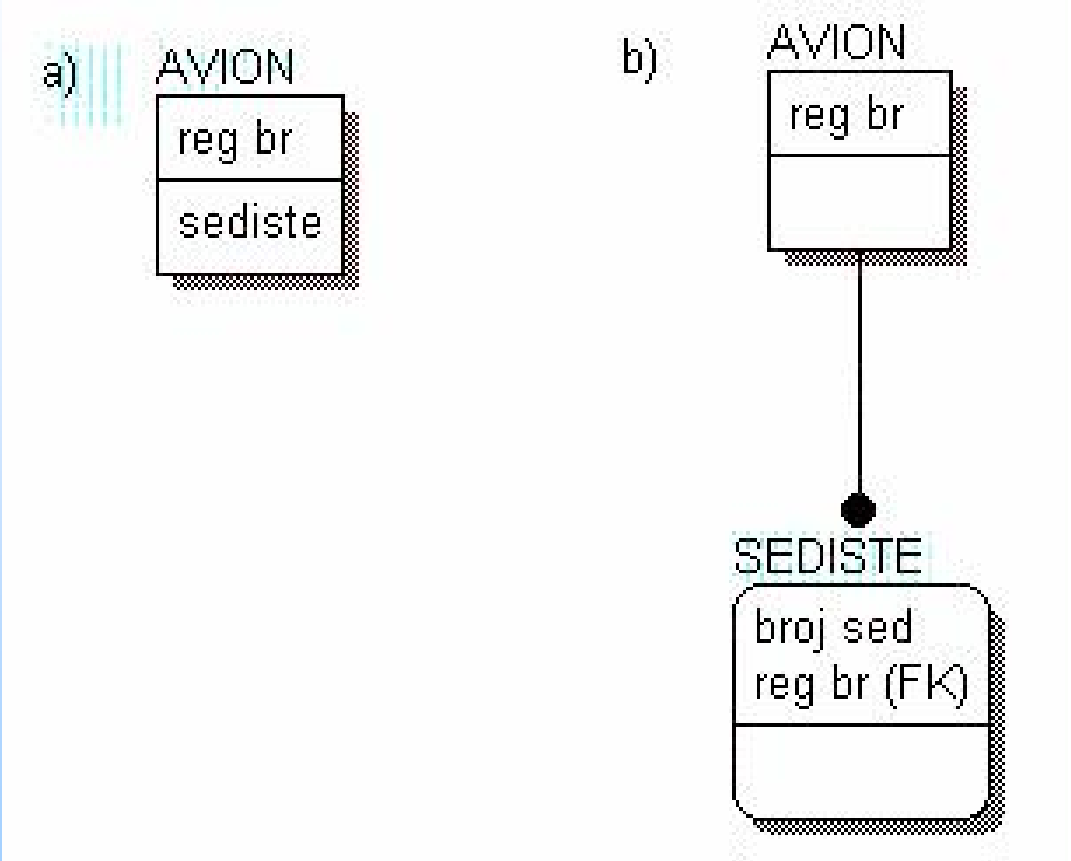


Viseznačni atributi

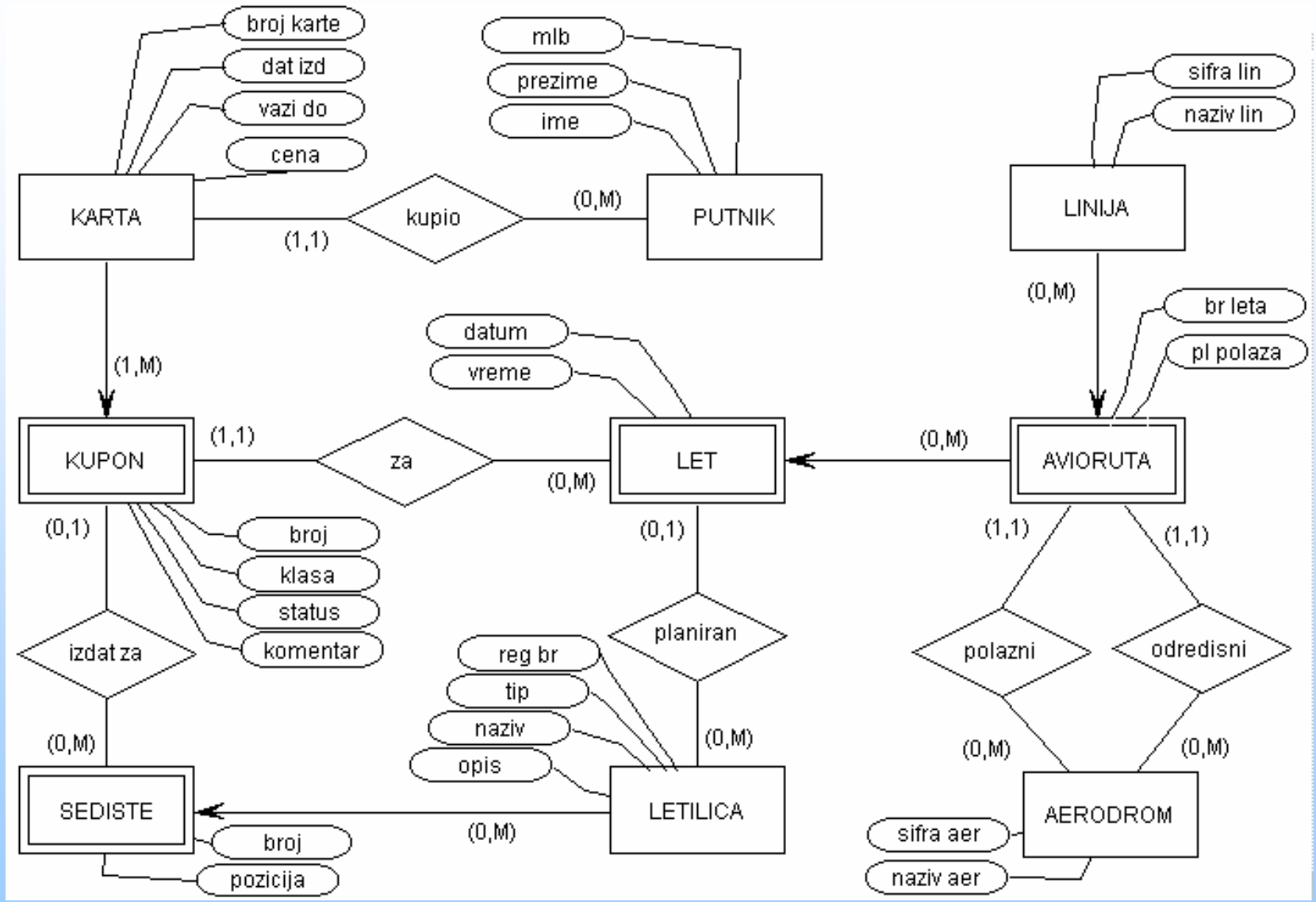


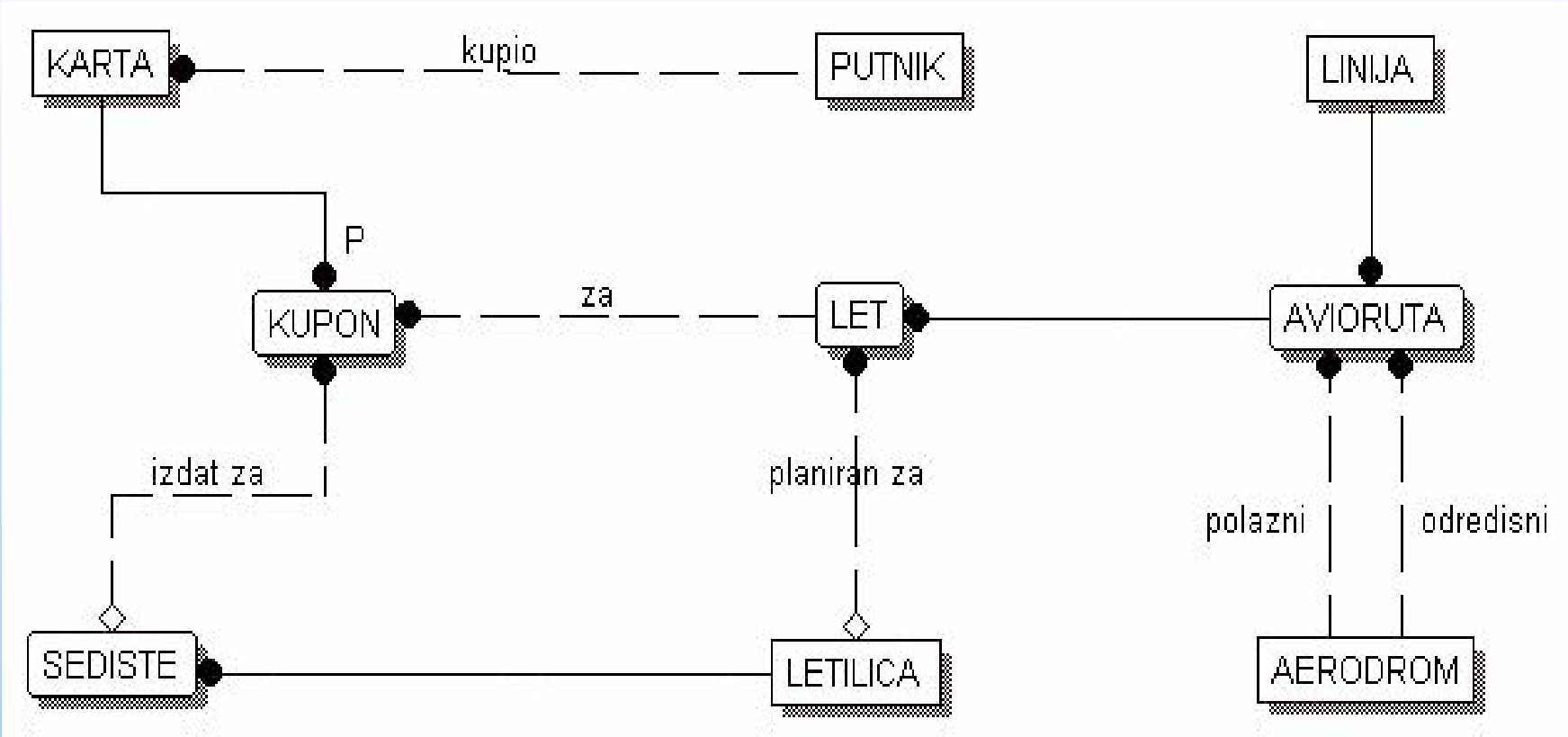


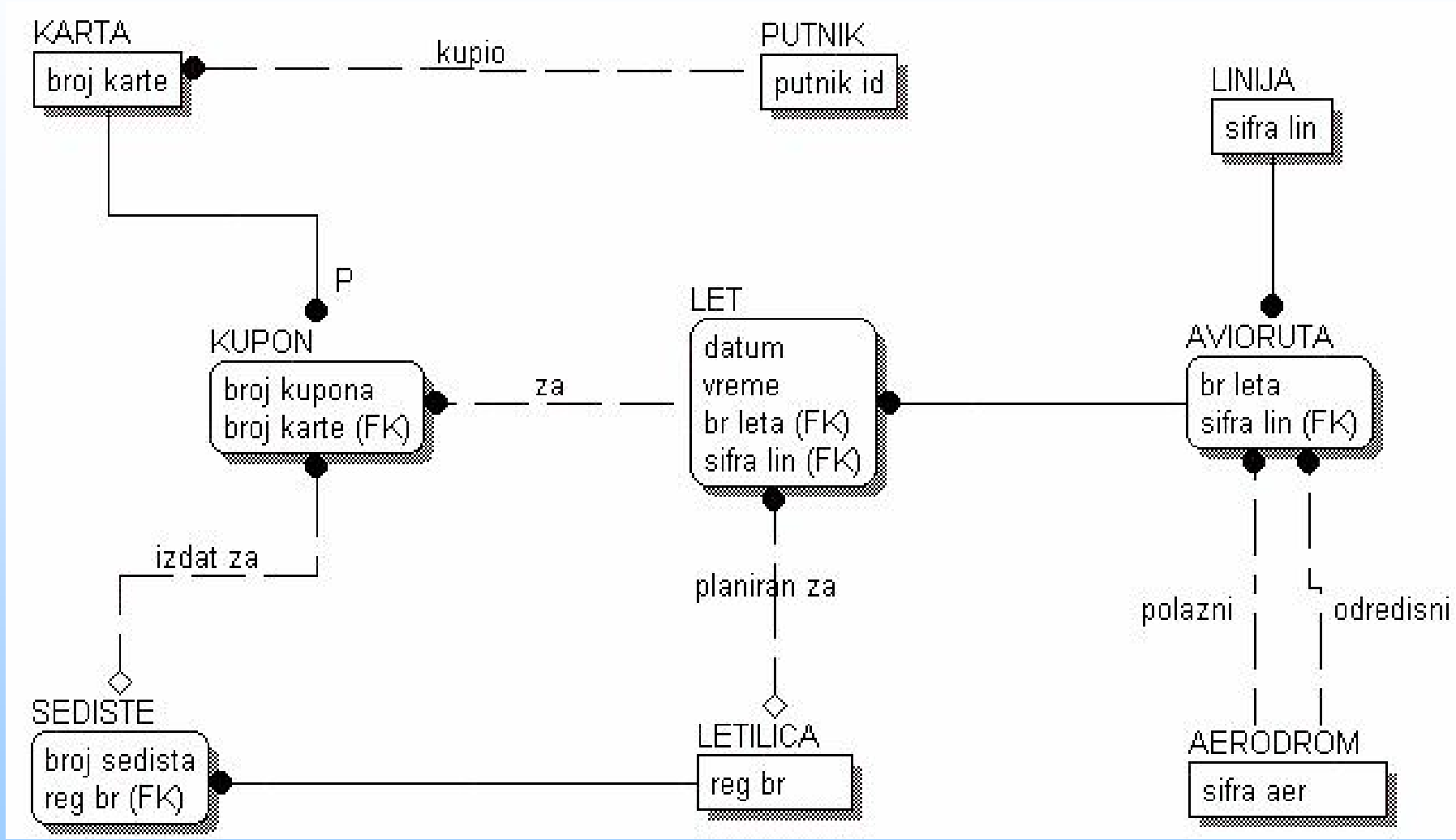
Viseznačni atributi

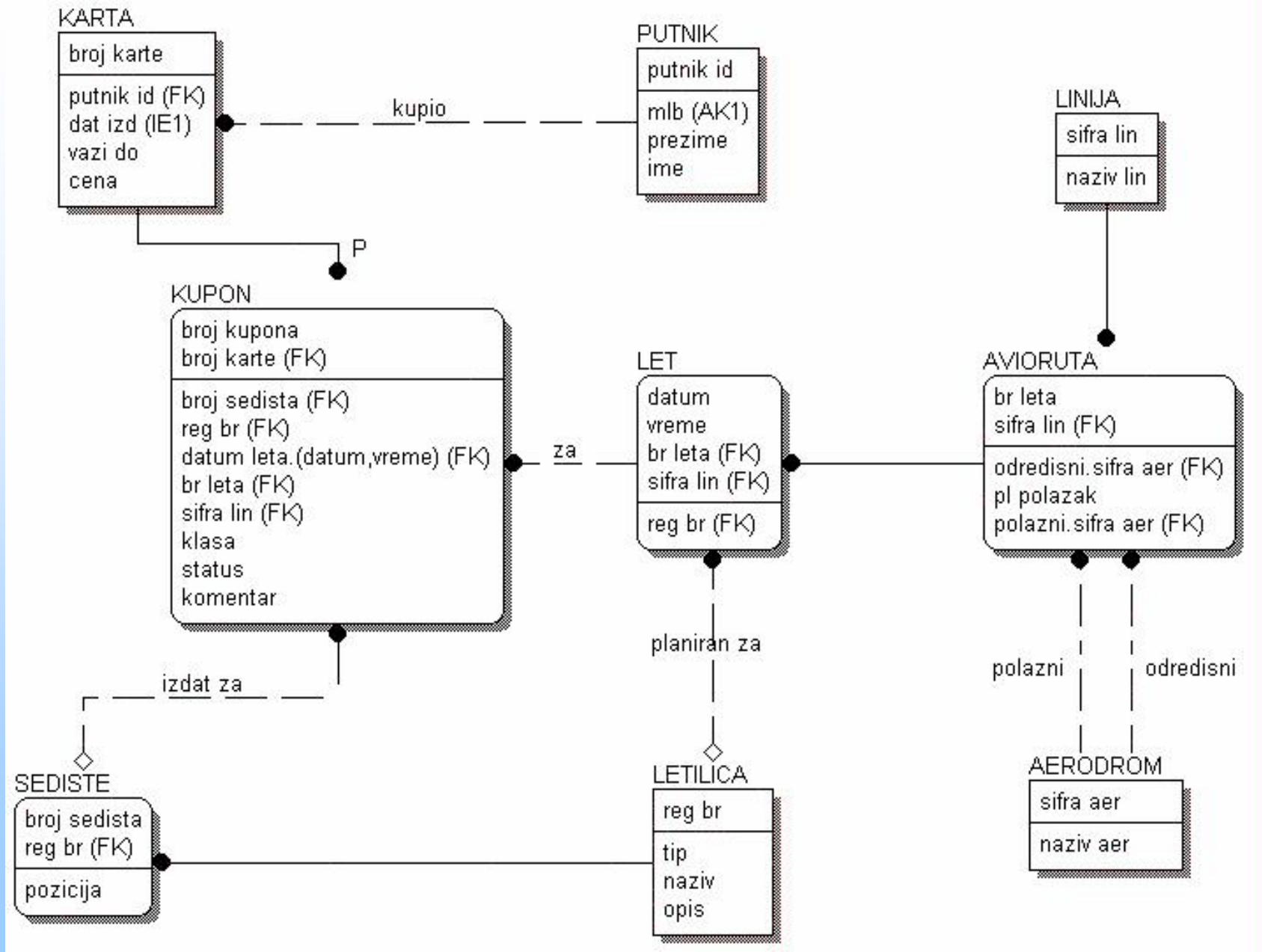


Primer 1: *Avionska karta za jednu standardnu avio-liniju može biti sastavljena od više kupona. Jedna linija može da uključi više letova na relaciji izmedju mesta polaska i mesta krajnjeg odredišta. Svaki avion obično ima nekoliko letova u toku dana (let je identifikovan preko datuma i vremena poletanja aviona). Karta sadrži podatke o avionskoj liniji, prezimenu i imenu putnika, mestu polazišta, mestu krajnjeg odredišta, datumu izdavanja, roku važenja i ceni. Kuponi karte sadrže identične podatke i podatke o pojedinačnim letovima izmedu polazišta i krajnjeg odredišta: mesto poletanja, mesto sletanja, osnovni podaci o avionu. broj leta, klasa sedišta, datum i vreme poletanja*.

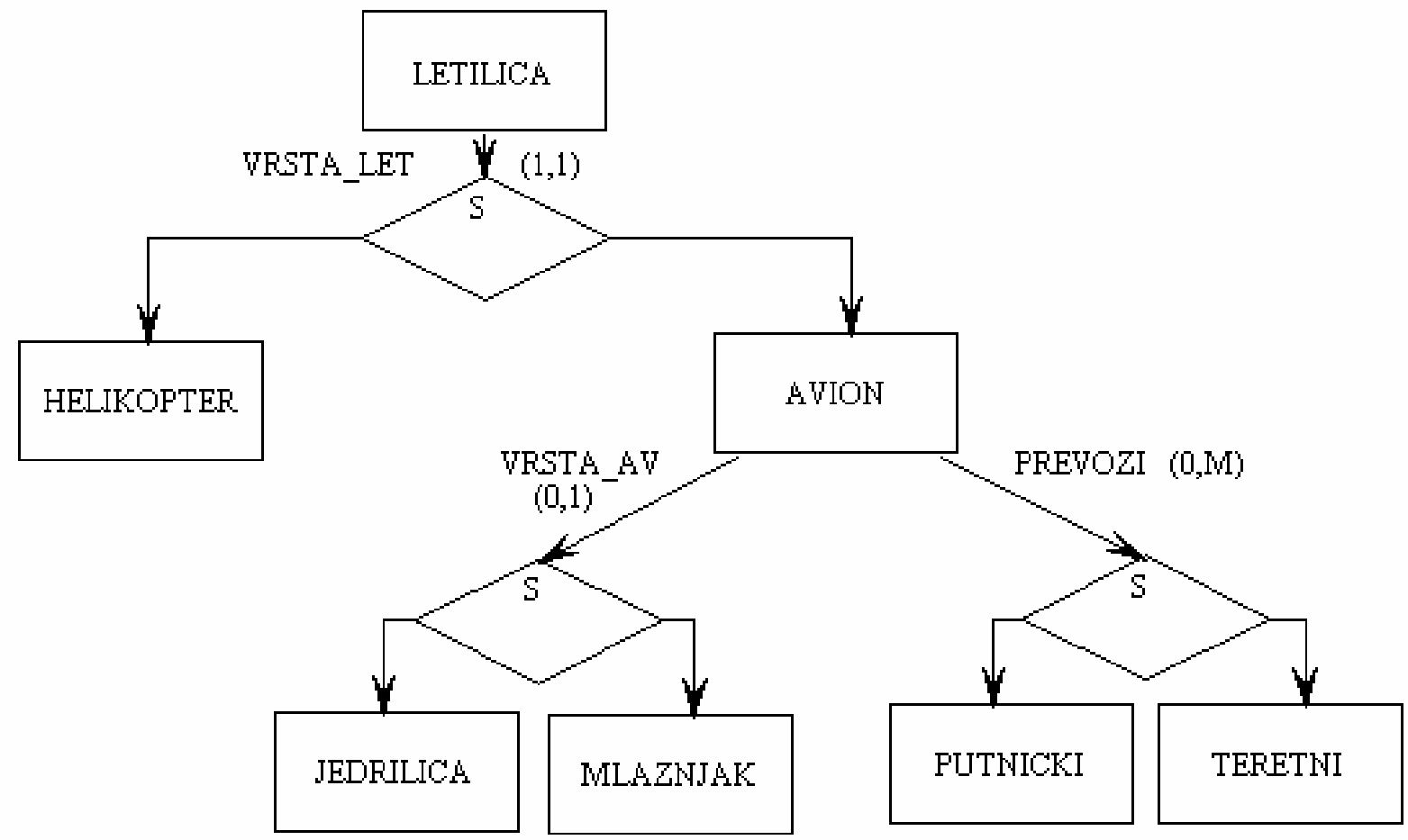

broj karte
dat izd
vazi do
cena
KARTA
kupio
(1,1)
(0,M)
PUTNIK
mlb
prezime
ime
sifra lin
naziv lin
LINIJA
(0,M)
br leta
pl polaza
(1,M)
datum
vreme
KUPON
(1,1)
za
(0,M)
LET
(0,M)
AVIORUTA
(0,1)
broj
klasa
status
komentar
(0,1)
(1,1)
(1,1)
izdat za
planiran
polazni
odredisni
reg br
tip
naziv
opis
(0,M)
(0,M)
(0,M)
(0,M)
SEDISTE
LETILICA
AERODROM
broj
pozicija
(0,M)
sifra aer
naziv aer


KARTA
kupio
PUTNIK
LINIJA
P
KUPON
za
LET
AVIORUTA
izdat za
planiran za
polazni
odredisni
SEDISTE
LETILICA
AERODROM

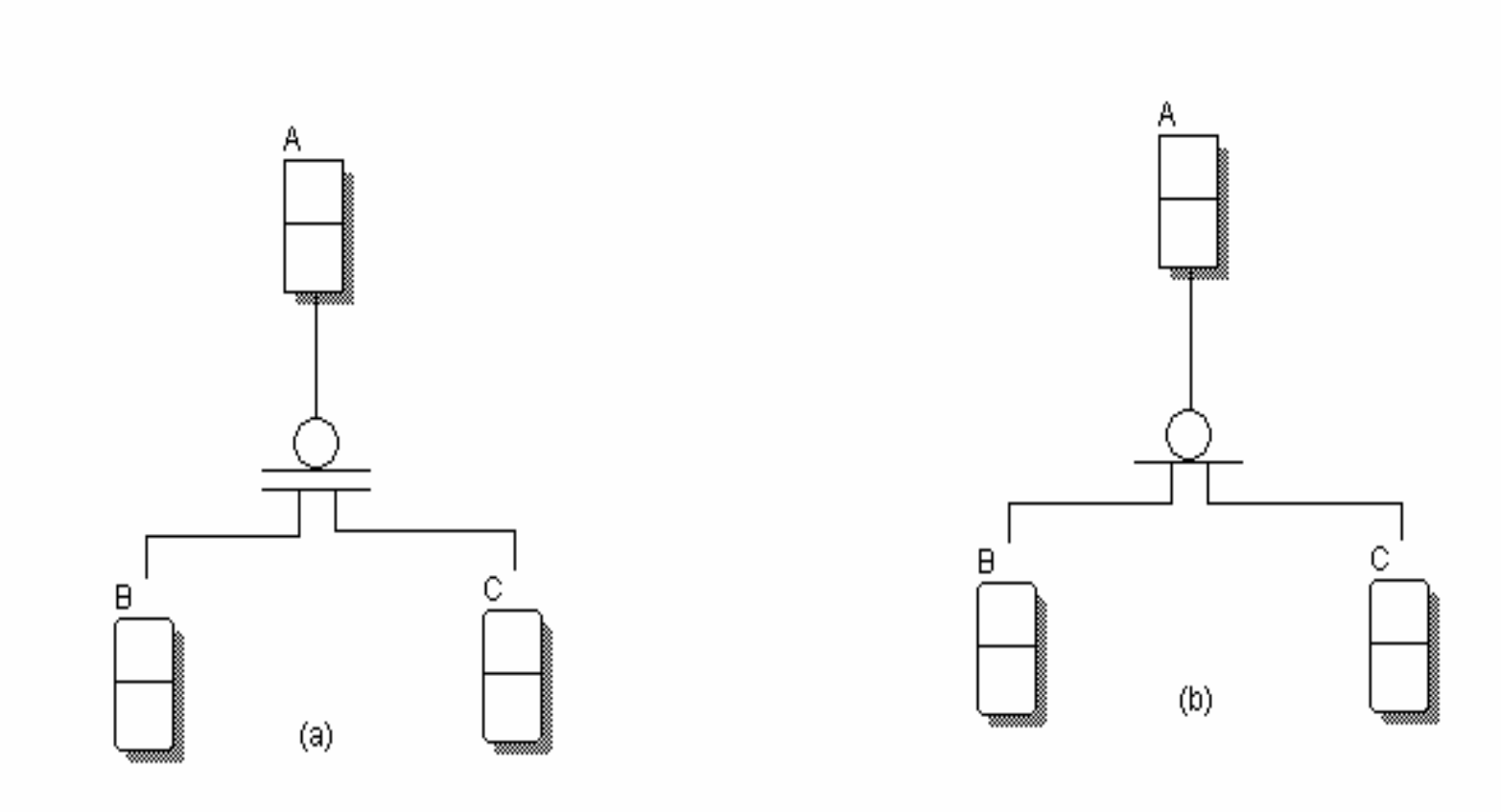

KARTA
broj karte
kupio
PUTNIK
putnik id
LINIJA
sifra lin
P
KUPON
broj kupona
broj karte (FK)
za
LET
datum
vreme
br leta (FK)
sifra lin (FK)
AVIORUTA
br leta
sifra lin (FK)
izdat za
planiran za
polazni
odredisni
SEDISTE
broj sedista
reg br (FK)
LETILICA
reg br
AERODROM
sifra aer


KARTA
broj karte
putnik id (FK)
dat izd (IE1)
vazi do
cena
kupio
PUTNIK
putnik id
mlb (AK1)
prezime
ime
LINIJA
sifra lin
naziv lin
P
KUPON
broj kupona
broj karte (FK)
broj sedista (FK)
reg br (FK)
datum leta.(datum,vreme) (FK)
br leta (FK)
sifra lin (FK)
klasa
status
komentar
za
LET
datum
vreme
br leta (FK)
sifra lin (FK)
reg br (FK)
AVIORUTA
br leta
sifra lin (FK)
odredisni.sifra aer (FK)
pl polazak
polazni.sifra aer (FK)
planiran za
polazni
odredisni
izdat za
SEDISTE
broj sedista
reg br (FK)
pozicija
LETILICA
reg br
tip
naziv
opis
AERODROM
sifra aer
naziv aer

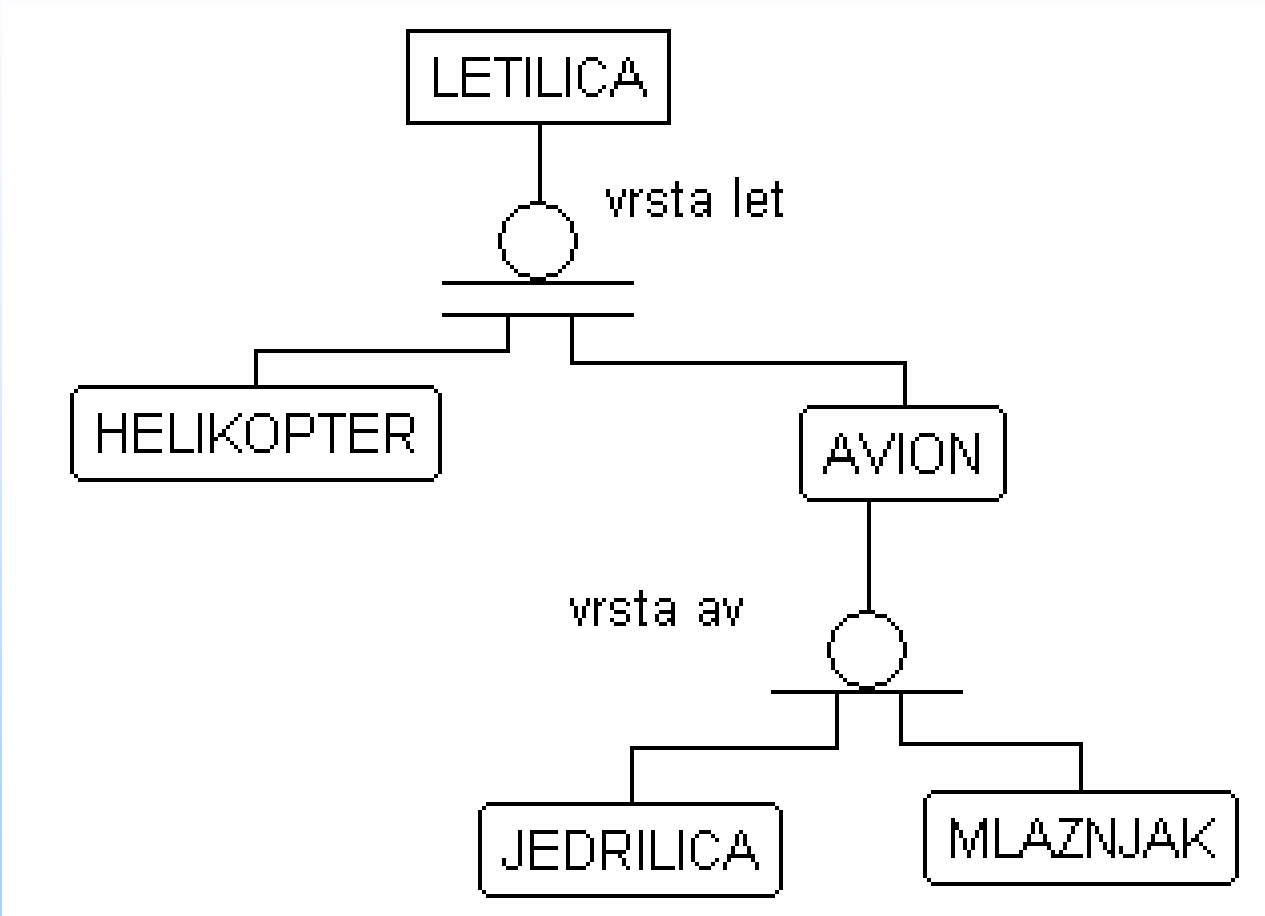
Generalizacija i specijalizacija



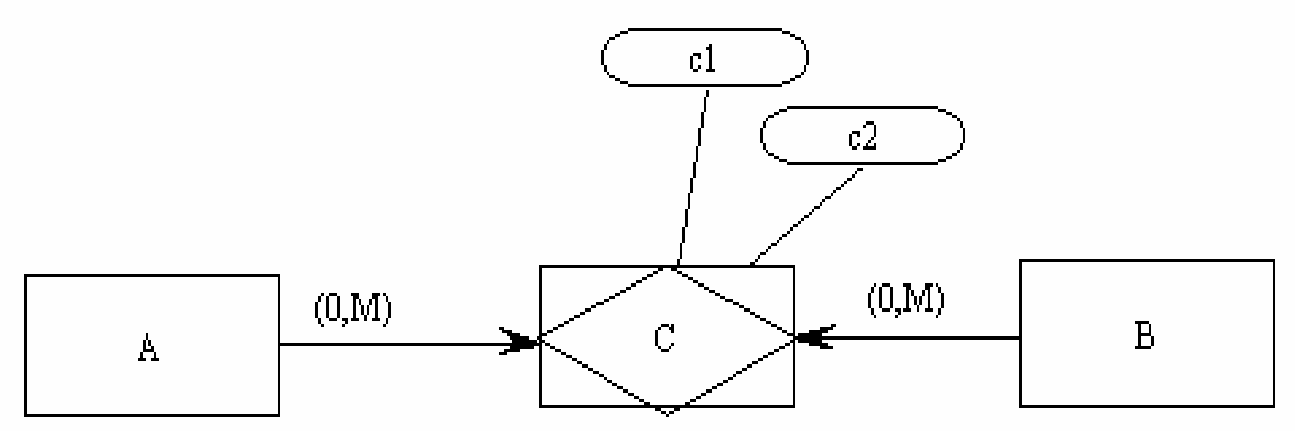
VERZIJE MOV-a: IDEF1x standard

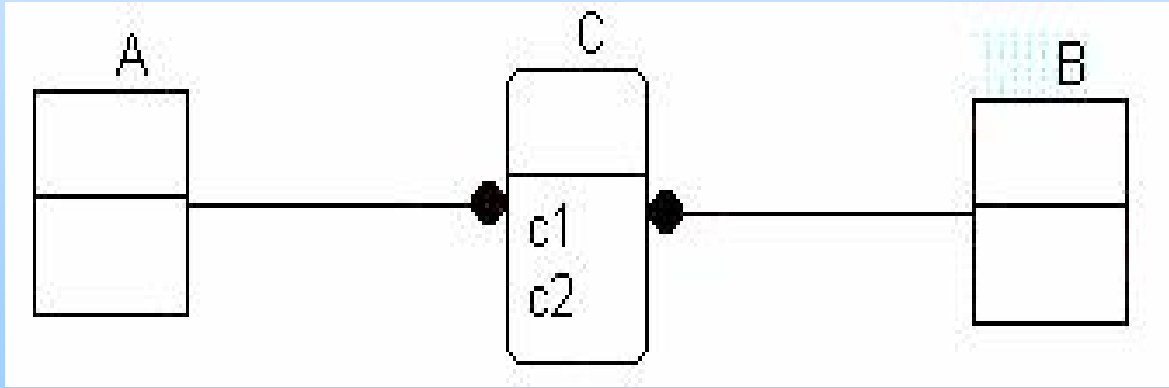


VERZIJE MOV-a: IDEF1x standard

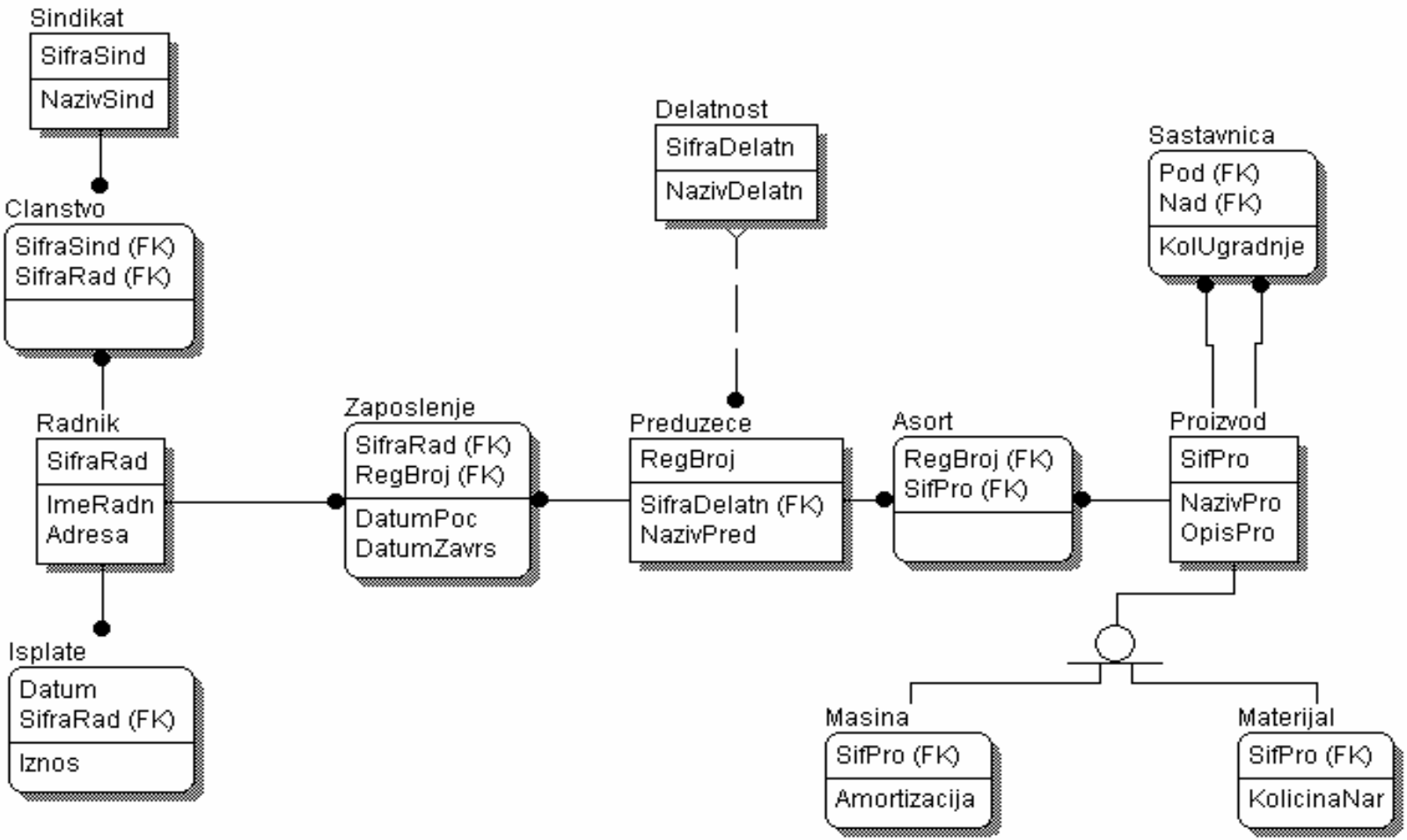


Agregacija





VERZIJE MOV-a: IDEF1x standard



RELACIONI MODEL

RELACIONI MODEL

Dve karakteristike čine relacioni model još uvek najpopularnijim i najšire primenjivanim:

- Struktura modela je veoma jednostavna, prihvatljiva svakom korisniku, jer relaciona baza podataka predstavlja skup tabela. I same operacije, koje iz skupa datih tabela generišu novu, su jednostavne i lako prihvatljive.

- Moguća je formalno-matematička interpretacija tabela. Određene vrste tabela se mogu tretirati kao matematičke relacije i zatim iskoristiti bogata teorijska osnova odgovarajućeg matematičkog aparata.

STRUKTURA RELACIONOG MODELA

- **Skup.** Pojam skupa je osnovni pojam u matematici i on se formalno ne definiše. Neformalno se može reći da je skup "objedinjavanje nekog mnoštva elemenata u celinu".

  Skup elemenata x, y, z,..., v se može označiti sa S = {x, y, z,..., v}.

- **Ekstenzija skupa** - Navodjenje svih elemenata skupa
- **Intenzija skupa** - Navodjenje osobina koje svaki element skupa mora da zadovolji:
  S = {x | P(x)},
  gde je P(x) uslov koji svi elementi treba da zadovolje

STRUKTURA RELACIONOG MODELA

- Dekartov (Kartezijanski) proizvod skupova. Neka je data kolekcija skupova $D_1, D_2, \ldots, D_n$ (ne neophodno različitih). Dekartov proizvod ovih n skupova

$$D_1 \times D_2 \times \ldots \times D_n$$

je skup svih mogućih uređenih n-torki

$$(d_1, d_2, \ldots, d_n),$$

tako da je $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$.

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

$A \times B = \{(1,4), (1,6), (1,8), (2,4), (2,6), (2,8), (3,4), (3,6), (3,8), (4,4), (4,6), (4,8)\}$.

STRUKTURA RELACIONOG MODELA

- Relacija koja najopštije predstavlja odnos izmedju elemenata nekih ne neophodno različitih skupova definiše se kao podskup Dekartovog proizvoda tih skupova.

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

(Podskup sadrži one n-torke Dekartovog proizvoda koje zadovoljavaju odnos koji relacija predstavlja)

Primer: Neka je na skupovima A i B iz Primera 1 zadata relacija

$$R \subseteq A \times B = \{(a,b) \mid a = b/2\}$$

Na osnovu definicije relacije očigledno je da je

$$R = \{(2,4), (3,6), (4,8)\}$$

STRUKTURA RELACIONOG MODELA

- **Domen relacije.** Skupovi D_1, D_2, ..., D_n se nazivaju domenima relacije R.
- Broj domena na kojima je definisana neka relacija se naziva **stepen relacije**.
- Kardinalnost relacije je broj n-torki u relaciji.

STRUKTURA RELACIONOG MODELA

- Atribut relacije se formalno može definisati kao preslikavanje:

Atribut: n_torka_relacije ---> Domen

odnosno kao par **(naziv domena, vrednost domena)**

Ovakva definicija atributa omogućava da se **relacija predstavi kao tabela.**

STRUKTURA RELACIONOG MODELA

STUDENT $\subseteq$ BrInd x Ime x Starost =
{<001, Miloš, 19>,<007, Ana, 19>, <035, Ana, 22>}

(REDOSLED N-TORKI JE BITAN)

BRIND	IME	STAROST
001	Miloš	19
007	Ana	19
035	Ana	35

STRUKTURA RELACIONOG MODELA: PRVA NORMALNA FORMA

- Definišu se sledeći uslovi koje tabela mora da zadovolji da bi bila relacija:

  (1) Ne postoje duplikati vrsta tabele;

  (2) Redosled vrsta nije značajan;

  (3) Redosled kolona nije značajan.

  (4) Sve vrednosti atributa u relacijama su atomske, ili drugim rečima, nisu dozvoljeni atributi ili grupe atributa "sa ponavljanjem"), odnosno nije dozvoljeno da vrednosti nekih atributa u relaciji budu relacije (nisu dozvoljene "tabele u tabeli").

- Ako relacija zadovoljava navedene uslove tada je ona u **Prvoj normalnoj formi (1NF)**.

NENORMALIZOVANA I NORMALIZOVANA RELACIJA

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

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

ŠEMA RELACIJA

Uobičajeno je se relacije predstavljaju na sledeći način koji se naziva i "šema relacija":

STUDENT(BRIND, IME, STAROST)
STUD_ISPIT(BRIND,IME,NAZPRED,OCENA)

Naziv relacije se navodi ispred zagrade, a nazivi atributa se navode u zagradi.

Relaciona baza podataka je kolekcija vremenski promenljivih relacija.

DOMENI RELACIJE

Domeni su skupovi iz kojih atributi relacije uzimaju svoje vrednosti. Uobičajeno je da se domeni podele na:

- **Predefinisane domene** (tipove podataka), koji postoje u jezicima baze podataka **(int, char).**
- **Semantičke domene,** koji se još nazivaju apstraktnim ili korisnički definisanim domenima. (Apstraktni tipovi podataka, odnosno objekti koje definiše korisnik).

Atributi relacija bi trebalo uvek da budu definisani nad semantičkim domenima, jer samo tako korisnik može imati punu kontrolu u izvršavanju operacija nad bazom podataka.

DOMENI RELACIJE

- Dva atributa u modelu su sematički ekvivalentna samo ako su definisana nad istim domenom. Sintaksna konstrukcija za definisanje semantičkog domena mogla bi da ima konstrukciju:

```
CREATE  DOMAIN naziv domena
[(ograničenje)] [ (lista operacija)];
```

- Većina komercijalnih relacionih SUBP ne podržava koncept semantičkog domena, ***Objektno-relacione baze podataka*** *ga* podržavaju i one su obrađene u posebnom poglavlju.

KLJUČEVI RELACIJE

- Činjenica da su sve n-torke u relaciji različite, govori da postoji jedan atribut (**prost ključ)** ili više atributa zajedno **(složen ključ)** čije vrednosti jedinstveno identifikuju jednu n-torku u relaciji (jednu vrstu u tabeli).

Ključ relacije R je takva kolekcija K njenih atributa koja zadovoljava sledeća dva uslova:

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

KLJUČEVI RELACIJE

- Može u jednoj relaciji postojati više različitih kolekcija K atributa koje zadovoljavaju definiciju ključa. Sve takve kolekcije se nazivaju kandidati za ključ. Jedan od kandidata koji se izabere da praktično služi za identifikaciju n-torke relacije se tada naziva primarni ključ. Ostali (neizabrani) kandidati se tada nazivaju alternativnim ključevima.

- Atributi koji učestvuju u ključevima (koji su deo kandidata za ključ) nazivaju se ključnim atributima. Ostali atributi u realciji su neključni (ili sporedni) atributi.

SPOLJNI KLJUČ

- Spoljni ključ je atribut (ili grupa atributa) u relaciji R1 koji u njoj nije primarni ključ, ali je primarni ključ u nekoj drugoj relaciji baze podataka. Vrednost spoljneg ključa relacije R1, koristi se za povezivanje sa vrednošću primarnog ključa u odgovarajućoj relaciji R2.

Neka je R2 bazna relacija. Spoljni ključ SK u R2 je poskup njenih atributa takav da:

- postoji bazna relacija R1 sa kandidatom za ključ KK
- svaka vrednost SK u relaciji R2 jednaka je nekoj vrednosti KK u relaciji R1

RELACIONA BAZA PODATAKA

STUDENT(BRIND,MLB,IME,STAROST,POL,ŠIFSMER)
PREDMET(ŠIFPRED, NAZIVPRED, BROJČASOVA)
PRIJAVA(BRIND, ŠIFPRED, DATUMPOL, OCENA)
SMER(ŠIFSMER, NAZIVSMERA, ŠN)
NASTAVNIK(ŠN, IMENAST,ZVANJE,ŠIFPRED)

Relacije u nekoj bazi podataka mogu se podeliti na "bazne”i "izvedene". Izvedena relacija (pogled) je relacija koja se može izvesti iz skupa datih baznih i izvedenih relacija, preko operacija koje se definišu nad relacijama.

NULA VREDNOSTI

- Termin "nula vrednost" (koji ćemo obeležavati sa ?) se koristi da označi "nedostatak imformacija u bazi, odnosno "još nepoznatu vrednost" za neki atribut u nekim n-torkama relacija u relacionoj bazi podataka.

BRIND	IME	STAROST
001	Miloš	19
007	Ana	?
035	?	35

NULA VREDNOSTI

- Pored nula vrednosti koja ima smisao "još nepoznata vrednost", može se uvesti i druga nula vrednost koja ima smisao "neprimenljivo svojstvo".Ona treba da omogući da se iskaže činjenica da je neki atribut neprimenjivo svojstvo za neka pojavljivanja objekata predstavljenih n-torkama date relacije.

- Relacioni SUBP ili treba da podrže obe “nula vrednosti”, ili projektovanjem relacija treba izbeći “nula vrednosti” sa smislom "neprimenljivo svojstvo"

DINAMIČKA PRAVILA INTEGRITETA

- Pravila integriteta definišu dozvoljena stanja i dozvoljene prelaze sistema iz stanja u stanje. Pravilo integriteta u relacionom modelu se iskazuje definisanjem ograničenja na vrednosti atributa i akcijama koje se preduzimaju kada neka operacija ažuriranja naruši posmatrano ograničenje. Opšta sintaksna konstrukcija za iskazivanje pravila integriteta je:

```
CREATE  INTEGRITY RULE naziv pravila
        ograničenje |  naziv_ograničenja
        [ (ON ATTEMPTED VIOLATION akcija)
 | (zapeta_lista parova operacija ažuriranja,  akcija)]
```

OGRANIČENJA I PRAVILA INTEGRITETA

- Uobičajeno je da se u relacionom modelu definišu dve vrste pravila integriteta:
  - **Pravila integriteta modela**, koja su posledica strukture relacinog modela, pa su zbog toga opšta i moraju da važe u svakom konkretnom relacionom modelu.
  - **Poslovna pravila integriteta**, odnosno specifična ograničenja za dati relacioni model. Naziv "poslovna" proističe iz činjenice da se preko ovih ograničenja iskazuju specifični odnosi vrednosti atributa koji važe u datom realnom (najčešće poslovnom) sistemu.

PRAVILA INTEGRITETA MODELA

- Definišu se dva opšta pravila integriteta relacionog modela:

(1) Integritet entiteta (integritet ključa): *Ni jedan atribut koji je primarni ključ ili deo primarnog ključa neke bazne relacije ne može da uzme nula vrednost.*

(2) Referencijalni integritet. *Ako neka bazna relacija (recimo R2) poseduje spoljni ključ (recimo SK) koji ovu relaciju povezuje sa nekom drugom baznom relacijom (recimo R1), preko primarnog ključa (recimo PK), tada svaka vrednost SK mora biti bilo jednaka nekoj vrednosti PK, ili biti nula vrednost. Relacije R1 i R2 ne moraju biti različite.*

REFERENCIJALNI INTEGRITET

FORMALNA DEFINICIJA REFERENCIJALNOG INTEGRITETA:

FOREIGN KEY (lista atributa koji ga čine)
REFERENCES naziv rel. primarnog ključa
 DELETE opcija
 UPDATE opcija
opcija :: RESTRICTED | CASCADES
 | NULLIFIES | DEFAULT

REFERENCIJALNI INTEGRITET

- **RESTRICTED.** Operacija se odbija ako narušava referencijalno ograničenje.
- **CASCADES.** "Okida" se odgovarajuća operacija i na relaciji u kojoj se nalazi spoljni ključč, da bi se, na taj način, zadovoljilo ograničenje.
- **NULLIFIES.** Zamenjije se vrednost spoljnjeg ključa sa "nula vrednošću" za sve n-torke kojima odgovara izbačeni ili promenjeni spoljni ključ.
- **DEFAULT.**Zamenjije se vrednost spoljnjeg ključa sa predefinisanom "default vrednošću" za sve n-torke kojima odgovara izbačeni ili promenjeni spoljni ključ.

POSLOVNA PRAVILA INTEGRITETA

Uobičajeno je da se ova pravila integriteta podele na sledeće podtipove:

- ***Pravila integriteta za domene,*** preko kojih se specifikuje koje vrednosti postoje u domenu;
- ***Pravila integriteta za atribute,*** preko kojih se definišu dozvoljene vrednosti nekog atributa nezavisno od vrednosti drugih atributa u bazi;
- ***Pravila integriteta za relacije,*** preko kojih je moguće vezati vrednost jednog, za vrednost drugog atributa u jednoj relaciji;
- ***Pravila integriteta za bazu,*** preko kojih je moguće povezati vrednosti atributa iz više relacija.

PRAVILA INTEGRITETA ZA DOMENE

Definisanje pravila integriteta za domene svodi se na definisanje semantičkog domena. Pri tome se može koristiti sledeća sintaksa:

CREATE DOMAIN naziv_domena predefinisani domen predikat;

CREATE DOMAIN Kol INTEGER

∀ Kol (Kol > 5000 AND Kol < 50000 AND MOD (Kol, 50) =0);

ili

CREATE DOMAIN Boja CHAR(6) IN ('Crvena', 'Plava', 'Bela', 'Zelena');

Ograničenje je zadovoljeno ako predikat dobije vrednost "T" (True- istinito).

PRAVILA INTEGRITETA ZA ATRIBUTE

Očigledno je da se pravila integriteta za atribute mogu definisati preko sledeće četvorke, odnosno tabele:

<naziv atributa, domen, ograničenje, akcija>

PRAVILA INTEGRITETA ZA RELACIJE

Sintaksa za iskazivanje ove vrste integriteta, bez navođenja triger procedure, može da bude:

CREATE INTEGRITY RULE naziv ograničenja predikat;

Promenljive u predikatu mogu da budu samo atributi jedne relacije. Atribut kao promenljiva se označava sa "dot notacijom". Primer pravila integriteta za relacije bi mogao biti:

CREATE INTEGRITY RULE Starost_smer

∀ Student (IF Student.ŠSmer = 01 THEN Student.Starost < 28);

PRAVILA INTEGRITETA ZA BAZU

Preko ovog pravila moguće je iskazati bilo kakvo složeno ograničenje na vrednosti atributa u bazi podataka, ograničenje koje povezuje vrednosti atributa iz više relacija. Sintaksa za iskaz ove vrste ograničenja je jednaka onoj za ograničenja nad jednom relacijom.

Primer ove vrste ograničenja je:

CREATE INTEGRITY RULE Ocene_smer
∀ Student (IF Student.ŠifSmer = 01 THEN
∃ Prijava (Prijava.Ocena >7 AND
Student.BrInd = Prijava.BrInd
AND Prijava. ŠifPred = Predmet. ŠifPred
AND Predmet. Nazivpred = 'Matematika'.));

PRAVILA INTEGRITETA PRELAZA IZ STANJA U STANJE

Preko ovog pravila moguće je iskazati bilo kakvo složeno ograničenje na vrednosti atributa u bazi podataka, ograničenje koje povezuje vrednosti atributa iz više relacija. Sintaksa za iskaz ove vrste ograničenja je jednaka onoj za ograničenja nad jednom relacijom.

Na primer, pretpostavimo pravilo: "Student sa smera sa ŠifSmer = 01 ne može se prebaciti na smer sa ŠifSmer = 05"

CREATE INTEGRITY RULE Prelaz sa smera_na_smer

$\forall$ Student, Student' (IF Student'.ŠifSmer=01

THEN Student.ŠifSmer $\neq$ 05);

OPERACIJE RELACIONOG MODELA

Relaciona algebra
Relacioni račun

RELACIONA ALGEBRA

- Relaciona algebra definiše skup operacija pomoću kojih je moguće dobiti željenu relaciju (tabelu) iz skupa datih relacija (tabela).
  - **Konvencionalne skupovne operacije, unija, presek, razlika i Dekartov proizvod;**
  - **Specijalne relacione operacije selekcija, projekcija, spajanje i deljenje;**
  - **Dodatne operacije relacione algebre,** operacije koje su se kasnije dodavale orginalnoj relacionoj algebri da bi se povećala njena "moć" kao upitnog jezika- Skalarna računanja u relacionoj algebri;
  - **Operacije ažuriranja baze;**
  - **Operacije u prisustvu nula vrednosti;**

SKUPOVNE OPERACIJE

- **RELACIJA JE SKUP:**
  - UNIJA, PRESEK, RAZLIKA I DEKARTOV PROIZVOD

- **Uslov kompatibilnosti** relacija R_1 i R_2 za izvođenje operacija unije, preseka i diferencije:

Relacije R_1 i R_2 moraju imati isti broj atributa (isti stepen), a njihovi odgovarajući atributi moraju biti definisani nad istim domenima da bi se nad njima mogle izvršiti operacije unije, preseka i diferencije.

Tabele za primer

S_1

BrInd	MLB	Ime	Starost	[ifSmer
152/97	16309723331981	Ana	19	01
223/95	13975673331981	Mirko	21	01
021/94	11145276418976	Zoran	20	02

S_2

BrInd	MLB	Ime	Starost	[ifSmer
223/95	13975673331981	Mirko	21	01
021/94	11145276418976	Zoran	20	02
003/94	23456786418976	Milo{	22	01

UNIJA

$S_3 := S_1 \cup S_2$

S_3

BrInd	MLB	Ime	Starost	[ifSmer
152/97	16309723331981	Ana	19	01
223/95	13975673331981	Mirko	21	01
021/94	11145276418976	Zoran	20	02
003/94	23456786418976	Milo{	22	01

RAZLIKA

$S_4 := S_1 - S_2$

S_4

BrInd	MLB	Ime	Starost	[ifSmer
152/97	1630972333 1981	Ana	19	01

PRESEK

$S_5 := S_1 \cap S_2$

S_5

BrInd	MLB	Ime	Starost	[ifSmer
223/95	13975673331981	Mirko	21	01
021/94	111452764 18976	Zoran	20	02

DEKARTOV PROIZVOD

R_1

C	D	E
1	2	3
2	4	6

R_2

A	B
a1	b1
a2	b2
a3	b3

$R_3 = R_1 \times R_2$

C	D	E	A	B
1	2	3	a1	b1
1	2	3	a2	b2
1	2	3	a3	b3
2	4	6	a1	b1
2	4	6	a2	b2
2	4	6	a3	b3

Nekotrolisano spajanje.

SPECIJALNE RELACIONE OPERACIJE

- **Projekcija**
- **Selekcija**
- **Spajanje**
- **Deljenje**

PROJEKCIJA

- Neka je $R(A_1, A_2, ..., A_n)$ relacija, a X podskup njenih atributa. Označimo sa Y komplement $\{A_1, A_2, ..., A_n\}$ - X. Rezultat operacije projekcije relacije R po atributima X je

$$\pi_X(R) = \{x \mid \exists\, y\ ,\ <x,y> \in R\}.$$

Projekcija - primer

Građanin

MLB	Ime	Starost	MestoRo|
16309723331981	Ana	19	Beograd
13975673331981	Mirko	21	Valjevo
11145276418976	Zoran	20	Beograd
23243723331981	Ana	19	Ni{
22222223331981	Mirko	21	Beograd
11145276418976	Zoran	20	Novi Sad
23456786418976	Milo{	22	Beograd

$Pr_1 := \pi_{Ime, Starost}$ Građanin

Pr_1

Ime	Starost
Ana	19
Mirko	21
Zoran	20
Milo{	22

SELEKCIJA

Data je relacija $R(A_1, A_2, ..., A_n)$ i predikat Θ definisan nad njenim atributima. Rezultat operacije selekcije

$$\sigma_{\Theta}(R) = \{x \mid x \in R \text{ AND } \Theta(x)\}$$

(skup n-torki x relacije R koje zadovoljavaju predikat (uslov) Θ).

SELEKCIJA - PRIMER

Primer: Prikaži građane koji su stariji od 20 godina i rođeni su u Beogradu.

$Pr_2 := \sigma_{\text{Starost} > 20 \text{ AND MestoRođ} = \text{“Beograd”}} (\text{Građanin})$

Pr_2

MLB	Ime	Starost	MestoRo|
22222223331981	Mirko	21	Beograd
23456786418976	Milo{	22	Beograd

Restrikcija – koje n-torke mogu da se jave u rezultatu

SPAJANJE - JOIN

Date su relacije $R_1(A_1, A_2, ..., A_n)$ i $R_2(B_1,B_2, ..., B_m)$ i predikat Θ definisan nad njihovim atributima. Obeležimo sa X i Y skupove atributa relacija R_1 i R_2, respektivno. Rezultat operacije spajanja ovih relacija (tzv. ***teta spajanje***) je

$$R_1[{}_X\Theta]R_2 = \{<x,y> \mid x \in R_1 \text{ AND } y \in R_2 \text{ AND } \Theta(x,y)\}.$$

Oznaka ${}_X\Theta$ za operaciju spajanja ukazuje na činjenicu, očiglednu iz definicije teta spajanja, da ova operacija nije primitivna operacija relacione algebre, već se može izvesti uzastopnom primenom operacije Dekartovog proizvoda (x) nad relacijama koje se spajaju i selekcije po predikatu Θ nad tako dobijenom relacijom.

SPAJANJE

- Ako je predikat Θ definisan sa $A_k = B_j$, s tim da su i atributi A_k i B_j definisani nad istim domenima, tada se takvo spajanje naziva ***ekvispajanje.***

- Očigledno je da se u rezultatu ekvispajanja uvek pojavljuju dve iste kolone, u gornjem primeru dve iste kolone MLB. Ako se jedna od te dve kolone izbace, takvo spajanje se naziva ***prirodno spajanje.*** Predpostavlja se da su atributi spajanja istoimeni.

SPAJANJE - PRIMERI

Građanin

MLB	Ime	Starost	MestoRođ
16309723331981	Ana	19	Beograd
13975673331981	Mirko	21	Valjevo
11145276418976	Zoran	20	Beograd
23243723331981	Ana	19	Niš
22222223331981	Mirko	21	Beograd
11145276418976	Zoran	20	Novi Sad
23456786418976	Miloš	22	Beograd

Student

BrInd	MLB	Smer
152/97	16309723331981	01
223/95	13975673331981	01
021/94	11145276418976	02
003/94	23456786418976	01

SPAJANJE - PRIMERI

Pr_3 := Građanin [Građanin.MLB = Student.MLB] Student

Pr_3

MLB	Ime	Starost	MestoRo|	MLB	BrInd	Smer
16309723331981	Ana	19	Beograd	16309723331981	152/87	01
13975673331981	Mirko	21	Valjevo	13975673331981	223/95	01
11145276418976	Zoran	20	Beograd	11145276418976	021/94	02
23456786418976	Milo{	22	Beograd	23456786418976	003/94	01

Pr_4 := Građanin * Student

Pr_4

MLB	Ime	Starost	MestoRo|	BrInd	Smer
16309723331981	Ana	19	Beograd	152/87	01
13975673331981	Mirko	21	Valjevo	223/95	01
11145276418976	Zoran	20	Beograd	021/94	02
23456786418976	Milo{	22	Beograd	003/94	01

DELJENJE

1. Deljenje je operacija pogodna za upite u kojima se javlja reč "svi" ("sve", "sva"). Formalno se definiše na sledeći način:

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 odgovarajući domeni su im jednaki. Rezultat operacije deljenja

$$A[Y \div Z]B = R(X)$$

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).

DELJENJE

Prijava

Predmet		Brind	ŠifPred
ŠifPred		152/97	P1
P1		152/97	P2
P2		021/94	P1
P3		003/94	P3
		152/97	P3

Prijava [Prijava.ŠifPred ÷ Predmet.ŠifPred] Predmet

BrInd
152/97

DELJENJE

peracija deljenja nije primitivna operacija relacione algebre, već se može izvesti pomoću drugih operacija na sledeći način:

$$(X, Y)\ [Y \div Z]B(Z) = \pi_X A - \pi_X ((\pi_X A \times B) - A)$$

bjašnjenje:

- $_X$ **A** daje sve n-torke koje mogu da učestvuju u rezultatu,
- π_X **A x B)** daje relaciju u kojoj se za svaku vrednost z iz B pojavljuju parovi <x,z> sa svim vrednostima x,
-

RELACIONA ALGEBRA

9. ***Poluspajanje (SEMIJOIN).*** Operacija poluspajanja uvedena je za potrebe distribuiranih baza podataka. Ako se, u nekoj distribuciji relacione baze podataka, relacije koje se spajaju nalaze na različitim lokacijama, spajanje je veoma skupa operacija, pogotovo ako se rezultat zahteva sa neke treće lokacije. Posmatrajmo sledeći primer:

Radnik(ŠifRadn, ImeRadn, AdresaRadn, Staž, ŠifOdel) - lokacija a

Odeljenje(ŠifOdel, NazivOdel) - lokacija b,

a na lokaciji c se izvršava operacija

Radnik * Odeljenje

RELACIONA ALGEBRA

Skalarna računanja u relacionoj algebri

- **EXTEND** koji uključuje tzv "horizontalno računanje" u relacionu algebru, odnosno formira vrednost atributa relacije preko aritmetičkog izraza nad vrednostima ostalih atributa iste n-torke;

- **SUMMARIZE** koji uključuje tzv "vertikalno računanje" u relacionu algebru, odnosno formira novu relaciju kao projekciju polazne po nekim atributima, proširenu za atribut koji je rezultat nekog "vertikalnog", agregirajućeg, aritmetičkog izraza nad vrednostima odgovarajućih atributa.

RELACIONA ALGEBRA

Operacije sa nula vrednostima

- **MOŽDA_SELEKCIJA (MAYBE_SELECT).** Selektuju se one n-torke relacije za koje se predikat selekcije, na osnovu trovrednosnih tablica istinitosti, sračunava u nula vrednost.
- **SPOLJNO_SPAJANJE (OUTER_JOIN):**
  - centralno, levo i desno
- **SPOLJNA_UNIJA (OUTER_UNION).**

RELACIONI RAČUN

- Postoje dva oblika relacionog računa:
  (1) **Relacioni račun n-torki** i
  (2) **Relacioni račun domena**

- **Relacioni račun n-torki** je Predikatski račun prvog reda u kome promenljive uzimaju vrednosti n-torki relacija date baze podataka.

- **Relacioni račun** domena je Predikatski račun prvog reda u kome promenljive uzimaju vrednosti iz nekih domena atributa relacija date baze podataka.

PREDIKATSKI RAČUN PRVOG REDA

(1)Afirmativna rečenica, koja ima smisla i koja je istinita ili neistinita naziva se sud.

(2)Afirmativna rečenica koja ima smisla i koja sadrži jedan ili više promenljivih parametara i koja postaje sud uvek kada parametri iz rečenice dobiju konkretnu vrednost naziva se predikat. Broj parametara u predikatu se naziva dužina predikata. (Primer predikata je $x^2 + y^2 \leq 1$).

(3)Činjenica da u predikatskom računu promenljiva x uzima svoju vrednost iz nekog skupa R označava se sa x : R.

(4)Predikatski ili kvantifikatorski račun je matematička teorija čiji su objekti formule koje predstavljaju predikate.

(5) Definišu se **atomske formule** i **pravila (sintaksa) izvođenja složenijih formula**

RELACIONI RAČUN N-TORKI

- U RELACIONOM RAČUNU N-TORKI:
  - Promenljive su n-torke relacija;
  - Atomske formule se definišu nad atributima n-torki;
  - Pravila izvođenja su standardna pravila Predikatskog računa prvog reda.
- U OBJEKTNOM RAČUNU:
  - Promenljive su objekti klasa.
  - Atomske formule se definišu nad atributima i operacijama objekata;
  - Pravila izvođenja su standardna pravila Predikatskog računa prvog reda.

RELACIONI RAČUN N-TORKI

- Činjenica da u relacionom računu n-torki promenljiva x uzima kao svoju vrednost n-torku relacije R označava se sa x : R.
- Atomi u relacionom računu n-torki su:
  - **x.A Θ y.B** (x.A je vrednost atributa A relacije R1 za n-torku x koja iz ove relacije uzima vrednost. Na isti način y.B je vrednost atributa B neke druge relacije R2 iz koje promenljiva y uzima n-torke kao svoje vrednosti (x: R1, y: R2). Pretpostavlja se da su atributi A i B definisani nad istim domenom, a Θ je operacija poređenja definisana nad tim domenom.)
  - **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 atribut A.

RELACIONI RAČUN N-TORKI

- Formule se formiraju od atoma preko sledećih pravila (sintakse):

•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 sadrži neku slobodnu promenljivu s tada su i ∃ s (P1(s)) i ∀ s (P1(s)) takođe formule
(∃ - "postoji", egzistencijalni kvantifikator,
∀ - "za svako", univerzalni kvantifikator).

RELACIONI RAČUN N-TORKI

- **Opšti iskaz relacionog računa.** Neka su R1, R2, ..., Rn relacije u nekoj bazi podataka. Neka su A, B, ..., C atributi ovih relacija, respektivno i neka je F formula. Opšti izraz relacionog računa n-torki je tada:

t: R1, u: R2, . . . , v: Rn
t.A, u.B, . . . , v.C **WHERE** F

(Prikaži 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

PRIMERI ĆE BITI PRIKAZANI NA SLEDEĆJEDNOSTAVNOJ ŠEMI RELACIONE BAZE PODATAKA:

STUDENT(BI,IME, STAROST,ŠSMER)
PRIJAVA(BI,ŠP,OCENA)
PREDMET(ŠP, NAZIP,BRČAS)
SMER(ŠSMER,NAZIVS)

PRIMERI RELACIONOG RAČUNA N-TORKI

(a) *Prikaži brojeve indeksa i imena studenata koji su stariji od 20 godina.*

```
x: STUDENT
x.BI, x.IME WHERE   x.STAROST > 20;
```

(b) *Prikaži imena studenata koji studiraju smer "Informacioni sistemi".*

```
x:  STUDENT, y:  SMER
 x. IME  WHERE  ∃ y (y.ŠSMER  = x. ŠSMER
AND y.NAZIVS = 'InfSist');
```

PRIMERI RELACINOG RAČUNA N-TORKI

(c)Prikaži imena studenata koji su položili sve predmete.

x: STUDENT, y: PRIJAVA, z: PREDMET
x.IME WHERE ∀ z (∃ y (y.BI = x.BI AND y. ŠP = z. ŠP));

INTERPRETACIJA SQL-a PREKO RELACIONE ALGEBRE I RELACIONOG RAČUNA

- SQL se može interpretirati kao korisniku prilagođena sintaksa i za relacionu algebru i za relacioni račun n-torki.
- U relacionoj algebri, opšti izraz preko koga se može dobiti željena tabela iz skupa datih tabela je "**projekcija po navedenim atributima, selekcije po datom uslovu, Dekartovog proizvoda potrebnih tabela",** odnosno

Π [**R1.A1, R2A2, ..., Rm.An**] ($\Sigma\Epsilon\Lambda$ [Θ] **(R1 x R2 x ... x Rm)**)

INTERPRETACIJA SQL-a PREKO RELACIONE ALGEBRE I RELACIONOG RAČUNA

Isti upit se u SQL-u može izraziti na sledeći način:

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

Očigledne su sledeće ekvivalencije:

- Ključna reč **SELECT** odgovara operaciji projekcije (Π)
- Ključna reč **FROM** odgovara Dekartovom proizvodu;
- Ključna reč **WHERE** odgovara operaciji selekcije ($\Sigma E \Lambda$) po uslovu Θ.

INTERPRETACIJA SQL-a PREKO RELACIONE ALGEBRE I RELACIONOG RAČUNA

Ekvivalencija relacionog računa n-torki i SQL-a, može se pokazati ako se prethodni SQL upit napiše u obliku:

```
SELECT x.A1, y.A2, ..., z.An
        FROM R1 x, R2 y,  ..., Rm z
        WHERE  Θ;
```

Interpretacija je sledeća:

- Ključna reč **FROM** služi za definisanje promenljivih
- Ključna reč **SELECT** definiše "desnu stranu" opšteg iskaza relacionog računa
- Ključna reč **WHERE** ima istu ulogu kao i u opštem iskazu relacionog računa - iza nje se navodi predikat (uslov) koji treba da bude zadovoljen.

IMPLEMENTACIJA OBJEKTNOG RAČUNA - QUEL

Najpoznatiji upitni jezik koji predstavlja direktnu implementaciju relacionog računa n-torki je **QUEL (Query Language).** Promenljive koje uzimaju za svoje vrednosti n-torke pojedinih relacija navode se iskazom:

RANGE OF naziv-promenljive **IS** naziv-relacije

čime se iskazuje da neka promenljiva uzima za svoje vrednosti n-torke navedene relacije. Opšti oblik upita u QUEL-u je:

RETRIEVE lista-atributa **WHERE** predikat;

PRIMERI UPITA U QUEL-U SU:

(1) *Prikazati brojeve indeksa i imena radnika koji studiraju smer "Informacioni sistemi".*

```
RANGE OF x IS STUDENT
RANGE OF y IS  SMER
        RETRIEVE  (x. BI, x.IME )
        WHERE  x.ŠIFSMER = y.ŠIFSMER
        AND y.NAZIVSMER = 'Infsist';
```

(2) *Prikaži parove imena studenata iste starosti.*

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

PRIMERI JEDNOSTAVNIH SQL UPITA

Prikaži brojeve indeksa i imena studenata koji su stariji od 20 godina.

```
SELECT STUDENT.BI, STUDENT.IME
FROM STUDENT
WHERE STUDENT.STAROST >20;
```

Prikaži imena studenata koji studiraju smer "Informacioni sistemi".

```
SELECT STUDENT.BI
FROM STUDENT, SMER
WHERE STUDENT.ŠSMER = SMER.ŠSMER AND
        SMER.NAZIVS= ‘InfSist’;
```

RELACIONI RAČUN DOMENA

U relacionom računu domena promenljive uzimaju vrednosti iz nekih domena atributa relacija posmatrane relacione baze podataka. Ovde se, pored navedenih, definiše još jedna atomska formula, tzv. "uslov članstva" (membership condition

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 članstva se sračunava u TRUE ako postoji n-torka u relaciji R koja ima zadate vrednosti navedenih atributa.

STUDENT (BI = '152/97' , ŠIFSMER =01)

RELACIONI RAČUN DOMENA

Opšti izraz relacionog računa domena je:

x, y, ..., z **WHERE** F

gde su x, ..., z promenljive a F je formula koja uključuje i uslov članstva.

(a) *Prikaži brojeve indeksa i imena studenata starijih od 20 godina*

x, y WHERE ∃ z > 25 AND
STUDENT (BRIND: x, IME:y, STAROST: z)

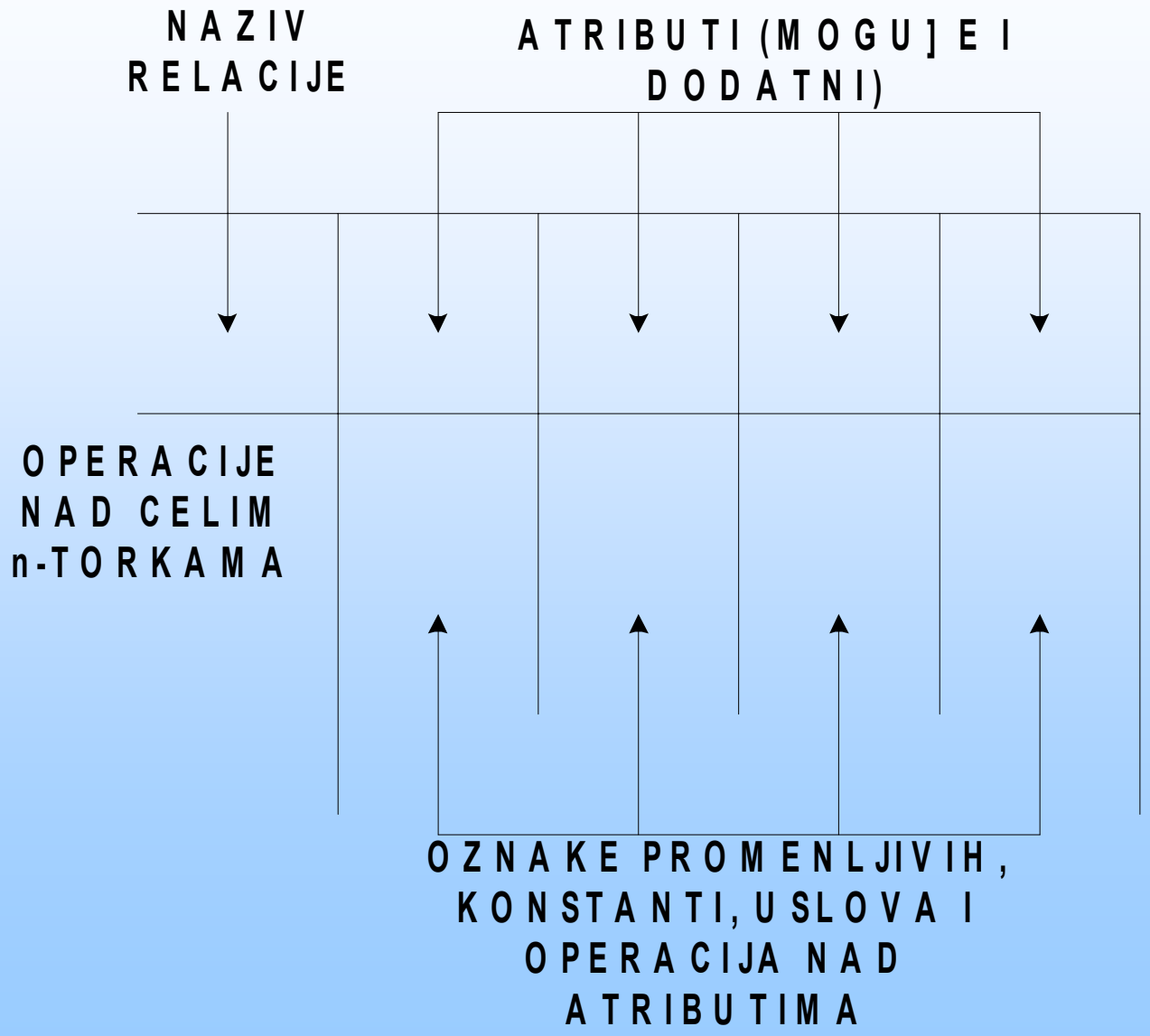
(b) *Prikaži imena studenata sa smera "InfSist".*

x WHERE ∃ y ((IME: x, ŠIFSMER: y)
AND SMER (ŠIFSMER: y, NAZIVS:'InfSis'))

RELACIONI RAČUN DOMENA

- QBE (Query By Example) je upitni jezik koji predstavlja implementaciju relacionog računa domena, preko specifične "dvodimenzione" sintakse, koja je vrlo bliska korisniku, jer se preko nje, direktno, u skeletu tabele predstavljene na ekranu, zadaje "primer odgovora" koji korisnik želi. Otuda i ime Query By Example (upit na osnovu primera).

QBE (Query By Example)



PRIMER QBE

STUDENT	BI	IME	STAROST	[SMER
P.	_127/97			

PREDMET	[P	NAZIP	BR^AS
	_p01	'Matemat'	

PRIJAVA	BI	[P	OCENA
	_127/97	_p01	> 8

Standardni upitni jezik SQL

Standardni upitni jezik SQL

- CHAMBERLIN, IBM RESEARCH LABORATORY, SAN JOSE CALIFORNIA (1974)

- ISTORIJAT
  - SQL-86
  - SQL-89
  - SQL-92 (SQL2)
  - SQL:1999
  - SQL:2003

Standardni upitni jezik SQL

- Osnovna literatura
  - Marjanovi}, Z., *SQL:1999 standard,* Breza, Beograd, 2002.
  - Melton, J., Simon, A., *SQL:1999, Understanding Relational Language Components*, Academic Press, 2002.
  - American National Standars for Information Technology - Database Languages - SQL, ANSI/ISO/IEC 9075-x-1999, 1999.

- KARAKTERISTIKE
  - SQL je u stalnom razvoju. Na po~etku je bio prili~no jednostavan, blizak korisniku i u velikoj meri deklarativan (neproceduralan)
  - Danas se za SQL mo`e re}i da je kompleksan, proceduralno/deklarativan jezik.

- Zaklju~no sa SQL-92 standardom SQL naredbe su svrstavane u jednu od slede}e tri kategorije:

    - naredbe za definisanje podataka (data definition statements)
    - naredbe za manipulisanje (rukovanje) podacima (data manipulation statements) i
    - naredbe za kontrolne (upravlja~ke) funkcije (data control statements).

- Naredbe za definisanje podataka omogu}uju definisanje objekata baze. Primeri naredbi ove kategorije su:

    - **CREATE TABLE** (kreiranje tebele baze podataka)
    - **CREATE VIEW** (kreiranje virtuelne tabele - "pogleda")
    - **CREATE INDEX** (kreiranje indeksa nad kombinacijom kolona tabele)
    - **ALTER TABLE** (izmena definicije tabele)
    - **DROP TABLE** (izbacivanje tabele iz baze podataka)

- Naredbe za manipulisanje (rukovanje) podacima omogu}uju a`uriranje i prikaz podataka baze:

  - **SELECT** (prikaz sadr`aja relacione baze podataka)
  - **UPDATE** (izmena vrednosti kolona tabele)
  - **DELETE** (izbacivanje redova tabele)
  - **INSERT** (dodavanje redova postoje}oj tabeli)

- Naredbe za kontrolne (upravlja~ke) funkcije omogu}uju oporavak, konkurentnost, sigurnost i integritet relacione baze podataka:

    - **GRANT** (dodela prava kori{}enja sopstvene tabele drugim korisnicima)
    - **REVOKE** (oduzimanje prava kori{}enja sopstvene tabele od drugih korisnika)
    - **COMMIT** (prenos dejstava transakcije na bazu podataka)
    - **ROLLBACK** (poni{tavanje dejstava transakcije)

- SQL:1999 standard razvrstava SQL naredbe u sedam kategorija:
    1. **Naredbe za {emu baze podataka** (SQL-schema statements), koje se koriste za kreiranje, izmenu i izbacivanje {ema i objekata {ema (CREATE, ALTER, DROP)
    2. **Naredbe za podatke** (SQL-data statements), koje se koriste za prikaz i a`uriranje podataka baze (SELECT, INSERT, UPDATE, DELETE)
    3. **Naredbe za transakcije** (SQL-transaction statements), koje se koriste za startovanje, zavr{avanje i postavljanje parametara za transakcije (COMMIT, ROLLBACK)
    4. **Naredbe za kontrolu** (SQL-control statements), koje se koriste za kontrolu izvr{avanja sekvence SQL naredbi (CALL, RETURN)
    5. **Naredbe za konekcije** (SQL-connection statements), koje se koriste za uspostavljanje i prekidanje SQL konekcije (CONNECT, DISCONNECT)
    6. **Naredbe za sesije** (SQL-session statements), koje se koriste za postavljanje default vrednosti i drugih parametara SQL sesije (SET)
    7. **Naredbe za dijagnostiku** (SQL-diagnostic statements), koje koriste dijagnosti~ke podatke i signaliziraju izuzetke u SQL rutinama (GET DIAGNOSTICS)

- Dva osnovna na~ina kori{}enja SQL-a su:
  - direktno (interaktivno) kori{}enje SQL-a i
  - povezivanje SQL-a sa klasi~nim programskim jezicima ("ugradjeni" SQL).

  Kursor i naredbe za rad sa kursorom:

  a) **OPEN** <kursor> - konceptualno prouzrokuje generisanje skupa n-torki kao rezultat izvr{avanja upitnog izraza tj. SELECT upitnog bloka definisanog za <kursor>,

  b) **FETCH** <kursor> - izvr{ava se jedanput za svaku n-torku kada je kursor otvoren naredbom OPEN <kursor> i prouzrokuje pomeranje kursora na slede}u n-torku generisanog skupa i prikaz te n-torke,

  c) **CLOSE** <kursor> - zatvara kursor nakon obrade n-torki generisanog skupa.

  d) **DECLARE** <kursor> - unutar te naredbe navodi se i upitni izraz, odnosno SELECT upitni blok koji, pri otvaranju kursora, generi{e skup n-torki.

Relaciona {ema i model

```
ODELJENJE (ODELJENJE#, NAZIV, GRAD)
RADNIK (SRADNIK, IME, POSAO, SRUKOV, DATZAP, PLATA,
             PREMIJA, ODELJENJE#)
```

relaciona shema

ODELJENJE#	NAZIV	GRAD
10	RAD_ZAJ	BEOGRAD
20	PRIPREMA	NIS
30	PROJEKTOVANJE	BOR
40	ISTRAZIVANJE	SARAJEVO

Relaciona {ema i model

SRADNIK	IME	POSAO	SRUKOV	DATZAP	PLATA	PREMIJA	ODELJENJE#
3069	Steva	ANALITICAR	3602	1980-12-17	80000		20
3199	Milan	TRG_PUTNIK	3398	1981-02-20	160000	30000	30
3221	Petar	TRG_PUTNIK	3398	1981-02-22	125000	50000	30
3266	Marko	RUKOVODILAC	3539	1981-04-02	297500		20
3354	Josip	TRG_PUTNIK	3398	1981-09-28	125000	140000	30
3398	Dragan	RUKOVODILAC	3539	1981-05-01	285000		30
3482	Ivan	RUKOVODILAC	3539	1981-06-09	245000		10
3488	Pavle	SAVETNIK	3266	1981-11-09	300000		20
3539	Jovan	PREDSEDNIK		1981-11-17	500000		10
3544	Goran	TRG_PUTNIK	3398	1981-09-08	150000	0	30
3576	Adam	ANALITICAR	3488	1981-09-23	110000		20
3600	Janko	ANALITICAR	3398	1981-12-03	95000		30
3602	Filip	SAVETNIK	3266	1981-12-03	300000		20
3634	Dejan	ANALITICAR	3482	1982-01-23	130000		10

Definisanje koncepata strukture

- SQL tipovi podataka
- domeni
- tabele i kolone - osnovna svojstva
- {ema baze podataka
- katalog

SQL tipovi podataka

- SQL:1999 standard podr`ava numeri~ke, tekstualne, binarne, datumske, intervalne i logi~ke tipove podataka
- Numeri~ki tipovi (ta~ni i pribli`ni)
    - INTEGER (ili INT)
    - SMALLINT
    - NUMERIC (p, d)
    - DECIMAL (p, d) (ili DEC)

    - REAL
    - DOUBLE PRECISION
    - FLOAT (p)

SQL tipovi podataka

- Tekstualni tipovi
  - CHARACTER (ili CHAR)
  - CHARACTER VARYING (ili CHAR VARYING ili VARCHAR)
  - CHARACTER LARGE OBJECT (ili CLOB ili CHAR LARGE OBJECT)

SQL tipovi podataka

- Binarni tipovi
  - BIT
  - BIT VARYING
  - BINARY LARGE OBJECT (ili BLOB)

SQL tipovi podataka

- Datumski tipovi

  Definisani su u odnosu na univerzalno koordinisano vreme (UTC)

  - DATE
  - TIME
  - TIMESTAMP
  - TIME WITH TIME ZONE
  - TIMESTAMP WITH TIME ZONE

SQL tipovi podataka

- Intervalni tipovi
  - Godina-mesec interval

    INTERVAL YEAR, INTERVAL YEAR (p), INTERVAL MONTH, INTERVAL MONTH (p), INTERVAL YEAR TO MONTH, INTERVAL YEAR (p) TO MONTH
  - Dan-vreme interval

    INTERVAL DAY TO HOUR, INTERVAL DAY (6) TO MINUTE, INTERVAL SECOND (7), INTERVAL DAY (5) TO SECOND (10), INTERVAL MINUTE (3) TO SECUNDE (4)

SQL tipovi podataka

- Logi~ki tipovi
  - BOOLEAN

    Tri logi~ke vrednosti TRUE, FALSE, UNKNOWN

Domeni

```
CREATE DOMAIN <naziv domena> [AS] <predefinisani tip>
   [DEFAULT <vrednost>]
   [[ CONSTRAINT <naziv ogranicenja>] CHECK (<ogranicenje>)]
   ...
```

Primer:

```
CREATE DOMAIN novac AS DECIMAL (7,2)

CREATE DOMAIN vrsta_racuna AS CHAR (1)
   DEFAULT 'L'
   CONSTRAINT vrsta_racuna_provera
   CHECK (value IN ('L', 'R', 'P'))
```

Domeni

Definicija domena menja se naredbom ALTER DOMAIN:

```
ALTER DOMAIN <naziv domena>
SET DEFAULT <vrednost> |
DROP DEFAULT |
ADD [CONSTRAINT <naziv ogranicenja>] CHECK (<ogranicenje>) |
DROP CONSTRAINT <naziv ogranicenja>
```

Domen se uni{tava slede}om naredbom:

```
DROP DOMEN <naziv domena>
```

Tabele i kolone - osnovna svojstva

- Bazne tabele
  - Perzistentne bazne tabele
  - Globalne privremene tabele
  - Kreirane lokalne privremene tabele
  - Deklarisane lokalne privremene tabele
- Tabele pogleda
- Izvedene tabele

Kreiranje tabele - osnovna sintaksa

```
CREATE TABLE <naziv tabele>
   (<naziv kolone1> <tip podataka> [not null],
    <naziv kolone2> <tip podataka> [not null],
        ...
     )
```

Kreiranje tabele - osnovna sintaksa

Tabele RADNIK i ODELJENJE kreirane su slede}im naredbama:

```
CREATE TABLE RADNIK
  (SRADNIK INTEGER NOT NULL,
   IME VARCHAR (20) NOT NULL,
   POSAO VARCHAR (20),
   SRUKOV INTEGER,
   DATZAP DATE,
   PLATA NUMERIC (10),
   PREMIJA NUMERIC (10),
   ODELJENJE# INTEGER NOT NULL);

CREATE TABLE ODELJENJE
  (ODELJENJE# INTEGER NOT NULL,
   NAZIV VARCHAR (20) NOT NULL,
   GRAD VARCHAR (20) );
```

Kreiranje tabele - osnovna sintaksa

- Pri definisanju kolone, umesto tipa podataka, mogu}e je navesti domen
- Pored navedenih osnovnih svojstava, definicija kolone mo`e da obuhvati i specifikaciju default vrednosti i ograni~enja na vrednosti kolone.
- Za default vrednost neke kolone mogu}e je specificirati:
  - literal odgovaraju}eg tipa,
  - neku od datumskih funkcija (CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME ili LOCALTIMESTAMP)
  - identifikator korisnika (CURRENT_USER, SESSION_USER ili SYSTEM_USER) ili NULL
- Kolona DATZAP mogla je biti specificirana na slede}i na~in:

```
DATZAP DATE DEFAULT CURRENT_DATE
```

Kreiranje tabele - osnovna sintaksa

- Globalna privremena tabela definiše se na sledeći način:

```
CREATE GLOBAL TEMPORARY TABLE <naziv tabele>
   (<naziv kolone1> <tip podataka> [NOT NULL],
    <naziv kolone2> <tip podataka> [NOT NULL],
        ...                )
```

- Sadržaj jedinstven u SQL sesiji

Kreiranje tabele - osnovna sintaksa

- Kreirana lokalna privremena tabela defini{e se slede}om naredbom:

```
CREATE LOCAL TEMPORARY TABLE <naziv tabele>
   (<naziv kolone1> <tip podataka> [NOT NULL],
    <naziv kolone2> <tip podataka> [NOT NULL],
     . . .      )
```

- Sadr`aj jedinstven unutar modula ili “embedded” sql programa jedne SQL sesije

Kreiranje tabele - osnovna sintaksa

- Deklarisana lokalna privremena tabela defini{e se na slede}i na~in:

```
DECLARE LOCAL TEMPORARY TABLE MODULE.<naziv
  tabele>
  (<naziv kolone1> <tip podataka> [not null],
   <naziv kolone2> <tip podataka> [not null],
    . . .      )
```

- Sadr`aj jedinstven unutar procedure modula

Izmena definicije tabele

- Definicija postoje}e tabele se mo`e izmeniti:
  - dodavanjem nove kolone
  - izmenom definicije postoje}e kolone
  - izbacivanjem kolone i
  - dodavanjem ili izbacivanjem ograni~enja na vrednosti podataka tabele

Izmena definicije tabele

- Ukoliko `elimo da dodamo novu kolonu postoje}oj tabeli op{ti oblik naredbe je:

```
ALTER TABLE <naziv tabele>
ADD [COLUMN] <definicija kolone>
```

- Strukturu tabele ODELJENJE mo`emo izmeniti dodavanjem nove kolone BROJ_RADNIKA, {to se realizuje slede}om naredbom:

```
ALTER TABLE ODELJENJE
ADD COLUMN BROJ_RADNIKA NUMERIC (4)
```

Izmena definicije tabele

- Izmena definicije postoje}e kolone realizuje se slede}om naredbom:

```
ALTER TABLE <naziv tabele>
ALTER [COLUMN] <naziv kolone>
  SET DEFAULT <vrednost> |
  DROP DEFAULT.
```

- Ovim oblikom ALTER TABLE naredbe odredjuje se, menja se ili izbacuje default vrednost kolone, kao u slede}im primerima:

```
ALTER TABLE ODELJENJE
ALTER COLUMN GRAD
  SET DEFAULT 'BEOGRAD'

ALTER TABLE RADNIK
ALTER COLUMN DATZAP
  DROP DEFAULT
```

Izmena definicije tabele

- Izbacivanje kolone iz tabele realizuje se slede}om naredbom:

```
ALTER TABLE <naziv tabele>
DROP [COLUMN] <naziv kolone>
```

- Kolonu BROJ_RADNIKA mo`emo izbaciti iz tabele ODELJENJE na slede}i na~in:

```
ALTER TABLE ODELJENJE
DROP COLUMN BROJ_RADNIKA
```

Izmena definicije tabele

- Dodavanje ili izbacivanje ograni~enja na vrednosti podataka tabele realizuje se slede}om naredbom:

```
ALTER TABLE <naziv tabele>
ADD [CONSTRAINT <naziv ogranicenja>]
                <ogranicenje tabele> |
DROP CONSTRAINT <naziv ogranicenja>
```

Izbacivanje tabele

- Ukoliko ho}emo da izbacimo iz baze podataka definiciju tabele, zajedno sa podacima koje sadr`i, koristi se DROP TABLE naredba.

  Op{ti oblik naredbe je:

  ```
  DROP TABLE <naziv tabele>
  ```

- Tabelu PREMIJA mo`emo izbaciti iz baze podataka naredbom:

  ```
  DROP TABLE PREMIJA;
  ```

Indeksi

- Tipi~na sintaksa za kreiranje indeksa je:

```
CREATE [UNIQUE] INDEX <naziv indeksa>
ON <naziv tabele> (<naziv kolone1>
                        [, naziv kolone2, ...]);
```

- Indeks se izbacuje naredbom:

```
DROP INDEX <naziv indeksa>;
```

[ema baze podataka

- [ema je kolekcija svih objekata koji dele isti prostor imenovanja. Svaki objekat (tabela, pogled, itd.) pripada ta~no jednoj {emi. Pod pripadno{}u se ne podrazumeva fizi~ka pripadnost, ve} hijerarhijska veza u kojoj, na primer, {ema sadr`i nula ili vi{e tabela, a svaka tabela logi~ki pripada ta~no jednoj {emi.
- [ema ima naziv, koji se mo`e koristiti za kvalifikovanje naziva objekata koji pripadaju {emi.
- [ema se kreira CREATE SCHEMA naredbom, iza koje slede naredbe za kreiranje objekata {eme, prvenstveno tabela, domena i pogleda:

```
CREATE SCHEMA <naziv seme> ...

CREATE TABLE <naziv tabele1> ...
CREATE TABLE <naziv tabele2> ...
...
```

[ema baze podataka

- [ema se izbacuje DROP SCHEMA naredbom. Standard zahteva da se obavezno navede CASCADE ili RESTRICT na~in zadovoljavanja integriteta pri izbacivanju {eme. Naredba:

  ```
  DROP SCHEMA <naziv seme> CASCADE
  ```

  izbaci}e {emu i sve objekte koji joj pripadaju.
- Naredba:

  ```
  DROP SCHEMA <naziv seme> RESTRICT
  ```

  izbaci}e {emu samo ako je prazna, odnosno ako se u njoj ne nalazi ni jedan objekat. RESTRICT klauzula spre~ava izbacivanje {eme, ako je u njoj bar jedan objekat.

Katalog

- Katalog je imenovana kolekcija {ema baze podataka u SQL okru`enju. SQL okru`enje sadr`i nula ili vi{e kataloga, a katalog sadr`i jednu ili vi{e {ema. Svaki katalog sadr`i {emu sa nazivom INFORMATION_SCHEMA, koja predstavlja re~nik podataka. Nju ~ini skup pogleda, odnosno sistemskih tabela, koje sadr`e sve bitne informacije o SQL okru`enju. Sadr`aj sistemskih tabela se automatski odr`ava.
- U SQL standardu ne postoje naredbe za kreiranje i uni{tavanje kataloga. Na~in njihovog kreiranja i uni{tavanja je implementaciono-definisan (implementation-defined), odnosno prepu{ten je vlasnicima softverskih proizvoda koji implementiraju SQL okru`enje.

Katalog

- Pun naziv objekata {eme ima tri komponente, razdvojene ta~kama: naziv kataloga, naziv {eme i naziv objekta. Ako je posmatrani objekat tabela, tada se njen pun naziv specificira na slede}i na~in:

```
<naziv kataloga>.<naziv seme>.<naziv tabele>.
```

- Objekti {eme se mogu referencirati sa eksplicitnim ili implicitnim nazivom kataloga i {eme:

```
FROM <naziv tabele>  -- nekvalifikovan naziv
FROM <naziv seme>.<naziv tabele>
                              -- delimi~no kvalifikovan naziv
FROM <naziv kataloga>.<naziv seme>.<naziv tabele>
                              -- potpuno kvalifikovan naziv
```

Operacije - upiti

- Osnova SQL-a je upitni blok oblika:

```
SELECT < lista atributa >
FROM < lista relacija >
WHERE < kvalifikacioni izraz >.
```

- Listom atributa zadaje se operacija PROJEKCIJE. Kvalifikacionim izrazom zadaju se uslovi SELEKCIJE i SPAJANJA, odnosno iskazi sli~ni iskazima u relacionom ra~unu.
- Klauzule SELECT i FROM su obavezne, dok klauzula WHERE nije.

Upiti nad jednom tabelom kojima se prikazuje prost, neizmenjen sadr`aj te tabele

- *Prikazati {ifre, nazive i lokacije svih odeljenja iz relacije o odeljenjima.*

```
SELECT ODELJENJE#, NAZIV, GRAD
FROM ODELJENJE;

ODELJENJE# NAZIV          GRAD
---------- -------------- -------------
        10 RAD_ZAJ        BEOGRAD
        20 PRIPREMA       NIS
        30 PROJEKTOVANJE  BOR
        40 ISTRAZIVANJE   SARAJEVO
```

Upiti nad jednom tabelom kojima se prikazuje prost, neizmenjen sadr`aj te tabele

- *Kada se tra`e svi atributi neke relacije, umesto navo|enja svakog atributa pojedina~no, mogu}e je koristiti znak "***** "sa istim dejstvom.*

```
SELECT *
FROM ODELJENJE;
```

```
ODELJENJE# NAZIV          GRAD
---------- -------------- -------------
 10 RAD_ZAJ        BEOGRAD
 20 PRIPREMA       NIS
 30 PROJEKTOVANJE  BOR
 40 ISTRAZIVANJE   SARAJEVO
```

- Pored prikazivanja svih atributa neke relacije (tj. kolona neke tabele) koriste}i SELECT klauzulu mogu}e je:
    - izdvojiti specifi~ne atribute relacije
    - kontrolisati redosled pojavljivanja atributa
    - spre~iti selekciju duplih n-torki
- *Prikaziti nazive i {ifre svih odeljenja*

```
SELECT NAZIV, ODELJENJE#
FROM ODELJENJE;

NAZIV          ODELJENJE#
-------------- ----------
RAD_ZAJ                10
PRIPREMA               20
PROJEKTOVANJE          30
ISTRAZIVANJE           40
```

- *Prikazati sve poslove radnika*

```
SELECT POSAO
FROM RADNIK;

POSAO
------------
ANALITICAR
TRG_PUTNIK
TRG_PUTNIK
RUKOVODILAC
TRG_PUTNIK
RUKOVODILAC
RUKOVODILAC
SAVETNIK
PREDSEDNIK
TRG_PUTNIK
ANALITICAR
ANALITICAR
SAVETNIK
ANALITICAR
```

- Koriste}i klauzulu DISTINCT obezbedjujemo prikazivanje samo razli~itih poslova.

```
SELECT DISTINCT POSAO
FROM RADNIK;

POSAO
------------
ANALITICAR
PREDSEDNIK
RUKOVODILAC
SAVETNIK
TRG_PUTNIK
```

WHERE klauzula

- Where klauzula nije obavezna, a mo`e se koristiti sa SELECT, UPDATE i DELETE naredbama. Kori{}ena u SELECT bloku ona omogu}uje:
  1. Selekciju specifi~nih n-torki relacije (redova tabele)
  2. Selekciju n-torki koje zadovoljavaju vi{estruke uslove
  3. Selekciju n-torki koje zadovoljavaju bar jedan od vi{e uslova
  4. Selekciju n-torki koje ne zadovoljavaju odredjene uslove
  5. Selekciju n-torki istovremenim kori{}enjem AND i OR logi~kih operatora
  6. Selekciju n-torki unutar izvesnog raspona
  7. Selekciju n-torki koje zadovoljavaju vrednost u listi vrednosti
  8. Selekciju n-torki koje sadr`e odredjenu kombinaciju karaktera

- *Prikazati sve podatke o radnicima koji rade u odeljenju 30*

```
SELECT *
FROM RADNIK
WHERE ODELJENJE# = 30;
```

SRADNIK	IME	POSAO	SRUKOV	DATZAP	PLATA	PREMIJA	ODELJENJ
3199	Milan	TRG_PUTNIK	3398	1981-02-20	160000	30000	30
3221	Petar	TRG_PUTNIK	3398	1981-02-22	125000	50000	30
3354	Josip	TRG_PUTNIK	3398	1981-09-28	125000	140000	30
3398	Dragan	RUKOVODILAC	3539	1981-05-01	285000		30
3544	Goran	TRG_PUTNIK	3398	1981-09-08	150000	0	30
3600	Janko	ANALITICAR	3398	1981-12-03	95000		30

- Pored operatora "**=**" mogu}e je koristiti i ostale operatore poredjenja:

  - RAZLI^ITO ("**<>**") WHERE ODELJENJE# <> 30
  - VE]E ("**>**") WHERE PREMIJA > PLATA,
  - MANJE ("**<**") WHERE PREMIJA < PLATA,
  - VE]E ILI JEDNAKO ("**>=**")
    WHERE DATZAP >= ‘1981-09-08’
  - MANJE ILI JEDNAKO ("**<=**").
    WHERE DATZAP <= ‘1981-09-08’

- *Prika`i ime, posao i platu svakog radnika iz odeljenja 20 koji zaradjuje vi{e od 200000 din. (kori{}enje logi~kog operatora AND)*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE ODELJENJE# = 20
  AND PLATA > 200000;
```

```
IME        POSAO              PLATA
---------- ------------ ----------
Marko      RUKOVODILAC      297500
Pavle      SAVETNIK         300000
Filip      SAVETNIK         300000
```

- *Prikazati sve podatke o rukovodiocima odeljenja i predsedniku (kori{}enje logi~kog operatora OR)*

```
SELECT *
FROM RADNIK
WHERE POSAO = 'RUKOVODILAC'
   OR POSAO = 'PREDSEDNIK';
```

```
SRADNIK IME    POSAO       SRUKOV DATZAP     PLATA PREMIJA ODELJENJENJE#
------- ------ ----------- ------ --------- ------ ------- -------------
   3266 Marko  RUKOVODILAC   3539 02-APR-81 297500                    20
   3398 Dragan RUKOVODILAC   3539 01-MAY-81 285000                    30
   3482 Ivan   RUKOVODILAC   3539 09-JUN-81 245000                    10
   3539 Jovan  PREDSEDNIK         17-NOV-81 500000                    10
```

- *Prika`i ime, platu, posao i {ifru odeljenja za rukovodioce koji ne rade u odeljenju 30 (kori{}enje logi~kog operatora NOT)*

```
SELECT IME, PLATA, POSAO, ODELJENJE#
FROM RADNIK
WHERE POSAO = 'RUKOVODILAC'
  AND NOT (ODELJENJE# = 30);

IME              PLATA POSAO        ODELJENJE#
---------- ---------- ------------ ----------
Marko            297500 RUKOVODILAC          20
Ivan             245000 RUKOVODILAC          10
```

- *Prika`i ime, platu, posao i {ifru odeljenja za rukovodioca i analiti~are u odeljenju 10 (kori{}enje zagrada da bi se definisao redosled ispitivanja uslova kod istovremene primene AND i OR logi~kih operatora)*

```
SELECT IME, PLATA, POSAO, ODELJENJE#
FROM RADNIK
WHERE (POSAO = 'RUKOVODILAC' OR
       POSAO = 'ANALITICAR' )
  AND ODELJENJE# = 10;

IME              PLATA POSAO        ODELJENJE#
---------- ---------- ------------ ----------
Ivan             245000 RUKOVODILAC          10
Dejan            130000 ANALITICAR           10
```

- *Prika`i ime, posao i platu radnika koji zaradjuju izmedju 120000 i 150000 (kori{}enje klauzule BETWEEN, koja proverava da li je tra`ena vrednost atributa u definisanom rasponu)*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE PLATA BETWEEN 120000 AND 150000;
```

```
IME        POSAO              PLATA
---------- ------------ ----------
Petar      TRG_PUTNIK       125000
Josip      TRG_PUTNIK       125000
Goran      TRG_PUTNIK       150000
Dejan      ANALITICAR       130000
```

Isti uslov mogu}e je realizovati i na slede}i na~in:

```
WHERE PLATA >=120000 AND PLATA <= 150000
```

- *Prika`i ime, posao i sifru odeljenja radnika koji nisu analiti~ari, savetnici niti trgova~ki putnici (kori{}enje operatora IN koji menja vi{estruku primenu operatora OR)*

```
SELECT IME, POSAO, ODELJENJE#
FROM RADNIK
WHERE POSAO NOT IN ('ANALITICAR', 'SAVETNIK',
                    'TRG_PUTNIK');

IME        POSAO        ODELJENJE#
---------- ------------ ----------
Marko      RUKOVODILAC          20
Dragan     RUKOVODILAC          30
Ivan       RUKOVODILAC          10
Jovan      PREDSEDNIK           10
```

- *Prikazati ime, posao i {ifru odeljenja radnika ~ija imena po~inju sa M. (Kori{}enje klauzule LIKE)*

```
SELECT IME, POSAO, ODELJENJE#
FROM RADNIK
WHERE IME LIKE 'M%';

IME        POSAO        ODELJENJE#
---------- ------------ ----------
Milan      TRG_PUTNIK           30
Marko      RUKOVODILAC          20
```

- Klauzula LIKE omogu}uje pretra`ivanje na osnovu "uzorka" odnosno dobijanje informacija i kada ne znamo potpun naziv (tj. vrednost) odredjenog atributa tipa character. Ona koristi dva specijalna karaktera ("**%**", "_") sa slede}im zna~enjem:

  "**%**" predstavlja string od 0 ili vi{e karaktera
  "_" predstavlja poziciju ta~no jednog karaktera.

  Ostali karakteri imaju uobi~ajeno zna~enje.

- *Primeri:*

  ... gde se ime zavr{ava sa N.

  ```
  WHERE IME LIKE '%N'
  ```

  ... gde je tre}i karakter imena R.

  ```
  WHERE IME LIKE '__R%'
  ```

  ... gde je ime duga~ko 5 karaktera.

  ```
  WHERE IME LIKE '_____'
  ```

  ... gde ime nije duga~ko 5 karaktera.

  ```
  WHERE IME NOT LIKE '_____'
  ```

  ... gde je u imenu slovo G posle R.

  WHERE IME LIKE '%R%G%'

- Ponekad vrednosti tekstualnih kolona sadr`e i karaktere '%' i '_'. Klauzula ESCAPE omogu}uje nala`enje takvih kolona tako {to privremeno, unutar upita, poni{tava specijalni efekat karaktera '%' ili '_'. Escape karakter sami biramo navodjenjem u ESCAPE klauzuli i on poni{tava specijalni efekat karaktera ispred koga se neposredno nalazi u uzorku za pretra`ivanje.
- *Prikazati sve podatke o radnicima koji u nazivu posla imaju karakter '_'.*

```
SELECT *
FROM RADNIK
WHERE POSAO LIKE '%*_%' ESCAPE '*';
```

SRADNIK	IME	POSAO	SRUKOV	DATZAP	PLATA	PREMIJA	ODELJENJE#
3199	Milan	TRG_PUTNIK	3398	1981-02-20	160000	30000	30
3221	Petar	TRG_PUTNIK	3398	1981-02-22	125000	50000	30
3354	Josip	TRG_PUTNIK	3398	1981-09-28	125000	140000	30
3544	Goran	TRG_PUTNIK	3398	1981-09-08	150000	0	30

Kori{}enje NULL vrednosti

- dva osnovna tipa NULL vrednosti
  - jo{ nepoznata vrednost
  - neprimenjivo svojstvo
- za testiranje null vrednosti koristi se sintaksa:

IS NULL ili IS NOT NULL

- *Prikazati ime, posao i premiju radnika koji ne primaju premiju*

```
SELECT IME, POSAO, PREMIJA
FROM RADNIK
WHERE PREMIJA IS NULL;

IME        POSAO           PREMIJA
---------- ------------ ----------
Steva      ANALITICAR
Marko      RUKOVODILAC
Dragan     RUKOVODILAC
Ivan       RUKOVODILAC
Pavle      SAVETNIK
Jovan      PREDSEDNIK
Adam       ANALITICAR
Janko      ANALITICAR
Filip      SAVETNIK
Dejan      ANALITICAR
```

- *Prikazati ime, posao i premiju radnika koji primaju premiju*

```
SELECT IME, POSAO, PREMIJA
  FROM RADNIK
 WHERE PREMIJA IS NOT NULL;

IME        POSAO           PREMIJA
---------- ------------ ----------
Milan      TRG_PUTNIK        30000
Petar      TRG_PUTNIK        50000
Josip      TRG_PUTNIK       140000
Goran      TRG_PUTNIK            0
```

ORDER BY klauzula

- Kori{}enjem ORDER BY klauzule mogu}e je sortirati rezultuju}u tabelu po jednom ili vi{e atributa u rastu}em ili opadaju}em redosledu.
- Za specifikaciju rastu}eg redosleda koristi se klauzula ASC, a za specifikaciju opadaju}eg redosleda klauzula DESC. Rastu}i redosled se podrazumeva, pa klauzulu ASC nije neophodno navoditi, za razliku od klauzule DESC koju uvek treba navesti kada se sortira u opadaju}em redosledu.
- ORDER BY je uvek poslednja klauzula u SELECT bloku.

- *Prika`i ime, posao i platu radnika u odeljenju 30 uredjene u rastu}em redosledu poslova i opadaju}em redosledu plata.*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE ODELJENJE# = 30
ORDER BY POSAO ASC, PLATA DESC;

IME        POSAO             PLATA
---------- ------------ ----------
Janko      ANALITICAR         95000
Dragan     RUKOVODILAC       285000
Milan      TRG_PUTNIK        160000
Goran      TRG_PUTNIK        150000
Petar      TRG_PUTNIK        125000
Josip      TRG_PUTNIK        125000
```

Kada se sortiranje vr{i po koloni koja sadr`i NULL vrednosti, redosled prikaza n-torki sa NULL vrednostima u koloni sortiranja uslovljen je realizacijom SQL-a u konkretnom SUBP-u.

Upiti nad jednom tabelom uz prikaz modifikovanog sadr`aja tabele

- Kori{}enjem izraza, agregatnih funkcija i funkcija nad pojedina~nim redovima mogu}e je obraditi podatke baze podataka i prikazati izvedene podatke
- Funkcije i izrazi pi{u se u SELECT listi i ne menjaju sadr`aj tabela baze podataka.

Agregatne funkcije

- Pored prikazivanja prostih vrednosti memorisanih u tabelama baze podataka, SQL ima vi{e funkcija koje se koriste za dobijanje izvedenih, sumarnih informacija.
- Te funkcije se obi~no nazivaju agregatnim funkcijama.
- Primena agregatne funkcije zahteva da redovi tabele na koju se agregatna funkcija primenjuje budu grupisani na neki na~in.
- Svaka agregatna funkcija generi{e jedan rezultuju}i red za svaku grupu redova tabele. Ukoliko grupisanje redova tabele nije eksplicitno specificirano, ~itava tabela tretira se kao jedna grupa.
- Najzna~ajnije agregatne funkcije su:

  AVG (naziv_kolone) - izra~unava srednju vrednost

  SUM (naziv_kolone) - izra~unava ukupnu vrednost

  MIN (naziv_kolone) - nalazi minimalnu vrednost

  MAX (naziv_kolone) - nalazi maksimalnu vrednost

  COUNT (*) - nalazi broj redova u grupi

  COUNT ([**ALL**] naziv_kolone) - nalazi broj definisanih (not null) vrednosti kolone

  COUNT (**DISTINCT** naziv_kolone) - nalazi broj razli~itih definisanih (not null) vrednosti kolone

- *Na}i minimalnu, srednju i maksimalnu platu, kao i broj radnika u odeljenju 10*

```
SELECT MIN (PLATA), AVG (PLATA),
       MAX (PLATA), COUNT (*)
  FROM RADNIK
 WHERE ODELJENJE# = 10;

MIN(PLATA) AVG(PLATA) MAX(PLATA)   COUNT(*)
---------- ---------- ---------- ----------
    130000 291666.667     500000          3
```

- *Na}i ukupnu platu i ukupnu premiju za trgova~ke putnike*

```
SELECT SUM (PLATA), SUM (PREMIJA)
FROM RADNIK
WHERE POSAO = 'TRG_PUTNIK';

SUM(PLATA) SUM(PREMIJA)
---------- ------------
    560000       220000
```

- Pored toga {to argumenti agregatnih funkcija mogu biti izrazi, i same agregatne funkcije mogu biti operandi izraza. To je ilustrovano slede}im upitom:
- *Koliko je srednje godi{nje primanje trgova~kih putnika*

```
SELECT AVG (PLATA + PREMIJA) * 12
FROM RADNIK
WHERE POSAO = 'TRG_PUTNIK';

AVG(PLATA+PREMIJA)*12
---------------------
              2340000
```

GROUP BY klauzula

- Prouzrokuje dobijanje jednog rezultuju}eg reda za svaku razli~itu vrednost kolone po kojoj se vr{i grupisanje.
- *Na}i minimalnu, srednju i maksimalnu platu kao i ukupan broj radnika u svakom odeljenju*

```
SELECT MIN (PLATA), AVG (PLATA),
       MAX (PLATA), COUNT (*), ODELJENJE#
FROM RADNIK
GROUP BY ODELJENJE#;
```

```
MIN(PLATA) AVG(PLATA) MAX(PLATA)   COUNT(*) ODELJENJE#
---------- ---------- ---------- ---------- ----------
    130000 291666.667     500000          3         10
     80000     217500     300000          5         20
     95000 156666.667     285000          6         30
```

- Grupisanje se mo`e vr{iti po vi{e kolona. U tom slu~aju svaka razli~ita postoje}a kombinacija vrednosti kolona ~ini grupu.
- *Izra~unati broj radnika koji obavljaju razli~iti posao unutar svakog odeljenja*

```
SELECT ODELJENJE#, POSAO, COUNT(*)
FROM RADNIK
GROUP BY ODELJENJE#, POSAO;

ODELJENJE# POSAO          COUNT(*)
---------- ------------ ----------
        10 ANALITICAR            1
        10 PREDSEDNIK            1
        10 RUKOVODILAC           1
        20 ANALITICAR            2
        20 RUKOVODILAC           1
        20 SAVETNIK              2
        30 ANALITICAR            1
        30 RUKOVODILAC           1
        30 TRG_PUTNIK            4
```

HAVING klauzula

- odredjuje kriterijume za selekciju grupa koje su prethodno specificirane GROUP BY klauzulom .
- *Prikazati koje poslove obavlja vi{e od dva radnika u svakom odeljenju*

```
SELECT ODELJENJE#, POSAO, COUNT(*)
FROM RADNIK
GROUP BY ODELJENJE#, POSAO
HAVING COUNT (*) > 2;

ODELJENJE# POSAO          COUNT(*)
---------- ------------ ----------
        30 TRG_PUTNIK             4
```

- SQL naredbe mogu sadr`ati aritmeti~ke izraze sastavljene od imena kolona i konstantnih vrednosti povezanih aritmeti~kim operatorima (“+”, “*”, “-”, “/”)
- *Koliko je srednje godi{nje primanje svih trgova~kih putnika.*

```
SELECT AVG (PLATA + PREMIJA) * 12
FROM RADNIK
WHERE POSAO = 'TRG_PUTNIK';

AVG(PLATA + PREMIJA)*12
-----------------------
                2340000
```

- WHERE i GROUP BY klauzula mogu se koristiti zajedno, pri ~emu WHERE klauzula uvek ide pre GROUP BY klauzule.
- *Odrediti srednju godi{nju platu za svako odeljenje. Pri ra~unanju ne uzimati u obzir predsednika i rukovodioce odeljenja.*

```
SELECT ODELJENJE#, AVG (PLATA) * 12
FROM RADNIK
WHERE POSAO NOT IN ('RUKOVODILAC', 'PREDSEDNIK')
GROUP BY ODELJENJE#;

ODELJENJE# AVG(PLATA)*12
---------- -------------
        10       1560000
        20       2370000
        30       1572000
```

Funkcije EVERY, ANY i SOME

- Pored navedenih najzna~ajnijih agregatnih funkcija (AVG, SUM, MIN, MAX i COUNT), SQL:1999 standard uvodi i funkcije:
  - EVERY
  - ANY i
  - SOME
- Ove funkcije su definisane nad logi~kim (BOOLEAN) tipovima podataka.
  - Funkcija EVERY vrati}e vrednost TRUE (istinito) ako i samo ako je vrednost izraza, koji je argument funkcije, istinita (TRUE) u svakom redu grupe nad kojom je funkcija definisana.
  - Funkcije ANY i SOME su sinonimi. One vra}aju vrednost TRUE ako i samo ako je vrednost izraza, koji je argument funkcije, istinita bar u jednom redu grupe nad kojom je funkcija definisana.

Funkcije i izrazi za obradu pojedina~nih redova

- SQL:1999 standard sadr`i ~etiri tipa funkcija za obradu pojedina~nih redova. To su:
  - Numeri~ke funkcije
  - Tekstualne funkcije
  - Datumske funkcije
  - Intervalne funkcije
- Funkcije su grupisane prema tipu rezultuju}e vrednosti. Numeri~ka funkcija, na primer, mo`e imati argumente bilo kog tipa, ali kao rezultat uvek vra}a numeri~ku vrednost.

Numeri~ke funkcije za obradu pojedina~nih redova

- Najzna~ajnije numeri~ke funkcije su
  - POSITION
  - CHARACTER_LENGTH
  - OCTET_LENGTH
  - BIT_LENGTH
  - EXTRACT
  - ABS
  - MOD

Funkcija POSITION

- Funkcija POSITION ima slede}u sintaksu:

```
POSITION (string1 IN string2)
```

- String je bilo koji, eksplicitno ili implicitno odredjeni niz karaktera.
- Funkcija POSITION tra`i string1 u string2. Ukoliko ga nadje, vra}a njegovu po~etnu poziciju, ina~e, ukoliko ga ne nadje, vra}a vrednost 0.

- Slede neki primeri primene funkcije:

```
POSITION ('4' IN '48 sati')
```

vra}a vrednost 1

```
POSITION ('48' IN '48 sati')
```

vra}a vrednost 1

```
POSITION ('48' IN 'Narednih 48 sati')
```

vra}a vrednost 10

```
POSITION ('Zabava' IN 'Bolji zivot')
```

vra}a vrednost 0 ukazuju}i da tra`eni string nije pronadjen

```
POSITION ('' IN 'Istok Zapad')
```

vra}a vrednost 1; ako je string1 du`ine 0 funkcija uvek vra}a vrednost 1

```
POSITION (B'101' IN B'000101001010')
```

vra}a vrednost 4

- Oracle trenutno ne podr`ava funkciju POSITION.
- Umesto nje na raspolaganju je logi~ki ekvivalentna funkcija INSTR slede}e sintakse:

  `INSTR (string2, string1 [, pocetak])-` tra`i string1 u string2 polaze}i od pozicije "pocetak". Ako je nadjen, vra}a se njegova pozicija, ina~e 0
- Primer:

  `INSTR ('48 sati', '4') -` vra}a vrednost 1

Funkcija EXTRACT

- Funkcija EXTRACT ima slede}u sintaksu:

```
EXTRACT (polje_datuma FROM datum)
EXTRACT (polje_datuma FROM interval)
EXTRACT (vremenska_zona FROM datum)
```

- Datum uklju~uje vreme i vremensku zonu, polje_datuma je datumski ili vremenski deo datuma (godina, mesec, ..., sekund), a interval je godina-mesec (year-month) ili dan-vreme (day-time) interval.
- Funkcija izdvaja datumski ili vremenski deo iz datuma ili intervala, odnosno vremensku zonu iz datuma.

- Slede neki primeri primene funkcije:

```
EXTRACT (YEAR FROM DATE '1996-06-28')
```

vra}a vrednost 1996

```
EXTRACT (MONTH FROM DATE '1996-06-28')
```

vra}a vrednost 6

```
EXTRACT (DAY FROM DATE '1996-06-28')
```

vra}a vrednost 28

```
EXTRACT (HOUR FROM TIMESTAMP '1996-06-28 18:00:01.99')
```

vra}a vrednost 18

```
EXTRACT (MINUTE FROM TIMESTAMP '1996-06-28 18:00:01.99')
```

vra}a vrednost 0

```
EXTRACT (SECOND FROM TIMESTAMP '1996-06-28 18:00:01.99')
```

vra}a vrednost 1.99

```
EXTRACT (HOUR FROM INTERVAL '12 17:10:30.11' DAY(2) TO SECOND)
```

vra}a vrednost 17

```
EXTRACT (SECOND FROM INTERVAL '10:30' MINUTE TO SECOND)
```

vra}a vrednost 30

Funkcije CHARACTER_LENGTH i CHAR_LENGTH

- Funkcije CHARACTER_LENGTH i CHAR_LENGTH su sinonimi. Njihova sintaksa je:

  ```
  CHARACTER_LENGTH (string)
  CHAR_LENGTH (string).
  ```

- Funkcija vra}a numeri~ku vrednost koja odgovara du`ini stringa u karakterima.
- Slede neki primeri primene funkcija:

  ```
  CHARACTER_LENGTH ('Baze podataka')
  ```

  - vra}a vrednost 13

  ```
  CHAR_LENGTH ('Istinite lazi')
  ```

  - vra}a vrednost 13

- *Koliko su duga imena odeljenja ?*

```
SELECT NAZIV, CHAR_LENGTH (NAZIV) AS DUZINA
FROM ODELJENJE;

NAZIV                DUZINA
-------------- ----------
RAD_ZAJ                   7
PRIPREMA                  8
PROJEKTOVANJE            13
ISTRAZIVANJE             12
```

- Logi~ki ekvivalentna funkcija u Oracle-u je funkcija LENGTH slede}e sintakse:

```
LENGTH (string)
```

Funkcije OCTET_LENGTH i BIT_LENGTH

- Funkcije OCTET_LENGTH i BIT_LENGTH imaju slede}u sintaksu:

```
OCTET_LENGTH (string)
BIT_LENGTH (string)
```

- OCTET_LENGTH vra}a du`inu stringa u oktetima, a BIT_LENGTH u bitovima.
- Dalje slede neki primeri primene funkcija:

  `OCTET_LENGTH ('Baze podataka')`- vra}a vrednost 13, podrazumevaju}i jedan oktet po karakteru; za multioktet implementacije vra}ena vrednost bi bila razli~ita

  `OCTET_LENGTH (B'1011100001')`- vra}a vrednost 2, jer 10 bita zahteva bar 2 okteta za reprezentaciju (oktet = 8 bita)

  `BIT_LENGTH (B'01111110')` - vra}a vrednost 8
- Oracle trenutno ne podr`ava funkcije OCTET_LENGTH i BIT_LENGTH

Funkcije ABS i MOD

- Funkcija ABS ima slede}u sintaksu:

  ```
  ABS (numerik)
  ```

- Funkcija ABS nalazi apsolutnu vrednost broja.

- Funkcija MOD ima slede}u sintaksu:

  ```
  MOD (numerik1, numerik2).
  ```

- Funkcija nalazi ostatak deljenja prve numeri~ke vrednosti drugom
- Slede primeri:

  `MOD (35, 3)` - vra}a vrednost 2

  `MOD (35, -8)` - vra}a vrednost 3

- Komercijalni relacioni sistemi obi~no imaju {iri skup funkcija u odnosu na funkcije definisane SQL standardom.
- Njima se zna~ajno pove}ava efikasnost rada u konkretnom implementacionom okru`enju.
- Dalje sledi spisak nekih numeri~kih funkcija koje su raspolo`ive u Oracle okru`enju:

  `POWER (broj, e)` - di`e broj na e-ti stepen

  `ROUND (broj [,d])`- zaokru`uje broj na d decimala

  `TRUNC (broj [,d])`- odbacuje ostatak od d-tog decimalnog mesta

- `SIGN (broj)` - vra}a +1 ako je broj >0, 0 ako je broj =0, a -1 ako je broj <0.

  `SQRT (broj)` - nalazi pozitivan kvadratni koren broja

- Broj, e i d su eksplicitne ili implicitne numeri~ke vrednosti.

- Slede}i primer ilustruje primenu funkcije ROUND na numeri~ki izraz
- *Koji radnici zaradjuju vi{e od 1000 dinara po satu. Zaradu po satu zaokru`iti na 2 decimale. (Podrazumeva se da postoje 22 radna dana u mesecu i 8 radnih sati u danu)*

```
SELECT IME,
       ROUND (PLATA / (22 * 8), 2) AS "ZARADA PO CASU"
FROM RADNIK
WHERE PLATA / (22 * 8) > 1000;

IME        ZARADA PO CASU
---------- --------------
Marko              1690.34
Dragan             1619.32
Ivan               1392.05
Pavle              1704.55
Jovan              2840.91
Filip              1704.55
```

Tekstualne funkcije za obradu pojedina~nih redova

- Najzna~ajnije tekstualne funkcije su:
  - SUBSTRING
  - UPPER
  - LOWER
  - TRIM i
  - OVERLAY
- Funkcija SUBSTRING ima slede}u sintaksu:

```
SUBSTRING (string FROM pocetak [FOR duzina])
```

  - String je bilo koji, eksplicitno ili implicitno odredjeni niz karaktera ili niz bitova.
  - Funkcija SUBSTRING izdvaja podniz iz datog niza (string) po~ev od startne pozicije (pocetak) u definisanoj du`ini (duzina).
  - To je mo}na funkcija jer za tekstualne kolone omogu}uje kontrolu vrednosti do nivoa pojedina~nog karaktera, a za binarne do nivoa pojedina~nog bita.

- Dalje slede neki primeri primene funkcije :

```
SUBSTRING ('Narednih 48 sati FROM 1 FOR 10)
```

vra}a vrednost 'Narednih 4'

```
SUBSTRING ('abcdef' FROM -8 FOR 2)
```

vra}a vrednost '' (prazan string)

```
SUBSTRING ('abcdef' FROM -2 FOR 6)
```

vra}a vrednost 'abc'

```
SUBSTRING ('abcdef' FROM 0 FOR 4)
```

vra}a vrednost 'abc'

```
SUBSTRING ('abcdef' FROM 3 FOR -2)
```

vra}a poruku o gre{ci

```
SUBSTRING ('abcdef' FROM 7 FOR 3)
```

vra}a vrednost '' (prazan string)

```
SUBSTRING ('abcdef' FROM 3)
```

vra}a vrednost 'cdef'

```
SUBSTRING (B'101101' FROM 3 FOR 2)
```

vra}a vrednost B'11'

- Logi~ki ekvivalentna u Oracle-u je funkcija SUBSTR slede}e sintakse:

```
SUBSTR (string, pocetak [, duzina])
SUBSTR ('PROJEKTOVANJE', 1, 4) -
```

vra}a vrednost 'PROJ'.

- Funkcija UPPER ima slede}u sintaksu:

  `UPPER (string)`

  – Funkcija menja sva mala slova u velika, kao u slede}em primeru:

  `UPPER ('Hocu da sam veliki !!!')`

  vra}a 'HOCU DA SAM VELIKI !!!'.

- Funkcija LOWER ima slede}u sintaksu:

  `LOWER (string)`

  – Funkcija menja sva velika slova u mala, kao u slede}em primeru:

  `LOWER ('LAKSE JE BITI MALI !!!')`

  vra}a 'lakse je biti mali !!!'.

- Funkcija TRIM ima slede}u sintaksu:

  `TRIM ([BOTH|LEADING|TRAILING] karakter FROM string)`

  – Funkcija elimini{e navedeni karakter sa po~etka i kraja (opcija BOTH), samo sa po~etka (opcija LEADING) ili samo sa kraja (opcija TRAILING) datog stringa.

  `TRIM (LEADING ' ' FROM ' TEST ')-` vra}a vrednost 'TEST ‘

  `TRIM (TRAILING ' ' FROM ' TEST ')-` vra}a vrednost ' TEST‘

  `TRIM (BOTH ' ' FROM ' TEST ')-` vra}a vrednost 'TEST‘

  `TRIM (BOTH 'T' FROM 'TEST')-` vra}a vrednost 'ES'.

- Funkcija OVERLAY ima slede}u sintaksu:

```
OVERLAY (string1 PLACING
          string2 FROM pocetak [FOR duzina]).
```

  – Funkcija omogu}uje zamenu dela niza karaktera ili niza bitova (string1) novim nizom (string2) od date startne pozicije (pocetak) u navedenoj du`ini (duzina), kao u slede}em primeru :

```
OVERLAY ('Nikad subotom' PLACING 'Uvek' FROM 1 FOR 5)
```

  vra}a vrednost 'Uvek subotom'.

- Pored navedenih funkcija za rad sa tekstualnim i binarnim podacima na raspolaganju je i operator konkatenacije (||).
  – On omogu}uje formiranje izraza kojima se spajaju nizovi karaktera ili nizovi bitova.
  – Sintaksa operatora konkatenacije:

```
string1 || string2
```

- *Prika`i imena radnika odeljenja 30 iza kojih neposredno treba da dodje posao koji obavljaju. Sortirati rezultuju}u tabelu u rastu}em redosledu vrednosti kolone POSAO*

```
SELECT IME||', '||POSAO AS RADNIK
FROM RADNIK
WHERE ODELJENJE# = 30
ORDER BY POSAO;

RADNIK
-------------------------
Janko, ANALITICAR
Dragan, RUKOVODILAC
Milan, TRG_PUTNIK
Petar, TRG_PUTNIK
Goran, TRG_PUTNIK
Josip, TRG_PUTNIK
```

Datumske funkcije za obradu pojedina~nih redova

- SQL standard podr`ava slede}e datumske funkcije:
  - CURRENT_DATE
  - CURRENT_TIME
  - CURRENT_TIMESTAMP
  - LOCALTIME i
  - LOCALTIMESTAMP.
- Prve tri funkcije definisane su SQL-92 standardom, a poslednje dve su uvedene u SQL:1999 standardu.
- Funkcija CURRENT_DATE je bez argumenata. Ona prikazuje teku}i datum, na primer 2002-01-07. Vra}ena vrednost je tipa DATE.
- Ostale funkcije imaju jedan argument, preciznost, kojim je odredjen deo sekunde (desetinka, stotinka, hiljaditi deo itd.) sa kojim je potrebno iskazati ta~nost vremena.

- Funkcije CURRENT_TIME i LOCALTIME prikazuju teku}e vreme
  - Prva funkcija daje teku}e vreme sa vremenskim odstupanjem od univerzalno koordinisanog vremena (UTC) za datu vremensku zonu, a druga je bez odstupanja za vremensku zonu.
  - Rezultuju}a vrednost prve funkcije je tipa TIME WITH TIME ZONE, a druge TIME.
- Sledi sintaksa i primeri primene navedenih funkcija:

```
CURRENT_TIME [(preciznost)]
LOCALTIME [(preciznost)]
```

`CURRENT_TIME (2)` - vra}a teku}e vreme, npr. 17:12:30.32 +01:00, sa ta~no{}u na stote delove sekunda i sa vremenskim odstupanjem

`LOCALTIME (2)` - vra}a teku}e vreme, npr. 17:12:30.32, sa ta~no{}u na stote delove sekunda.

- Funkcije CURRENT_TIMESTAMP i LOCALTIMESTAMP
  - prikazuju teku}e "datum + vreme“
  - prva funkcija vr{i prikaz sa odstupanjem od univerzalno koordinisanog vremena (UTC) za datu vremensku zonu
  - druga je bez odstupanja za vremensku zonu.
  - Rezultuju}a vrednost prve funkcije je tipa TIMESTAMP WITH TIME ZONE, a druge TIMESTAMP.
- Sledi sintaksa i primeri primene navedenih funkcija:

  `CURRENT_TIMESTAMP [(preciznost)]`

  `LOCALTIMESTAMP [(preciznost)]`

  `CURRENT_TIMESTAMP (1)` - vra}a teku}e "datum + vreme", npr. 2002-01-07:17:12:30.3 +01:00, sa ta~no{}u na desete delove sekunda i sa vremenskim odstupanjem

  `LOCALTIMESTAMP (1)`- vra}a teku}e "datum + vreme", npr. 2002-01-07:17:12:30.3, sa ta~no{}u na desete delove sekunda.

- Navedene datumske funkcije, kombinovane sa vrednostima intervalnog tipa, mogu formirati izraze sa rezultuju}om vredno{}u datumskog tipa.
- Na primer, izraz

  ```
  CURRENT_DATE + INTERVAL '1' DAY
  ```

  vrati}e istu vrednost koju }e funkcija CURRENT_DATE imati sutra.
- Medjutim, ako od jednog datuma oduzmemo drugi datum, dobi}emo rezultuju}u vrednost intervalnog tipa, kao u slede}em slu~aju

  ```
  (CURRENT_DATE - DATZAP) YEAR TO MONTH.
  ```

Intervalne funkcije za obradu pojedina~nih redova

- Postoji samo jedna intervalna funkcija, a uvedena je u SQL:1999 standardu.
- To je funkcija ABS i potpuno je analogna funkciji ABS koja se primenjuje na numeri~ke vrednosti.
- Funkcija ABS ima jedan operand, koji mora biti intervalnog tipa, i vra}a rezultuju}u vrednost koja je istog tipa i iste preciznosti kao i operand funkcije.
- Na primer, funkcija :

```
ABS (TIME '12:00:00' - TIME '13:00:00'),
```

vrati}e slede}u rezultuju}u vrednost:

```
INTERVAL +'1:00:00' HOUR TO SECOND.
```

Izrazi CASE, NULLIF, COALESCE i CAST

- Pored prethodno navedenih osnovnih funkcija i izraza, SQL:1999 standard podr`ava i kompleksne uslovne izraze (CASE), izraze kojima se konvertuju podaci jednog tipa u drugi (CAST) i specijalne slu~ajeve uslovnih, CASE izraza (NULLIF i COALESCE).
- CASE izraz ima slede}u sintaksu:

```
CASE
   WHEN uslov1 THEN rezultat1
   WHEN uslov2 THEN rezultat2
   ...
   WHEN uslovn THEN rezultatn
   ELSE rezultatx
END.
```

- CASE izraz je sli~an CASE naredbi nekog klasi~nog programskog jezika.
  - Su{tinska razlika je u tome da CASE izraz nije izvr{na naredba, ve} uslovni izraz, koji se u SQL naredbama mo`e koristiti na bilo kom mestu gde je dozvoljeno kori{}enje vrednosnog izraza.
- Jedna od va`nih primena CASE izraza je za transformaciju nedefinisane (NULL) vrednosti u odredjenu konkretnu vrednost.
  - NULL vrednost se ne koristi pri izra~unavanju izraza i funkcija. Da bi se izra~unavanje ipak omogu}ilo koristi se CASE izraz koji privremeno, unutar upita, menja NULL vrednost sa vredno{}u za koju se sami odlu~imo, naj~e{}e vredno{}u koja je neutralna u odnosu na `eljenu operaciju.

- *Prika`i ukupnu mese~nu zaradu radnika u odeljenju 30*

```
SELECT IME, POSAO, PLATA, PREMIJA,
       PLATA + (CASE WHEN PREMIJA IS NULL THEN 0
                     ELSE PREMIJA
                END) AS PRIMANJA
FROM RADNIK
WHERE ODELJENJE# = 30;
```

IME	POSAO	PLATA	PREMIJA	PRIMANJA
Milan	TRG_PUTNIK	160000	30000	190000
Petar	TRG_PUTNIK	125000	50000	175000
Josip	TRG_PUTNIK	125000	140000	265000
Dragan	RUKOVODILAC	285000		285000
Goran	TRG_PUTNIK	150000	0	150000
Janko	ANALITICAR	95000		95000

- Sli~an efekat CASE izraz ima i kao argument funkcija
- *Za odeljenje 30 izra~unaj srednju platu, srednju premiju, srednju mese~nu zaradu za radnike koji primaju premiju i srednju mese~nu zaradu za sve radnike.*

```
SELECT AVG (PLATA), AVG (PREMIJA),
       AVG (PLATA + PREMIJA),
       AVG (PLATA + (CASE WHEN PREMIJA IS NULL THEN 0
                          ELSE PREMIJA
                     END)) AS "PROSEK SVIH"
FROM RADNIK
WHERE ODELJENJE# = 30;

AVG(PLATA) AVG(PREMIJA) AVG(PLATA+PREMIJA) PROSEK SVIH
---------- ------------ ------------------ -----------
156666.667        55000             195000  193333.333
```

- Mogu}e je koristiti i skra}eni oblik CASE izraza, koji je slede}eg izgleda :

```
CASE izraz
   WHEN izraz1 THEN rezultat1
   WHEN izraz2 THEN rezultat2
   ...
   WHEN izrazn THEN rezultatn
   ELSE rezultatx
END
```

- Navedeni CASE izraz je samo skra}ena verzija slede}eg op{teg CASE izraza:

```
CASE
   WHEN izraz = izraz1 THEN rezultat1
   WHEN izraz = izraz2 THEN rezultat2
   ...
   WHEN izraz = izrazn THEN rezultatn
   ELSE rezultatx
END.
```

- Primenu skra}enog oblika CASE izraza ilustrova}emo slede}im primerom:
- *Koriste}i kolonu POSAO formiraj kolonu KLASA tako {to za posao analiti~ara vrednost klase treba da bude 1, rukovodioca - 3, predsednika - 5, a za svaki drugi - 2*

```
SELECT IME, POSAO,
   CASE POSAO WHEN 'ANALITICAR' THEN 1
              WHEN 'RUKOVODILAC' THEN 3
              WHEN 'PREDSEDNIK' THEN 5
              ELSE 2
   END AS KLASA
FROM RADNIK;
```

IME	POSAO	KLASA
Steva	ANALITICAR	1
Milan	TRG_PUTNIK	2
Petar	TRG_PUTNIK	2
Marko	RUKOVODILAC	3
Josip	TRG_PUTNIK	2
Pavle	SAVETNIK	2
Jovan	PREDSEDNIK	5
...		

- Korisnici se ponekad, i pored toga {to SQL podr`ava rad sa NULL vrednostima, odlu~uju da NULL vrednosti predstave na tradicionalan na~in, nekom konkretnom vredno{}u, recimo -1, koja se ne pojavljuje kao mogu}a vrednost posmatrane kolone.
- Ako `elimo da takvoj koloni vratimo SQL funkcionalnost rada sa NULL vrednostima, mo`emo koristiti slede}i CASE izraz:

```
CASE
   WHEN naziv_kolone = -1 THEN NULL
   ELSE naziv_kolone
END.
```

- Kako je procenjeno da se potreba za prethodnom transformacijom relativno ~esto javlja, uveden je NULLIF izraz slede}e sintakse:

  NULLIF ($izraz_1$, $izraz_2$).

- Semantika NULLIF izraza je:
  - ako je vrednost izraza1 jednaka vrednosti izraza2, vrati NULL vrednost, ina~e vrati vrednost izraza1.
  - NULLIF izraz za prikazani CASE izraz bio bi slede}eg izgleda:

```
NULLIF (naziv_kolone, -1).
```

- COALESCE izraz ima slede}u sintaksu:

  COALESCE ($izraz_1$, $izraz_2$, ..., $izraz_n$).

  – Ako je vrednost izraza1 definisana, not null vrednost, rezultat COALESCE izraza je vrednost izraza1.
  – Ako je vrednost izraza1 nedefinisana, null vrednost, vr{i se provera vrednosti izraza2 na identi~an na~in.
  – COALESCE izraz vra}a vrednost NULL ako nijedna od vrednosti izraza iz liste nije definisana, not null vrednost.
  – COALESCE izraz je samo specijalan slu~aj CASE izraza.
  – COALESCE ($izraz_1$, $izraz_2$, $izraz_3$) je ekvivalent slede}oj CASE naredbi:

```
CASE
   WHEN izraz1 IS NOT NULL THEN izraz1
   WHEN izraz2 IS NOT NULL THEN izraz2
   ELSE izraz3
END.
```

- Prikaz ukupne mese~ne zarade radnika u odeljenju 30 mogli smo, kori{}enjem COALESCE izraza, re{iti i na slede}i na~in:

```
SELECT IME, POSAO, PLATA, PREMIJA,
   COALESCE (PLATA + PREMIJA, PLATA) AS PRIMANJA
FROM RADNIK
WHERE ODELJENJE# = 30;
```

IME	POSAO	PLATA	PREMIJA	PRIMANJA
Milan	TRG_PUTNIK	160000	30000	190000
Petar	TRG_PUTNIK	125000	50000	175000
Josip	TRG_PUTNIK	125000	140000	265000
Dragan	RUKOVODILAC	285000		285000
Goran	TRG_PUTNIK	150000	0	150000
Janko	ANALITICAR	95000		95000

- CAST izraz ima slede}u sintaksu
  `CAST (izraz AS tip_podataka).`
  - Ona omogu}uje transformaciju vrednosti jednog tipa podataka u drugi.
  - Dalje sledi primer primene funkcije:
- *Za svakog radnika odeljenja 30 prika`i {ifru, iza koje neposredno treba da dodje ime, a iza imena posao koji obavlja.*

```
SELECT CAST (SRADNIK AS CHAR(4))||', '||IME||', '||POSAO
       AS RADNIK
FROM RADNIK
WHERE ODELJENJE# = 30;
 RADNIK
------------------------------
3199, Milan, TRG_PUTNIK
3221, Petar, TRG_PUTNIK
3354, Josip, TRG_PUTNIK
3398, Dragan, RUKOVODILAC
3544, Goran, TRG_PUTNIK
```

Ulaganje upita nad jednom relacijom u upit nad drugom relacijom

- U SQL-u rezultat jednog upita mo`e biti dinami~ki zamenjen u WHERE klauzuli drugog upita.
- *Prikazati ime i posao svakog radnika koji ima isti posao kao Dejan*

```
SELECT IME, POSAO
  FROM RADNIK
 WHERE POSAO = (SELECT POSAO FROM RADNIK
                WHERE IME = 'Dejan');
```

```
IME        POSAO
---------- ------------
Steva      ANALITICAR
Adam       ANALITICAR
Janko      ANALITICAR
Dejan      ANALITICAR
```

- *Prikazati ime i posao radnika koji rade u Beogradu*

```
SELECT IME, POSAO
FROM RADNIK
WHERE ODELJENJE# = (SELECT ODELJENJE#
                    FROM ODELJENJE
                    WHERE GRAD = 'BEOGRAD');
```

```
IME        POSAO
---------- ------------
Ivan       RUKOVODILAC
Jovan      PREDSEDNIK
Dejan      ANALITICAR
```

- *Prikazati ime, posao i platu radnika u odeljenju 20 koji imaju isti posao kao radnici odeljenja projektovanje*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE ODELJENJE# = 20
  AND POSAO IN
         (SELECT POSAO FROM RADNIK
          WHERE ODELJENJE# IN
                 (SELECT ODELJENJE# FROM ODELJENJE
                  WHERE NAZIV = 'PROJEKTOVANJE'));
```

```
IME        POSAO              PLATA
---------- ------------ ----------
Steva      ANALITICAR          80000
Adam       ANALITICAR         110000
Marko      RUKOVODILAC        297500
```

- *Ko je najbolje pla}eni radnik u svakom odeljenju?*

```
SELECT IME, ODELJENJE#, POSAO, PLATA
FROM RADNIK
WHERE (ODELJENJE#, PLATA) IN
          (SELECT ODELJENJE#, MAX (PLATA)
           FROM RADNIK
           GROUP BY ODELJENJE#);
```

IME	ODELJENJE#	POSAO	PLATA
Jovan	10	PREDSEDNIK	500000
Pavle	20	SAVETNIK	300000
Filip	20	SAVETNIK	300000
Dragan	30	RUKOVODILAC	285000

- Upit koji se izvr{ava jedanput za svaki selektovani red spoljnog upita
- *Prika`i {ifru odeljenja, ime i platu svakog radnika ~ija je plata ve}a od prose~ne plate svog odeljenja.*

```
SELECT ODELJENJE#, IME, PLATA
FROM RADNIK R
WHERE PLATA > (SELECT AVG(PLATA)
               FROM RADNIK
               WHERE ODELJENJE# = R.ODELJENJE#);
```

```
ODELJENJE# IME          PLATA
---------- ---------- -------
        30 Milan       160000
        20 Marko       297500
        30 Dragan      285000
        20 Pavle       300000
        10 Jovan       500000
        20 Filip       300000
```

Primena operatora unije (UNION), preseka (INTERSECT) i razlike (EXCEPT)

- Da bi skupovne operacije mogle da se primene SELECT blokovi (upiti) na koje se operacije odnose moraju imati isti broj rezultuju}ih kolona i rezultuju}e kolone moraju odgovarati po tipu.
- U cilju ilustracije da}emo primer primene operacije EXCEPT na dve tabele. Pretpostavimo da je kreirana i tabela o trgova~kim putnicima ~ija bi shema bila slede}a:

  TRGOVACKI_PUTNIK (SRADNIK, PREMIJA).
- *Zahtev da se prika`u {ifre radnika koji nisu trgova~ki putnici mogao bi da se re{i na slede}i na~in*

```
SELECT SRADNIK                  SRADNIK
FROM RADNIK                    ----------
EXCEPT                            3069
SELECT SRADNIK                    3266
FROM TRGOVACKI_PUTNIK;            3398
                                   ...
```

- SQL standard je odstupio od relacione teorije i dozvoljava postojanje duplikata u rezultatu primene navedenih skupovnih operacija.
- O~uvanje duplikata - klju~na re~ ALL odmah iza naziva operatora.
- Eliminisanje duplikata - klju~na re~ DISTINCT (DISTINCT se podrazumeva po default-u).
- U cilju potpunog razja{njenja pokaza}emo primenu ovih operatora sa klauzulama ALL i DISTINCT na slede}e dve tabele

```
T1                  T2

ID NAZIV            ID NAZIV
-- -----            -- -----
10 AAA              10 AAA
10 AAA              10 AAA
10 AAA              40 DDD
20 BBB              50 EEE
30 CCC              60 FFF
40 DDD
```

```
SELECT *
FROM T1
UNION ALL
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
10 AAA
10 AAA
10 AAA
10 AAA
10 AAA
20 BBB
30 CCC
40 DDD
40 DDD
50 EEE
60 FFF
```

```
SELECT *
FROM T1
UNION DISTINCT
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
20 BBB
30 CCC
40 DDD
50 EEE
60 FFF
```

```
SELECT *
FROM T1
INTERSECT ALL
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
10 AAA
40 DDD
```

```
SELECT *
FROM T1
INTERSECT DISTINCT
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
40 DDD
```

```
SELECT *
FROM T1
EXCEPT ALL
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
20 BBB
30 CCC
```

```
SELECT *
FROM T1
EXCEPT DISTINCT
SELECT *
FROM T2;

ID NAZIV
-- -----
20 BBB
30 CCC
```

- Pored ALL i DISTINCT neposredno iza naziva operatora mo`e se navesti i klju~na re~ CORRESPONDING.
- Ona omogu}uje primenu operatora UNION, INTERSECT i EXCEPT na tabele koje nisu kompatibilne na uniju, ali imaju neke 'zajedni~ke' kolone (kolone identi~nih naziva).
- CORRESPONDING mo`e biti sa listom ili bez liste zajedni~kih kolona.
- Za ilustraciju, posmatrajmo slede}e dve tabele:

T1

ID	NAZIV	TEZINA
10	AAA	100
10	AAA	100
10	AAA	100
20	BBB	200
30	CCC	400
40	DDD	300

T2

ID	NAZIV	BOJA
10	AAA	PLAVA
10	AAA	PLAVA
40	DDD	ZELENA
50	EEE	BELA
60	FFF	CRNA

- Mada tabele T1 i T2 nisu kompatibilne na uniju, kori{}enjem klauzule CORRESPONDING operator UNION (takodje i INTERSECT i EXCEPT) se mo`e primeniti na njih na slede}i na~in:

```
SELECT *
FROM T1
UNION CORRESPONDING
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
20 BBB
30 CCC
40 DDD
50 EEE
60 FFF
```

- Pri obradi i izvr{avanju prikazanog upita odigrava se slede}a logi~ka sekvenca akcija:
  - Zvezdica (*) iz SELECT liste prevodi se u kompletnu listu kolona tabele.
  - Lista se redukuje tako da ostaju samo kolone koje su implicitno ili eksplicitno odredjene CORRESPONDING klauzulom.
  - Rezultuju}a tabela ne sadr`i duplikate, jer je DISTINCT default opcija.
- Napomenimo na kraju da bi se identi~an rezultat dobio izvr{avanjem slede}e naredbe:

```
SELECT ID, NAZIV
FROM T1
UNION
SELECT ID, NAZIV
FROM T2;
```

JOIN - spajanje dve ili vi{e tabela

- Kada su nam potrebni podaci iz vi{e tabela, koristi se JOIN. JOIN povezuje n-torke razli~itih tabela koriste}i zajedni~ke atribute.
- U skladu sa relacionom teorijom podr`ani su razli~iti tipovi spajanja:
  - dekartov proizvod
  - ekvispajanje (spajanje na jednakost)
  - prirodno spajanje i
  - spoljno spajanje.
- Operacija spajanja mo`e se zahtevati:
  - implicitno - zadavanjem uslova spajanja u WHERE klauzuli upita
    - Ovaj na~in je klasi~an na~in spajanja, koji je bio i jedini u SQL-86 i SQL-89 standardu.
  - eksplicitno - navodjenjem tipa i uslova spajanja u FROM klauzuli upita
    - SQL-92 i SQL:1999 standard, pored implicitnog, omogu}uju i eksplicitno navodjenje tipa spajanja.

Ekvispajanje

- *Prika`i za svakog radnika ime, posao i podatke o odeljenju u kom radi.*

```
SELECT IME, POSAO, RADNIK.ODELJENJE#,
       ODELJENJE.ODELJENJE#, NAZIV, GRAD
FROM RADNIK, ODELJENJE
WHERE RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;
```

IME	POSAO	ODELJENJE#	ODELJENJE#	NAZIV	GRAD
Ivan	RUKOVODILAC	10	10	RAD_ZAJ	BEOGRAD
Jovan	PREDSEDNIK	10	10	RAD_ZAJ	BEOGRAD
Dejan	ANALITICAR	10	10	RAD_ZAJ	BEOGRAD
Steva	ANALITICAR	20	20	PRIPREMA	NIS
Adam	ANALITICAR	20	20	PRIPREMA	NIS
Filip	SAVETNIK	20	20	PRIPREMA	NIS
Pavle	SAVETNIK	20	20	PRIPREMA	NIS
Marko	RUKOVODILAC	20	20	PRIPREMA	NIS
Milan	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR
Dragan	RUKOVODILAC	30	30	PROJEKTOVANJE	BOR
Josip	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR
Janko	ANALITICAR	30	30	PROJEKTOVANJE	BOR
Goran	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR
Petar	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR

Pored prikazane klasi~ne sintaksne podr{ke operacije ekvispajanja, SQL-92 i SQL:1999 standard omogu}uju eksplicitno navodjenje operacije spajanja u FROM klauzuli na jedan od slede}a dva na~ina:

```
SELECT *
FROM RADNIK JOIN ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;

SELECT *
FROM RADNIK JOIN ODELJENJE
     USING (ODELJENJE#);
```

Prirodno spajanje

- Ekvispajanje uvek daje rezultuju}u tabelu sa dve identi~ne kolone, odnosno kolone sa identi~nim sadr`ajem.
- Kada se iz rezultata ekvispajanja izbaci jedna od dve identi~ne kolone dobija se prirodno spajanje.
- Prirodno spajanje tabela RADNIK i ODELJENJE kori{}enjem klasi~ne sintakse mo`e se prikazati slede}om SQL naredbom:

```
SELECT RADNIK.*, NAZIV, GRAD
FROM RADNIK, ODELJENJE
WHERE RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;
```

- Eksplicitnim navodjenjem operacije prirodnog spajanja dobija se slede}a SQL naredba:

```
SELECT *
FROM RADNIK NATURAL JOIN ODELJENJE;
```

- Uslov spajanja nije eksplicitno naveden. Spajanje se vr{i po svim kolonama sa identi~nim nazivima u obe tabele.

Uslov spajanja i uslov selekcije u istom upitu

- *Za svakog analiti~ara prika`i ime i grad u kome radi.*

  Ako se upit realizuje klasi~nom sintaksom i uslov spajanja i uslov selekcije bi}e u WHERE klauzuli.

```
SELECT IME, GRAD
FROM RADNIK, ODELJENJE
WHERE RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#
  AND POSAO = 'ANALITICAR';

IME        GRAD
---------- -------------
Dejan      BEOGRAD
Steva      NIS
Adam       NIS
Janko      BOR
```

Ako se upit realizuje eksplicitnim navodjenjem operacije spajanja u FROM klauzuli, u WHERE klauzuli osta}e samo uslov selekcije.

```
SELECT IME, GRAD
FROM RADNIK JOIN ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#
WHERE POSAO = 'ANALITICAR';

ili
SELECT IME, GRAD
FROM RADNIK JOIN ODELJENJE
     USING (ODELJENJE#)
WHERE POSAO = 'ANALITICAR';
```

Dekartov proizvod

- Ukoliko se u upitu izostavi uslov spajanja dobija se dekartov proizod
- *Ponovimo predhodni primer zapisan klasi~nom sintaksom izostavljaju}i uslov spajanja .*

```
SELECT IME, GRAD
FROM RADNIK, ODELJENJE
WHERE POSAO = 'ANALITICAR';
```

```
 IME        GRAD
---------- -------------
Steva      BEOGRAD
Adam       BEOGRAD
Janko      BEOGRAD
Dejan      BEOGRAD
Steva      NIS
Adam       NIS
Janko      NIS
Dejan      NIS
Steva      BOR
Adam       BOR
Janko      BOR
Dejan      BOR
Steva      SARAJEVO
Adam       SARAJEVO
Janko      SARAJEVO
Dejan      SARAJEVO
```

- Eksplicitnim navodjenjem operacije dekartovog proizvoda prethodni upit bi se realizovao SQL naredbom slede}eg izgleda:

```
SELECT IME, GRAD
FROM RADNIK CROSS JOIN ODELJENJE
WHERE POSAO = 'ANALITICAR';
```

Spajanje tabele sa samom sobom (SELF JOIN)

- Tabela mo`e biti spojena sa samom sobom po kolonama koje sadr`e isti tip informacija. SELF JOIN spaja redove tabele sa ostalim ili istim redovima te iste tabele.
- *Prika`i ime i posao svakog radnika i njegovog pretpostavljenog.*

```
SELECT PODR.IME, PODR.POSAO, PODR.SRUKOV,
       NADR.SRADNIK AS SEF,
       NADR.IME AS SEF_IME, NADR.POSAO AS SEF_POSAO
FROM RADNIK AS PODR, RADNIK AS NADR
WHERE PODR.SRUKOV = NADR.SRADNIK;
```

IME	POSAO	SRUKOV	SEF	SEF_IME	SEF_POSAO
Pavle	SAVETNIK	3266	3266	Marko	RUKOVODILAC
Filip	SAVETNIK	3266	3266	Marko	RUKOVODILAC
Milan	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Petar	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Janko	ANALITICAR	3398	3398	Dragan	RUKOVODILAC
Goran	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Josip	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Dejan	ANALITICAR	3482	3482	Ivan	RUKOVODILAC
Adam	ANALITICAR	3488	3488	Pavle	SAVETNIK
Marko	RUKOVODILAC	3539	3539	Jovan	PREDSEDNIK
Ivan	RUKOVODILAC	3539	3539	Jovan	PREDSEDNIK
Dragan	RUKOVODILAC	3539	3539	Jovan	PREDSEDNIK
Steva	ANALITICAR	3602	3602	Filip	SAVETNIK

- Eksplicitnim navodjenjem operacije spajanja prethodni upit bi se realizovao SQL naredbom slede}eg izgleda:

```
SELECT PODR.IME, PODR.POSAO, PODR.SRUKOV,
       NADR.SRADNIK AS SEF,
       NADR.IME AS SEF_IME, NADR.POSAO AS
  SEF_POSAO
FROM RADNIK AS PODR JOIN RADNIK AS NADR
     ON PODR.SRUKOV = NADR.SRADNIK;
```

Spoljno spajanje (OUTER JOIN)

- Spajanje tabela mo`e biti unutra{nje (INNER) ili spoljno (OUTER).
- Ukoliko se vrsta spajanja ne navede eksplicitno podrazumeva se unutra{nje spajanje.
- Ponovimo primere za ekvispajanje i prirodno spajanje sa eksplicitnim navodjenjem klauzule za unutra{nje (INNER) spajanje:

```
SELECT *
FROM RADNIK INNER JOIN ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;

SELECT *
FROM RADNIK NATURAL INNER JOIN ODELJENJE;
```

- Spoljno spajanje mo`e biti levo (LEFT), desno (RIGHT) i centralno (FULL).
  - Levo spoljno spajanje omogu}uje uklju~ivanje u rezultuju}u tabelu svih redova tabele sa leve strane JOIN klauzule tako {to se praznim redom pro{iruje tabela sa desne strane.
  - Desno spoljno spajanje omogu}uje uklju~ivanje u rezultuju}u tabelu svih redova tabele sa desne strane JOIN klauzule tako {to se praznim redom pro{iruje tabela sa leve strane.
  - Centralno spoljno spajanje omogu}uje uklju~ivanje u rezultuju}u tabelu svih redova i leve i desne tabele tako {to se obe tabele pro{iruju praznim redom.

- Levo, desno ili centralno spoljno spajanje tabela RADNIK i ODELJENJE mo`e se realizovati jednom od slede}e dve SQL naredbe:

```
SELECT *
FROM RADNIK {LEFT | RIGHT | FULL} [OUTER] JOIN
     ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;

SELECT *
FROM RADNIK {LEFT | RIGHT | FULL} [OUTER] JOIN
     ODELJENJE
     USING (ODELJENJE#);
```

- Prika`i naziv odeljenja kao i ime i posao svakog radnika koji u njemu radi, uklju~uju}i i odeljenja bez radnika.

  SQL naredba zapisana originalnim Oracle dijalektom bila bi slede}eg izgleda:

```
SELECT NAZIV, IME, POSAO
FROM ODELJENJE, RADNIK
WHERE ODELJENJE.ODELJENJE# = RADNIK.ODELJENJE#(+);
NAZIV          IME        POSAO
-------------- ---------- ------------
RAD_ZAJ        Ivan       RUKOVODILAC
RAD_ZAJ        Jovan      PREDSEDNIK
RAD_ZAJ        Dejan      ANALITICAR
PRIPREMA       Steva      ANALITICAR
PRIPREMA       Adam       ANALITICAR
PRIPREMA       Filip      SAVETNIK
PRIPREMA       Pavle      SAVETNIK
PRIPREMA       Marko      RUKOVODILAC
PROJEKTOVANJE  Milan      TRG_PUTNIK
PROJEKTOVANJE  Dragan     RUKOVODILAC
PROJEKTOVANJE  Josip      TRG_PUTNIK
PROJEKTOVANJE  Janko      ANALITICAR
PROJEKTOVANJE  Goran      TRG_PUTNIK
PROJEKTOVANJE  Petar      TRG_PUTNIK
ISTRAZIVANJE
```

- SQL naredba sa eksplicitnim navodjenjem spoljnog spajanja bila bi slede}eg izgleda:

```
SELECT NAZIV, IME, POSAO
FROM ODELJENJE LEFT [OUTER] JOIN RADNIK
     ON ODELJENJE.ODELJENJE# = RADNIK.ODELJENJE#;
ili
SELECT NAZIV, IME, POSAO
FROM ODELJENJE LEFT [OUTER] JOIN RADNIK
     USING (ODELJENJE#);
```

A`uriranje baze podataka
(naredbe INSERT, UPDATE i DELETE)

- A`uriranje u {irem smislu zna~enja te re~i obuhvata dodavanje, izmenu sadr`aja i brisanje reda ili redova tabele. Te osnovne operacije realizuju se SQL naredbama INSERT, UPDATE i DELETE sa slede}im zna~enjem:

  - **INSERT-** dodavanje reda ili redova u tabelu
  - **UPDATE-** izmena sadr`aja postoje}eg reda ili redova tabele
  - **DELETE-** brisanje postoje}eg reda ili redova tabele

Dodavanje novih redova - INSERT

- Razmatra}e se slede}i tipovi insert naredbe:
  1. Ubacivanje vrednosti svih atributa n-torke;
  2. Ubacivanje vrednosti nekih atributa n-torke;
  3. Ubacivanje podataka iz jedne tabele u drugu.

1. U prvom slu~aju nije potrebno specificirati nazive atributa, pa insert naredba ima oblik:

```
INSERT INTO naziv_tabele
VALUES (vrednost_atr1, vrednost_atr2, . . . ).
```

- Za svaki atribut mora postojati vrednost, pri ~emu je NULL dozvoljena opcija za svaki atribut koji nije NOT NULL.

- *Ubaciti podatke o Bojanu, pravniku, koji po~inje da radi u odeljenju 10 7.04.1984 sa platom od 450000 i bez prava na premiju. Neposredni rukovodilac jo{ nije poznat.*

```
INSERT INTO RADNIK VALUES
     (3545, 'Bojan', 'PRAVNIK', NULL,
     '1984-04-07', 450000,      NULL, 10)
```

- *Prika`imo rezultat insert naredbe*

```
SELECT *
FROM RADNIK
WHERE IME = 'Bojan';
```

```
SRADNIK IME   POSAO   SRUKOV DATZAP     PLATA PREMIJA ODELJENJE#
------- ----- ------- ------ ---------- ----- ------- ----------
   3545 Bojan PRAVNIK        1984-04-07 450000                10
```

2. Ako `elimo da unesemo vrednost za samo neke atribute, nazivi tih atributa moraju se eksplicitno navesti. U tom slu~aju naredba insert ima oblik:

```
INSERT INTO naziv_tabele (atr1, atr2, . . . )
VALUES (vrednost_atr1, vrednost_atr2, . . . ).
```

Vrednosti za NOT NULL atribute moraju biti unete.

- *Uneti podatke o Srdjanu, planeru, koji je primljen u odeljenje 20.*

```
INSERT INTO RADNIK
 (IME, SRADNIK, ODELJENJE#, POSAO)
VALUES ('Srdjan', 3562, 20, 'PLANER')
```

- *Prika`imo rezultat insert naredbe*

```
SELECT *
FROM RADNIK
WHERE IME = 'Srdjan';

 SRADNIK IME    POSAO    SRUKOV DATZAP  PLATA  PREMIJA ODELJENJE#
------- ------ ------- ------- ------ ------ -------- ----------
   3562 Srdjan PLANER                                         20
```

3. Ukoliko obe tabele imaju isti broj atributa i ukoliko su atributi identi~no definisani, naredba INSERT je oblika:

```
INSERT INTO tabela1
SELECT * FROM tabela2;
```

ina~e:

```
INSERT INTO tabela1 (atribut1, atribut2, . . . )
SELECT atribut, izraz
FROM tabela2
WHERE kriterijum selekcije
```

- *Dati svim analiti~arima i savetnicima premiju u iznosu od 10% njihove plate. Te informacije uneti u tabelu premija zajedno sa datumom zaposlenja.*

```
INSERT INTO PREMIJA (SRADNIK, PREMIJA, POSAO, DATZAP)
SELECT SRADNIK, PLATA * .10, POSAO, DATZAP
FROM RADNIK
WHERE POSAO IN ('ANALITICAR', 'SAVETNIK');
```

- *Prikazati izgled tabele premija.*

```
SELECT * FROM PREMIJA;
```

SRADNIK	POSAO	PLATA	PREMIJA	DATZAP
3069	ANALITICAR		8000	1980-12-17
3488	SAVETNIK		30000	1981-11-09
3576	ANALITICAR		11000	1981-09-23
3600	ANALITICAR		9500	1981-12-03
3602	SAVETNIK		30000	1981-12-03
3634	ANALITICAR		13000	1982-01-23

Izmena sadr`aja - UPDATE

- Op{ti oblik naredbe je:

```
UPDATE tabela
SET atribut1=izraz1[,atribut2=izraz2]
[WHERE kriterijum selekcije n-torki],
```

odnosno:

```
UPDATE tabela
SET(atribut1, atribut2, ...)=(podupit)
[WHERE kriterijum selekcije n-torki].
```

- Podupit mora vratiti najvi{e po jednu vrednost za svaki od atributa u listi atributa iza SET klauzule.

- *Novoprimljeni pravnik Bojan ima}e predsednika za neposrednog rukovodioca. Realizovati tu odluku.*

```
UPDATE RADNIK
SET SRUKOV = 3539
WHERE IME = 'Bojan';
```

- *Srdjan po~inje sa radom 19.09.84 sa platom od 350000 i ima neposrednog rukovodioca sa {ifrom 3266. Realizovati te informacije.*

```
UPDATE RADNIK
SET SRUKOV = 3266,
    DATZAP = '1984-09-19',
    PLATA = 350000
WHERE IME = 'Srdjan';
```

- *Dati svim analiti~arima i savetnicima u odeljenju 20 15% povi{icu na platu .*

```
UPDATE RADNIK
SET PLATA = PLATA * 1.15
WHERE POSAO IN ('ANALITICAR', 'SAVETNIKK')
  AND ODELJENJE# = 20;
```

Brisanje n-torki tabele - DELETE

- Op{ti oblik naredbe je:

```
DELETE [FROM] tabela
[WHERE kriterijum selekcije n-torki].
```

- *Izbaciti podatke o Bojanu, jer je on napustio radnu organizaciju.*

```
DELETE FROM RADNIK
WHERE IME = 'Bojan';
```

- *Izbaciti podatke o svim radnicima odeljenja priprema .*

```
DELETE RADNIK
WHERE ODELJENJE# IN (SELECT ODELJENJE#
                     FROM ODELJENJE
                     WHERE NAZIV = 'PRIPREMA');
```

- Mogu}e je kontrolisati vremenski trenutak prenosa dejstava transakcije na bazu podataka pomo}u SQL naredbi COMMIT i ROLLBACK.
- Pod transakcijom se podrazumevaju sve operacije (INSERT, UPDATE, DELETE) od poslednjeg prenosa dejstava na bazu, tj. od poslednjeg izdavanja naredbe COMMIT.
- Naredbom COMMIT se, dakle, prenose dejstva transakcije na bazu podataka.
- Naredbom ROLLBACK poni{tavaju se sva dejstva od poslednjeg izdavanja naredbe COMMIT.

Pogled - VIEW

- [ta je pogled ?
  - Pogled je "prozor" kroz koji se vide podaci baze podataka,
  - Pogled je virtuelna tabela (sa njim se radi gotovo kao sa baznom tabelom, mada nema svoje podatke i ne zauzima nikakav memorijski prostor).
- Preciznije re~eno, pogled se koristi kao bilo koja druga tabela pri izve{tavanju.
- A`uriranje baze podataka preko pogleda ima brojna ograni~enja.
  - Ne mo`e se vr{iti a`uriranje preko pogleda ukoliko je pogled definisan nad vi{e tabela. Takvo a`uriranje bi zna~ilo da se jednom naredbom vr{i a`uriranje vi{e tabela baze podataka, {to je suprotno definiciji INSERT, DELETE i UPDATE naredbi.
  - Zatim se ne mo`e vr{iti a`uriranje preko pogleda ukoliko se u definiciji pogleda iza SELECT klauzule nalaze funkcije i aritmeti~ki izrazi.
  - Isto tako, a`uriranje se ne mo`e vr{iti ukoliko u definiciju pogleda nisu uklju~ene sve NOT NULL kolone tabele nad kojom je pogled definisan.

- Dakle, da bi mogli vr{iti a`uriranje preko pogleda:
  - pogled mora biti definisan nad jednom tabelom
  - u definiciju pogleda moraju biti uklju~ene sve NOT NULL kolone te tabele i
  - kolone pogleda moraju sadr`ati samo prost, neizmenjen sadr`aj kolona tabele nad kojom je pogled definisan.
- Za{to koristiti pogled ?
  - Jednostavnost kori{}enja - upro{}ava upite,
  - Tajnost - mo}an mehanizam kontrole pristupa podacima,
  - Performanse - ~uva se u kompajliranom obliku,
  - Nezavisnost podataka - menjaju se definicije pogleda, a ne aplikacioni programi koji koriste podatke baze podataka preko pogleda.

- Op{ti oblik naredbe za kreiranje pogleda je:

```
CREATE VIEW  naziv-pogleda [(nazivi atributa pogleda)] AS
    SELECT . . .
    FROM    . . .}  bilo koja ispravna select naredba
    [WITH [LOCAL|CASCADED] CHECK OPTION]
```

- *Kreirati pogled koji se sastoji od imena, {ifara i poslova zaposlenih u odeljenju 30.*

```
CREATE VIEW ODELJENJE30 AS
SELECT SRADNIK, IME, POSAO
FROM RADNIK
WHERE ODELJENJE# = 30;
```

- *Kreirati pogled koji }e davati informacije o platama i premijama radnika bez navodjenja imena radnika.*

```
CREATE VIEW ZARADA AS
SELECT SRADNIK, ODELJENJE#, PLATA, PREMIJA
FROM RADNIK;
```

- *Pogledajmo {ta se mo`e videti kroz taj pogled.*

```
SELECT *
FROM ZARADA;
```

SRADNIK	ODELJENJE#	PLATA	PREMIJA
3069	20	80000	
3199	30	160000	30000
3221	30	125000	50000
3266	20	297500	
3354	30	125000	140000
3398	30	285000	
3482	10	245000	
3488	20	300000	
3539	10	500000	
3544	30	150000	0
3576	20	110000	
3600	30	95000	
3602	20	300000	
3634	10	130000	
3545	10	450000	
3562	20		

- Pored originalnih vrednosti kolona baznih tabela pogled mo`e sadr`ati i izvedene vrednosti svojih virtuelnih kolona.
- *Kreirati pogled sa atributima odeljenje, posao, ukupna i srednja primanja radnika odredjenog zanimanja u odredjenom odeljenju.*

```
CREATE VIEW FINANSIJE
  (ODELJENJE#, POSAO, UKUPNAPLATA, SREDNJAPLATA) AS
SELECT ODELJENJE#, POSAO, SUM (PLATA), AVG (PLATA)
FROM RADNIK
GROUP BY ODELJENJE#, POSAO;
```

- *[ta se vidi kroz ovaj pogled ?*

```
SELECT * FROM FINANSIJE;
ODELJENJE# POSAO        UKUPNAPLATA SREDNJAPLATA
---------- ------------ ----------- ------------
        10 ANALITICAR        130000       130000
        10 PRAVNIK           450000       450000
        10 PREDSEDNIK        500000       500000
        10 RUKOVODILAC       245000       245000
        20 ANALITICAR        190000        95000
        20 PLANER
        20 RUKOVODILAC       297500       297500
        20 SAVETNIK          600000       300000
        30 ANALITICAR         95000        95000
        30 RUKOVODILAC       285000       285000
        30 TRG_PUTNIK        560000       140000
```

- Izbacivanje definicije pogleda (DROP VIEW)

```
DROP VIEW FINANSIJE;
```

- Kako pogled nema svoje podatke, ovom naredbom se izbacuje definicija pogleda iz re~nika podataka.

Ograni~enja

- SQL standard podr`ava slede}e vrste ograni~enja:
  - Ograni~enja domena
  - Ograni~enja tabela i kolona
  - Op{ta ograni~enja
- **Ograni~enje domena** je CHECK ograni~enje. Primenjuje se na sve kolone definisane nad posmatranim domenom.
- **Ograni~enje tabele** definisano je za jednu baznu tabelu. Ograni~enje tabele je:
  - UNIQUE ograni~enje,
  - PRIMARY KEY ograni~enje,
  - referencijalno (FOREIGN KEY) ograni~enje ili
  - CHECK ograni~enje.

- UNIQUE ograni~enje obezbedjuje jedinstvenost vrednosti jedne ili vi{e kolona bazne tabele. UNIQUE ograni~enje je zadovoljeno ako i samo ako ne postoje dva reda bazne tabele sa istim definisanim (NOT NULL) vrednostima u kolonama nad kojima je ograni~enje specificirano.
- PRIMARY KEY ograni~enje defini{e primarni klju~ tabele. Ono je zadovoljeno ako i samo ako ne postoje dva reda bazne tabele sa istim vrednostima u kolonama nad kojima je ograni~enje specificirano i ne postoji ni jedna nedefinisana vrednost ni u jednoj od navedenih kolona.
- Referencijalno ograni~enje realizuje referencijalni integritet na taj na~in {to specificira jednu ili vi{e kolona bazne tabele kao **referenciraju}e kolone** i njima odgovaraju}e **referencirane kolone** u nekoj (ne neophodno razli~itoj) baznoj tabeli.
  - Tabela sa referenciraju}im kolonama naziva se **referenciraju}a tabela**, a tabela sa referenciranim kolonama - **referencirana tabela**.
  - Nad referenciranim kolonama referencirane tabele definisano je PRIMARY KEY ili UNIQUE ograni~enje.

- Referencijalno ograni~enje je zadovoljeno ako su, za svaki red referenciraju}e tabele, vrednosti referenciraju}ih kolona jednake vrednostima odgovaraju}ih referenciranih kolona nekog reda referencirane tabele.
- Ako neka od referenciraju}ih kolona sadr`i null vrednost, zadovoljenje referencijalnog integriteta zavisi od tretmana null vrednosti (***match*** opcija).
- Kao deo specifikacije referencijalnog ograni~enja mogu se navesti i akcije za zadovoljavanje ograni~enja, kojima se defini{u promene koje treba sprovesti nad referenciraju}om tabelom, a bez kojih bi promene nad referenciranom tabelom dovele do naru{avanja referencijalnog ograni~enja.

– CHECK ograni~enje defini{e uslov. Ograni~enje je naru{eno ako je za bilo koji red tabele rezultat evaluacije uslova FALSE (la`no), a zadovoljeno ako je rezultat TRUE (istinito) ili UNKNOWN (nepoznato).

- **Op{te ograni~enje** (assertion) je CHECK ograni~enje, kojim se defini{e uslov nad podacima vi{e tabela baze podataka. Ograni~enje je naru{eno ukoliko je rezultat evaluacije uslova FALSE (la`no), a zadovoljeno ako je rezultat TRUE (istinito) ili UNKNOWN (nepoznato).
- SQL:1999 standard, kao i SQL-92, defini{e ograni~enje kolone kao sintaksnu skra}enicu ograni~enja tabele . Medjutim, pored UNIQUE, PRIMARY KEY, FOREIGN KEY i CHECK ograni~enja, ograni~enje kolone mo`e biti i NOT NULL.

UNIQUE ograni~enje

- Pretpostavimo da tabela RADNIK ima i kolonu MLB (mati~ni li~ni broj). UNIQUE ograni~enje obezbedi}e jedinstvenost vrednosti navedene kolone. Ako bi se UNIQUE ograni~enje specificiralo kao ograni~enje kolone, onda bi kolona MLB bila definisana na slede}i na~in:

```
CREATE TABLE RADNIK
  (...
   MLB CHAR (13) UNIQUE,
   ...
   ...);
```

- UNIQUE ograni~enje ne podrazumeva not null karakteristiku, odnosno dozvoljava da kolone nad kojima je definisano imaju null vrednosti.
- U SQL:1999 terminologiji UNIQUE ima zna~enje "nije jednako".

- Ako ho}emo da kolona MLB ima definisanu vrednost u svakom redu tabele RADNIK, neophodno je da se NOT NULL ograni~enje eksplicitno navede:

```
CREATE TABLE RADNIK
  (...
   MLB CHAR (13) NOT NULL UNIQUE,
   ...
 ...);
```

- Ako bi se UNIQUE ograni~enje specificiralo kao ograni~enje tabele, bilo bi zapisano na slede}i na~in:

```
CREATE TABLE RADNIK
  (...
   MLB CHAR (13) NOT NULL,
   ...
   UNIQUE (MLB),
 ... );
```

- UNIQUE ograni~enje nad vi{e kolona, kojim se obezbedjuje da ne postoje dva reda sa identi~nom kombinacijom vrednosti navedenih kolona, specificira se isklju~ivo kao ograni~enje tabele:

```
UNIQUE (<naziv kolone1>,
        < naziv kolone2>,
        ...,
        < naziv kolonen>).
```

- Ako se zahteva da svi redovi tabele budu razli~iti, odnosno da budu razli~ite kombinacije vrednosti svih kolona tabele, UNIQUE ograni~enje ima slede}i skra}eni oblik:

```
UNIQUE (VALUE).
```

PRIMARY KEY ograni~enje

- Primarni klju~evi tabela RADNIK i ODELJENJE mogu se specificirati kao ograni~enja kolona SRADNIK i ODELJENJE#:

```
CREATE TABLE RADNIK
  (SRADNIK INTEGER PRIMARY KEY,
   ...
   ...);

CREATE TABLE ODELJENJE
  (ODELJENJE# INTEGER PRIMARY KEY,
   ...
   ...);
```

- SQL-89 standard zahtevao je da se specificira NOT NULL PRIMARY KEY.
- SQL:1999 standard, kao i SQL-92, dozvoljava da se, sa istom semantikom, specificira samo PRIMARY KEY. To zna~i da je u PRIMARY KEY ograni~enje uklju~eno i NOT NULL ograni~enje.

- Ako bi se prethodna PRIMARY KEY ograni~enja specificirala kao ograni~enja tabela, bila bi zapisana na slede}i na~in:

```
CREATE TABLE RADNIK
  (SRADNIK INTEGER NOT NULL,
   ...
   CONSTRAINT RADNIK_PK PRIMARY KEY (SRADNIK),
   ... );

CREATE TABLE ODELJENJE
  (ODELJENJE# INTEGER NOT NULL,
   ...
   CONSTRAINT ODELJENJE_PK PRIMARY KEY (ODELJENJE#),
  ... );
```

- Svako ograni~enje ima naziv
  - korisnik mo`e eksplicitno da imenuje svako ograni~enje
  - ukoliko korisnik to ne uradi, sistem sam dodeljuje naziv svakom ograni~enju.
- Kada je primarni klju~ slo`en, odnosno sastavljen od vi{e kolona, specificira se isklju~ivo kao ograni~enje tabele:

```
[CONSTRAINT <naziv ogranicenja>]
PRIMARY KEY (<naziv kolone1>, ..., < naziv kolonen>).
```

Referencijalno (FOREIGN KEY) ograni~enje

- Referenciraju}a tabela u na{em primeru je tabela RADNIK sa referenciraju}om kolonom RADNIK.ODELJENJE#, a referencirana tabela je tabela ODELJENJE sa referenciranom kolonom ODELJENJE.ODELJENJE#.
- Referenciraju}a kolona RADNIK.ODELJENJE# je spoljni klju~ tabele RADNIK, a referencirana kolona ODELJENJE.ODELJENJE# je primarni klju~ tabele ODELJENJE.
- Referencijalno ograni~enje specificira se kao deo definicije tabele RADNIK. Ono se zadaje ili kao ograni~enje kolone, kroz definiciju njene kolone ODELJENJE#, ili kao ograni~enje tabele.
- Osnovna sintaksa specifikacije referencijalnog ograni~enja kao ograni~enja kolone mo`e se ilustrovati slede}im primerom:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER NOT NULL
       REFERENCES ODELJENJE (ODELJENJE#),
   ... );
```

- U prikazanom primeru ograni~enja nisu eksplicitno imenovana. Eksplicitnim imenovanjem referencijalnog i NOT NULL ograni~enja imali bi slede}i izgled naredbe:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER
     CONSTRAINT ODELJENJE_NOT_NULL NOT NULL
     CONSTRAINT ODELJENJE_FK
       REFERENCES ODELJENJE (ODELJENJE#),
   ... );
```

- Navedeno referencijalno ograni~enje mo`emo da zapi{emo i kao ograni~enje tabele:

```
CREATE TABLE RADNIK
  (...
   ...
   ...
   CONSTRAINT ODELJENJE_FK FOREIGN KEY (ODELJENJE#)
       REFERENCES ODELJENJE (ODELJENJE#),
  ... );
```

- Kada je referencirana kolona primarni klju~ referencirane tabele, ne mora se eksplicitno navesti u specifikaciji referencijalnog ograni~enja:

```
CREATE TABLE RADNIK
  (...
   ...
   ...
   CONSTRAINT ODELJENJE_FK FOREIGN KEY (ODELJENJE#)
       REFERENCES ODELJENJE,
   ... );
```

- Kada je spoljni klju~ slo`en, odnosno sastavljen od vi{e kolona, specificira se isklju~ivo kao ograni~enje tabele

- Svi do sada navedeni primeri spre~avali su izvr{avanje svake SQL naredbe koja bi naru{ila definisano referencijalno ograni~enje.
- To je bio i jedini na~in zadovoljavanja referencijalnog integriteta u SQL-89 standardu.
- SQL:1999 standard podr`ava pet referencijalnih akcija kao odgovor na operacije nad referenciranom tabelom koje potencijalno naru{avaju referencijalni integritet.
- Operacije koje potencijalno naru{avaju referencijalni integritet su:
  - brisanje (DELETE) ili
  - izmena (UPDATE) redova referencirane tabele.
- Referencijalne akcije su
  - NO ACTION,
  - RESTRICT,
  - CASCADE,
  - SET NULL i
  - SET DEFAULT.
- Pri specifikaciji referencijalnog ograni~enja mogu}e je izabrati jednu referencijalnu akciju za operaciju DELETE, a drugu za operaciju UPDATE.

- NO ACTION referencijalna akcija poni{tava efekte SQL naredbe ako je **na kraju** njenog izvr{avanja naru{eno referencijalno ograni~enje.
- SUBP prikazuje poruku o gre{ci.
- U slede}em primeru specificirano je referencijalno ograni~enje sa NO ACTION referencijalnom akcijom i za brisanje i za izmenu redova referencirane tabele:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER NOT NULL
       REFERENCES ODELJENJE
         ON DELETE NO ACTION
         ON UPDATE NO ACTION,
   ... );
```

- NO ACTION je default referencijalna akcija, {to zna~i da se primenjuje ukoliko nijedna akcija nije eksplicitno navedena.

- RESTRICT i NO ACTION su veoma sli~ne referencijalne akcije. Razlika je u tome {to NO ACTION dozvoljava da referencijalno ograni~enje bude naru{eno **u toku** izvr{avanja SQL naredbe, a RESTRICT ne.
- CASCADE referencijalna akcija zadovoljava referencijalno ograni~enje tako {to nastavlja zapo~etu operaciju (DELETE ili UPDATE) nad odgovaraju}im redovima referenciraju}e tabele.
- U slede}em primeru specificirano je referencijalno ograni~enje koje obezbedjuje da brisanje reda tabele ODELJENJE ima za posledicu brisanje svih redova tabele RADNIK koji su referencirali taj red tabele ODELJENJE:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER NOT NULL
       REFERENCES ODELJENJE
         ON DELETE CASCADE,
   ... );
```

- SET NULL referencijalna akcija zadovoljava referencijalno ograni~enje tako {to postavlja vrednosti referenciraju}ih kolona na NULL, a SET DEFAULT na default vrednosti. Naravno, SET NULL akcija je mogu}a samo ukoliko referenciraju}e kolone nisu definisane kao NOT NULL.

CHECK ograni~enje

- CHECK ograni~enje je najfleksibilnije i verovatno najkorisnije ograni~enje.
- Njime se defini{e uslov, koji mo`e ograni~avati dozvoljene vrednosti domena ili kolone, odrediti medjuzavisnosti vrednosti kolona i redova jedne tabele ili definisati op{te ograni~enje, koje se prostire na vi{e tabela baze podataka.
- Slede}e CHECK ograni~enje je specificirano kao ograni~enje kolone IME i obezbedjuje da sva imena budu zapisana velikim slovima u bazi podataka:

```
CREATE TABLE RADNIK
  (...
   IME VARCHAR (20) NOT NULL CHECK(IME = UPPER (IME)),
   ...
   ...);
```

- Slede}e ograni~enje specificirano je kao ograni~enje tabele RADNIK. Ono ne dozvoljava da ukupan broj radnika bude ve}i od 100:

```
CREATE TABLE RADNIK
  (...
   ...
   CONSTRAINT BROJ_RADNIKA
     CHECK((SELECT COUNT(*)
            FROM RADNIK) <= 100),
   ...);
```

- Op{te ograni~enje specificira se slede}om sintaksom:

```
CREATE ASSERTION <naziv ogranicenja>
CHECK ( <uslov> ).
```

Provera ograni~enja

- Ograni~enja se, po default-u, proveravaju na kraju izvr{avanja svake SQL naredbe.
- SQL:1999 standard dozvoljava da se provera ograni~enja odlo`i i izvr{i na kraju transakcije.
- Za svako ograni~enje je mogu}e specificirati da li je ili ne dozvoljeno odlaganje provere do kraja transakcije, {to se zapisuje navodjenjem opcije [NOT] DEFERRABLE.
- Ako je izabrana opcija DEFERRABLE, tada je neophodno navesti da li je na po~etku transakcije provera ograni~enja odlo`ena (INITIALLY DEFERRED) ili ne (INITIALLY IMMEDIATE).
- Provera ograni~enja, koje je specificirano kao INITIALLY IMMEDIATE DEFERRABLE, mo`e se odlo`iti do kraja transakcije kori{}enjem naredbe SET CONSTRAINTS.

OBJEKTNE BAZE

OBJEKTNE BAZE

- Poslednja decenija u softverskom inženjerstvu je **decenija "objektne" orijentacije**. Objektna orijentacija je pristup u kome se neki sistem organizuje kao kolekcija međusobno povezanih objekata koji, sarađujući, ostvaruju postavljene ciljeve. Objektna orijentacija je danas preovlađujuća u **programiranju i programskim jezicima, metodološkim pristupima razvoju sofvera i informacionih sistema**, a postepeno preuzima primat i **u sistemima za upravljanje bazom podataka, bilo preko "čistih" objektnih SUBP, bilo preko objektno relacionih SUBP** koji predstavljaju proširenje relacionih sa objektnim konceptima.

OBJEKTNE BAZE- TIPOVI PODATAKA

- Jedna od najbitnijih karakteristika SUBP-a je skup tipova podataka koje on podržava.
- Konvencionalni SUBP (hijerarhijski, mrežni i relacioni) su zasnovani na tipu rekorda koji predstavlja agregaciju polja (atributa u relacionom modelu) definisanih nad standardnim tipovima. U relacionom modelu se još definiše i tabela kao skup rekorda, u mrežnom se uvodi pojam "rekord seta", a u hijerarhijskom pojam "stabla" rekorda.
- Objektni SUBP su postavili sebi cilj da stvore mnogo bogatiji skup tipova i time omoguće prirodniju manipulaciju podacima u aplikacijama koje se razvijaju nad bazom.

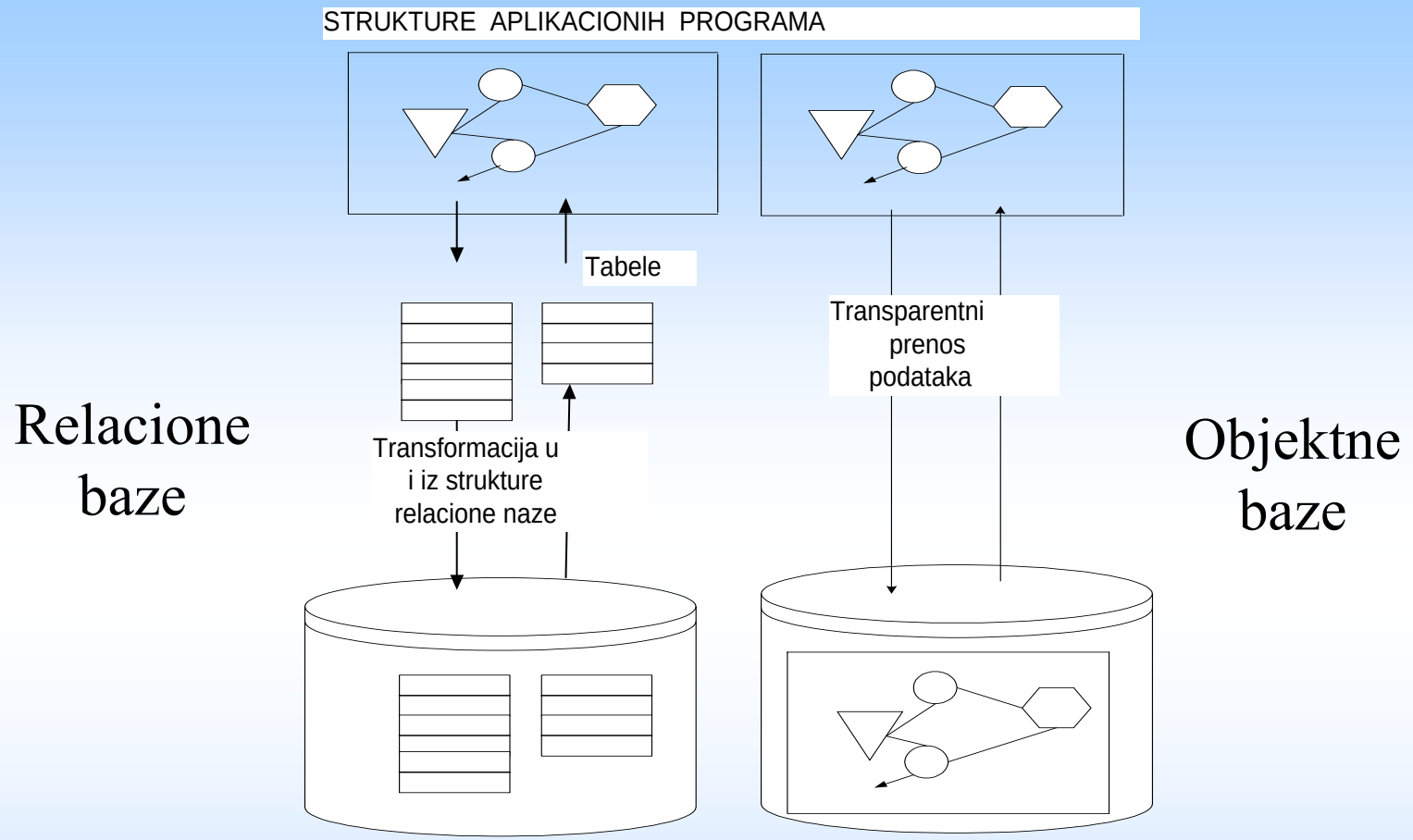
OBJEKTNE BAZE

- U konvencionalnim SUBP, tronivoska ANSI/SPARC arhitektura imala je za cilj da omogući da se baza podataka tretira kao samostalna komponenta nekog IS. Zbog toga je Jezik baze podataka (DDL i DML) u ovim SUBP potpuno odvojen (nezavisan) od programskih jezika u kojima se razvijaju aplikacije, omogućujući na taj način da se nad jednom bazom podataka razvijaju i izvršavaju aplikacije napisane i u različitim programskim jezicima.

  Nezavisnost Jezika baze podataka od programskih jezika, bez obzira na probleme koji su proisticali iz njihove moguće nesaglasnosti, tretirala se kao prednost, a ne kao nedostatak ovih SUBP-a.

OBJEKTNE BAZE

- Objektni SUBP teže da integrišu funkcije SUBP-a u programski jezik da bi se nesaglasnost Jezika baze podataka i programskog jezika eliminisale i značajno poboljšale performanse sistema.
- Teži se tome da se izjednače u potpunosti objekti baze podataka i objekti aplikacija napisanih u nekom objektnom programskom jeziku. Objekti u aplikacijama su ***tranzijentni***, životni vek im je jednak trajanju aplikacije, dok su objekti u bazi podataka ***perzistentni***, nezavisni od postojanja aplikacija koje ih koriste.
- Ne eliminiše se mogućnost da više programa, napisanih čak i u različitim objektnim jezicima, koriste istu bazu podataka.
- Raspolažući sa mnogo bogatijim skupom tipova, objektni SUBP treba da podrže i znatno moćniji upitni jezik od relacionog (SQL-a).



U nekom objektnom jeziku jednostavno se mogu realizovati složene strukture podataka i takve usladištiti u OBP. U slučaju relacionih baza podataka, ta struktura podataka se prvo transformiše u odgovarajući skup tabela relacione baze, što je ponekad složen i vremenski zahtevan postupak.

OBJEKTNE BAZE

- Rani komercijalni objektni SUBP nisu bili zasnovani na jedinstvenom standardu, navedene ciljeve su ostvarivali, sa manje ili više uspeha, na različite načine. Njihov komercijalni prodor je bio zanemarljiv.
- Uspeh relacionih SUBP, pored već diskutovanih prednosti, bio je i u tome što su zasnovani na jedinstvenom standardu, SQL-u, što je značajno podiglo njihovu prihvatljivost, prenosivost i mogućnost kooperacije različitih sistema.
- Zbog toga se u poslednjih nekoliko godina intenzivno razvija standard za objektne baze podataka u okviru takozvane ODMG (Object Database Management Group). Verzija 1.2. ODMG standarda pojavila se 1996, a značajno izmenjena i dopunjena verzija 2.0, 1997. godine. Ovde se izlažu elementi ODMG 2.0 standarda.

ARHITEKTURA OBJEKTNIH SUBP – komponente

- Objektni model.
- Objektni specifikacioni jezici
- Objektni upitni jezik
- C++, Smalltalk, Java Language Binding

ARHITEKTURA OBJEKTNIH SUBP – komponente

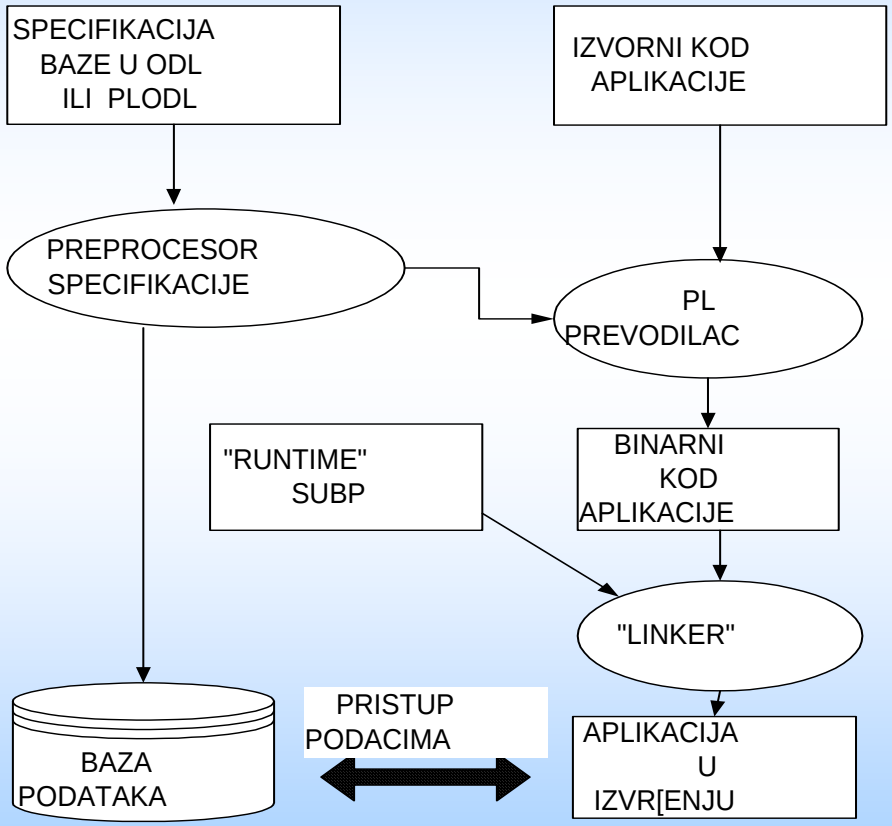
- ***Objektni model.*** Objektni model na kome treba da bude zasnovan svaki objektni SUBP (ODMG model) izveden je iz OMG (Object Management group) objektnog modela. OMG je objektni model definisan kao zajednička osnova za objektne programske jezike, komunikaciju objekata u nekoj klijent server arhitekturi (Object Request Brockers, ORB, na primer CORBA) i objektne baze podataka.

- ***Objektni specifikacioni jezici*** koji služe da opišu sistem kao skup međusobno povezanih objekata. Objektni specifikacioni jezik - ODL (Object Definition Language) ima istu ulogu koju JOP (DDL) ima u konvencionalnim jezicima.

ARHITEKTURA OBJEKTNIH SUBP – komponente

- ***Objektni upitni jezik***, neproceduralni jezik za postavljanje upita i modifikaciju baze podataka. Ovaj upitni jezik nazvan OQL (Object Query Language) je namerno veoma sličan SQL-u, sa većim mogućnostima koje pruža objektni model.
- ***C++, Smalltalk, Java Language Binding***, su posebne nezavisne komponente arhitekture koje pokazuju kako se u C++ (Smallltalk, Java) ugrađuju mogućnosti manipulacije perzistentnim objektima, mehanizmi za povezivanje sa OQL-om, upravljanje transakcijama i slično. Kao što je već rečeno, za razliku od konvencionalnih JMP jezika, JMP u objektnim bazama se "kroje" prema "jeziku domaćinu".

ARHITEKTURA OBJEKTNIH SUBP – korišćenje SUBP



OBJEKTNI MODEL

- Bilo koji sistem se može posmatrati kao ***skup međusobno povezanih objekata.*** Pod objektima u nekom sistemu se podrazumevaju fizički objekti, koncepti, apstrakcije, bilo šta što ima jasne granice i jasno značenje, što se jasno razlikuje od drugih objekata u sistemu.
- U realnom sistemu, objekti i način ostvarivanja njihovih veza mogu da budu veoma raznovrsni.
- U nekoj vrsti ***modela realnog sistema*** svi ti raznovrsni objekti i njihove veze predstavljaju se sa određenim (malim) brojem precizno definisanih koncepata. Modeli koji služe za opis baze podataka, nazivaju se ***modeli podataka***.

OBJEKTNI MODEL: OSNOVNI POJMOVI

- Osnovni primitivni koncepti objektnog modela su ***objekat i literal.***
- Pod objektom se podrazumeva entitet koji je sposoban da čuva svoja ***stanja*** i koji stavlja okolini na raspolaganje skup ***operacija*** preko kojih se ta stanja prikazuju ili menjaju.
- Literal je u osnovi vrednost, podatak koji se koristi u modelu. Brojevi i karakteri su primeri "atomskih" literala, a datum je primer složenog (struktuiranog) literala.
- Analogija sa OO Programskim jezicima: "Objekti neke klase su bilo vrednosti datog tipa (immutable objects - literal) ili promenljive koje mogu uzimati vrednosti datog tipa (muttable object - objekat)".
- Posebno je potrebno naglasiti sledeću razliku objekta i literala: *objekat ima jedinstveni identifikator, a literal nema*.

OBJEKTNI MODEL: OSNOVNI POJMOVI

- Objekti i literali se kategorizuju u ***tipove.*** Svi objekti, odnosno literali istog tipa imaju zajednički skup stanja i jedinstveno ponašanje. Konkretan objekat se ponekad *naziva* ***pojavljivanje (instanca)*** datog tipa***.*** Za pojavljivanja literala se pretpostavlja da implicitno postoje.
- ***Stanje objekta*** se predstavlja vrednostima njegovih ***osobina.*** **O**sobine su ***atributi objekta*** i njegove ***veze*** sa drugim objektima u sistemu.
- ***Ponašanje objekta*** se opisuje preko skupa **operacija** koje on izvršava ili se nad njim izvršavaju. Svaka operacija ima kao implicitni argument objekat kome je pridružena. Operacija može da ima i listu ulaznih parametara definisanih tipova, a može i da vrati tipizovan rezultat.

OBJEKTNI MODEL: OSNOVNI POJMOVI

- ***Baza podataka*** skladišti objekte i stavlja ih na korišćenje većem broju korisnika, odnosno aplikacija. Baza podataka se opisuje preko svoje ***šeme*** koja se specifikuje preko ODL-a. U šemi se definišu tipovi objekata čija se pojavljivanja čuvaju u bazi.

SPECIFIKACIJA I IMPLEMENTACIJA TIPA

- Svaki tip ima jednu **specifikaciju** i jednu ili više **implementacija.**
- Specifikacija definiše eksterne karakteristike, vidljive korisniku tipa: **operacije** koje mogu biti pozvane, osobine kojima se može pristupiti i ***izuzeci*** (exceptions) koje operacije pobuđuju.
- Implementacija tipa definiše interne karakteristike objekata datog tipa. Moguće je definisati više implementacija jedne specifikacije u jednom ili u više različitih jezika.
- Odvajane specifikacije od implementacije tipa je veoma bitno, jer se na taj način ostvaruje ***učaurenje*** (encapsulation) objekta, korišćenje servisa koji objekat pruža okolini, bez poznavanja implementacije.

SPECIFIKACIJA I IMPLEMENTACIJA TIPA

- Specifikacija tipa se daje preko:
  - ***interfejsa*** koji predstavlja samo apstraktno ponašanje
  - ***klase*** koja predstavlja i apstraktno stanje i apstraktno ponašanje tipa objekta.
  - definicija literala daje samo apstraktno stanje tipa literala.
    - **interface** Radnik {...};
    - **class** Lice {...};
    - **struct** Kompleks {**float** reldeo, **float** imdeo};
- Klasa je tip koji se može direktno instancirati, pojavljivanja ovoga tipa mogu se kreirati u nekom programu ili bazi. Interfejs je tip koji se ne može direktno instancirati.

SPECIFIKACIJA I IMPLEMENTACIJA TIPA

- Implementacija tipa, u bilo kom programskom jeziku, ostvaruje se preko ***strukture podataka*** za opis stanja i skupa **metoda**, preko kojih su implementirane operacije tipa. U većini objektnih jezika postoji ***koncept klase***, pa se može reći da je u njima klasa osnovni mehanizam za implementaciju tipova objekata. Koncept klase u objektnim jezicima ne treba izjednačavati sa pojmom klase u ODMG.
- Specifikacija tipova neke baze podataka se definiše preko ODL-a, a njihova implementacija u okviru povezivanja sa jezicima (language bindings). Oni definišu način transformacije klase ODMG u sopstveni implementacioni koncept klase.

UGRAĐENI TIPOVI

- *Tipovi literala*
  - *Atomski tipovi*

    long, short, unsigned long, unsigned short, float, double, boolean, octet, char, string, enum <t>
  - *Kolekcije literala*

    set <t>, bag<t>, list<t>, array<t>, dictionary<t,p>
  - *Struktuirani literali*

    date, time, timestamp, interval, structure<t>

UGRAĐENI TIPOVI

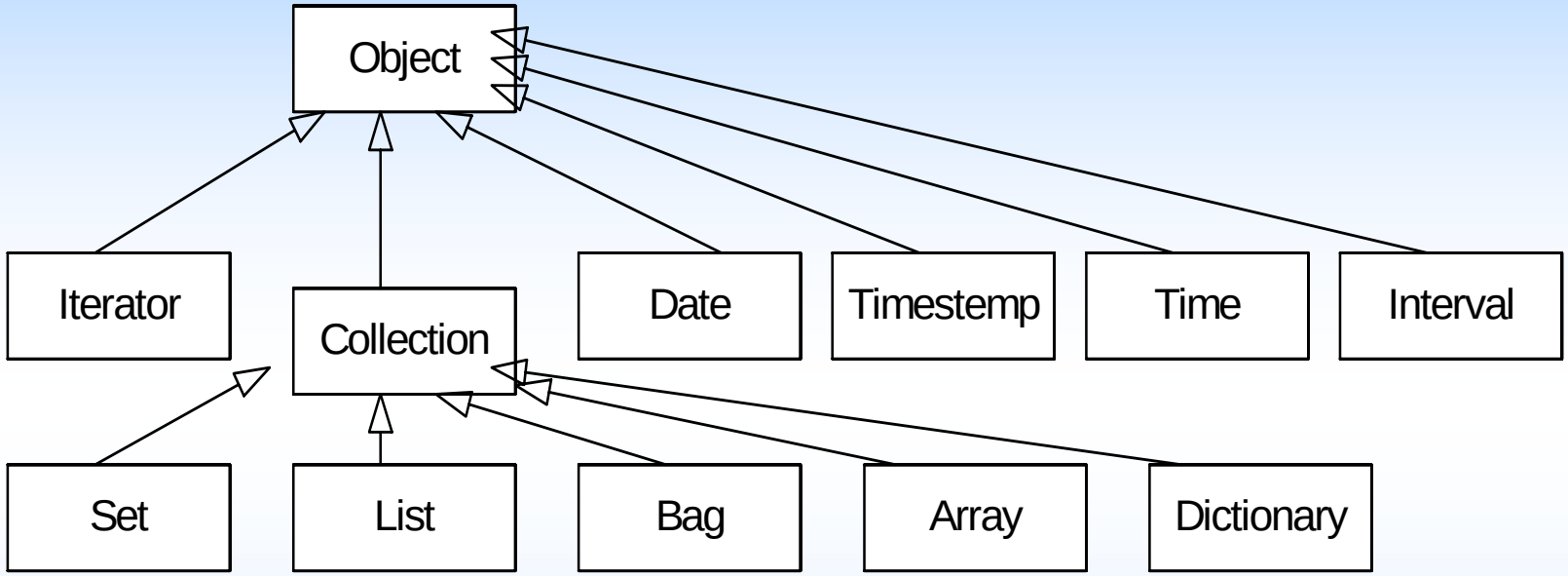
- *Tipovi objekata*
  - *Kolekcije objekata*

    Set<t>, Bag<t>, List<t>,
    Array<t>, Dictionary<t>
  - *Stuktuirani objekti*

    Date, Time, Timestamp, Interval

- Ne postoji ugrađeni tip atomskog objekta. Svi atomski objekti su korisnički definisani.

HIJERARHIJA UGRAĐENIH TIPOVA OBJEKATA



U ODMG svi objekti nasleđuju operacije osnovnog interfejsa **Object.** Objekti se kreiraju pozivajući operacije interfejsa **ObjectFactory,** a kolekcije pozivanjem operacija interfejsa **CollectionFactory**

UGRAĐENI TIPOVI

```
interface ObjectFactory {
        Object new();
    };
interface Object {
        ....
        boolean same_as(in Object  Objekat);
        Object copy();
        void delete();
    };
```

Operacije se primenjuju na objekte korišćenjem **"dot"** notacije.

```
o.same_as(p);
```

UGRAĐENI TIPOVI

Pošto jedan objektni SUBP može da upravlja sa više baza podataka definiše se objekat za "fabrikovanje" baza kao i objekat koji definiše samu bazu.

```
interface  databaseFactor{
   Database   new();};
interface Database {
   exception ElementNotFound {};
   void open(in string  ime_base);
   void close ();
   void bind (in any neki_obj, in string ime_obj);
   Object  unbind(in string ime_objekta);
   Object  lookup (in string ime_objekta)
                        raises(ElementNotFound);
   ....};
```

UGRAĐENI TIPOVI

```
interface CollectionFactory : ObjectFactory {
        Collection    new_of_size(in long velicina);
   };
```

UGRAĐENI TIPOVI

```
interface Collection {
        exception           InvalidColectionType{};
        exception           Element not found{ bilo koji el};
        unsigned long       cardinality();
        boolean             is_empty();
        boolean             is_ordered();
        boolean             allows_duplicates();
        boolean             contains_element(in neki element);
        void                insert_element (in neki element);
        viod                remove_element(in neki element);
                                                raises (ElementNotFound);
        Iterator      create_iterator(in boolean stable);
        BidirectionalIterator   create_bidirectional_iterator(in
                                                boolean stable)

        raises(InvalidCollectionType);;}
```

UGRAĐENI TIPOVI

```
interface Iterator {
    exception NoMoreElements{};
    exception InvalidCollectionType{};
    boolean is_stable();
    bolean at-end();
    void reset();
    any         get_element() raises (NoMoreElements);
    void        next_position (in neki element);
    void        replace_element( in  neki element)
                raises IvalidCollectionType);
        };
```

UGRAĐENI TIPOVI

```
interface Set: Collection{
    Set         create_union(in Set drugi-skup);
    Set         create_intersection(in Set drugi-skup);
    Set         create_difference(in Set drugi-skup);
    boolean     is_subset_of(in drugi-skup);
    boolean     is_proper_subset_of(in drugi-skup);
    boolean     is_superset_of(in drugi-skup);
    boolean     is_proper_superset_of(in drugi-skup);
};
```

MODELIRANJE STANJA

- Stanje pojavljivanja nekog tipa objekta iskazuje se preko vrednosti njegovih osobina. Postoje dve vrste osobina: **atributi i veze.**

- **Atribut**. Atributi nekog tipa objekta se specifikuju preko tipova literala ili tipova objekata. Drugim rečima, vrednost atributa može da bude literal ili identifikator nekog objekta.

PRIMER DEFINISANJA ATRIBUTA KLASE

```
interface Lice {
        attribute short      starost;
        attribute string     ime;
        attribute enum       pol {muški, ženski};
        attribute      Adresa kućna-adresa;
        attribute set <String>     telefon;
        attribute      Organizacija  zaposlen;
    };
```

Pretpostavlja se da je definisan objekat Organizacija i

```
struct Adresa {string grad,
                    string ulica_i_broj};
```

MODELIRANJE STANJA - VEZE

- ODMG podržava samo binarne veze (relationship), tj. veze između dva tipa objekta.
- Veza se definiše implicitno, unutar opisa tipa objekta, kao tzv "prelazna putanja" (traversal path).
- "Prelazna putanja" se uvek deklariše u paru, u okviru deklaracije oba tipa koji su u vezi. Deklariše se, na primer, da radnik **radi** u preduzeću (u okviru tipa Radnik) i da preduzeće **zapošljava** radnika (u okviru tipa Preduzeće).
- Činjenica da ovakve dve deklaracije predstavljaju jednu vezu iskazuje se u ODL-u preko klauzule **inverse.**

PRIMER DEKLARISANJA VEZE

```
interface  Radnik {
    .....
    relationship Preduzeće radi
          inverse Preduzeće:: zapošljava;
    .....
};
interface Preduzeće {
    .....
    relationship set <Radnik> zapošljava
          inverse Radnik:: radi;
    .....
};
```

VEZE

- Kardinalnost veze se definiše preko činjenice da veza "uzima" kao vrednost jedan atomski objekat (kardinalnost "jedan") ili neku njihovu kolekciju (kardinalnost "više").

- Objektni SUBP treba da ostvari integritet veze: ako je objekat koji učestvuje u vezi izbrisan iz baze tada se i svaka "prelazna putanja" prema tom objektu, takođe briše.

- Veza u OMG-u je uvek dvosmerna. Pri implementaciji veze u nekom pridruženom jeziku veza se može učiniti jednosmernom. Isto tako, kao što je ranije pokazano, atribut čija je vrednost identifikator nekog objekta, može da posluži za iskaz jednosmernih veza.

MODELOVANJE PONAŠANJA – OPERACIJE

- Ponašanje tipa predstavlja se skupom operacija
- Sintaksa za specifikaciju operacije:

Specifik_oper:: tip_rezult naziv_oper(lista_argum)
raises(lista_naziv_izuzetka)

lista_argum:: argument | argument, lista_argum

argument:: parametar_argumenta
specifik_ tipa_argumenta naziv_argumenta

parametar_argumenta:: **in** | **out** | **inout**

lista_naziv_izuzetka:: naziv_izuzetka |
naziv_izuzetka, lista_naziv_izuzetka

Primer operacije je:

Boolean upisan_na_ predmet (**in unsigned short** kurs)
raises (nema_uslova, popunjen_broj)

NASLEĐIVANJE

- **U ODMG se definišu dve posebne vrste nasleđivanja: *nasleđivanje ponašanja* i *nasleđivanje stanja.***
- Za nasleđivanje ponašanja se koristi veza nadtip-podtip (koja se ponekad naziva veza generalizacija-specijalizacija ili ISA veza. Nadtip u nasleđivanju ponašanja mora da bude Interfejs.
- Za nasleđivanje stanja koristi se specifična veza EXTENDS (Proširenje). U ovoj vezi između klasa, podređena klasa nasleđuje celokupno stanje i ponašanje klase koju proširuje.

NASLEĐIVANJE PONAŠANJA

Za nasleđivanje ponašanja se koristi veza nadtip-podtip. Činjenica da je Nastavnik podtip Radnika, a Asistent podtip Nastavnika označava se na sledeći način**:**

intrerface Radnik {...specifikacija tipa Radnik..}

interface Nastavnik: Radnik {...specifikacija tipa Nastavnik..}

interface Asistent: Nastavnik {...specifikacija tipa Asistent..}

class StalniRadnik: Radnik {...specifikacija tipa StalniRadnik..}

class PrivremeniRadnik:Radnik {...specifikacija tipa PrivremeniRadnik..}

NASLEĐIVANJE PONAŠANJA

- U podtipu se može i redefinisati ponašanje specifikovano u nadtipu. Na primer, ako je u tipu Radnik specifikovana operacija ObračunZarade, ista operacija se može različito implementirati za podtipove StalniRadnik i PrivremeniRadnik. Osobina da se ista operacija izvršava na različit način u zavisnosti od tipa objekta sa kojim se izvršava, naziva se **polimorfizam**.

- U ODMG modelu podržano je višestruko nasleđivanje ponašanja. Tada se, međutim, može desiti da dve (ili više) nasleđene operacije imaju isti naziv, a različit broj i/ili tip argumenata. Da bi se ovaj problem izbegao, nije dozvoljeno "preopterećenje" (overloading) imena operacija (davanje istih imena operacijama različitih tipova) u takvoj hijerarhiji nasleđivanja.

NASLEĐIVANJE STANJA

- ODMG model uvodi EXTENDS vezu za definisanje nasleđivanja stanja. Na primer,

```
class Lice {
        attribute string ime;
        attribute Date datumRođenja;};
 class ZaposlenoLice extends Lice: Radnik {
        attribute Date datumZapošljavanja;
        attribute Carency  plata;
        relationship Rukovodilac šef inverse Rukovodilac ::
                                podređeni;};
 class Rukovodilac extends ZaposlenoLice {
        relationship set <ZaposlenoLice> podređeni inverse
                                ZaposlenoLice šef;};
```

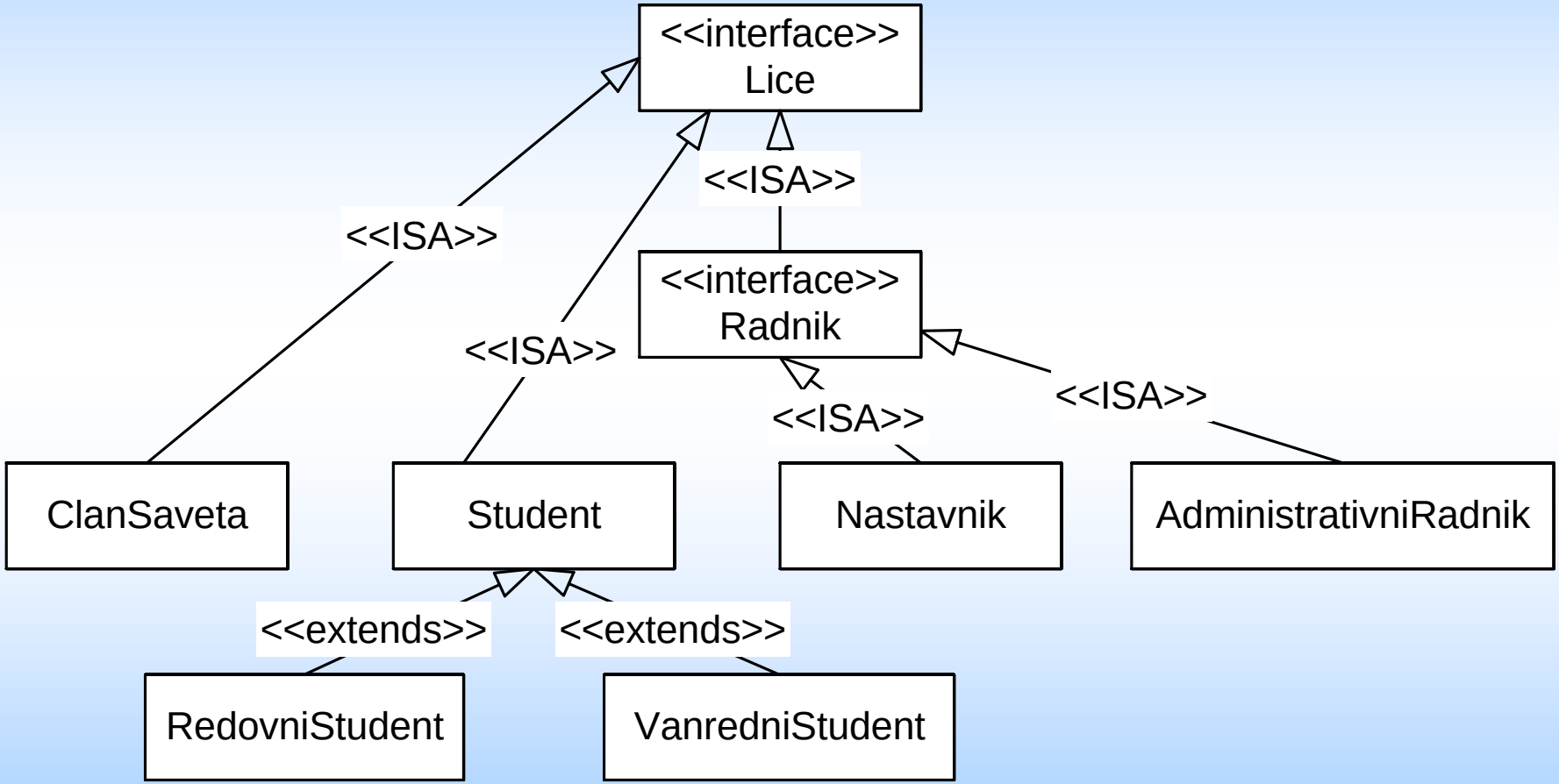
NASLEĐIVANJE

- Odnos tip-podtip (generalizacija-specijalizacija) u ODMG modelu se primenjuje samo na nasleđivanje ponašanja. Zbog toga interfejs i klasa mogu biti podtipovi samo interfejsa. Interfejsi i klase ne mogu biti podtipovi klasa.

- Pošto se interfejs ne može instancirati, on (slično apstraktnoj klasi u nekim OO jezicima) služi samo za to da se iz njega naslede neke operacije.

- Pošto se preko odnosa podtip/nadtip nasleđuje samo ponašanje, ako se želi u podtipu isti atribut koji postoji u interfejsu koji je nadtip, on se u podtipu mora “kopirati”.

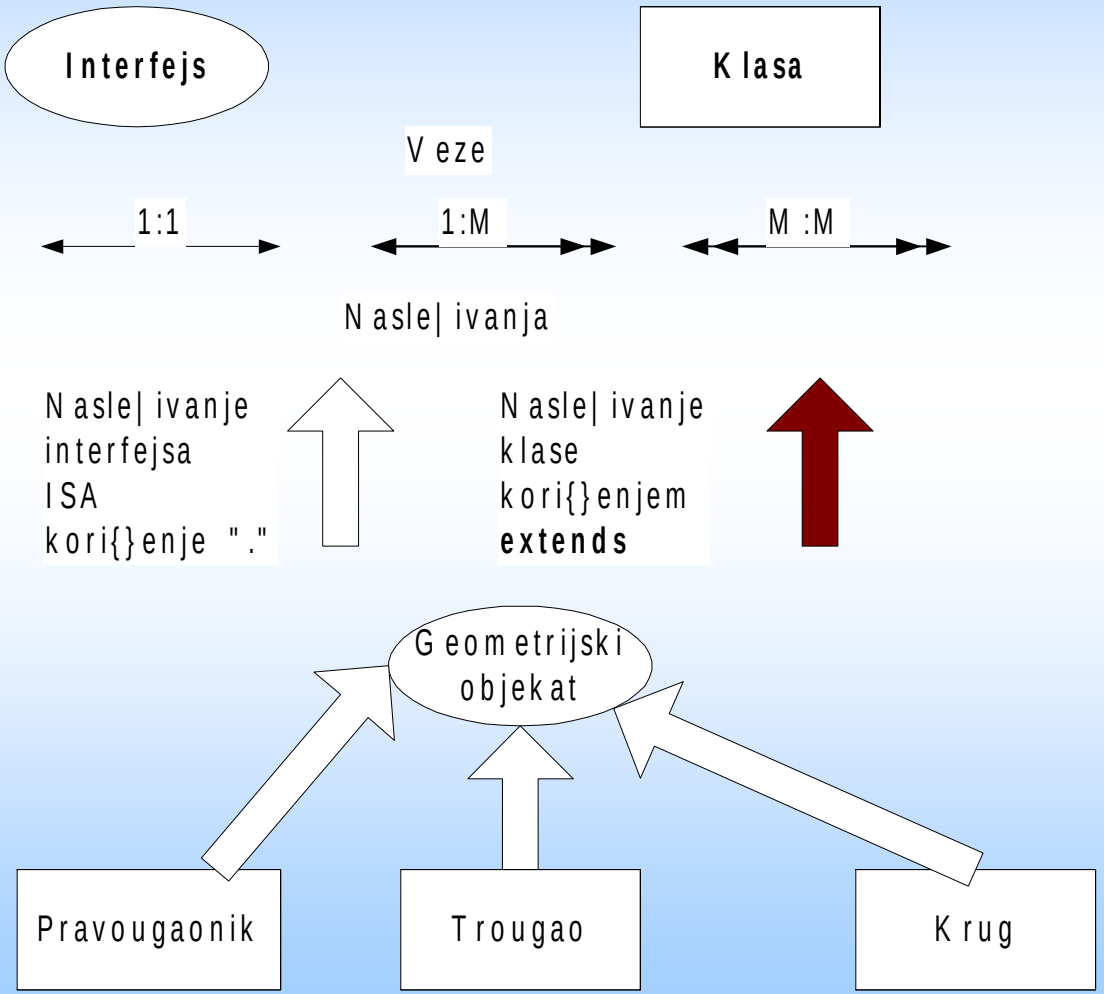
NASLEĐIVANJE - UML notacija



OPSEG (EXTENT) TIPA

- Opseg tipa je skup svih instanci datog tipa u okviru posmatrane baze podataka. Ako je neki objekat instanca tipa A, tada je on elemenat opsega A.
- U konvencionalim bazama podataka čuvaju se sva pojavljivanja tipova definisanih u njima. U objektnim bazama, projektant može da odluči da li će se opseg tipa čuvati u bazi ili ne. Objektni SUBP treba da obezbedi ubacivanje i izbacivanje pojavljivanja u i iz opsega, kao i kreiranje indeksa.
- Moguće je definisati da jedan atribut (za prost) ili više atributa (za složen) predstavljaju korisnički ključ tipa. Ključ služi za jedinstvenu identifikaciju, preko atributa definisanih u njegovoj specifikaciji, jednog pojavljivanja u opsegu tipa.

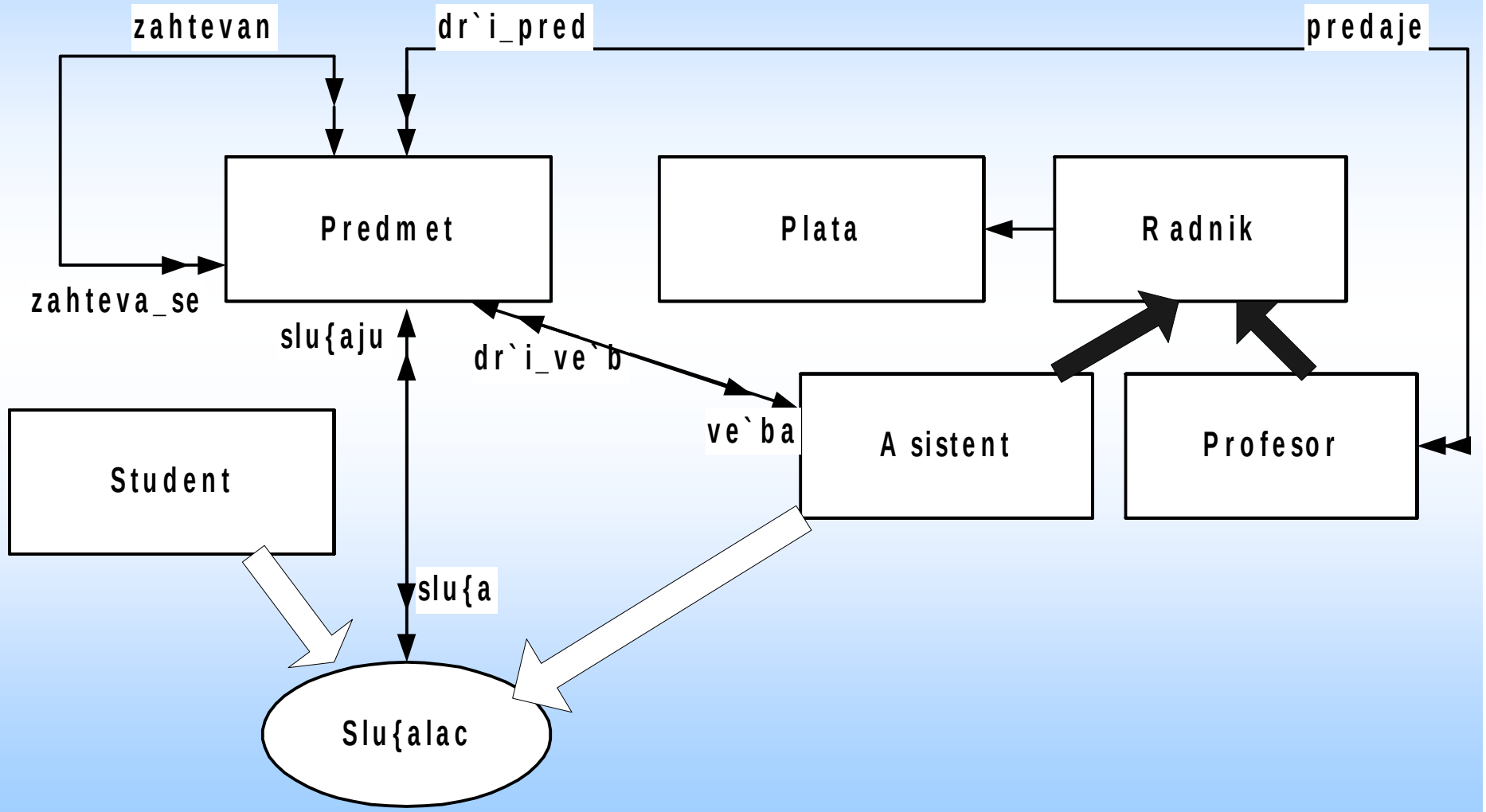
GRAFIČKE OZNAKE ZA PRIKAZ ŠEME BP U ODMG



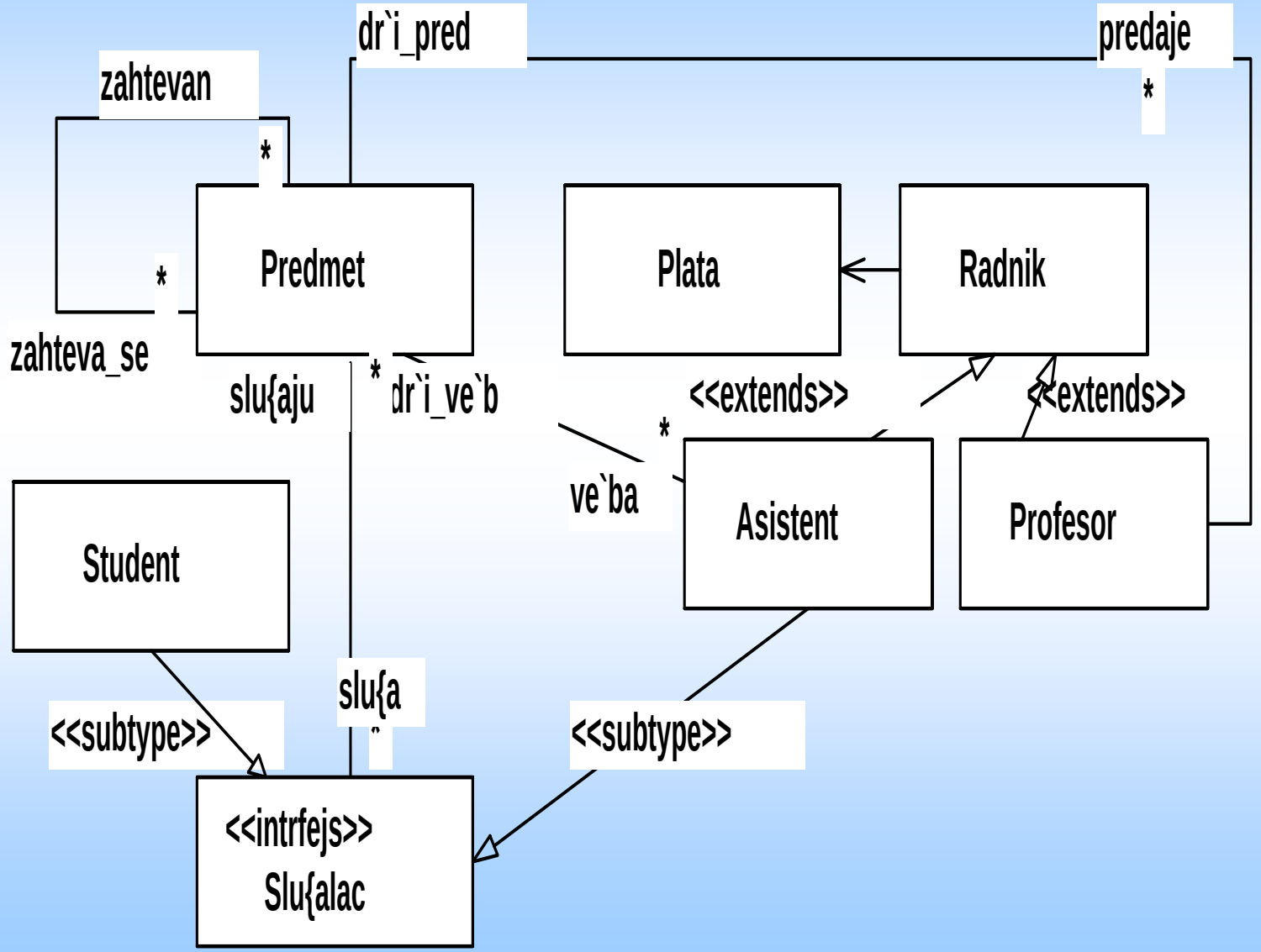
PRIMER1 - GOEMETRIJSKI OBLICI

```
interface GeometrijskiObjekat {
    atribute enum Oblik {Pravoug, Trougao, Krug} oblik;
    attribute struct Tačka {short x, short y} referentna_tačka;
    float obim ();
    float  površina ();
    void  pomeri (short x_pomeraj,  short y_pomeraj);
    void  rotiraj (short ugao);};
class Pravougaonik: Geometrijski oblik
    (extent pravougaonici                    ){
     attribute struct Tačka {short x, short y} referentna_tačka;
     attribute short dužina;
     attribute short širina;};
```

PRIMER 2 - PRIKAZ ŠEME SA ODMG OZNAKAMA



PRIMER2 PRIKAZ ŠEME SA UML OZNAKAMA



PRIMER 2

```
class Kurs
(extent kursevi
 key  sif_kurs)
{
attribute string sif_kurs;
attribute string naziv;
relationship set <Kurs> zahteva_se inverse Kurs:: zahtevan;
relationship set <Kurs> zahtevan inverse Kurs:: zahteva_se;
relationship set<Slušalac> slušaju inverse Slušalac:: sluša;
relationship  Asistent drži_vežb inverse Asistent:: vežba;
relationship Profesor drži_pred inverse Profesor:: predaje;
 boolean drzi_se (in unsigned short semestar) raises (vec_plan);
boolean izostavljen (in unsigned short semestar)
                    raises(vec_izost);
};
```

PRIMER 2

```
class Radnik
(   extent radnici
     key id)
{attribute short id;
    attribute string ime;
    attribute Plata mesecna_plata;
    void() zaposli;
    void otpusti() raises(nema_tog_radnika);};
class Plata {
    attribute float osnovna;
    attribute float prekovremena;
    attribute float minuli_rad;}:
```

PRIMER 2

```
class Profesor extends Radnik
(   extent radnici)
{attribute enum Zvanje{redovni, vanredni, docent} zvanje;
relationship set<Kurs>predaje inverse Kurs:: drži_pred;};

interface Slušalac
{struct Adresa {string grad, string ulica_i_ broj};
 attribute stri ng  ime;
 attribute string broj_indeksa;
 attribute Adresa adresa-stud;
 relationship set <Kurs> sluša inverse Kurs :: slušaju;
boolean upisan_za_kurs (in unsigned short Kurs)
raises (nema_preduslove, popunjeno);
void ispiši(in unsigned short Kurs) raises (nije_upisan);
};
```

PRIMER 2

class Asistent **extends** Radnik: Slušalac
{ **attribute string** ime;
attribute string broj_indeksa;
attribute Adresa adresa-stud;
relationship set <Kurs> sluša **inverse** Kurs :: slušaju;
relationship set <Kurs> vežba **inverse** Kurs :: drži_vezb;};
class Student: Slušalac (**extent** studenti)
{**attribute string** ime;
attribute string broj_indeksa;
attribute Adresa adresa-stud;
relationship set <Kurs> sluša **inverse** Kurs :: slušaju;};

UML DIJAGRAM KLASA
KAO
OBJEKTNI MODEL

KLASE, INTERFEJS I TIPOVI U UML-U

UML uglavnom poštuje ovaj objektni model. U njemu se definiše:

- *"Klasa kao opis skupa objekata koji imaju iste atribute, operacije veze i semantiku"*

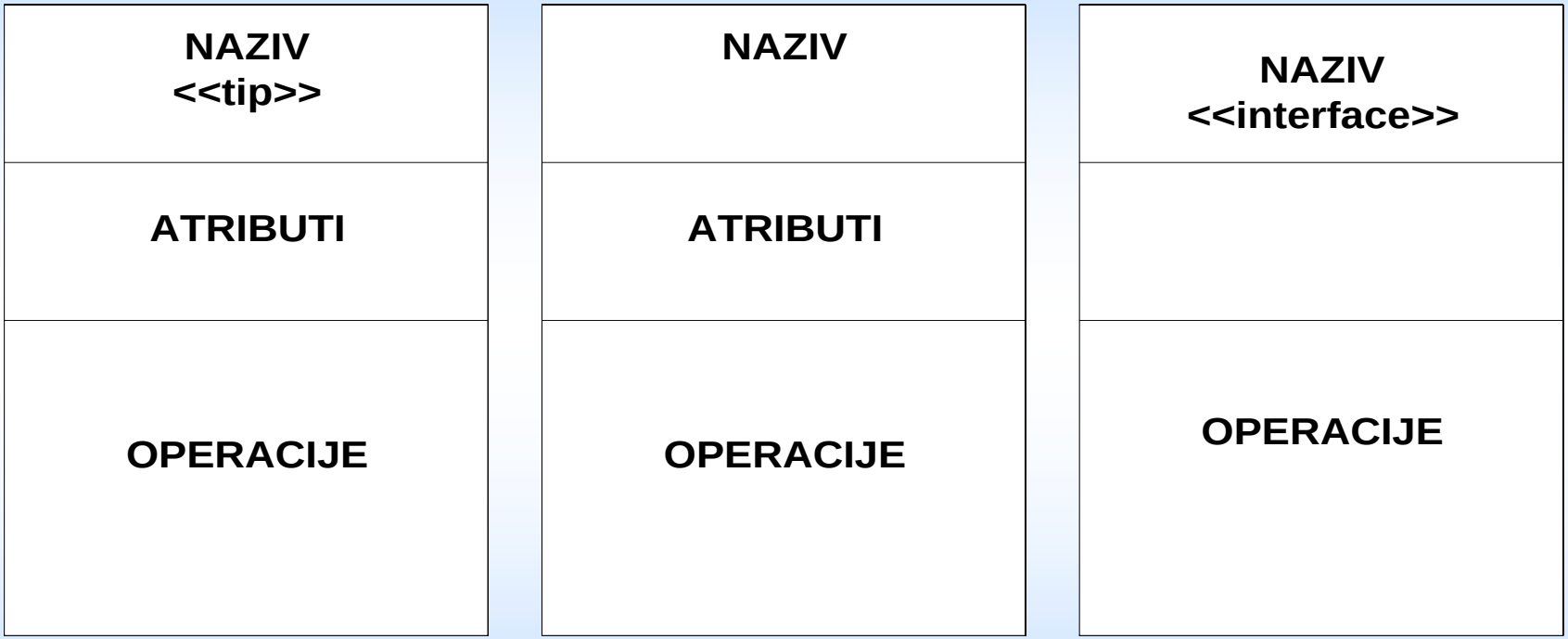
- *"Interfejs kao skup operacija koji definiše neki servis klase ili softverske komponente."* Jedna klasa može da ima više servisa, odnosno interfejsa. Drugim rečima može se reći da klasa implementira odgovarajući interfejs i za takav prikaz se definiše posebna vrsta veze u UML-u

PREDSTAVLJANJE TIPA OBJEKTA U UML-u

Za tip se ne uvodi poseban koncept već se **tip definiše kao *stereotip klase.***

Stereotip je mehanizam preko koga se može proširiti skup koncepata UML-a tako što se izvode novi koncepti iz prethodno definisanih. Stereotip mogu da definišu korisnici, da bi uveli koncepte pogodne za opis njihovog problema.

Neki stereotipovi su definisani i u samom UML-u


TIP
NAZIV
<<tip>>
ATRIBUTI
OPERACIJE
KLASA
NAZIV
ATRIBUTI
OPERACIJE
INTERFEJS
NAZIV
<<interface>>
OPERACIJE

SINTAKSA ZA SPECIFIKACIJU KLASE

NAZIV KLASE
vidljivost nazivi-atributa- 1:tip-podatka-1=pocetrna vrednost {iskaz osobina} **vidljivost naziv-atributa-2: tip-podatka-2=pocetna vrednost-2 {iskaz osobina}** **.....**
vidljivost naziv-operacije-1 (lista-argumenata-1): tip-rezultata-1 {iskaz osobina} **vidljivost naziv-operacije-2 (lista-argumenata-1): tip-rezultata-2 {iskaz osobina}** **.....**

SINTAKSA ZA DEFINISANJE ATRIBUTA I OPERACIJA

ARIBUTI:
[vidljivost] ime [kardinalnost] [:tip] [= poc.-vredn]
[{niz-karaktera}]
OPERACIJE:
[vidljivost] ime [{lista-paraetara}] [:povratni-tip]
[{niz-karaktera}]

Vidljivost može da bude:

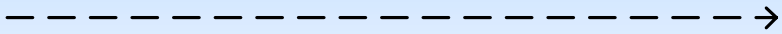
\+ **javna** (atribut, odnosno operacija su dostupni svim objektima)

\# **zaštićena** (atribut, odnosno operacija su dostupni samo nekim objektima)

\- **privatna** (atribut i operacija su dostupni samo posmatranoj klasi, odnosno njenim objektima)

VEZE U UML-U

ZAVISNOST: promena u jednom "predmetu" može da utiče na sematiku drugog

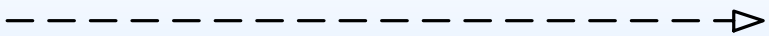


ASOCIJACIJA: opisuje vezu između pojavljivanja objekata, odnosno predstavlja "klasu" veza između pojavljivanja objekata

VEZE U UML-U

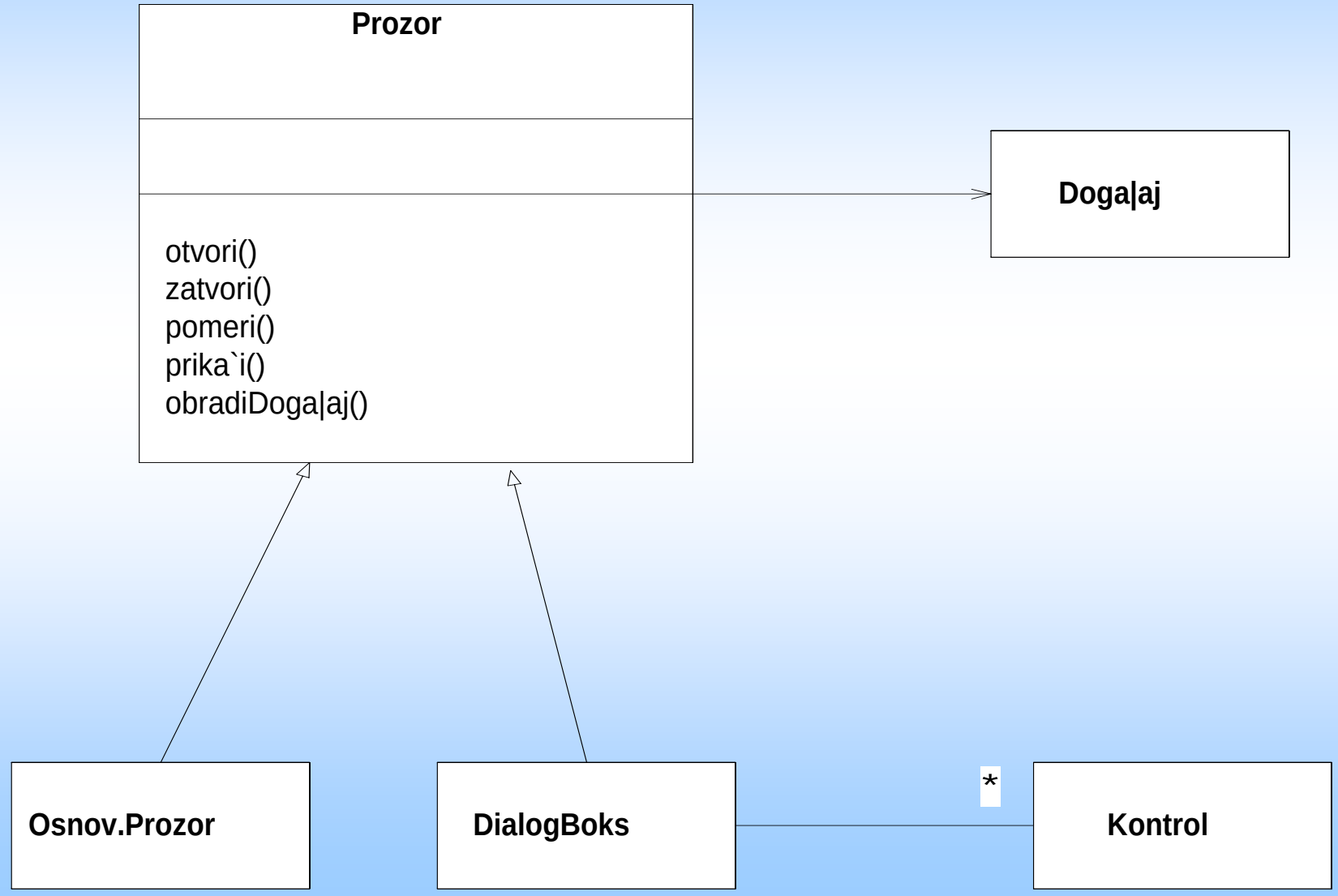
GENERALIZACIJA: 

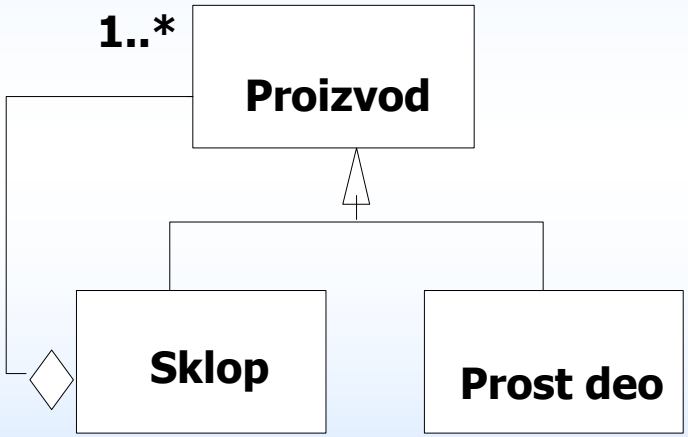
Veza između dva objekta u kojoj je
prvi objekat generalizacija (nadtip) drugog,
odnosno drugi specijalizacija (podtip) prvog.
Podtip nasleđuje stanje i ponašanje nadtipa.

REALIZACIJA: veza 

između specifikacije i implementacije
nekog predmeta (klasifikatora)

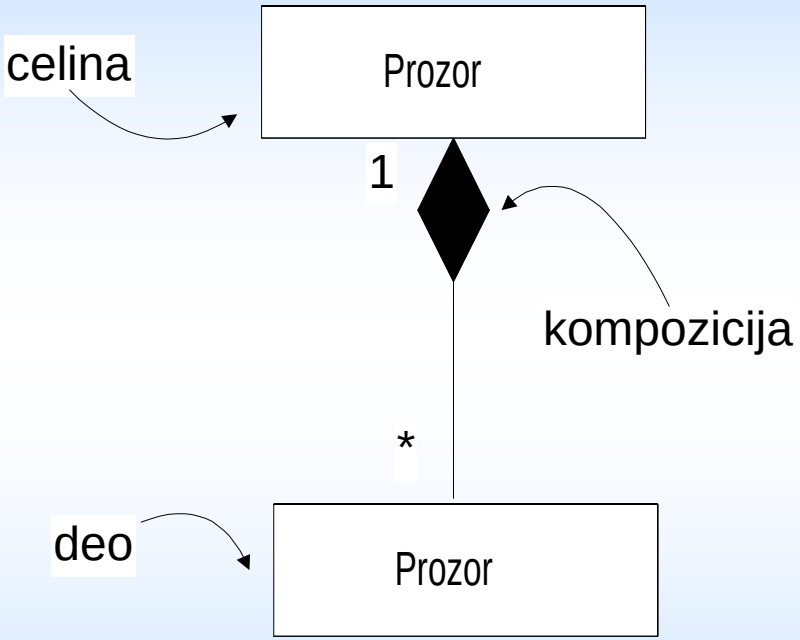
PRIMERI VEZA





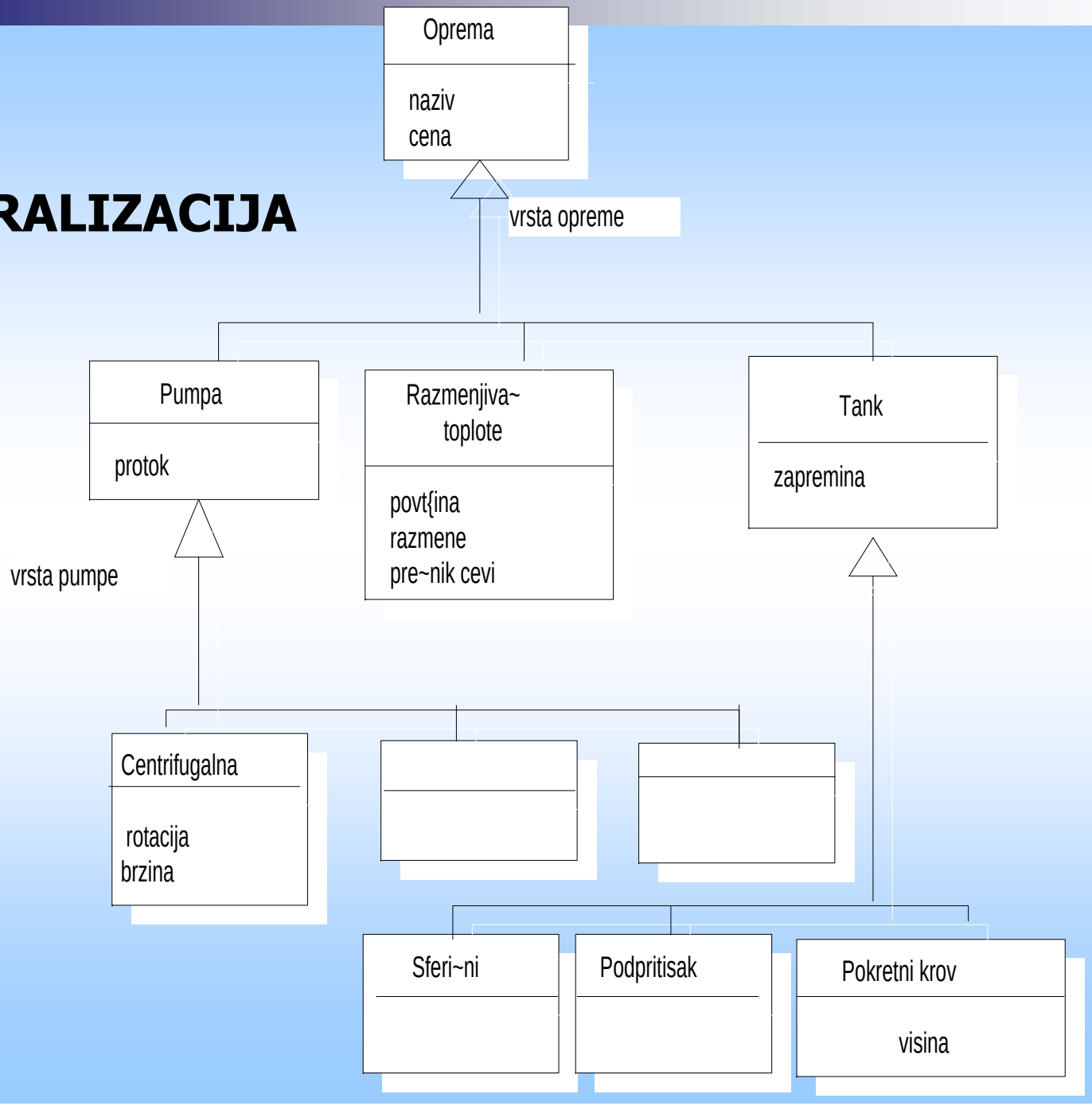
PRIMERI AGREGACIJE: specifična asocijacija, sa smislom "deo od"

KOMPOZICIJA – POSEBNA VRSTA AGREGACIJE



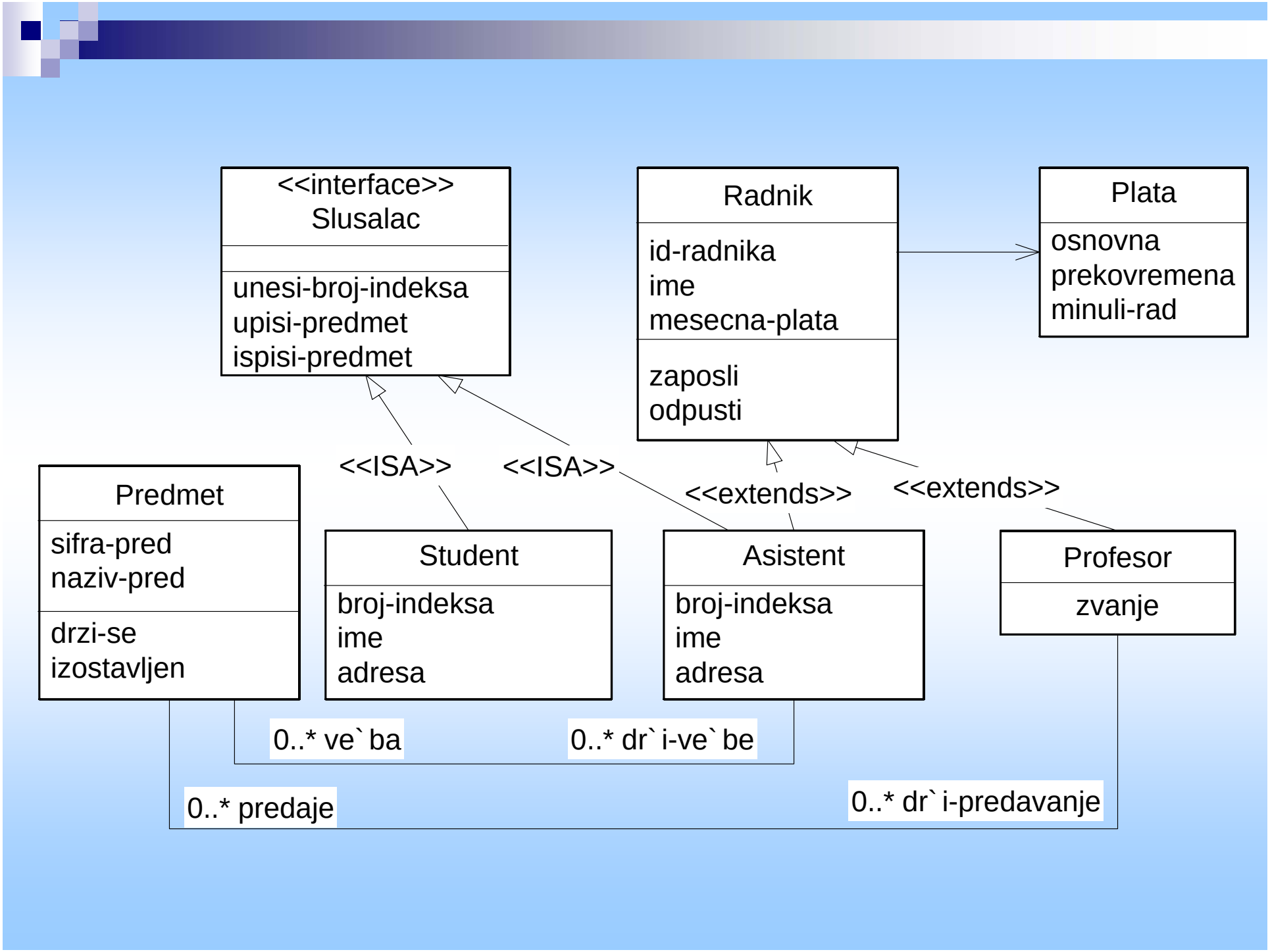
Jedan objekat u jednom trenutku vremena može biti deo samo jedne kompozicije

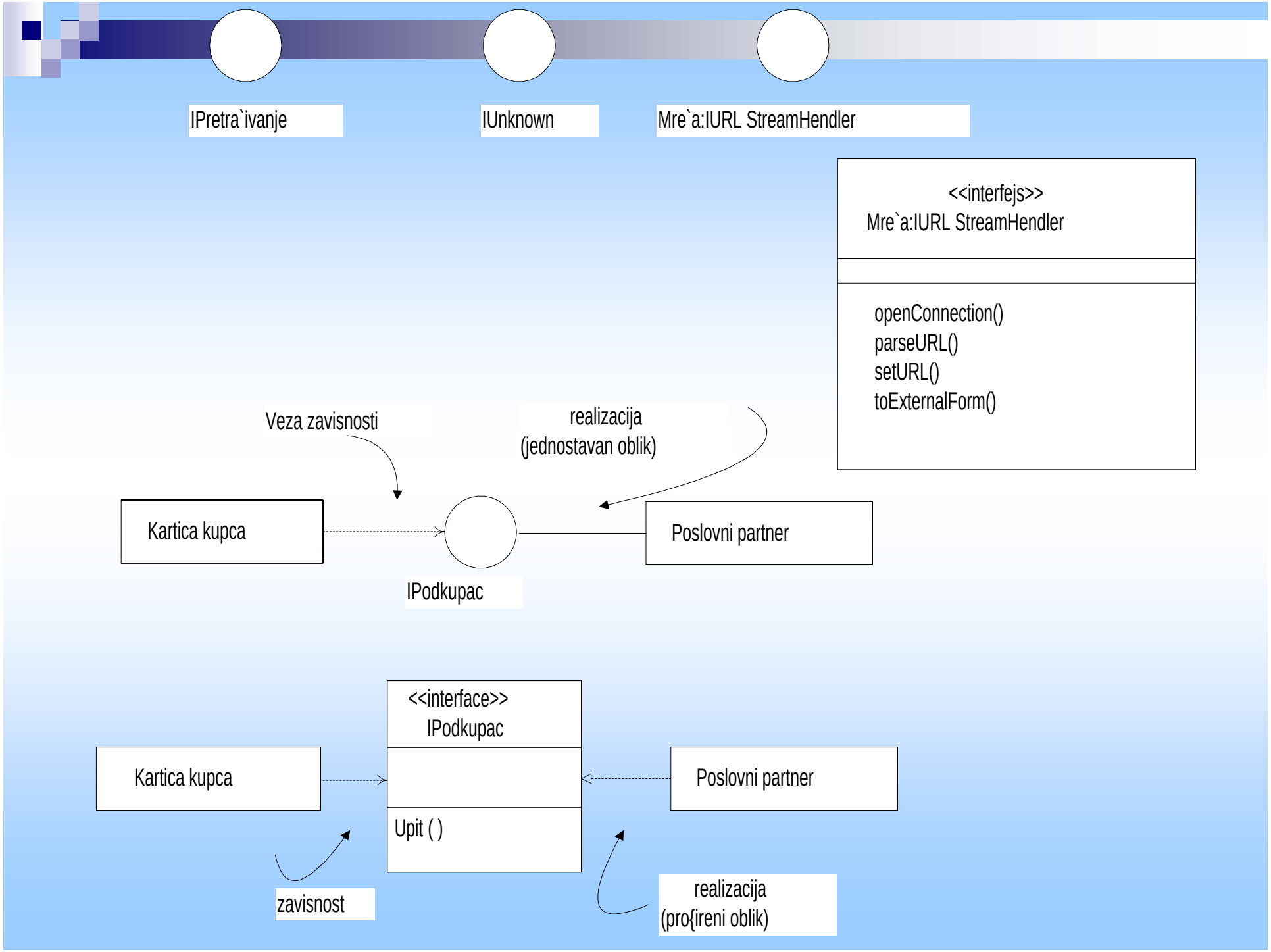
GENERALIZACIJA



INTERFEJS

- **Interfejs** je kolekcija operacija koje opisuju "servise" koje pruža klasa ili komponenta nekog sistema.
- **Interfejs** predstavlja liniju razgraničenja između specifikacije i implementacije neke apstrakcije.
- **Interfejs** se koristi da prikaže i dokumentuje granice između podsistema nekog sistema
- Deklarišući **interfejs** definiše se željeno ponašanje neke apstrakcije, nezavisno od načina njene implementacije


<<interface>>
Slusalac
unesi-broj-indeksa
upisi-predmet
ispisi-predmet
Radnik
id-radnika
ime
mesecna-plata
zaposli
odpusti
Plata
osnovna
prekovremena
minuli-rad
<<ISA>>
<<ISA>>
<<extends>>
<<extends>>
Predmet
sifra-pred
naziv-pred
drzi-se
izostavljen
Student
broj-indeksa
ime
adresa
Asistent
broj-indeksa
ime
adresa
Profesor
zvanje
0..* ve` ba
0..* dr` i-ve` be
0..* predaje
0..* dr` i-predavanje


IPretra`ivanje
IUnknown
Mre`a:IURL StreamHendler
<<interfejs>>
Mre`a:IURL StreamHendler
openConnection()
parseURL()
setURL()
toExternalForm()
Veza zavisnosti
realizacija
(jednostavan oblik)
Kartica kupca
Poslovni partner
IPodkupac
<<interface>>
IPodkupac
Kartica kupca
Poslovni partner
Upit ( )
zavisnost
realizacija
(pro{ireni oblik)

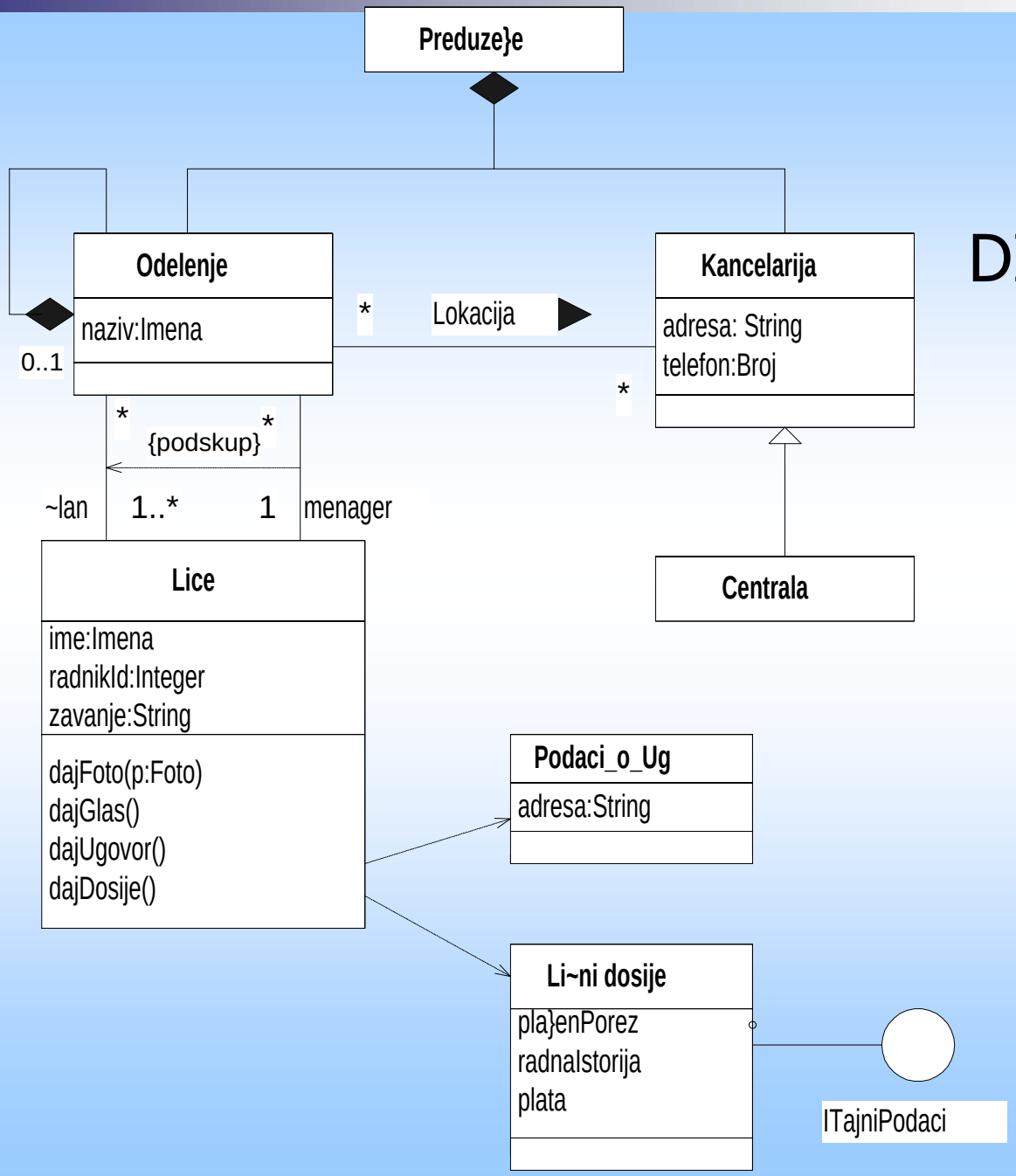
DIJAGRAMI KLASA

- Dijagram klasa najčešće korišćeni dijagam u OO pristupima razvoju softvera

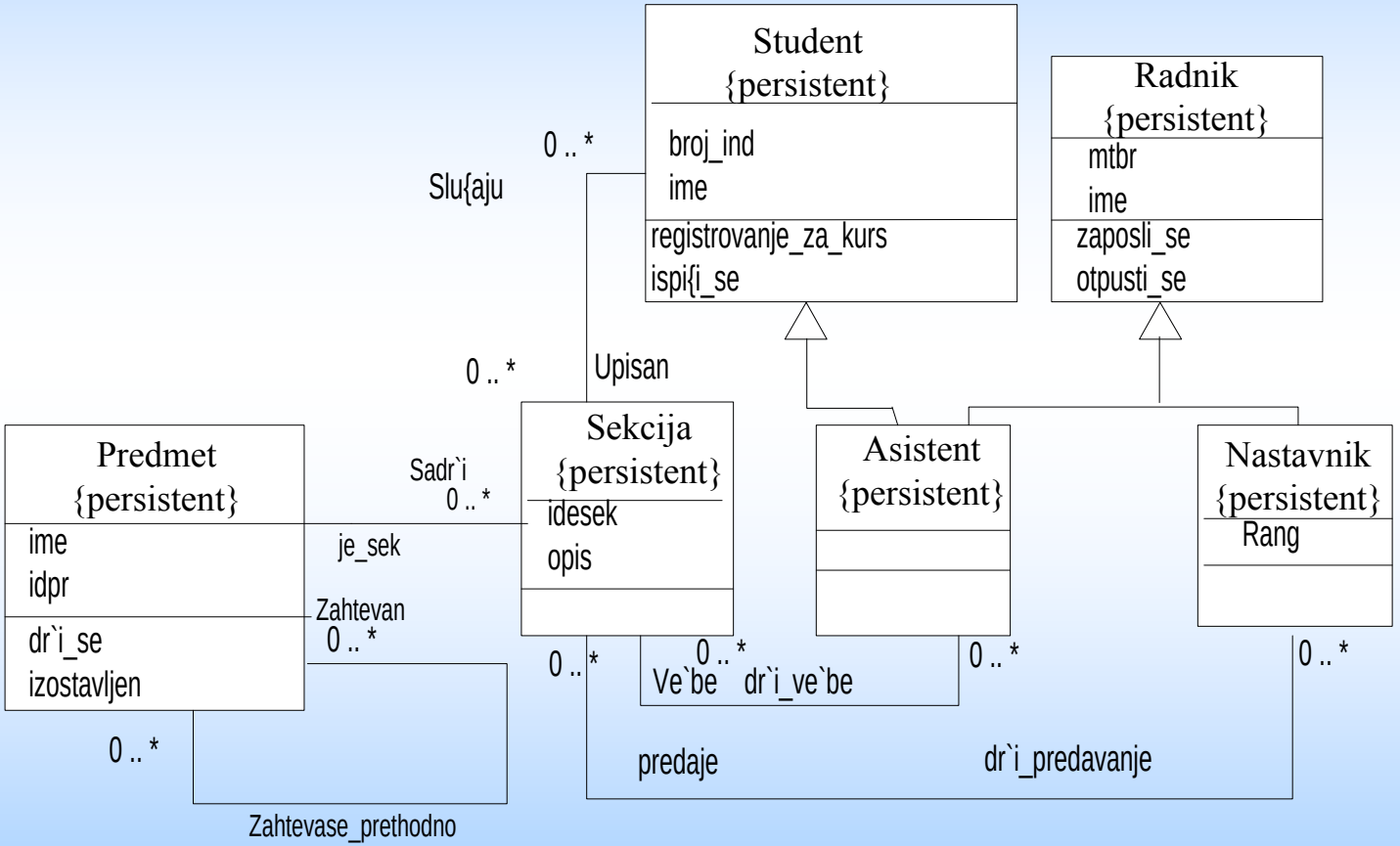
- On predstavlja skupove klasa, interfejsa, kolaboracija i njihove međusobne veze
- Koristi se da predstavi:

1. *Osnovni "rečnik sistema" – definiše pojmove koji se u tom sistemu koriste,*
2. *Opiše strukturu neke kolaboracije,*
3. *Predstavi logičku šemu baze podataka*

PRIMER DIJAGRAMA KLASA



PRIMER DIJAGRAMA KLASA KAO LOGIČKE ŠEME BAZE PODATAKA

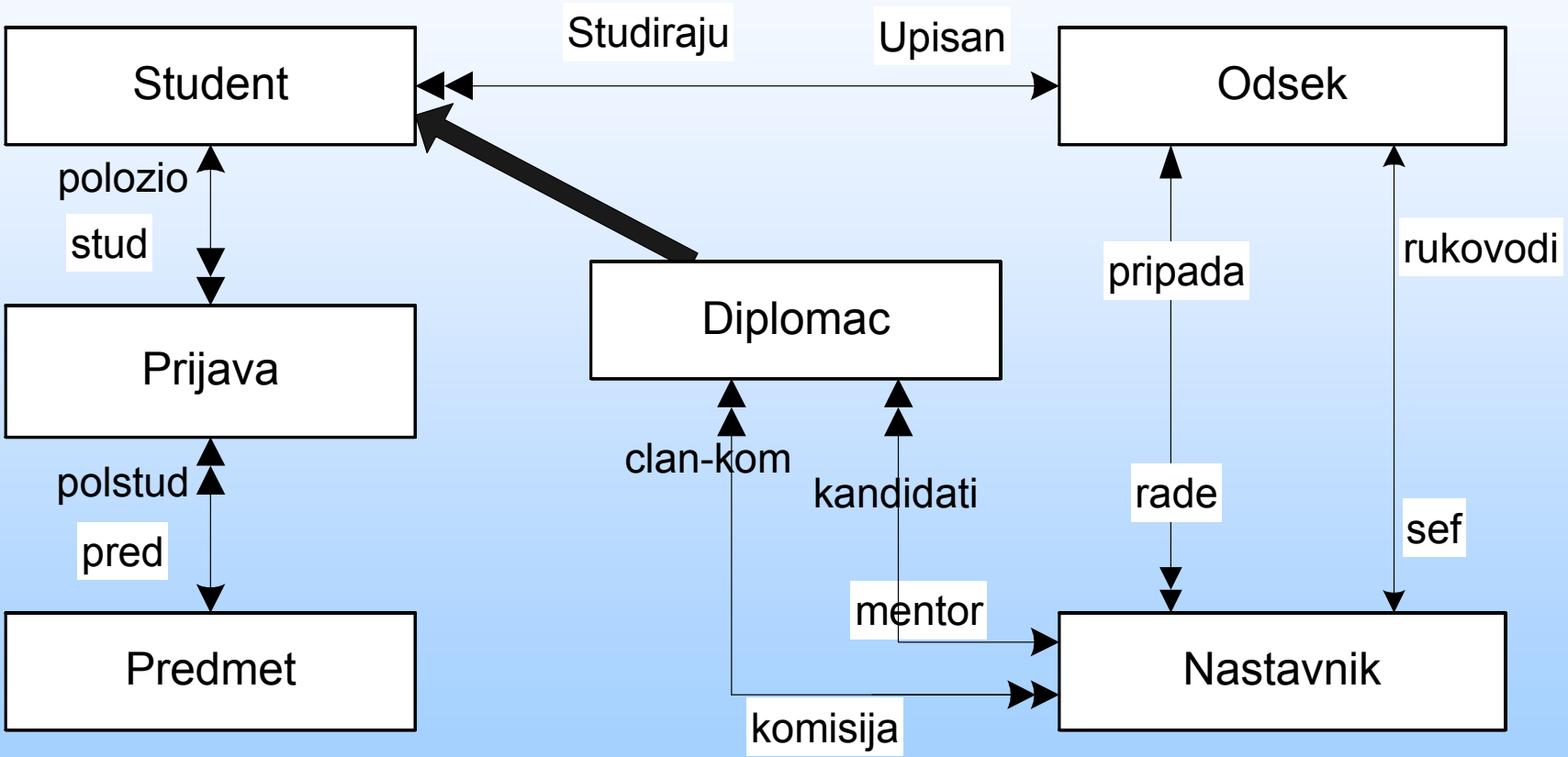


OBJEKTNI UPITNI JEZIK - OQL

OBJEKTNI UPITNI JEZIK - OQL

- POŠTO ODMG RASPOLAŽE SA MNOGO BOGATIJIM SKUPOM TIPOVA, OBJEKTNI UPITNI JEZIK TREBA DA BUDE ZNATNO MOĆNIJI OD RELACIONOG (SQL-A).
- OBJEKTNI UPITNI JEZIK (OQL) NIJE IZVEDEN IZ NEKOG STROGO TEORIJSKOG MODELA, NA PRIMER NEKOG OBJEKTNOG RAČUNA, VEĆ JE DEFINISAN KAO JEDAN JEZIK KOJI NE RADI SAMO SA SKUPOVIMA OBJEKATA, VEĆ I SA KONCEPTIMA STRUKTURE, LISTE I NIZA.
- OQL VEOMA LIČI NA SQL ALI GA ZNATNO PROŠIRUJE SA OO KONCEPTIMA: SLOŽENI OBJEKAT, DEFINICIJA PUTANJE, KORIŠĆENJE OPERACIJA, POLIMORFIZAM, KASNO POVEZIVANJE. KAO NI SQL ON NIJE RAČUNSKI KOMPLETAN.

OBJEKTNI UPITNI JEZIK - OQL: PRIMER OBJEKTNOG MODELA



PRIMER OBJEKTNOG MODELA

```
class Student
    (extent studenti key bi)
{   attribute string bi;
    attribute string ime;
    attribute short starost;
    attribute enum Polovi{M,Z} pol;
    relationship Odsek upisan inverse Odsek::Studiraju;
    relationship set<Prijava> polozio inverse Prijava::
    stud;
    float sred_ocena ( );
    void unesi_ocenu(in short sp; in Ocene ocena)
                raises (nekorektna_ocena, nekorektna_sp);};
```

PRIMER OBJEKTNOG MODELA

```
Class Odsek
   (extent odseci key sod)
{
   attribute short sod;
   attribute string nazivod;
   relationship set <Student>   studiraju
                  inverse Student:: upisan;
   relationship set <Nastavnik>rade
                  inverse Nastavnik:: pripada;
   relationship Nastavnik sef
                  inverse Nastavnik:: rukovodi;
};
```

PRIMER OBJEKTNOG MODELA

```
class Predmet
   (extent predmeti key sp)
{
   attribute short sp;
   attribute string nazivpr;
   attribute string brcas;
   relationship set <Prijava> polstud
                  inverse Prijava:: pred;
   float  sred_oc_pr()
};
```

PRIMER OBJEKTNOG MODELA

```
class Prijava
   (extent prijave)
{
   attribute enum Ocene {6,7,8,9,10} ocena;
   relationship Student stud inverse Student:: polozio;
   relationship Predmet pred inverse Predmet:: polstud;};

class Diplomac extends Student
   (extent diplomci)
   {relationship Nastavnik mentor
         inverse Nastavnik:: kandidati;
   relationship set<Nastavnik> komisija
         inverse Nastavnik:: clan_kom;
};
```

PRIMER OBJEKTNOG MODELA

```
class Nastavnik
    (extent nastavnici key snast)
    {
    attribute short snast;
    attribute string imenast;
    attribute string zvanje;
    relationship Odsek pripada inverse Odsek:: rade;
    relationship set<Diplomaci> kandidati inverse Diplomac ::
    mentor;
    relationship set<Diplomac> clan_kom inverse Diplomac::
    komisija;
    relationship Odsek rukovodi inverse Odsek:: sef;
    void izbor (in string novo_zvanje);
};
```

OBJEKTNI UPITNI JEZIK - OQL

- OQL JE FUNKCIONALNI JEZIK U KOME SE OPERATORI SLOBODNO KORISTE, VODEĆI RAČUNA DA OPERAND POŠTUJE ODGOVARAJUĆI TIP. DRUGIM REČIMA REZULTAT UPITA JE NEKI ODMG TIP I MOŽE BITI PONOVO PODVRGNUT UPITU.
- "ULAZNE TAČKE" U BAZU PODATAKA MOGU DA BUDU BILO KOJI OPSEG (EXTENT) NEKE KLASE ILI BILO KOJI PERZISTENTNI OBJEKAT ČIJE JE IME DEFINISANO PREKO OPERACIJE **bind** OBJEKTA **Database** .
- NA PRIMER "ULAZNA TAČKA" **odseci** UKAZUJE NA PERZISTENTNU KOLEKCIJU OBJEKATA.

OBJEKTNI UPITNI JEZIK - OQL

- KAD GOD SE KOLEKCIJA REFERENCIRA U OQL-u, NEOPHODNO JE NAD NJOM DEFINISATI PROMENLJIVU (TZV "ITERATORSKA" PROMENLJIVA) KOJA UZIMA VREDNOST IZ TE KOLEKCIJE. OVA PROMENLJIVA SE DEFINIŠE NA ISTI NAČIN KAO U SQL-u, u "**from**" DELU UPITA. UZ **"select"** DEO SE OPISUJE REZULTAT, A "**where"** DEFINIŠE USLOV ZA SELEKCIJU ELEMENATA KOLEKCIJE.
- PROMENLJIVU **x** JE MOGUĆE DEFINISATI NA SLEDEĆE NAČINE:

studenti x
x **in** studenti
studenti **as** x

OBJEKTNI UPITNI JEZIK - OQL

```
select x.ime
from studenti x
where  x.pol = "M";
```

PRIKAZANI UPIT KAO ODGOVOR
VRAĆA TIP **bag < string>**

```
select distinct x.ime
from studenti x
where  x.pol = "M";
```

PRIKAZANI UPIT KAO ODGOVOR
VRAĆA TIP **set < string>**

OBJEKTNI UPITNI JEZIK - OQL

- NIJE NEOPHODNO DA SE UPIT POSTAVLJA PREKO "**select…from…where'** KLAUZULE. U NAJPROSTIJEM SLUČAJU JE DOVOLJNO NAVESTI NAZIV PERZISTENTNOG OBJEKTA. SVAKI PERZISTENTNI OBJEKAT SAM PO SEBI SE U OQL-u TRETIRA KAO UPIT. NA PRIMER, UPIT

  odseci;

  VRAĆA REFERENCU NA KOLEKCIJU PERZISTENTNIH OBJEKATA TIPA Odsek. ILI, AKO JE DEFINISANO IME PERZISTENTNOG OBJEKTA "**odsekZaIS**" TADA UPIT

  odsekZaIS;

  VRAĆA REFERENCU NA TAJ POSEBAN OBJEKAT

OQL- PUTANJA

- "IZRAZ PUTANJE" (PATH EXPRESSION) OMOGUĆAVA DA SE, POLAZEĆI OD NEKE "ULAZNE TAČKE" DOSPE DO ŽELJENOG OBJEKTA ILI ATRIBUTA
- SPECIFIKOVANJE PUTANJE OBIČNO POČINJE OD IMENOVANOG PERZISTENTNOG OBJEKTA ILI OD PROMENLJIVE DEFINISANE NAD KOLEKCIJOM OBJEKATA. ONA JE ISTOVREMENO I UPIT. PRIMER:

```
odsekZaIS.sef; // vra}a objekat tipa Nastavnik
odsekZaIS.imenast; //vra}a literal tipa string
odsekZaIS.rade; //vra}a objekat tipa set<Nastavnik>
odsekZaIS.rade.imenast; //nije dozvoljeno
```

OQL- PUTANJA

- NAD KOLEKCIJOM odsek.rade TREBA DEFINISATI PROMENLJIVU I PREKO NJE POSTAVITI UPIT:

```
select f.imenast
from odsekZaIS.rade f;
------
select distinct f.imenast
from odsekZaIS.rade f;
```

PRVI UPIT KAO REZULTAT DAJE TIP **bag<string>**
DRUGI UPIT KAO REZULTAT DAJE TIP **set<string>**

OQL- PUTANJA

- OQL MOŽE DA VRATI KAO REZULTAT KOMPLEKSNU STRUKTURU DEFINISANU U SAMOM UPITU KORIŠĆENJEM KLJUČNE REČI **struct.**

```
select struct (osn_pod: struct (brind : x.bi, imestud : x.ime,
        ocene: (select struct (predmet:
                        element (select z.nazivpr
                                        from y.pred  z), ocena: y.ocena)
                    from  x.polozio y), srednjaoc: x.sred_ocena)
from studenti x;
```

OQL- STRUKTURA REZULTATA

PRETHODNI UPIT DAJE KAO REZULTAT STRUKTURU KOJA SE MOŽE PREDSTAVITI KAO NENORMALIZOVANA TABELA

brind	imest	ocene		srednjaoc

OQL-KREIRANJE POGLEDA

- KAO I U SQL-u, I U OQL-u SE MOGU KREIRATI POGLEDI (VIEWS) KOJI SE NAZIVAJU I "IMENOVANI UPITI". NA PRIMER:

```
define studenti_odseka( ime_odseka)
select x
from studenti x
where x.upisan.nazivod  = ime_odseka;
```

ISKAZ studenti_odseka(InfSist) DAĆE SKUP
STUDENATA OVOG ODSEKA I NAD NJIM SE MOGU
POSTAVLJATI UPITI.
PRETPOSTAVLJENO JE DA U KLASI STUDENT NIJE
 DEFINISANA VEZA **upisan,** VEĆ ATRIBUT
attribute Odsek upisan,
ODNOSNO SAMO JEDNOSMERNA VEZA STUDENT -> ODSEK.
GORNJA DEFINICIJA POGLEDA POKAZUJE KAKO SE U OVOM SLUČAJU
KREIRA INVERZNA VEZA

OQL - UREĐENJE ODGOVORA I NASLEĐIVANJE

- ATRIBUTI I OPERACIJE NADTIPA KORISTE SE U PODTIPU JER IH OVAJ NASLEĐUJE
- UREĐENJE ODGOVORA VRŠI SE NA NAČIN EKVIVALENTAN SQL-u

```
select struct (nastav: struct (imen : x.imenast,
        rang: x. zvanje, mentor_stud :
            (select struct (imes: y. ime, prosek: y. sred_ocena)
            from  x.kandidati y
            order by  prosek desc))
from studenti x
order by rang asc,  imen asc;
```

OQL - OPERACIJE NAD KOLEKCIJAMA

- OPRACIJE NAD KOLEKCIJAMA SU:
  - (1) AGREGIRANE FUNKCIJE (**min, max, count, sum, avg)**
  - (2) ISPITIVANJE ČLANSTVA U KOLEKCIJI (**in)**
  - (3) KVANTIFIKATORSKI IZRAZI (**exists, for all**)

```
count( select  x
       from  prijava x
       where  x. stud.bi = “33/99”);
```

```
select  x.imenast
       from nastavnici  x
       where count (x.clan_kom) > 100);
```

```
select x.ime, x.upisan. nazivod
from  student x
where "Baze podataka"  in
            (select y.pred. nazivpr
             from  x.polozio);
```

Pripadnost kolekciji

(exists v **in** c:b)	Egzistencijalni kvantifikator vra}a **true** ako postoji barem jedan elemenat u c koji zadovoljava b.
(**for all** v **in** c:b)	Univerzalni kvantifikator (**za svako)** **v**ra}a **true** ako svi elementi kolekcije c zadovoljavaju b.

exists g in (**select x**
from studenti x
where x.upisan = “InfSist”)
:g. sred_oc_pred = 10;

(Da li postoji student Odseka za InfSist koji ima prose~nu ocenu 10.)

for all g **in** (**select** x
from diplomci x
where x.mentor.pripada = “InfSist”)
: g. sred_oc_pred > 8.5;

Da li svi diplomci ~iji su mentori sa odseka za InfSist imaju prosek ve}i od 8.5

OQL-OPERACIJE AŽURIRANJA BAZE

- ODL KLASE NISU POTPUNO UČAURENE, POŠTO SE POJEDINIM KOMPONENTAMA NE PRISTUPA ISKLJUČIVO PREKO OPERACIJA, VEĆ I PREKO UPITNOG JEZIKA.

- OQL ZBOG TOGA **NEMA OPERACIJE AŽURIRANJA**, JER SAMO PRETRAŽIVANJE BAZE NIJE BITNO NARUŠAVANJE UČAURENJA.

Objektno-relacioni model

Osnovne karakteristike

- **Korisnički definisani tipovi**
  - **Distinct tip**
  - **Struktuirani tip**
  - **Metode**
- **Konstruisani tipovi**
  - **Referentni tipovi**
  - **Tip vrsta**
  - **Kolekcija**

Korisnički definisani tipovi

Distinkt tip

- Distinkt tip je jednostavan, perzistentni, imenovani korisnički definisani tip, čijim uvodjenjem je podržano strogo tipiziranje.
- Distinkt tipovi su uvek konačni (FINAL), što znači da ne mogu imati podtipove, odnosno da za njih nije podržano nasledjivanje.
- Distinkt tip i njegov izvorni predefinisani tip nisu direktno uporedivi.

Korisnički definisani tipovi

Distinkt tip

Sintaksa definisanja:

CREATE TYPE <naziv distinkt tipa>
AS <predefinisani tip> FINAL
[**<cast opcije prevodjenja>**]

Primer:

```
CREATE TABLE Soba (SobaID    CHAR(10),
                   Duzina    INTEGER,
                   Sirina    INTEGER,
                   Povrsina  INTEGER);
UPDATE Soba
SET Povrsina = Duzina;  =>        OK
```

Korisnički definisani tipovi

Distinkt tip

SQL:1999 standard, uvodjenjem distinkt tipova, rešava prethodni problem:

```
CREATE TYPE identifikator_sobe AS CHAR(10) FINAL;
CREATE TYPE metar AS INTEGER FINAL;
CREATE TYPE kvadratni_metar AS INTEGER FINAL;

CREATE TABLE Soba(SobaID    identifikator_sobe,
                  Duzina    metar,
                  Sirina    metar,
                  Povrsina kvadratni_metar);

UPDATE Soba
SET Povrsina = Duzina;   => GRESKA
```

Korisnički definisani tipovi

Distinkt tip

- *Kada su argumenti izraza kolone, definisane nad distinkt tipovima podataka, neophodno je transformisati njihove vrednosti u izvorne predefinisane tipove:*

*SELECT (Duzina + Sirina) * 2*
FROM Soba
WHERE SobaID = 'xxx'; *=> GRESKA*

Ovaj upit neće moći da se izvrši - operatori sabiranja (+) i množenja (*) definisani su nad numeričkim tipovima podataka

Korisnički definisani tipovi

Distinkt tip

- *Da bi izračunali obim sobe, upit treba modifikovati uključivanjem eksplicitne transformacije argumenata Duzina i Sirina u izvorne predefinisane tipove, korišćenjem funkcije CAST:*

```
SELECT (CAST (Duzina AS INTEGER) + CAST (Sirina AS INTEGER)) *
    2
FROM Soba
WHERE SobaID = 'xxx';          => OK
```

CAST (SOURCE AS DISTINCT) WITH <naziv distinkt tipa>
CAST (DISTINCT AS SOURCE) WITH <predefinisani tip>

Korisnički definisani tipovi

Struktuirani tip

- Omogućuje definisanje perzistentnih, imenovanih tipova, koji mogu imati jedan ili više atributa.
- Atributi mogu biti bilo kog tipa, uključujući druge struktuirane tipove, nizove...
- Sintaksa:

  CREATE TYPE <naziv struktuiranog tipa>

  [UNDER <naziv nadtipa>]

  AS (<naziv atributa> <tip atributa>, ...)

  [[NOT] INSTANTIABLE]

  NOT FINAL

  [<specifikacija referenciranja>]

  [<specifikacija metode>, ...]

Korisnički definisani tipovi

Struktuirani tip

- **U sledećem primeru kreiraju se dva struktuirana tipa: tip osoba, koji se ne instancira (nema svoja pojavljivanja) i koji može imati podtipove, i tip student, koji je podtip tipa osoba, instancira se i takodje može imati podtipove:**

CREATE TYPE osoba AS
(...) NOT INSTANTIABLE NOT FINAL

CREATE TYPE student UNDER osoba AS
(...) NOT FINAL

Korisnički definisani tipovi

- *U sledećem primeru dopunjena je definicija struktuiranog tipa osoba specificiranjem njegove strukture, odnosno atributa prime, jmbg i pol:*

```
CREATE TYPE osoba AS (prime   CHAR(30),
                      jmbg    INTEGER,
                      pol     CHAR(1))
NOT INSTANTIABLE NOT FINAL;
```

Korisnički definisani tipovi

Metode

Originalne metode se specificiraju sledećom sintaksom:

[INSTANCE | STATIC]
METHOD <naziv metode>
(<naziv parametra> <tip parametra>, ...)
RETURNS <tip rezultata>
[SPECIFIC <specifični naziv>]
[SELF AS RESULT]
[SELF AS LOCATOR]
[<karakteristike metode>]

Korisnički definisani tipovi

Metode

Redefinisane metode se specificiraju sledećom sintaksom:

OVERRIDING [INSTANCE | STATIC]
METHOD <naziv metode>
(<naziv parametra> <tip parametra>, ...)
RETURNS <tip rezultata>
[SPECIFIC <specifični naziv>]

Korisnički definisani tipovi

Transformacija se specificira sledećim iskazom:

{TRANSFORM | TRANSFORMS} FOR < naziv tipa >
<naziv transfomacione grupe>
(TO SQL WITH <to sql funkcija>
| FROM SQL WITH <from sql funkcija>)

Korisnički definisani tipovi

U primeru koji sledi kreira se struktuirani tip radnik sa strukturom koju čine atributi ime, osnovna_plata i bonus. Tip može imati pojavljivanja i može se specijalizovati u podtipove :

```
CREATE TYPE radnik AS(ime          CHAR(40),
                      osnovna_plata DECIMAL(9,2),
                      bonus        DECIMAL(9,2))
INSTANTIABLE NOT FINAL
METHOD plata() RETURNS DECIMAL(9,2);
```

Korisnički definisani tipovi

Nakon što je specificirana kao deo definicije tipa radnik, metoda plata() se kreira CREATE METHOD naredbom. Metoda mora biti kreirana u istoj šemi baze podataka kao i struktuirani tip za koji se definiše:

```
CREATE METHOD plata() FOR radnik
BEGIN
...
END;
```

Korisnički definisani tipovi

Sledećom naredbom možemo kreirati novi struktuirani tip menadzer koji je podtip struktuiranog tipa radnik. On ima jedan dodatni atribut udeo, može imati pojavljivanja i može se dalje specijalizovati:

```
CREATE TYPE menadzer UNDER radnik AS(udeo INTEGER)
INSTANTIABLE NOT FINAL;
--redefinisana
OVERRIDING METHOD plata() RETURNS DECIMAL(9,2);
--originalna
METHOD zaduzenje() RETURNS INTEGER ;
```

Korisnički definisani tipovi

*Prilikom definisanja struktuiranog tipa sistem automatski generiše jednu **observer** i jednu **mutator** metodu za svaki atribut. One imaju isti naziv kao i atribut za koji su generisane, služe za pristup, odnosno izmenu vrednosti atributa i ne mogu se redefinisati.*

***Konstruktor** metoda ili konstruktor kreira novu instancu tipa i postavlja atribute na default vrednosti. Ova metoda ima isti naziv kao struktuirani tip i nema argumente, što znači da se za struktuirani tip ST referencira sa ST(), a povratni tip je ST*

Konstruktor se može pozivati sa odgovarajućim argumentima, čime se odredjuju inicijalne vrednosti atributa. Korisnički definisani konstruktori pozivaju se sa operatorom NEW:

NEW <ime_metode> <lista parametara>

Korisnički definisani tipovi

U sledećem primeru prvo definišemo korisnički tip adresa i korisnički konstruktor, a zatim kreiramo nove instance tipa adresa uz pomoć sistemskog i korisničkog konstruktora i pristupamo atributima uz pomoć observer i mutator metoda:

```
CREATE TYPE adresa AS
 (ulica    CHAR (30),
  grad     CHAR (20),
  postbr INTEGER) NOT FINAL
METHOD adresa (ul CHAR (30), gr CHAR (20), pbr INTEGER)
RETURNS adresa

CREATE METHOD adresa (ul CHAR (30), gr CHAR (20), pbr INTEGER)
    RETURNS adresa
BEGIN
    SET self.ulica = ul;
    SET self.grad = gr;
    SET self.postbr = pbr;
   RETURN adresa;
END;
```

Korisnički definisani tipovi

```
CREATE TABLE Adrese OF adresa;

BEGIN
    DECLARE adr1, adr2  adresa;
    SET adr1 = adresa().ulica('Futoska 45').grad('Novi Sad').postbr(21000);
    SET adr2 = NEW adresa('Kosovska 17', 'Beograd', 11000);

  SELECT ulica()
  FROM Adrese
  WHERE grad() = 'Beograd';

  UPDATE Adrese
  SET postbr = 11010
  WHERE grad() = 'Beograd';
END;
```

Korisnički definisani tipovi

Tabele tipova

Ukoliko je tabela T definisana direktno nad struktuiranim tipom ST ona se naziva tabela tipa (*typed table*).

- Tabela tipa se koristi za uskladištenje instanci tipa
- Nad istim struktuiranim tipom može biti definisano više tabela
- Tabela tipa ima kolone koje po nazivu i tipu odgovaraju atributima struktuiranog tipa i jednu kolonu REFC koja je njena referentna kolona (*self-referencing*)
- Deklarisani tip kolone REFC je obavezno REF(ST), a njene vrednosti ne mogu biti NULL
- Deklaracija tabele tipa može se dopuniti elementima koji se ne preuzimaju od struktuiranog tipa, kao što je specifikacija primarnog ključa, spoljnih ključeva i ograničenja na nivou tabele

Korisnički definisani tipovi

Kreiranje tabele tipa:

CREATE TABLE <naziv tabele> OF <struktuirani tip>

Sledi primer kojim se pokazuje način definisanja tabele nad tipom. Prvo definišemo struktuirane tipove adresa, osoba i nekretnine.

```
CREATE TYPE adresa AS(...) NOT FINAL
CREATE TYPE osoba AS(...) NOT FINAL
CREATE TYPE nekretnina AS (nekretninaID INTEGER
                    opis        VARCHAR(50),
                    lokacija    adresa,
                    povrsina    DECIMAL(8,2),
                    vlasnik     osoba) NOT FINAL

CREATE TABLE vlasnistvo OF nekretnine;
```

Korisnički definisani tipovi

Hijerarhija i nasledjivanje

U SQL:1999 standardu je podržano jednostruko nasledjivanje i hijerarhija struktuiranih tipova (broj nivoa hijerarhije je proizvoljan).

- Vrednost struktuiranog tipa ST1 može se dodeliti promenljivoj struktuiranog tipa ST2 ako i samo ako je ST1 podtip od ST2.
- Struktuirani tip ST1 je direktni podtip tipa ST2 ako je ST1 podtip od ST2 i ne postoji tip ST3 koji je podtip od ST2 i nadtip od ST1.
- Tip koji nema nijedan odgovarajući nadtip naziva se maksimalni nadtip, a tip koji nema nijedan podtip naziva se tip list.
- Skup svih podtipova nekog maksimalnog nadtipa ST, koji mora biti jedinstven, naziva se familija podtipova od ST.

Poredjenje korisnički definisanih tipova

- Objekti, odnosno instance nekog korisnički definisanog tipa su apstraktni
- Čak i u slučaju da su sve komponente dva objekta identične, oni se neće smatrati jednakim sve dok se na neki način ne kaže sistemu da ih tretira kao jednake
- Ne možemo koristiti ORDER BY klauzulu niti poredjenje tipa "<" u WHERE klauzuli ukoliko nismo u mogućnosti da poredimo bilo koja dva elementa

Korisnički definisani tipovi

Da bi se omogućilo poredjenje i sortiranje objekata struktuiranih tipova u SQL:1999 standardu uvedena je CREATE ORDERING naredba:

CREATE ORDERING FOR T
EQUALS ONLY BY STATE;

Sledeći oblik CREATE ORDERING naredbe omogućuje primenu svih operatora poredjenja (<, <=, >, >=, = i <>) na objekte struktuiranog tipa T:

CREATE ORDERING FOR T
ORDERING FULL BY RELATIVE WITH F;

Korisnički definisani tipovi

- *Da bi definisali kako se objekti x_1 i x_2 tipa T porede uvodi se funkcija F čiji su argument objekti x_1 i x_2.*
- *Funkcija F se mora napisati tako da je $F(x_1,x_2) < 0$ kad god želimo da zaključimo da je $x_1 < x_2$. $F(x_1,x_2) = 0$ označava da je $x_1 = x_2$, a $F(x_1,x_2) > 0$ označava da je $x_1 > x_2$.*
- *Ukoliko "ORDERING FULL" zamenimo sa "EQUALS ONLY" funkcija $F(x_1,x_2) = 0$ označava da je $x_1 = x_2$, dok sve ostale vrednosti funkcije $F(x_1,x_2)$ označavaju da je $x_1 \neq x_2$.*
- *Poredjenje u odnosu na operator "<" je nemoguće u tom slučaju.*

Korisnički definisani tipovi

- *Promena definicije i izbacivanje korisnički definisanog tipa*

ALTER TYPE <naziv tipa> <akcija promene>

Akcija promene može biti jedna od sledećih:

- ADD ATTRIBUTE <definicija atributa>
- DROP ATTRIBUTE <naziv atributa>
- ADD <specifikacija originalne metode>
- ADD <specifikacija redefinisane metode>
- DROP <naziv metode>

Konstruisani tipovi

- *Konstruisani tipovi u SQL:1999 standardu su referentni tipovi (reference), tipovi vrste i kolekcije.*
- *Konstruišu se pomoću konstruktora tipova REF, ROW i ARRAY.*
- *Konstruisani tipovi mogu biti atomski i složeni (referentni tip je atomski tip, a tip vrste i kolekcija su složeni tipovi).*
- *U postojećoj verziji SQL standarda podržana je samo jedna vrsta kolekcije i to niz.*

Konstruisani tipovi

- *Referentni tip*

- Efekat identiteta objekata ostvaren je u SQL:1999 standardu uvodjenjem koncepta referentnog tipa, odnosno reference.

- Tabele definisane direktno nad struktuiranim tipovima mogu imati *referentnu kolonu*, koja služi kao identifikator n-torki. Referentna kolona može biti primarni ključ tabele ili, na primer, kolona sa jedinstvenim vrednostima koje automatski generiše SUBP.

- Da bi se podržalo referenciranje n-torki tabela sa referentnim kolonama omogućeno je da atributi budu definisani nad tipovima podataka koji su referentni tipovi.

Konstruisani tipovi

- *Referentni tip*

- Referenciranju može biti odredjen opseg (SCOPE), koji se specificira navodjenjem naziva relacije čije se n-torke referenciraju:

A REF(T) SCOPE R

Referentna kolona tabele čije se n-torke referenciraju odredjuje se dodavanjem sledeće klauzule u CREATE TABLE naredbu tabele:

REF IS <naziv atributa> <nacin generisanja>

Konstruisani tipovi

- *Referentni tip*

Naziv atributa je naziv dat koloni koja će služiti kao "identifikator objekta" za kolonu. Način generisanja je tipično:

- · SYSTEM GENERATED, sa značenjem da je SUBP odgovoran za održavanje jedinstvenosti vrednosti kolone u svakoj n-torki

- · DERIVED, sa značenjem da će SUBP koristiti vrednosti primarnog ključa relacije za izvodjenje jedinstvenih vrednosti kolone

Konstruisani tipovi

- *Tip vrsta*

- niz polja koja čine parovi (<naziv podatka>, <tip podatka>)

- novina u SQL:1999 je to da je sada moguće definisati promenljive i parametre koji su tipa vrsta, odnosno definisati kolonu u tabeli koja će imati kompleksnu strukturu

ROW (<naziv polja, tip polja> [{, <naziv polja, tip polja>} ...])

Tip vrste u tabeli može:

- predstavljati vrstu tabele ili
- pripadati jednoj koloni tabele kao složeni tip podatka

Konstruisani tipovi

- *Tip vrsta*

Primer:

```
CREATE TABLE adresa (ulica CHAR(30),
                     broj   INTEGER,
                     grad   CHAR(20));

CREATE TABLE  student (br_indeksa CHAR(6),
                       ime CHAR(15),
                       prezime CHAR(15),
                       adresa  ROW (ulica CHAR(30),
                                    broj INTEGER,
                                    grad CHAR(20)));
```

Konstruisani tipovi

- *Kolekcija*

- Kolekcija je grupa koja se sastoji od nula ili više elemenata istog tipa.

- Broj elemenata kolekcije se naziva kardinalnost kolekcije

- Niz A je uredjena kolekcija u kojoj je svaki elemenat povezan sa tačno jednom rednom pozicijom (koja se naziva indeks)

- Dva niza su uporediva ako i samo ako su tipovi njihovih elemenata uporedivi

- Formalna definicija niza je:

<tip elemenata niza> ARRAY [<maksimalna kardinalnost niza>]

Konstruisani tipovi

- *Kolekcija*

Sledećom naredbom se ubacuje jedna n-torka u tabelu sekcija i zatim se prikazuje naziv sekcije i član sekcije na drugoj poziciji u nizu kojim su predstavljeni članovi.

```
CREATE TABLE sekcija(naziv   CHAR(15),
                     clan    CHAR(20) ARRAY[20]);

INSERT INTO sekcija (naziv,clan)
VALUES ('dramska', ARRAY ['Markovic', 'Popovic', 'Denic']);

SELECT naziv, clan[2] AS ime FROM sekcija;
```

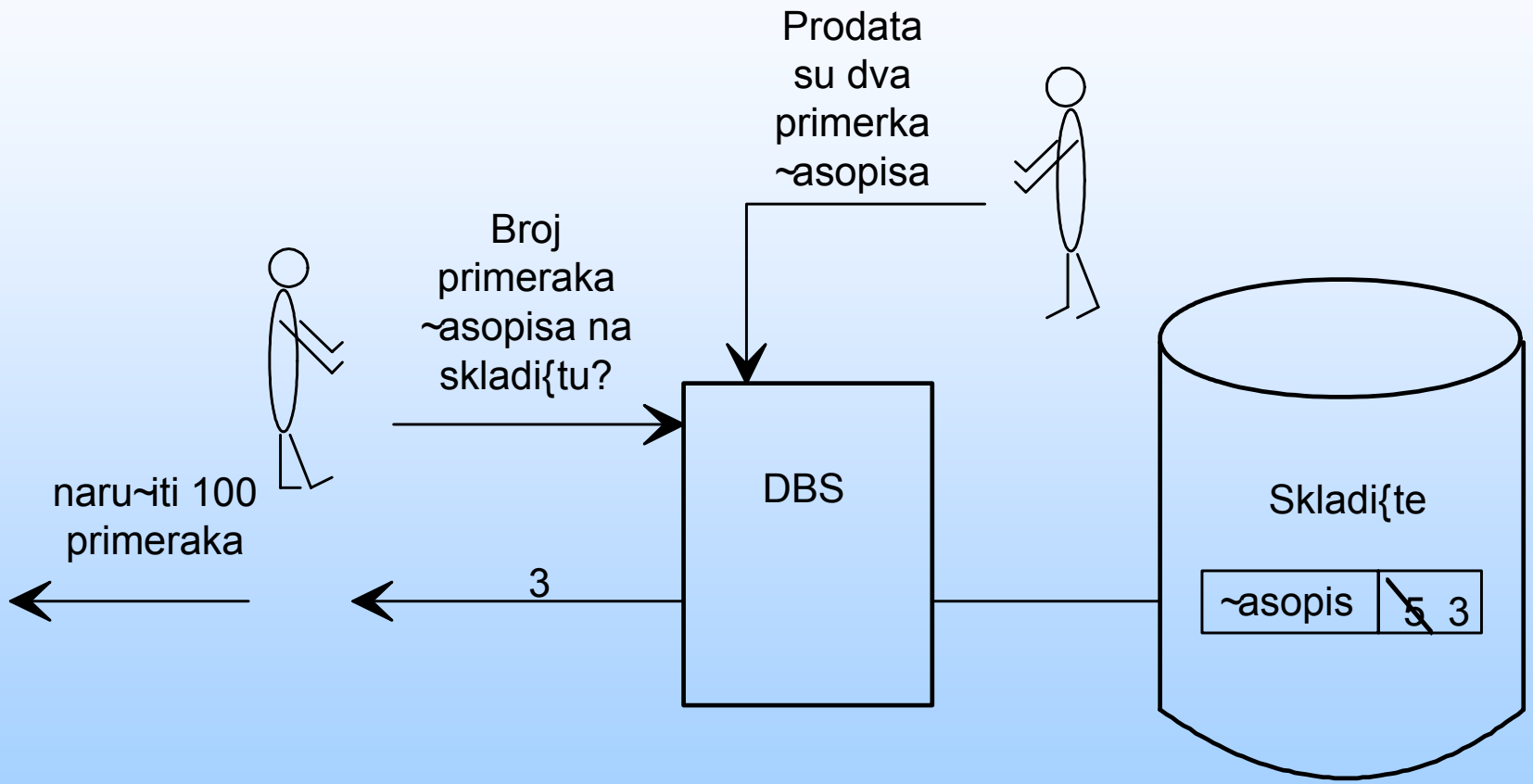
AKTIVNE BAZE PODATAKA

Aktivne baze podataka

Aktivni sistemi baza podataka proširuju funkcionalnost tradicionalnih baza podataka uvođenjem sistema pravila koji se koristi kao opšti, unificirani mehanizam za:

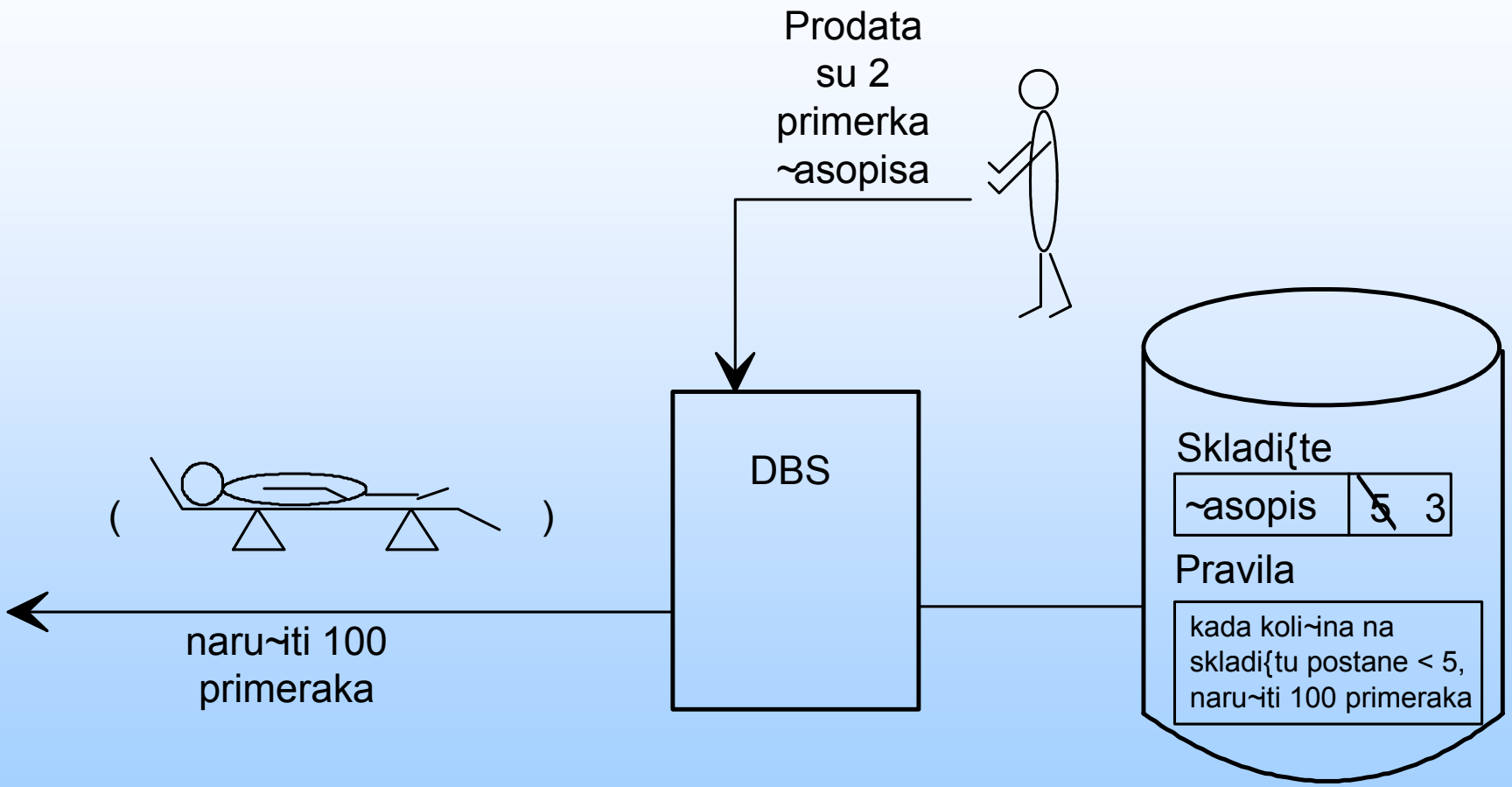
- kontrolu integriteta baze podataka,
- kontrolu prava pristupa,
- održavanje pogleda,
- generisanje izvedenih podataka,
- prikupljanje statističkih podataka,
- praćenje funkcionisanja baze i vođenje žurnala,
- ...

Konvencionalni SUBP



Pasivni sistem baze podataka

Aktivni SUBP



Aktivni sistem baze podataka

Aktivni SUBP

U osnovi aktivnih sistema nalazi se opšta ideja:

- prepoznati prethodno opisane **situacije** (događaje i uslove) u bazi podataka, aplikacijama i okruženju, i
- pokrenuti definisane **reakcije** (operacije nad bazom podataka ili programe) po nastanku situacija.

Pored logičke šeme i činjenica, aktivni sistemi omogućuju obuhvatanje i dodatne dinamike sistema iskazane kroz složena pravila integriteta.

Specifikacija pravila

Svako pravilo definiše situaciju i akcije koje se preduzimaju sa nastankom situacije. Pravila u aktivnim bazama podataka su poznata pod imenom **ECA** (**E**vent-**C**ondition-**A**ction***) produkciona pravila*** i specifikuju se na sledeći način:

ON	dogadjaj	"situacija"
IF	uslov	
DO	akcija	"(re)akcija"

Model znanja: semantika ECA-pravila

Da bi se pravilo moglo formirati prethodno moraju biti definisani događaji. Generalno govoreći, moguće je razlikovati **primitivne** i **složene** događaje. Primitivni događaji su:

- **Ažuriranje podataka**
- **Prikaz podataka**
- **Vreme**
- **Aplikativno definisan događaj**

Akcije se pokreću neposredno nakon (AFTER) ili neposredno pre (BEFORE) detektovanja događaja.

Model znanja: semantika ECA-pravila

Složeni događaji formiraju se kombinovanjem primitivnih i prethodno definisanih složenih događaja. Korisni operatori za kombinovanje događaja mogu biti:

- **Logički operatori**. Događaji mogu biti kombinovani korišćenjem logičkih operatora AND, OR, NOT, itd.
- **Sekvenca**. Pravilo može biti pokrenuto kada se dva ili više događaja pojave u određenom, definisanom redosledu.
- **Vremenska kompozicija**. Pravilo može biti pokrenuto kombinovanjem vremenskih i događaja koji nisu vremenski, na primer "*5 sekundi nakon događaja D1*" ili "*svaki sat nakon prvog pojavljivanja događaja D2*".

Neki aktivni sistemi dozvoljavaju parametrizaciju događaja.

Model znanja: semantika ECA-pravila

Uslov može biti predikat nad stanjem baze podataka, upit nad bazom podataka ili izvršavanje određene aplikativne procedure. U poslednja dva slučaja uslov je zadovoljen ako upit, odnosno aplikativna procedura vraća neke podatke.

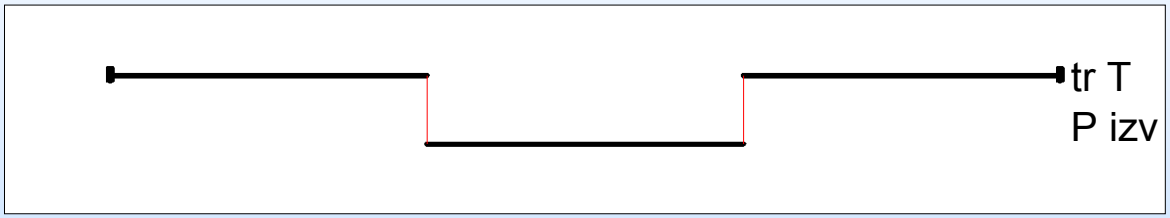
Akcija je bilo koji program. Takav program može da uključuje operacije nad bazom podataka i implicitno ili eksplicitno generiše nove događaje.

Model znanja treba da uključi i implicitne događaje (pravilo se izvršava uvek (ALWAYS) ili prvi put (FIRST) kada uslov bude zadovoljen), (selektivnu) aktivaciju/deaktivaciju pravila, preciziranje konteksta pravila kroz grupisanje, odnosno bolju organizaciju velikih baza pravila itd.

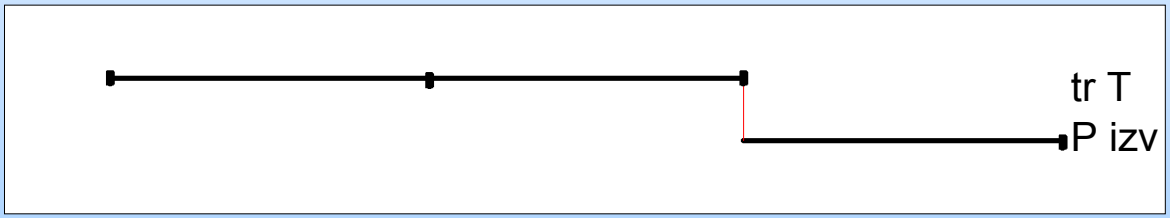
Model izvršavanja: obrada ECA-pravila

Postoji nekoliko različitih načina povezivanja transakcije, odnosno događaja koji pokreće pravilo i izvršavanja ECA-pravila.

a) trenutan način



b) odložen način

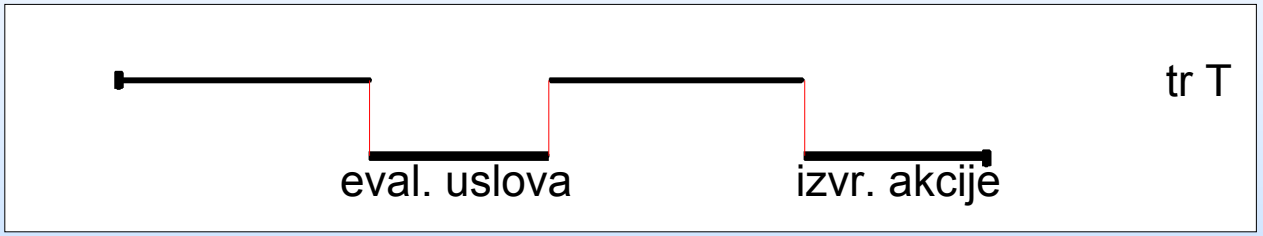


c) razdvojen način

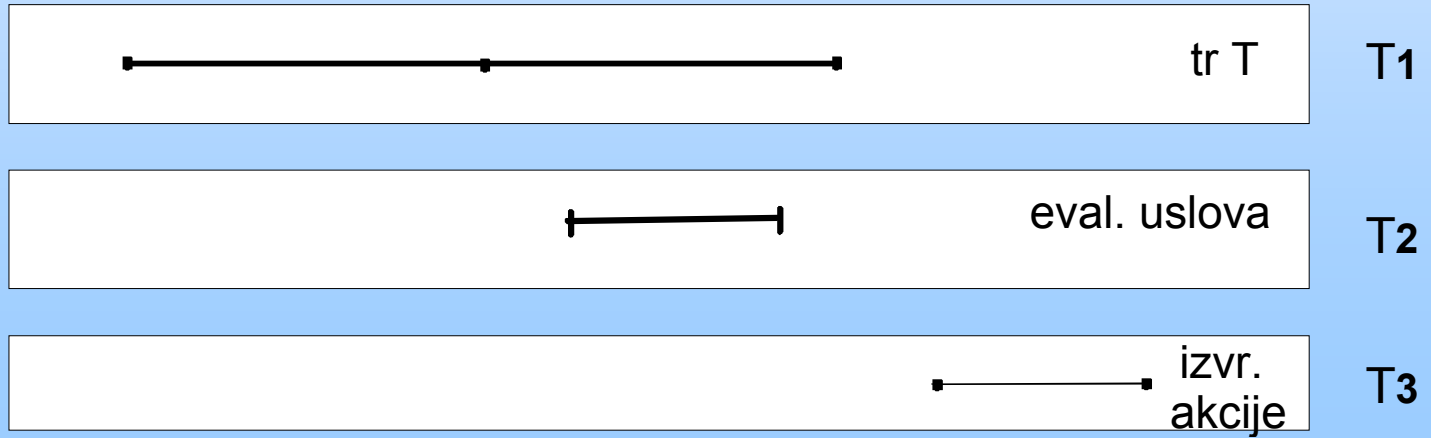
Model izvršavanja: obrada ECA-pravila

Moguće je koristiti i druge, složenije modele izvršavanja ECA-pravila

a) trenutan/odložen način



b) razdvojen/razdvojen način



Model izvršavanja: obrada ECA-pravila

Pored izbora i specifikacije odgovarajućeg načina povezivanja obrade transakcije i pravila, prilikom definisanja modela izvršavanja pravila potrebno je voditi računa i o izboru:

- redosleda izvršavanja pravila (sekvencijalni ili konkurentni) u slučaju da je za isti događaj definisano više pravila;
- najboljeg načina izvršavanja pravila čije akcije pokreću nova pravila (moguće rekurzivno pokretanje samog sebe);
- mehanizma oporavka.

Pregled pristupa razvoju aktivnih baza

Aktivni sistemi se, uslovno rečeno, razvijaju ili kao nadgradnja objektno orijentisanih sistema ili kao nadgradnja relacionih sistema. Najpoznatiji objektno orijentisan sistem za upravljanje bazom podataka, koji podržava ECA pravila, je **HiPAC** (CCA/XAIT, U. Wisconsin).

Najpoznatiji predstavnici aktivnih sistema koji su se razvijali kao nadgradnja relacionih sistema su **Postgres** (UC Berkley) i **Starburst** (IBM Almaden).

HiPAC (High Performance Active Database System)

HiPAC je najpoznatiji aktivni sistem, koji je zasnovan na objektno orijentisanom modelu podataka.

HiPAC poseduje veoma generalizovan jezik pravila. Pravila i komponente pravila su objekti baze podataka, što znači da:

- mogu biti kreirana, prikazana, izmenjena ili izbrisana slično drugim objektima;
- mogu biti grupisana i organizovana slično drugim objektima;
- mogu pripadati potklasama, imati atribute i biti povezana sa drugim objektima.

HiPAC (High Performance Active Database System)

Jedna od klasa podataka u HiPAC-u je klasa ***Rule*** koja je potklasa klase ***Entity***. Klasa *Rule* ima sledeće atribute i operacije:

Atributi	Operacije	
Event	*Create*	*Enable*
Condition	*Delete*	*Disable*
Action	*Modify*	*Fire*
ostali atributi		

Starburst

Starburst je prototipski sistem koji se, od 1985. godine, razvijao u IBM-ovom istraživačkom centru u San Jose-u u Kaliforniji. Jedan od najznačajnijih ciljeva Starburst projekta bio je izrada aktivnog relacionog sistema čija je osnovna karakteristika **mogućnost proširenja**, odnosno uključivanja nove tehnologije za svaku komponentu arhitekture sistema i davanje efikasne podrške za aplikacije bilo kog tipa.

Sintaksa pravila njegovog sistema pravila je zasnovana na SQL-u. Kako su relacioni upitni jezici skupovno orijentisani (operand je cela tabela, odnosno grupa n-torki, a ne pojedinačna n-torka tabele), i pravila su skupovno orijentisana - pokreću ih skupovi promena nad bazom podataka, a akcije pravila mogu takođe da proizvedu skupove promena nad bazom.

Starburst

Pravila su zasnovana na pojmu ***prelaza stanja (state transition)***. Prelaz stanja je promena stanja baze podataka nastala kao rezultat izvršavanja nedeljive sekvence operacija za ažuriranje baze (insert, update, delete). U razmatranje se uzimaju ***neto efekti*** prelaza stanja:

- ukoliko je n-torka ažurirana (menjana) više puta, neto efekat je samo jedno ažuriranje n-torke sa kombinovanim vrednostima pojedinačnih izmena;
- ukoliko je n-torka izmenjena, pa nakon toga izbačena, razmatra se samo izbacivanje;
- ukoliko je n-torka ubačena, pa izmenjena, neto efekat je ubacivanje n-torke sa izmenjenim vrednostima;
- ukoliko je n-torka ubačena, pa nakon toga izbačena, smatra se da nije ni došlo do promene stanja baze podataka.

Starburst

Sintaksa naredbe za kreiranje pravila je sledećeg izgleda:

create rule <naziv_pravila> **on** <tabela>
when <predikat_prelaza>
[**if** <uslov>]
then <lista_akcija>
[**precedes** <lista_pravila>]
[**follows** <lista_pravila>]

Uslov i akcija pravila, posredstvom SQL operacija, mogu da referenciraju tekuće stanje baze podataka. Pored toga, i uslov i akcija mogu da referenciraju ***tabele prelaza*** - logičke tabele koje odražavaju promene nastale u toku prelaza stanja. Postoje četiri tabele prelaza: **inserted**, **deleted**, **new-updated** i **old-updated**. Pravilo može da referencira bilo koju tabelu prelaza koja odgovara nekoj od operacija koje ga pokreću.

Starburst - primer pravila

Jezik pravila ilustrovaćemo pravilom koje kontroliše srednja primanja radnika. Pokreće se uvek kada se ubacuju podaci o novim radnicima ili menjaju plate postojećih. Ukoliko srednja primanja radnika pređu iznos od 100000, transakcija se poništava:

```
create rule kontrola-plata on radnik
when inserted, updated(plata)
if (select avg(plata) from radnik) > 100000
then (select * from inserted) union
     (select * from new-updated);
     rollback
```

Postgres

Postgres je prototipski sistem koji se razvijao na Kalifornijskom univerzitetu, Berkli, kao naslednik INGRES projekta. Postgres je najpre imao PRS I sistem pravila koji je zatim zamenjen PRS II sistemom. Sintaksa pravila oba sistema zasnovana je na Postquel-u.

PRS I sistem pravila se zasnivao na ideji da se bilo koja Postquel naredba može transformisati u pravilo promenom semantike naredbe, tako da se naredba logički izvršava ili ***uvek*** (ALWAYS) ili ***nikad*** (REFUSE) ili ***tačno jedanput*** (ONE-TIME).

Postgres

Sintaksa PRS I pravila je sledećeg izgleda:

{ALWAYS | REFUSE | ONE-TIME} Postquel-naredba

Događaji u PRS I sistemu pravila su ***implicitni***. Za razliku od Starburst pravila, koja su skupovno orijentisana, PRS I pravila su rekord (tuple) orijentisana; pokreće ih promena nad pojedinačnom n-torkom relacije. PRS I pravila nemaju mogućnost referenciranja "starih" i "novih" vrednosti.

Postgres

PRS II sistem pravila je nastao kao rezultat sagledanih nedostataka PRS I sistema i predstavlja povratak na konvencionalni način predstavljanja znanja u vidu produkcionih pravila. Sintaksa PRS II pravila je sledećeg izgleda:

define [**tuple** | **rewrite**] **rule** <naziv>
on <dogadjaj> **to** <objekat>
[**where** <uslov>]
do [**instead**] <lista akcija>

Postgres - primer pravila

Jezik pravila ilustrovaćemo pravilom koje obezbeđuje da Pavle uvek ima istu platu kao Milan:

```
define rule MilanPavle
on new to radnik.plata
where radnik.ime = "Milan"
   do replace radnik (plata = new.plata)
   where radnik.ime = "Pavle"
```

SQL:1999 Trigeri

Trigeri su specifična vrsta ECA (Event-Condition_Action) pravila.

Događaj je operacija ažuriranja baze podataka, uslov je proizvoljni SQL predikat, a akcija je sekvenca SQL naredbi.

Sekvenca SQL naredbi se izvršava ako je detektovan događaj (pokrenuta odgovarajuća operacija ažuriranja baze) i ako je uslov zadovoljen (sračunava se u true).

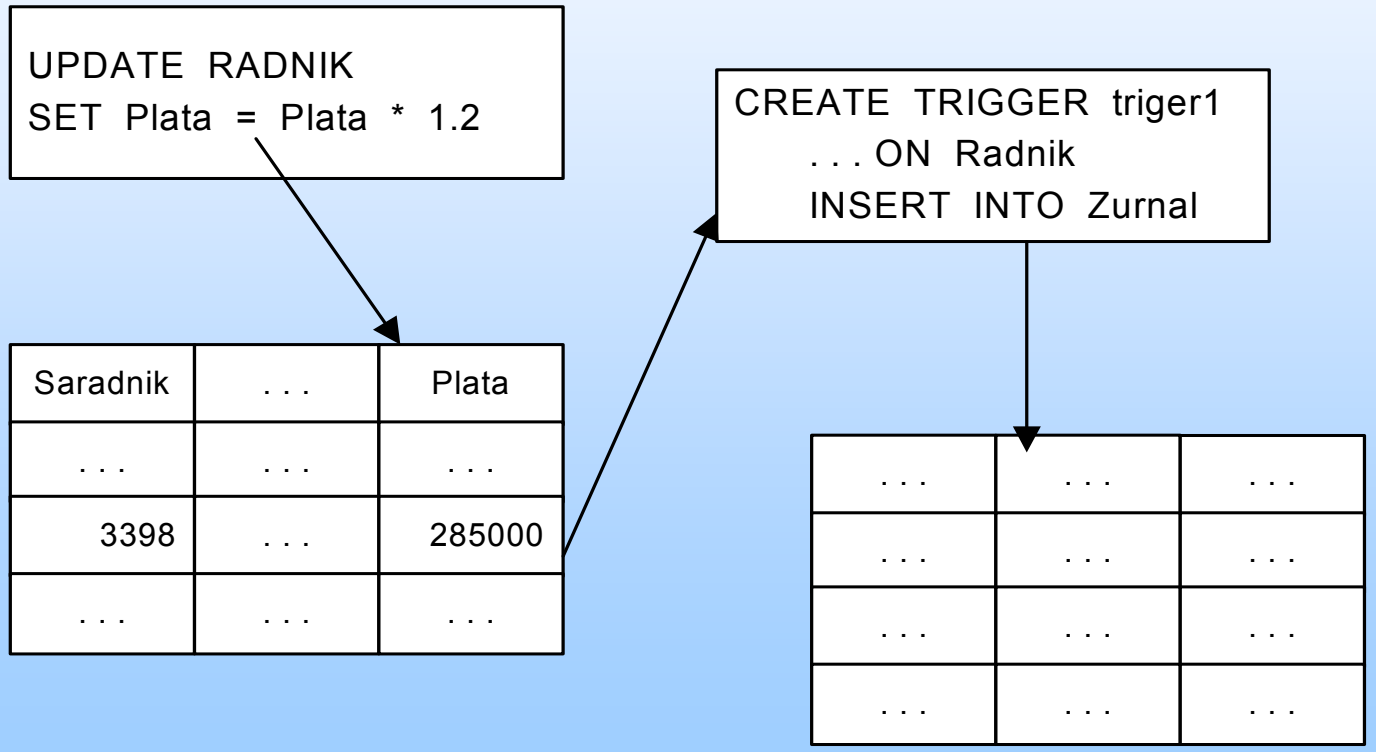
Osnovne karakteristike trigera

SQL:1999 standard definiše trigere kao objekte šeme baze podataka, koji su vezani za tačno jednu baznu tabelu i koji se pozivaju svaki put:

- kada se ubaci jedan ili više redova u tabelu za koju je definisan triger (INSERT naredba),
- kada se promeni jedan ili više redova u tabeli za koju je definisan triger (UPDATE naredba), ili
- kada se obriše jedan ili više redova iz tabele za koju je definisan triger (DELETE naredba).

Osnovne karakteristike trigera

Primer korišćenja trigera za vođenje žurnala o izvršenim operacijama nad bazom podataka:



Klasifikacija trigera

SQL:1999 trigeri mogu biti kategorizovani na više načina, u skladu sa različitim aspektima posmatranja:

- Trigeri mogu da se pozivaju neposredno pre (BEFORE) ili neposredno posle (AFTER) izvršavanja SQL naredbe (operacije ažuriranja) za koju je definisan triger. U skladu sa time mogu se razvrstati kao:

  - BEFORE trigeri
  - AFTER trigeri

- U zavisnosti od toga koji od tri tipa operacija ažuriranja baze podataka (INSERT, UPDATE, DELETE) predstavlja događaj koji izaziva pokretanje trigera, trigeri mogu biti:

  - INSERT trigeri
  - UPDATE trigeri
  - DELETE trigeri

Klasifikacija trigera

Treba napomenuti da SQL:1999 standard ne dozvoljava definisanje trigera za SELECT naredbu, odnosno za prikaz podataka.

Triger se može izvršiti jednom za operaciju ažuriranja koja ga pokreće ili jednom za svaki red koji je ažuriran (ubačen, izmenjen ili obrisan) operacijom koja pokreće triger. U skladu sa time razlikuju se:

- trigeri koji se pokreću na nivou naredbe (statement-level trigeri)
- trigeri koji se pokreću na nivou reda (row-level trigeri)

Klasifikacija trigera

U skladu sa opisanom klasifikacijom, moguće je, na primer, posmatrani triger opisati kao

"a before insert statement-level trigger" (triger koji se pokreće samo jednom neposredno pre izvršavanja insert naredbe nad tabelom za koju je definisan)

ili kao

"an after update row-level trigger" (triger koji se pokreće neposredno posle update naredbe nad tabelom za koju je definisan i to jedanput za svaki red koji se modifikuje tom naredbom).

Specifikacija trigera u SQL:1999 standardu

Za specifikaciju trigera u SQL:1999 standardu koristi se sledeća sintaksa:

1) CREATE TRIGGER <naziv trigera>
2) {BEFORE | AFTER}
3) {INSERT | DELETE | UPDATE [OF <naziv kolone$_1$>, <naziv kolone$_2$>,...]}
4) ON <naziv tabele>
5) [REFERENCING {OLD [ROW] [AS] <sinonim za red pre ažuriranja>
6) | NEW [ROW] [AS] <sinonim za red posle ažuriranja>
7) | OLD TABLE [AS] <sinonim za tabelu sa starim redovima>
8) | NEW TABLE [AS] <sinonim za tabelu sa novim redovima>}, ...]
9) [FOR EACH {ROW | STATEMENT}]
10) [WHEN (<uslov>)]
11) {<SQL naredba>
12) | BEGIN ATOMIC
13) {<SQL naredba>;}...
14) END}

Specifikacija trigera u SQL:1999 standardu

Sledi primer trigera kojim se ograničava povišica plate na najviše 20%:

```
CREATE TRIGGER KontrolaPovecanjaPlate
AFTER UPDATE OF Plata ON Radnik
REFERENCING
    OLD ROW AS StariRed
    NEW ROW AS NoviRed
FOR EACH ROW
WHEN (NoviRed.Plata > StariRed.Plata * 1.2)
    UPDATE Radnik
    SET Plata = StariRed.Plata * 1.2
    WHERE Sradnik = NoviRed.Sradnik;
```

Korisnik koji kreira triger mora da ima TRIGGER privilegiju nad tabelom za koju kreira triger. Vlasnik tabele već ima TRIGGER privilegiju, a korisniku koji nije vlasnik tabele TRIGGER privilegija mora da bude dodeljena naredbom GRANT.

Realizacija SQL:1999 trigera u komercijalnim sistemima

- Mnogi proizvođači sistema za upravljanje bazom podataka su realizovali trigere pre objavljivanja SQL:1999 standarda, uprkos tome što SQL-92 standard nije uključio specifikaciju za trigere. Posledica toga je postojanje razlike u implementaciji trigera u različitim komercijalnim proizvodima. Druga bitna karakteristika je da su mehanizmi trigera u komercijalnim proizvodima dosta moćniji u odnosu na karakteristike propisane SQL:1999 standardom. Većina komercijalnih proizvoda podržava SELECT operaciju i naredbe za definisanje podataka kao događaje koji pokreću trigere, mogućnost definisanja trigera nad pogledom i INSTEAD OF klauzulu, itd.

- Posebno je značajna mogućnost specificiranja INSTEAD OF klauzule pri definisanju trigera, umesto već opisanih BEFORE i AFTER klauzula. Time se omogućuje da se akcija trigera izvrši umesto, a ne kao dodatak na izvršavanje događaja trigera.

Realizacija SQL:1999 trigera u komercijalnim sistemima

• U cilju ilustracije podsetimo se definicije pogleda Odeljenje30:

```
CREATE VIEW Odeljenje30 AS
SELECT Sradnik, Ime, Posao
FROM Radnik
WHERE Odeljenje# = 30;
```

• Navedeni pogled nije ažurljiv jer ne uključuje atribut Odeljenje#, koji je obavezan (NOT NULL). Kako znamo da se kroz pogled Odeljenje30 ubacuju samo radnici odeljenja sa šifrom 30, možemo napisati sledeći triger, koji će navedeni pogled učiniti ažurljivim:

```
CREATE TRIGGER RadnikOdeljenja30
INSTEAD OF INSERT ON Odeljenje30
REFERENCING NEW ROW AS NoviRed
FOR EACH ROW
INSERT INTO Radnik (Sradnik, Ime, Posao, Odeljenje#)
VALUES (NoviRed.Sradnik, NoviRed.Ime, NoviRed.Posao, 30);
```

PROJEKTOVANJE RELACIONIH BAZA PODATAKA

1. PRIMENA NORMALNIH FORMI NA ANALIZU RELACIJA
2. PRIMENA NORMALNIH FORMI NA SINTEZU RELACIJA
3. TRANSFORMACIJA DRUGIH MODELA U RELACIONI

NORMALNE FORME - PROJEKTOVANJE RELACIJA NORMALIZACIJOM

- Normalizacija je postupak projektovanja logi~ke strukture baze podataka. Uobi~ajeno je da se koristi za projektovanje logi~ke strukture relacionog modela. Me|utim, postupak normalizacije ima op{tiji zna~aj i treba ga primenjivati i na druge modele baze podataka (mre`ni i hijerarhijski, relaciono-objektni), a i za projektovanje strukture zapisa u obradi podataka zasnovanoj na kolekciji izolovanih datoteka.

NENORMALIZOVANA RELACIJA

STUDENT

BI	IME	SEM	[SMER	IMERUK	[PRED	NAZPRED	OCENA
21	GORAN	5	01	BATA	121	MATEMAT	7
					323	BAZEPOD	8
					056	MARKSIZ	8
77	ANA	7	01	BATA	056	MARKSIZ	10
					121	MATEMAT	5
36	PERA	4	02	MIKA	323	BAZEPOD	8
					456	ELEKTRON	9
					442	FIZIKA	6
					056	MARKSIZ	8

ANOMALIJE U A@URIRANJU (DODAVANJU ZAPISA, IZBACIVANJU ZAPISA, IZMENI VREDNOSTI ATRIBUTA)

ANOMALIJE (NESIMETRI^NOST) U IZVE[TAVANJU

PRVA NORMALNA FORMA

- Relacija R je u Prvoj normalnoj formi (1NF) ako su sve vrednosti njenih atributa atomske

STUDENT

BI	IME	SEM	[SMER	IMERUK	[PRED	NAZPRED	OCENA
21	GORAN	5	01	BATA	121	MATEMAT	7
21	GORAN	5	01	BATA	323	BAZEPOD	8
21	GORAN	5	01	BATA	056	MARKSIZ	8
77	ANA	7	01	BATA	056	MARKSIZ	10
77	ANA	7	01	BATA	121	MATEMAT	5
36	PERA	4	02	MIKA	323	BAZEPOD	8
36	PERA	4	02	MIKA	456	ELEKTRON	9
36	PERA	4	02	MIKA	442	FIZIKA	6
36	PERA	4	02	MIKA	056	MARKSIZ	8

I DALJE POSTOJE ANOMALIJE U A@URIRANJU!

BAZA PODATAKA BEZ ANOMALIJA

Relacija Student predstavlja spoj (agregaciju) više različitih objekata sistema. Ona istovremeno predstavlja objekte Student, Predmet, Smer, Prijava i njihove međusobne veze. Dekompozicijom na relacije koje predstavljaju navedene objekte dobili bi slede{e relacije, koje su bez anomalija:

Student (BI, Ime, Sem, ŠSmer)

Smer(ŠSmer,ImeRuk)

Predmet(ŠPred, NazPred)

Prijava(BI, ŠPred, Ocena)

FUNKCIONALNE ZAVISNOSTI ATRIBUTA RELACIJE

1. Data je relacija R sa atributima X i Y, mogu}e slo`enim. Atribut Y je funkcionalno zavisan od atributa X (ili X funkcionalno odre|uje Y),

   R.X --> R.Y,

   ako i samo ako svakoj vrednosti X odgovara jedna i samo jedna vrednost Y.

BI, [PRED --> IME, SEM, [SMER, IMERUK,
NAZPRED, OCENA
BI --> IME, SEM, [SMER, IMERUK
[SMER --> IMERUK
[PRED --> NAZPRED

FUNKCIONALNE ZAVISNOSTI ATRIBUTA RELACIJE

Definicija funkcionalne zavisnosti se može dati i na sledeći način:

Atribut Y relacije R je funkcionalno zavisan od atributa X relacije R ako i samo ako, kad god dve n-torke relacije R imaju istu x-vrednost one moraju imati i istu y-vrednost.

Očigledno je da iz definicije funkcionalne zavisnosti sledi i nova definicija ključa i nadključa relacije:

Atribut X, moguće složeni, je nadključ neke relacije R ako i samo ako funkcionalno određuje sve ostale atribute relacije R.

Atribut X, moguće složeni, je ključ relacije R ako je nadključ relacije R, a nijedan njegov pravi podskup nema tu osobinu.

Očigledno je da je složeni atribut ***BI, ŠPred*** ključ relacije ***Student***.

FUNKCIONALNE ZAVISNOSTI ATRIBUTA RELACIJE

2. Atribut Y relacije R je potpuno funkcionalno zavisan od atributa X relacije R ako je funkcionalno zavisan od atributa X, a nije funkcionalno zavisan ni od jednog pravog podskupa atributa X.

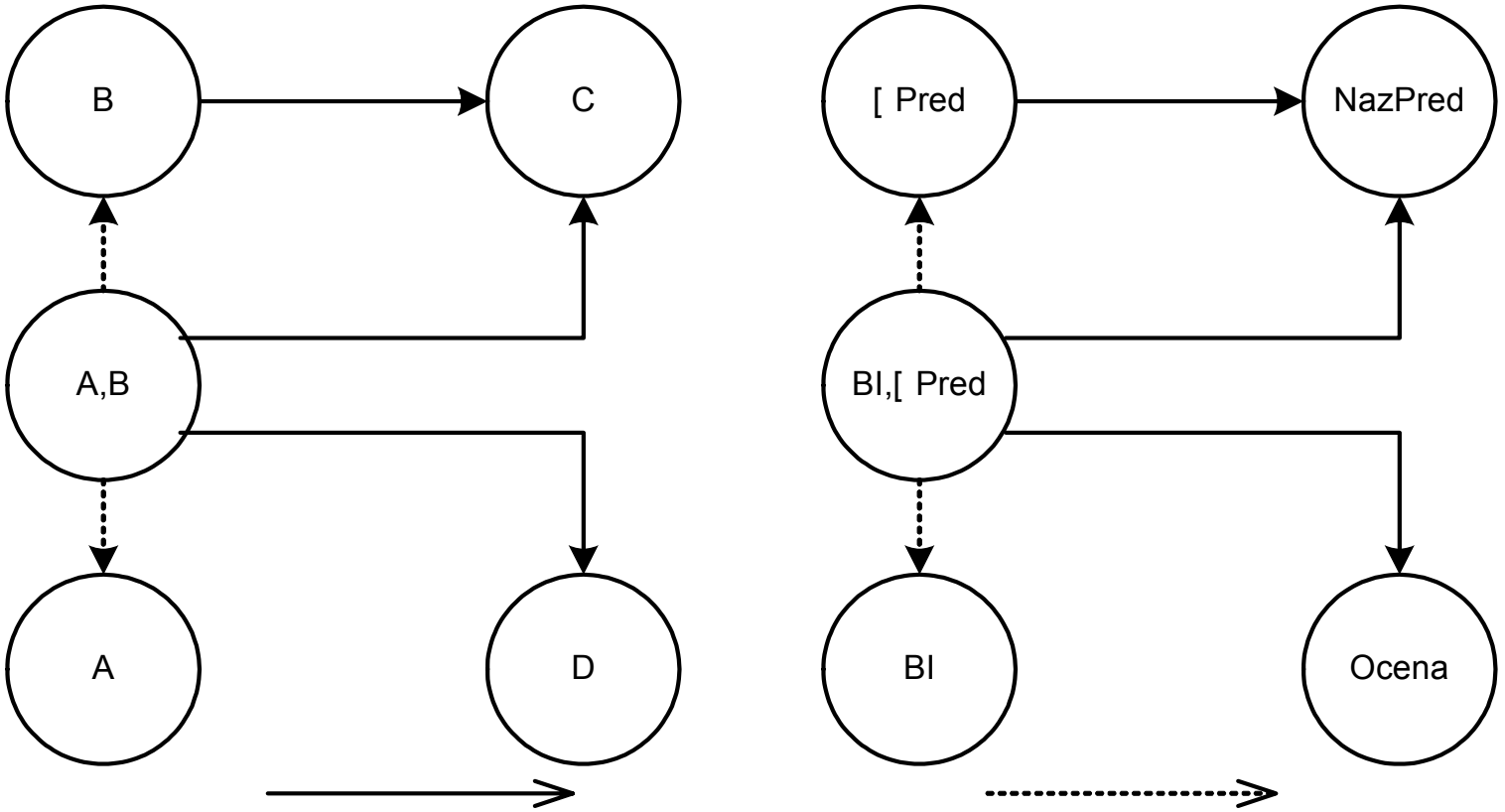
BI, [PRED ->OCENA
BI -/->OCENA
[PRED -/->OCENA

Atribut OCENA je potpuno funkcionalno zavisan od slo`enog atributa BI, [PRED

BI, [PRED ->NAZPRED
BI -/-> NAZPRED
[PRED ->NAZPRED

NAZPRED je nepotpuno funkcionalno zavisan od slo`enog atributa BI, [PRED, jer je funkcionalno zavisan i od njega i od jednog njegovog dela, [PRED

FUNKCIONALNE ZAVISNOSTI ATRIBUTA RELACIJE



(a) Graf definicije potpune funkcionalne zavisnosti

(b) Primer potpune funkcionalne zavisnosti.

FUNKCIONALNE ZAVISNOSTI ATRIBUTA RELACIJE

- Tranzitivna funkcionalna zavisnost se defini{e na slede}i na~in:

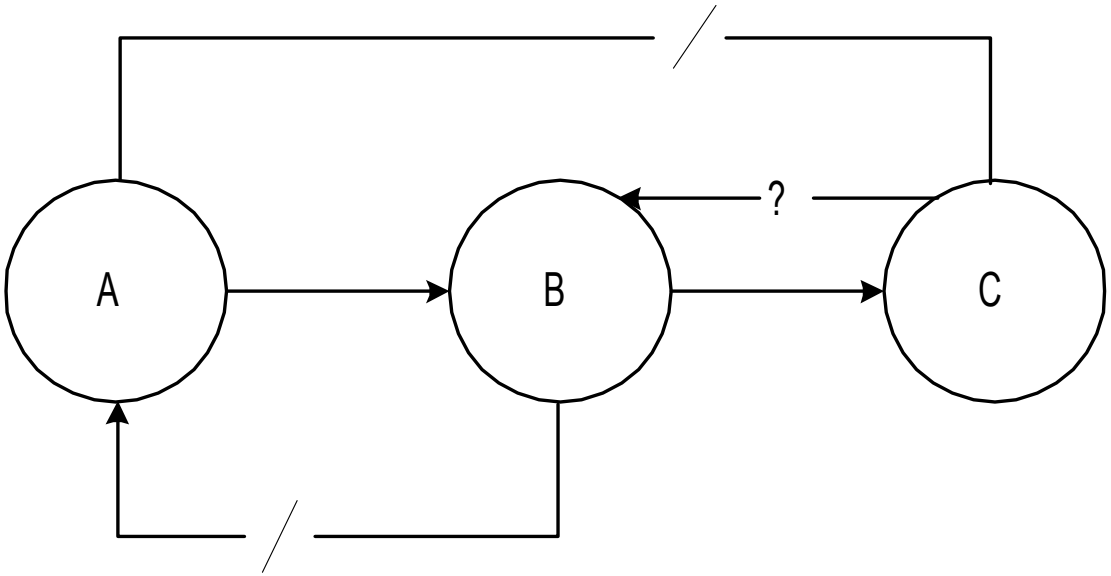
Data je relacija R sa atributima A, B i C, mogu}e slo`enim.
Ako u relaciji R va`i:

A	--->	B
B	--->	C
A	--->	C
B	-/->	A
C	-/->	A,

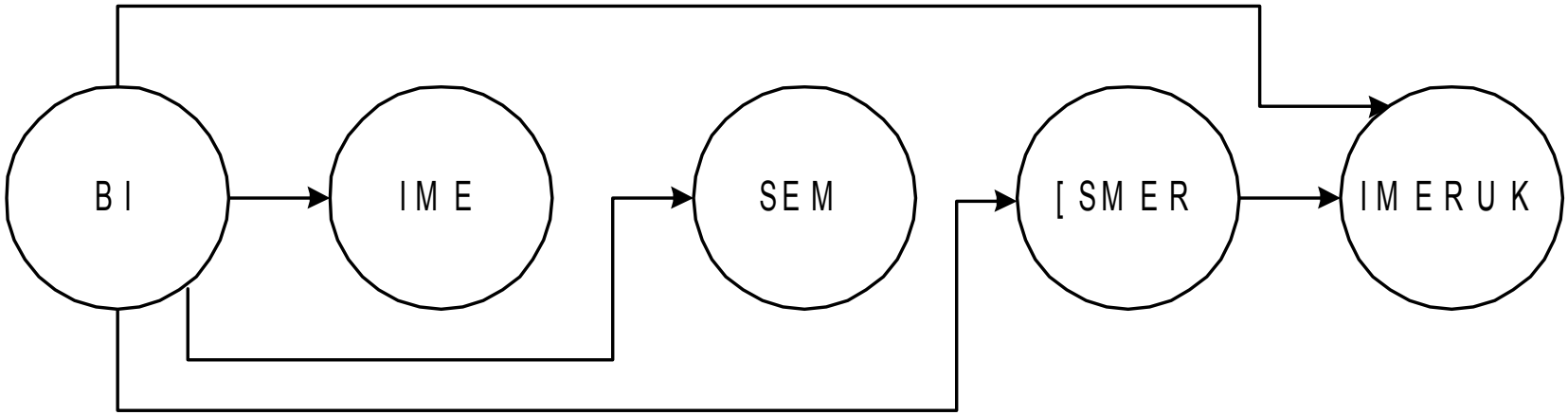
atribut C je tranzitivno funkcionalno zavisan od atributa A.

- Jednostavnije re~eno, *atribut C je tranzitivno funkcionalno zavisan od atributa A ako je funkcionalno zavisan od A i ako je funkcionalno zavisan od nekog atributa B koji je i sam funkcionalno zavisan od A*

DEFINICIJA I PRIMER TRANZITIVNE ZAVISNOSTI



IMERUK JE TRANZITIVNO ZAVISNO OD BI



DRUGA I TREJA NORMALNA FORMA

- Relacija R je u Drugoj normalnoj formi (2NF) ako i samo ako je u 1NF i svi njeni neklju~ni atributi potpuno funkcionalno zavise od primarnog klju~a.

  (Neklju~ni atributi su atributi koji nisu kandidati za klju~, niti deo kandidata za klju~)

- Relacija R je u Tre}oj normalnoj formi (3NF) ako i samo ako je u 2NF i ako svi njeni neklju~ni atributi netranzitivno zavise od primarnog klju~a.

Relacija R je u 2NF i u 3NF ako svi njeni atributi daju jednozna~ne ~injenice o celom klju~u i samo o celom klju~u. (NEFORMALNA DEFINICIJA)

NORMALIZACIJA

STUDENT(BI,[PRED,IME,SEM,[SMER,IMERUK,
NAZPRED,OCENA)

ZBOG NEPOTPUNIH ZAVISNOSTI:

BI, [PRED ->NAZPRED
[PRED ->NAZPRED

BI,[PRED ->IME,SEM,
[SMER, IMERUK
BI -> IME,SEM,
[SMER, IMERUK

RELACIJA **STUDENT** NIJE U DRUGOJ NORMALNOJ FORMI I DEKOMPONUJE SE NA SLEDE]E
PROJEKCIJE

NORMALIZACIJA

STUDENT1(BI,IME,SEM,[SMER,IMERUK)
PREDMET([PRED, NAZPRED)
PRIJAVA(BI,[PRED,OCENA)

U RELACIJI STUDENT1 POSTOJI TRANZITIVNA ZAVISNOST

BI->[SMER; BI->IMERUK; [SMER-> IMERUK

PA ONA NIJE U 3NF I DEKOMPONUJE SE NA PROJEKCIJE:

STUDENT2(BI,IME,SEM,[SMER)
SMER([SMER,IMERUK)

NORMALIZACIJA- BCNF

- Boyce-Codd-ova (BCNF) normalna forma uklanja neke nepreciznosti u definisanju 2NF i 3NF. Za definiciju BCNF uvodi se i pojam **determinante relacije.**

- Determinanta relacije R je bilo koji atribut, prost ili slo`en, od koga neki drugi atribut u relaciji potpuno funkcionalno zavisi.

- Relacija R je u Boyce-Codd-ovoj normalnoj formi (BCNF) ako i samo ako su sve determinante u relaciji i kandidati za klju~.

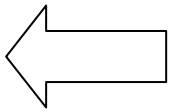
NORMALIZACIJA- BCNF: PRIMER

DATA JE RELACIJA PRIJAVA I NJENE
FUNKCIONALNE ZAVISNOSTI OZNA^ENE KAO
DETERMINANTE (D) ILI KANDIDATI ZA KLJU^ (K)

PRIJAVA(BI, [PRED, NAZPRED, OCENA)

BI, [PRED ---> NAZPRED, OCENA	(D)	(KK)
BI, NAZPRED ---> [PRED, OCENA	(D)	(KK)
[PRED ---> NAZPRED	(D)	
NAZPRED ---> [PRED	(D)	

PRIJAVA(BI,[PRED,OCENA)
PREDMET([PRED, NAZPRED)



PRIMENOM
DEFINICIJE
BCNF
DOBIJA SE

^ETVRTA, PETA I NORMALNA FORMA KJU^EVA I DOMENA

- Relacija R je u 4NF ako u njoj nisu date dve (ili vi{e) nezavisne vi{ezna~ne ~injenice.

- Relacija je u 5NF onda kad se njen informacioni sadr`aj ne mo`e rekonstruisati iz relacija ni`eg stepena, s tim {to se slu~aj relacija ni`eg stepena sa istim klju~em isklju~uje.

- Relacija je u Normalnoj formi klju~eva i domena (DK/NF) ako je svako ograni~enje na vrednosti njenih atributa posledica definicije klju~eva i domena.

^ETVRTA, PETA I NORMALNA FORMA KJU^EVA I DOMENA

Program

Predmet	Nastavnik	Knjiga
Inf-Sist	Branko Zoran Neša	Martin Date
Sist-Anal	Slađan Branko	DeMarco Sarson Martin

Nenormalizovana relacija

^ETVRTA, PETA I NORMALNA FORMA KJU^EVA I DOMENA

Predmet	Nastavnik	Knjiga
Inf-Sist	Branko	Martin
Inf-Sist	Branko	Date
Inf-Sist	Zoran	Martin
Inf-Sist	Zoran	Date
Inf-Sist	Neša	Martin
Inf-Sist	Neša	Date
Sist-Anal	Slađan	DeMarco
Sist-Anal	Slađan	Sarson
Sist-Anal	Slađan	Martin
Sist-Anal	Branko	DeMarco
Sist-Anal	Branko	Sarson
Sist-Anal	Branko	Martin

Relacija svedena na Prvu normalnu formu

^ETVRTA, PETA I NORMALNA FORMA KJU^EVA I DOMENA

U relaciji R(A, B, C) postoji višeznačna zavisnost A ->-> B ako za datu vrednost A, postoji skup od nula, jedne ili više vrednosti B, a taj skup vrednosti ni na koji način ne zavisi od vrednosti atributa C. Atributi A, B i C mogu biti složeni.

Formalna definicija višeznačnih zavisnosti može se dati i na sledeći način:

U relaciji R(A,B,C) postoji višeznačna zavisnost A ->-> B ako i samo ako kad god u njoj postoje n-torke <a,b,c> i <a,b',c'>, postoje takođe i n-torke <a,b,c'> i <a,b',c>. Atributi A, B i C mogu biti složeni.

U navedenom primeru relacije Program postoje sledeće višeznačne zavisnosti:

Predmet -->--> Nastavnik i Predmet -->--> Knjiga

^ETVRTA, PETA I NORMALNA FORMA KJU^EVA I DOMENA

U relaciji R(X, Y, ..., Z) postoji zavisnost spajanja ako i samo ako relacija R rezultuje iz prirodnog spajanja njenih projekcija po X,Y, ..., Z, gde su X,Y, ..., Z podskupovi atributa relacije R.

Relacija R je u Petoj normalnoj formi ako i samo ako se svaka zavisnost spajanja može pripisati kandidatu za ključ.

Rasp(Predmet, Nastavnik)
Udžbenik(Predmet, Knjiga)
Nast-Knj(Nastavnik, Knjiga)

Relacija ***Program*** se može rekonstruisati iz njenih projekcija prirodnim spajanjem relacija ***Nast-Knj*** i ***Udžbenik*** po atributu Knjiga, a zatim prirodnim spajanjem tako dobijenog rezultata sa relacijom ***Rasp*** po atributu Nastavnik. Kako atributi spajanja Knjiga i Nastavnik nisu i kandidati za ključ relacije Program, ona nije u 5NF.

PRIMENA NORMALNIH FORMI NA PROJEKTOVANJE BAZE PODATAKA ANALIZOM RELACIJA

- **DAT JE SKUP NENORMALIZOVANIH RELACIJA**. (NA PRIMER, SVAKA “STAVKA” RE^NIKA PODATAKA STRUKTURNE SISTEMSKE ANALIZE MO@E SE TRETIRATI KAO NENORMALIZOVANA RELACIJA)
- **PRIMENOM DEFINICIJA NORMALNIH FORMI, RELACIJE SE NORMALIZUJU**. PRI TOME SE MO@E ODMAH PRIMENITI DK/NF, ALI JE MNOGO PRAKTI^NIJE POSTEPENO PRIMENJIVATI DEFINICIJE 2NF, 3NF, BCNF 4NF, 5NF.
- **DOBIJENE RELACIJE SA ISTIM KLJU^EM MOGU DA SE KONSOLIDUJU** (SPOJE) AKO SE TIME NE NARU[AVA SEMANTIKA SISTEMA.

TRANSFORMACIJA MODELA OBJEKTI-VEZE U RELACIONI MODEL

P1. Pravilo za objekte. Svaki objekat PMOV postaje relacija sa odgovaraju}im imenom. Atributi ove relacije su:

- svi atributi posmatranog objekta,
- svi identifikatori objekata prema kojima posmatrani objekat ima preslikavanje sa donjom i gornjom granicom kardinalnosti jednakom 1. Pri tome treba imati u vidu i"trivijalna" preslikavanja koja se na DOV ne prikazuju (preslikavanje od slabog prema nadredjenom, od agregacije prema komponenti, od podtipa prema nadtipu), a ~ije kardinalnosti su uvek (1,1).

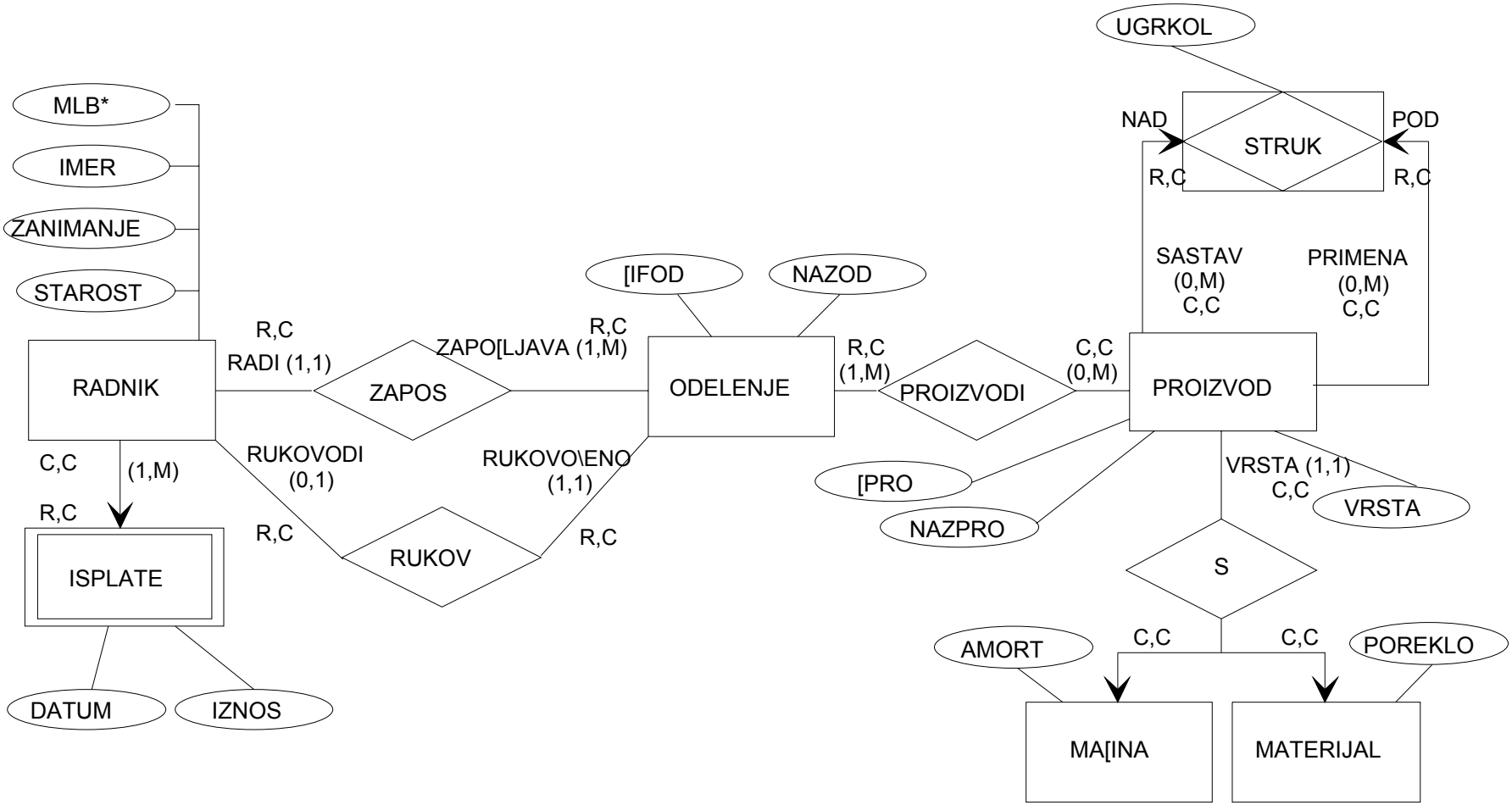
TRANSFORMACIJA MODELA OBJEKTI-VEZE U RELACIONI MODEL

- P1. Pravilo za objekte (nastavak)
- Klju~ relacije je:
  - identifikator objekta, za objekte "jezgra" (objekte koji nisu ni agregacije, ni slabi, ni podtipovi),
  - za slab objekat, identifikator nadre|enog objekta, pro{iren sa jednim ili vi{e atributa slabog, koji jedinstveno identifikuju slabi objekat u okviru nadredjenog,
  - za agregaciju, slo`eni kljuc koga ~ine identifikatori objekata koji ~ine agregaciju,
  - za podtip, identifikator nadtipa.

TRANSFORMACIJA MODELA OBJEKTI-VEZE U RELACIONI MODEL

- **P2. Dodatno pravilo za veze.** Preko pravila (P1) u relacioni model su prevedeni svi objekti i sve veze koje imaju barem jedno preslikavanje s kardinalno{}u (1,1) iz MOV.
  Sve ostale veze postaju posebne relacije. Ime relacije je ime veze.
  Atributi ovih relacija su identifikatori objekata koji ~ine vezu.
  Klju~ ovako dobijene relacije je:
  - slo`eni klju~ koga ~ine oba identifikatora, ako je gornja granica kardinalnosti oba preslikavanja M,
  - identifikator objekta sa gornjom granicom kardinalnosti preslikavanja 1, ako jedno preslikavanje ima gornju granicu 1, a drugo M
  - identifikator bilo kog objekta, ako oba preslikavanja imaju gornju granicu kardinalnosti preslikavanja jednaku 1.

PRIMER


UGRKOL
MLB*
IMER
ZANIMANJE
STAROST
NAD
STRUK
POD
R,C
R,C
SASTAV
(0,M)
C,C
PRIMENA
(0,M)
C,C
[IFOD
NAZOD
R,C
R,C
RADI (1,1)
ZAPO[LJAVA (1,M)
R,C
(1,M)
C,C
(0,M)
RADNIK
ZAPOS
ODELENJE
PROIZVODI
PROIZVOD
C,C
(1,M)
RUKOVODI
(0,1)
RUKOVO\ENO
(1,1)
[PRO
VRSTA (1,1)
C,C
VRSTA
R,C
R,C
RUKOV
R,C
NAZPRO
S
ISPLATE
AMORT
C,C
C,C
POREKLO
DATUM
IZNOS
MA[INA
MATERIJAL

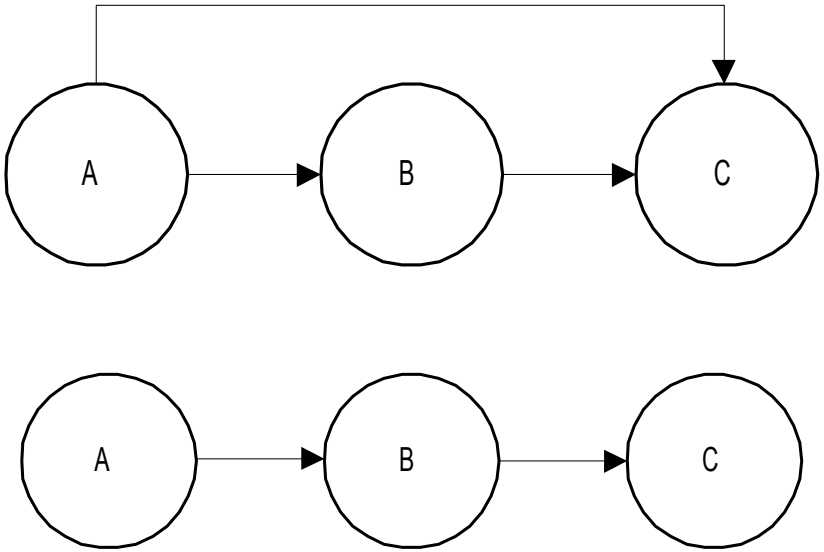
PRIMER

Na osnovu pravila P1 dobija se:

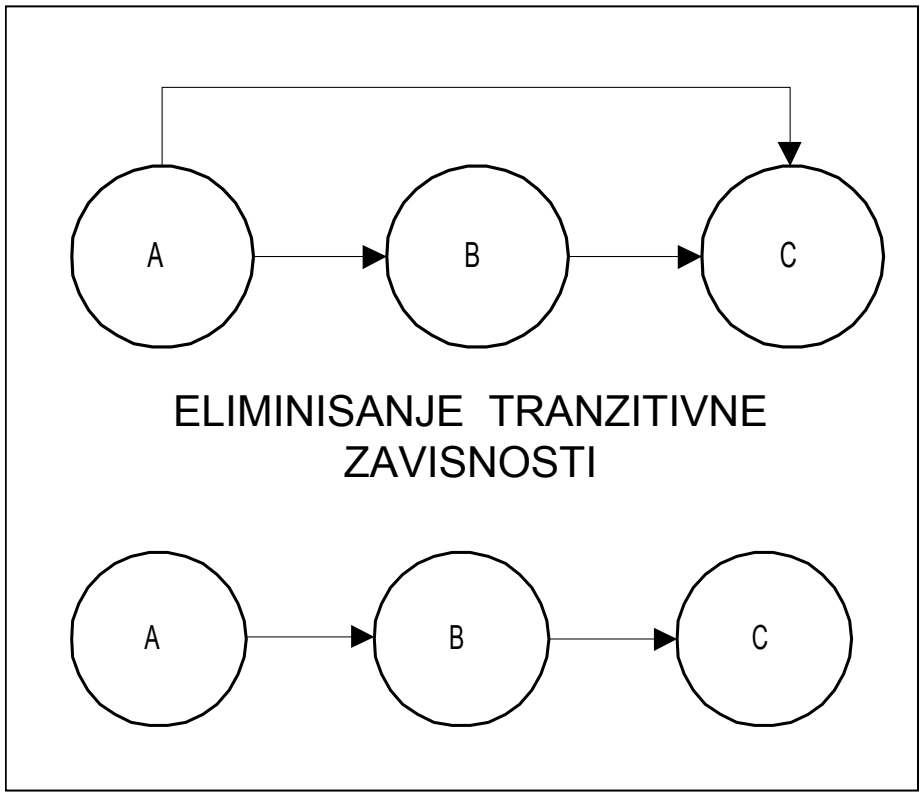
RADNIK(MLB, IMER, ZANIMANJE, STAROST, [IFOD)
PLATE(MLB,DATUM,IZNOS)
ODELENJE([IFOD, NAZOD, MLB)
PROIZVOD([PRO,NAZPRO,VRSTA)
MA[INA([PRO,AMORT)
MATERIJAL ([PRO, POREKLO)

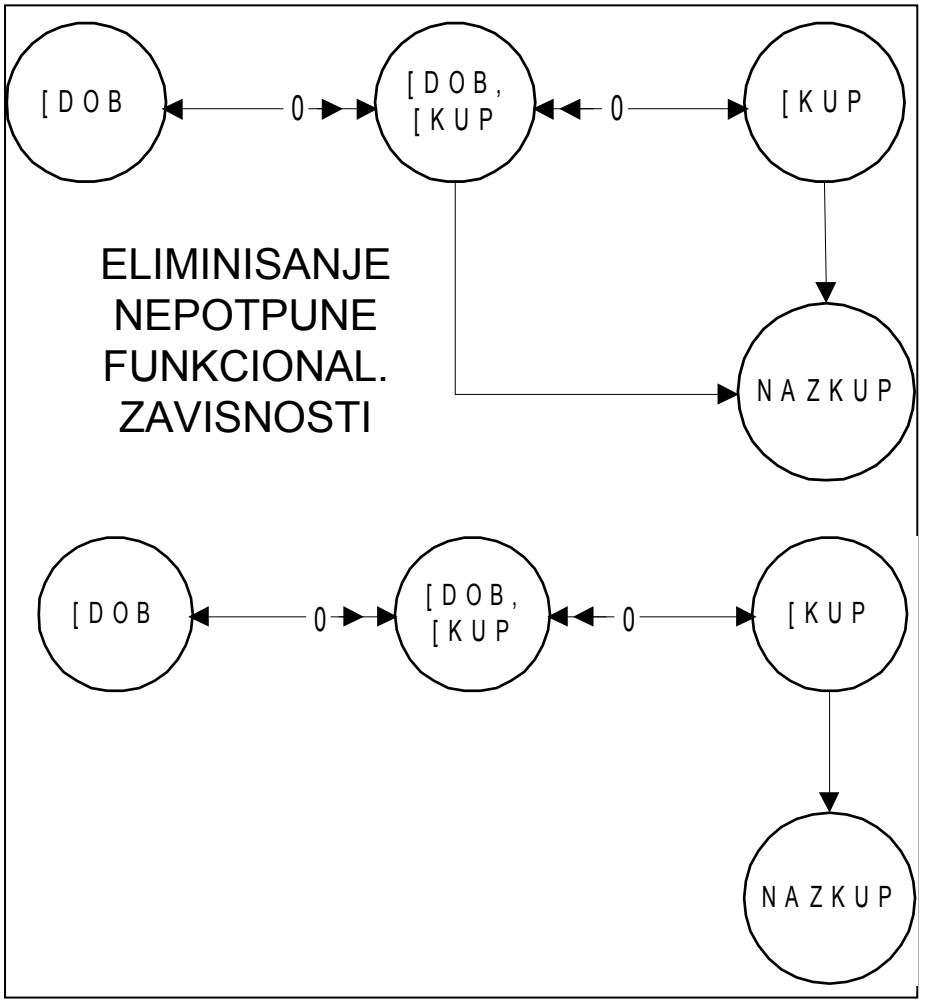
Na osnovu pravila P2 dobija se za preostale veze

PROIZVODI([IFOD,[IFPRO)
STRUKT([IFPRONAD, [IFPROPOD, UGRKOL)


A
B
C
A
B
C

SINTEZA RELACIJA NA OSNOVU TEORIJE FUNKCIONALNIH ZAVISNOSTI





SINTEZA RELACIJA NA OSNOVU TEORIJE FUNKCIONALNIH ZAVISNOSTI

- Prethodni grafovi ukazuju na mogu}nost definisanja postupka za sintezu relacija na osnovu zadatog skupa atributa i njihovih me|usobnih funkcionalnih zavisnosti Da bi se ovaj problem formalno definisao uvedimo i slede}e definicije:
- **Zatvaranje skupa funkcionalnih zavisnosti** F je skup F^+svih funkcionalnih zavisnosti koje se mogu dedukovati iz F, preko negog skupa pravila zaklju~ivanja.
- Posmatrajmo dva skupa funkcionalnih zavisnosti F i G. Ka`e se da je skup **G prekrivanje skupa F** ako je $G^+ = F^+$, tj ako su im zatvaranja ista. U tom slu~aju se ka`e da su skupovi G i F ekvivalentni

SINTEZA RELACIJA NA OSNOVU TEORIJE FUNKCIONALNIH ZAVISNOSTI

Formalno se problem projektovanja baze podataka mo`e definisati na slede}i na~in:

Za skup funkcionalnih zavisnosti definisan modelom sistema na}i minimalno prekrivanje i implementirati ga preko odgovaraju}eg skupa relacija

- PRAVILA ZAKLJU^IVANJA ZA IZVO\ENJE ZATVARANJA FUNKCIONALNIH ZAVISNOSTI?
- ALGORITMI ZA NALA@ENJE ZATVARANJA?
- ALGORITMI ZA NALA@ENJE MINIMALNOG PREKRIVANJA?

PRAVILA ZAKLJU^IVANJA ZA IZVO\ENJE ZATVARANJA

- Pravila zaklju~ivanja su osobine funkcionalnih zavisnosti na osnovu kojih, iz jedne ili vi{e funkcionalnih zavisnosti nekog skupa, mo`emo logi~ki dedukovati neke druge.

- Armstrong-ove aksiome predstavljaju, jedan neprotivre~an i potpun skup pravila zaklju~ivanja. Neprotivre~an, u smislu da se iz njega mogu dedukovati samo va`e}e funkcionalne zavisnosti, a potpun, u smislu da se na osnovu njega mogu dedukovati sve funkcionalne zavisnosti.

ARMSTRONG-ove AKSIOME

Oznaka U se nadalje koristi kao oznaka za skup atributa neke relacije.

A1. REFLEKSIVNOST. Ako je $Y \subseteq X \subseteq U$, tada va`i X -> Y. Ovo pravilo generi{e trivijalne funkcionalne zavisnosti. Trivijalne funkcionalne zavisnosti definisane su skupom atributa U neke relacije, a ne samim funkcionalnim zavisnostima koje u relaciji va`e.

A2. PRO[IRENJE. Ako va`i X -> Y i ako je $Z \subseteq U$, tada va`i i XZ -> YZ. (Oznaka XZ je skra}ena oznaka za X U Z).

A3. TRANZITIVNOST. Ako postoje funkcionalne zavisnosti X -> Y i Y -> Z, tada postoji i X -> Z.

ARMSTRONG-ove AKSIOME

Iz Armstrongovih aksioma mogu se izvesti i slede}a korisna dodatna pravila zaklju~ivanja:

D1.ADITIVNOST (pravilo unije). Ako postoje funkcionalne zavisnosti X -> Y i X -> Z, tada postoji i Y -> YZ

D2. PSEUDOTRANZITIVNOST. Ako postoje funkcionalne zavisnosti X -> Y i YW -> Z, tada postoji i funkcionalna zavisnost XW -> Z.

D3.DISTRIBUTIVNOST(pravilo dekompozicije). Ako postoji funkcionalna zavisnost X -> YZ, tada va`i i X -> Y i X -> Z.

USLOVI DA JE H MINIMALNI PREKRIVA^ F

1. Desne strane funkcionalnih zavisnosti u H su pojedina~ni atributi.
2. Za svaku funkciju f: X ---> A iz F, ako je ispunjeno

   $[F - \{X \text{ ---> } A\}]^+ = F^+$, tada $f \notin H$. (Elimini{e redundantne funkcionalne zavisnosti.)
3. Ni za jedno X ---> A iz H, za bilo koji pravi podskup Z od X ($Z \subset X$), [H - {X ---> A} U {Z ---> A}] nije ekvivalentno sa H. (Ne postoji nepotreban atribut na levoj strani)
4. $F^+ = H^+$, odnosno zatvara~i skupa funkcionalnih zavisnosti i njegovog minimalnog prekriva~a su jednaki. (Garantuje da su sve fukcionalne zavisnosti sa~uvane u minimalnom prekriva~u.)

BERNSTEIN-ov ALGORITAM ZA NALA@ENJE MINIMALNOG PREKRIVANJA

0. Na osnovu pravila dekompozicije, svesti skup funkcionalnih zavisnosti F na skup funkcionalnih zavisnosti F1 ~ije su desne strane pojedina~ni atributi.
1. Eliminisanje nepotrebnih atributa. Elimini{i nepotrebne atribute iz F1 i formiraj skup funkcionalnih zavisnosti G koji je ekvivalentan sa F1 ($G^{+} = F1^{+}$).
2. Nala`enje neredundantnog prekriva~a. Na|i neredundantni prekriva~ H skupa funkcionalnih zavisnosti G.
3. Podela i grupisanje. Podeli skup funkcionalnih zavisnosti u H u grupe H_k tako da u svakoj grupi leve strane budu identi~ne.

4. Spajanje ekvivalentnih klju~eva. Za svaki par grupa H_i i H_j , sa levim stranama X i Y spoji H_i i H_j zajedno ako postoji bijekcija X <-----> Y u H. Za svako A $\in$ Y izbaci X ---> A iz H. Za svako B $\in$ X izbaci Y ----> B iz H. Formiraj skup J u koji se stavljaju zavisnosti X ---> Y i Y -- -> X za svaku bijekciju grupa.

5. Eliminisanje novodobijenih tranzitivnih zavisnosti posle koraka 4. Na|i prekriva~ H' $\subseteq$ H tako da je (H', J)$^+$ = (H , J)$^+$ i da nijedan pravi podskup od H' nema tu osobinu.

6. Konstruisanje relacija. Za svaku grupu H_k konstrui{i jednu relaciju sa svim atributima koji se pojavljuju u grupi. Leva strana funkcionalnih zavisnosti u grupi je klju~ relacije.

BERNSTEIN-ov ALGORITAM ZA NALA@ENJE MINIMALNOG PREKRIVANJA

- Nala`enje minimalnog prekriva~a nije jednozna~an zadatak, odnosno da jedan skup funkcionalnih zavisnosti ima vi{e minimalnih prekriva~a.

- Osnovni problem u gornjem algoritmu je nala`enje minimalnog prekriva~a i njegovog zatvara~a $H^+$. Algoritam nala`enja minimalnog prekriva~a skupa funkcionalnih zavisnosti F mo`e se realizovati tako {to }e se, postepeno, eliminisati iz skupa F one funkcionalne zavisnosti $f_i$ za koje va`i $(F - f_i)^+ = F^+$. To zna~i da se algoritam nala`enja minimalnog prekriva~a svodi na algoritam nala`enja zatvara~a.

ALGORITAM ZA NALA@ENJE ZATVARA^A

- Nala`enje zatvara~a $F^+$ skupa funkcionalnih zavisnosti F direktnom primenom Armstrongovih aksioma je eksponencijalne slo`enosti. Zbog toga se primenjuje postupak nala`enja tzv. "**Zatvara~a skupa atributa**": Neka je F skup funkcionalnih zavisnosti nad skupom atributa U i neka je $X \subseteq U$. Tada je $X^+$, zatvara~ podskupa X u odnosu na F, skup atributa A takav da se X ---> A mo`e dedukovati iz F pomo}u Armstrongovih aksioma.
- Mo`e se dokazati slede}a lema: Neka funkcionalna zavisnost X ---> Y mo`e se dedukovati iz F pomo}u Armstrongovih aksioma, ako i samo ako je $Y \subseteq X^+$, a X^+ je zatvara~ skupa X a odnosu na F.

ALGORITAM ZA NALA@ENJE ZATVARA^A SKUPA ATRIBUTA

Ulaz: Skup atributa U, skup funkcionalnih zavisnosti F nad U i podskup $X \subseteq U$.

Izlaz: X^+, zatvara~ X u odnosu na F.

Postupak: Sra~unava se niz skupova atributa $X^{(0)}$, $X^{(1)}$, ..., preko pravila:

1. $X^{(0)} = X$
2. $X^{(i,1)} = X^{(i)}$ U V, gde je V takav skup atributa A da postoji neka funkcija Y ---> Z, gde je $A \in Z$, a $Y \subseteq X^{(i)}$. Kako se $X^{(j,1)}$ sra~unava samo na osnovu $X^{(j)}$ i kako je skup U kona~an, postupak se zavr{ava kada postane $X^{(i,1)} = X^{(i)}$.

PRIMER BERNSTEIN-ovog ALGORITMA

Dat je skup atributa U = {A, B, C, D, X1, X2}

Dat je skup funkcionalnih zavisnosti:

G = {X1X2 -> AD, CD -> X1X2, AX1 -> B, BX2-> C,
C -> A}

Korak 0:

f1: X1X2 ---> A

f2: X1X2 ---> D

f3: CD ---> X1

f4: CD ---> X2

f5: AX1 ---> B

f6: BX2 ---> C

f7: C ---> A

PRIMER BERNSTEIN-ovog ALGORITMA

Korak 1: Nema nepotrebnih atributa.

Korak 2: Proverava se redom da li se svaka fi mo`e dedukovati iz tranzitivnog zatvara~a preostalih, odnosno da li je

$fi \in [F - fi]^+$.

Za funkciju f1, na primer, to je ekvivalentno proveri da li je $A \in X1X2$ u odnosu na F - f1:

$X^{(0)} = X1X2$ $X^{(1)} = X1X2$ (zbog f2)

Prema tome X1X2 ---> A ne mo`e se dedukovati iz $(F - f1)^+$ pa zbog toga pripada neredundantnom prekriva~u. Isto se mo`e zaklju~iti i za preostale fi iz F.

PRIMER BERNSTEIN-ovog ALGORITMA

Korak 3: Podela i grupisanje. Vr{i se grupisanje funkcionalnih zavisnosti u slede}e grupe:

H1={f1, f2}, H2={f3, f4}, H3={f5}, H4={f6}, H5={f7}

Korak 4:Spajanje ekvivalentnih klju~eva. Proverava se da li izme|u levih strana funkcionalnih zavisnosti nekih grupa postoji bijekcija. Bijekcija postoji ako je $N \in M^+$, a $M \in N^+$.

Sra~unajmo $X1X2'$ u odnosu na skup H koji je u ovom slu~aju jednak skupu F:

$X^{(0)}$ = X1X2 $X^{(1)}$ = X1X2AD (zbog f1 i f2)

$X^{(2)}$ = X1X2ADB (zbog f5)

$X^{(3)}$ = X1X2ADBC (zbog f3, f4, f5, f6)

$X1X2'$ = X1X2ADBC

PRIMER BERNSTEIN-ovog ALGORITMA

Sra~unajmo CD^+ u odnosu na skup H:

$X^{(0)} = CD$

$X^{(1)} = CDX1X2A$ (zbog f3, f4 i f7)

$X^{(2)} = CDX1X2AB$ (zbog f5)

$X^{(3)} = CDX1X2AB$ (zbog f6)

$CD^+ = X1X2ADBC$

- Prema tome postoji bijekcija X1X2 <---> CD. Zbog toga se zavisnosti X1X2---> A, X1X2 ---> B, CD ---> X1, CD --->X2 spajaju u grupu (X1, X2, C, D, A). U skup J treba dodati gornju bijekciju, odnosno ~etiri zavisnosti: X1X2 ---> C, X1X2 ---> D, CD ---> X1 i CD ---> X2. Kako zadnje tri ve} postoje u F, dodaje se samo f8: X1X2 ---> C.

PRIMER BERNSTEIN-ovog ALGORITMA

Korak 5. Eliminisanje novodobijenih tranzitivnih zavisnosti. Novi skup funkcionalnih zavisnosti H , J obuhvata zavisnosti f1 do f8. Eliminisanje tranzitivnih zavisnosti se radi ponovo preko Koraka 2, uklju~uju}i i f8. Prvo se ispituje da li je sada f1 redundantna. Sra~unava se $X1X2'$ u odnosu na H , J - f1.

$X^{(0)} = X1X2$

$X^{(1)} = X1X2DC$ (na osnovu f2 i f8)

$X^{(2)} = X1X2DCA$ (na osnovu f7)

$X1X2' = X1X2DCA$

Kako je $A \in X1X2'$ funkcionalna zavisnost f1 je redundantna. Ostale funkcionalne zavisnosti (koje bi proveravali na isti na~in) nisu.

PRIMER BERNSTEIN-ovog ALGORITMA

Korak 6. Konstrui{u se slede}e relacije:

R1(X1, X2, C, D)

R2(A, X1, B)

R3(B, X2, C)

R4(C, A)

FIZI^KO PROJEKTOVANJE RELACIONIH BAZA PODATAKA

- FIZI^KO PROJEKTOVANJE BAZA PODATAKA SE VR[I NAKON POTPUNOG LOGI^KOG PROJEKTOVANJA, NA OSNOVU JASNE LOGI^KE STRUKTURE BAZE PODATAKA.
- FIZI^KO PROJEKTOVANJE BAZA PODATAKA VEOMA JE ZAVISNO OD KONKRETNOG SUBP.
- NIJE UOBI^AJENO DA SE VR[E NEKI DETALJNI “FIZI^KI PRORA^UNI”, RADIJE SE PRIMENJUJU EKSPERTSKA ZNANJA I KASNIJE PODE[AVANJE FIZI^KE STRUKTURE.

FIZI^KO PROJEKTOVANJE RELACIONIH BAZA PODATAKA - KORACI:

1. PRILAGO\AVANJE LOGI^KE STRUKTURE KONKRETNOM SKUPU APLIKACIJA - DENORMALIZACIJA
2. DISTRIBUCIJA BAZE PODATAKA - RAZLI^ITE “KLIJENT- SERVER” ARHITEKTURE
3. “KLASTEROVANJE” - PODACI KOJI SE ZAJEDNO KORISTE TREBA DA BUDU FIZI^KI BLISKI
4. ODRE\IVANJE METODA PRISTUPA (INDEKSIRANJE I EVENTUALNO “HE[ING”)

Upravljanje izvršenjem transakcija
Oporavak baze podataka
Sigurnost baze podataka
Katalog baze podataka

Upravljanje izvršenjem transakcija

Upravljanje izvršenjem transakcija

Baza podataka je zajednički resurs koji istovremeno (konkurentno) koristi veći broj programa. Pri ovakvom korišćenju baze podataka može doći do mnogih neželjenih efekata kao što su, na primer:

- otkaz sistema u toku izvršenja nekog programa koji može da ostavi bazu podataka u nekonzistentnom stanju,
- neželjena interferencija dva ili više programa nad podacima za koje istovremeno konkurišu, može, takođe, da dovede bazu podataka u nekonzistentno stanje.

Osnovni cilj baze podataka je da omogući efikasnu obradu transakcija. ***Transakcija*** je jedno izvršenje neke "logičke jedinice posla", jedno izvršenje neke logičke celine jednog programa, ili jedno izvršenje celog programa.

Problemi u izvr{avanju transakcija

Otkaz sistema u toku obrade transakcije. Prenos novca (N dinara) komitenta banke sa računa X na račun Y.

Gubljenje rezultata ažuriranja. Transakcija A podiže, a transakcija B ulaže novac na isti račun.

Problem nekorektne analize podataka. Ovim nazivom se definiše skup problema do kojih dolazi kada, za vreme nekog "sračunavanja" koje se obavlja u jednoj transakciji, druga promeni vrednost argument koji je prva već obradila.

Problemi u izvr{avanju transakcija

Otkaz sistema u toku obrade transakcije. Prenos novca (N dinara) komitenta banke sa računa X na račun Y.

Gubljenje rezultata ažuriranja. Transakcija A podiže, a transakcija B ulaže novac na isti račun.

Problem nekorektne analize podataka. Ovim nazivom se definiše skup problema do kojih dolazi kada, za vreme nekog "sračunavanja" koje se obavlja u jednoj transakciji, druga promeni vrednost argument koji je prva već obradila.

Transakcije

Transakcija mora da poseduje sledeći skup osobina (ACID osobine):

1. ***Atomnost*** (Atomicity). Transakcija, kao "logička jedinica posla", mora predstavljati atomski skup aktivnosti.

2. ***Konzistentnost*** (Consistency). Izvršenje transakcije treba da prevede bazu podataka iz jednog u drugo konzistentno stanje.

3. ***Izolacija*** (Isolation). Transakcija ne treba svoje promene baze podataka da učini vidljivim drugim transakcijama pre nego što se ona okonča, odnosno promene potvrde u bazi podataka.

4. ***Trajnost*** (Durability). Kada su, u bazi podataka, potvrđene promene koje je izvršila neka transakcija, ove promene se više ne mogu izgubiti.

Transakcije

Da bi se ACID osobine transakcije obezbedile skup instrukcija koje predstavljaju transakciju počinje, po pravilu, sa specijalnom instrukcijom BEGIN TRANSACTION,

a završava se bilo sa instrukcijom COMMIT (potvrđuju se promene u bazi podataka, ako su sve instrukcije transakcije uspešno izvršene)

ili instrukcijom ROLLBACK (poništavaju se promene u bazi, ako sve instrukcije nisu uspešno završene).

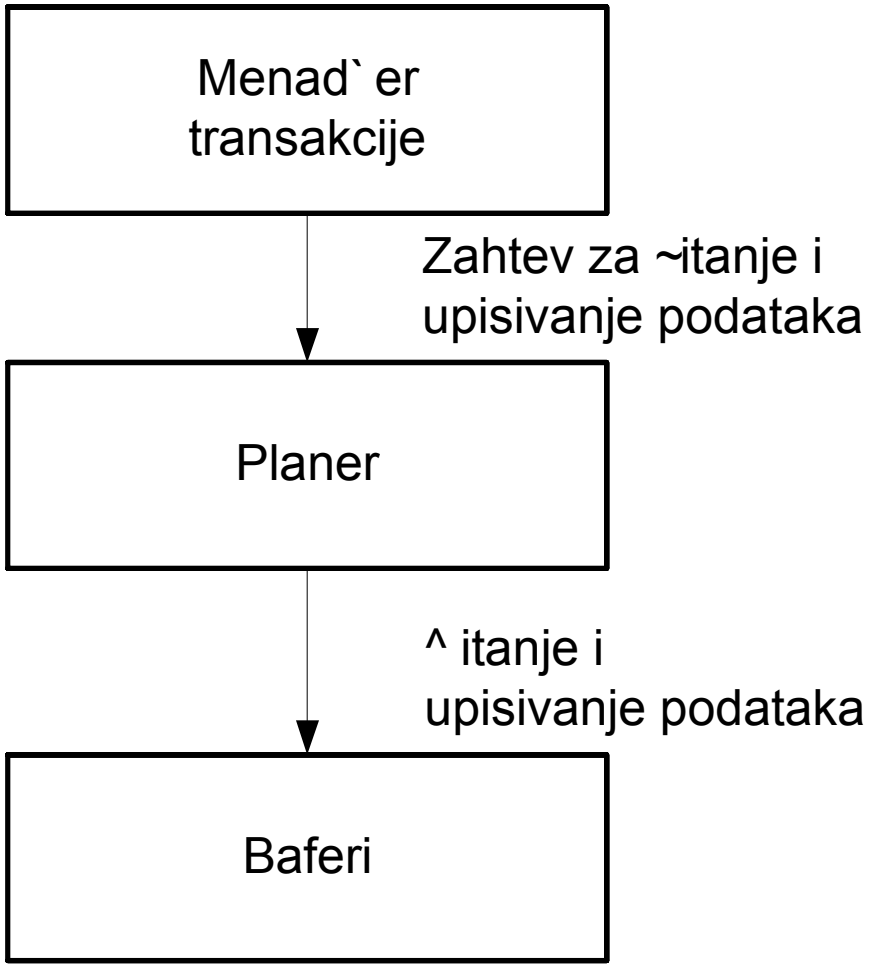
Konkurentna (uporedna) obrada transakcija

Komponenta SUBP-a koja vodi računa o redosledu izvršavanja akcija nad bazom podataka u skupu transakcija koje se konkurentno izvršavaju, naziva se ***Planer*** (***Scheduler***).

Komponenta koja upravlja celokupnim izvršenjem transakcija naziva se ***Menadžer transakcije*** (***Transaction manager***).

Na slede{oj slici prikazana je i razmena zahteva i podataka između ovih komponenti i bafera.

Konkurentna (uporedna) obrada transakcija



Planer izvršenja skupa transakcija

Konkurentna (uporedna) obrada transakcija

Rezultat serijskog izvršenja skupa transakcija u bilo kom redosledu se smatra korektnim.

Koristeći ovaj stav, možemo da uvedemo koncept serijabilnosti ili linearnosti izvršenja skupa transakcija.

Uporedno izvršavanje skupa transakcija je ***serijabilno (linearno)*** ako proizvodi isti rezultat kao i neko serijsko izvršenje istog skupa transakcija.

Usvaja se da je skup uporednih transakcija izvršen korektno, ako i samo ako je taj skup serijabilan.

Protokoli za ostvarivanje serijabilnosti izvršenja skupa transakcija

Različiti protokoli serijabilnosti izvršenja skupa transakcija zasnivaju se na različitim načinima utvrđivanja serijabilnosti i različitim postupcima ostvarivanja serijabilnosti.

View-serijabilnost

Za definisanje tzv. view-ekvivalnencije dva izvršenja skupa transakcija i view-serijabilnosti datog izvršenja skupa transakcija koriste se sledeći pojmovi:

- Ako u nekom izvršenju skupa transakcija neko upisivanje $w_j(X)$ prethodi nekom čitanju istog elementa baze $r_i(X)$ i nema nijedne druge operacije upisivanja $w_k(X)$ između njih, kaže se da "$r_i(X)$ čita iz $w_j(X)$", odnosno da između $r_i(X)$ i $w_j(X)$ postoji "čita iz" odnos.
- Operacija $w_i(X)$ se zove "krajnje upisivanje" , ako je to poslednje upisivanje elementa X u bazu.

View-serijabilnost

"View" ekivivalnecija i "view" serijabilnost se definišu na sledeći način:

- Dva izvršenja skupa transakcija su ***view-ekvivalenetna*** ($S_i \approx_V S_j$) ako poseduju iste "čita iz" odnose i ista "krajnja upisivanja".

Izvršenje skupa transakcija je ***view- serijabilno*** ako je view-ekvivalentno sa nekim serijskim izvršenjem.

Zbog kompleksnosti utvrdjivanja, view-serijabilnost se prakti~no ne koristi.

Konflikt-serijabilnost

Jedan od praktičnijih pristupa utvrđivanju serijabilnosti izvršenja skupa transakcija se ostvaruje preko definisanja koncepta konflikt-serijabilnosti. Pod ***konfliktom*** se ovde podrazumeva situacija u kojoj izmena redosleda dve operacije u izvršenju dovodi do izmene efekata na bazu barem jedne od transakcija iz posmatranog izvršenja.

Ako pretpostavimo da transakcije T_i i T_j pripadaju nekom izvršenju, tada sledeće parovi operacija neće biti u konfliktu:

1. $r_i(X)$, $r_j(Y)$ nije konflikt čak i kada je X =Y jer ni jedna ni druga operacija ne menjaju stanje baze podatka.
2. $r_i(X)$, $w_j(Y)$ očigledno nije konflikt, pod pretpostavkom da je $X \neq Y$.
3. $w_i(X)$, $r_j(Y)$ nije konflikt, pod pretpostavkom da je $X \neq Y$.
4. $w_i(X)$,$w_j(Y)$ nije konflikt.

Konflikt-serijabilnost

Planer ne može da izmeni redosled operacija u okviru jedne transakcije, jer se time menja semantika transakcije. Najopštije, dve susedne operacije različitih transakcija mogu zameniti mesta ako nije zadovoljen jedan od uslova:

1. Operacije se obavljaju nad istim elementom baze podataka i
2. Barem jedna od njih je upisivanje.

Dva izvršenja skupa transakcija su ***konflik-ekvivalentna*** ako se jedan u drugi mogu transformisati nekonfliktnim izmenama mesta susednih operacija.

Izvršenje skupa transakcija je ***konflikt-serijabilno*** ako je konflikt-ekvivalentno sa nekim serijskim izvršenjem.

Protokoli zaključavanja

Proveru serijabilnosti i preduzimanje odgovarajućih akcija planer izvršenja teško može da obavi u realnom vremenu. Zbog toga se serijabilnosti izvršavanja skupa transakcija najčešće ostvaruje "forsirano" primenjujući mehanizam ***zaključavanja (locking).***

Ekskluzivno zaključavanje (exclusive (XL) lock ili write lock). Ako neka transakcija postavi ekskluzivni lokot na objekat baze podataka, nijedna druga transakcija ne može na taj objekat da postavi bilo koji drugi lokot.

Deljivo zaključavanje (shared (SL) lock ili read lock). Ako neka transakcija postavi deljivi lokot na objekat baze podataka, neka druga transakcija takođe može da postavi deljivi lokot na isti objekat baze, ali nijedna druga ne može da postavi ekskluzivni lokot na taj objekat.

Matrica kompatibilnosti tipova zaključavanja

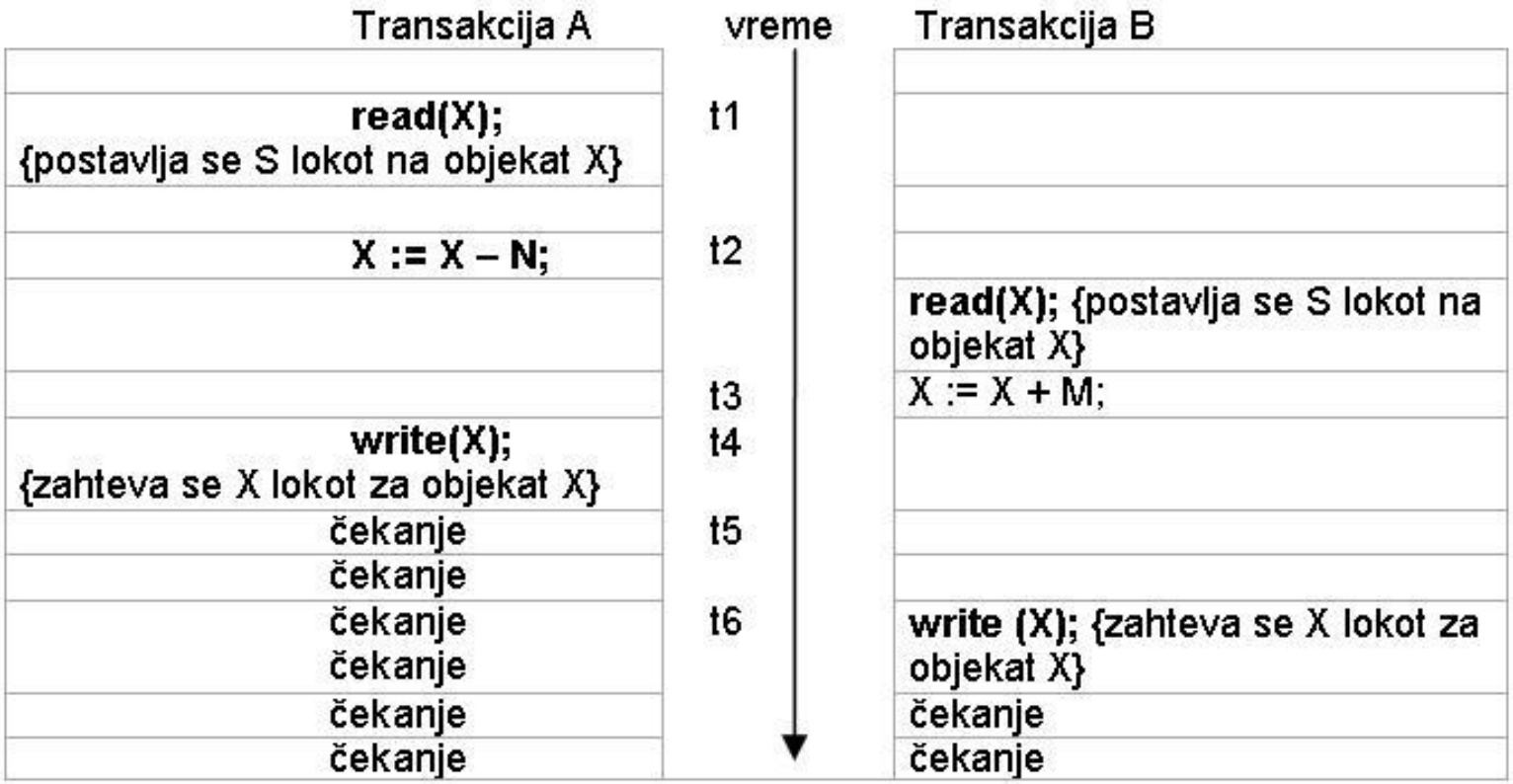
B / A	XL	SL	-
XL	NE	NE	DA
SL	NE	DA	DA
-	DA	DA	DA

Imajući u vidu ekskluzivni i deljivi lokot, ***protokol zaključavanja*** koji bi mogao da reši prikazane probleme može se iskazati na sledeći način:

1. Transakcija koja želi da pročita neki objekat baze podataka (n-torku, na primer) mora prvo da postavi deljivi lokot na taj objekat;
2. Transakcija koja želi da ažurira neki objekat baze podataka mora prvo da postavi ekskluzivni lokot na taj objekat. Ako je ta transakcije ranije postavila S lokot, ona treba da taj lokot transformiše u X lokot;
3. Ako transakcija nije uspela da postavi lokot na željeni objekat baze podataka, zbog toga što neka druga transakcija već drži nekompatibilan lokot nad posmatranim objektom, tada ona prelazi u stanje čekanja;
4. Transakcija oslobađa E lokot obavezno, a po pravilu i S lokot na kraju, sa COMMIT ili ROLLBACK naredbom.

- I X i S lokot se postavljaju implicitno, zajedno sa operacijama ažuriranja, odnosno čitanja podataka baze.

Mrtvi čvor

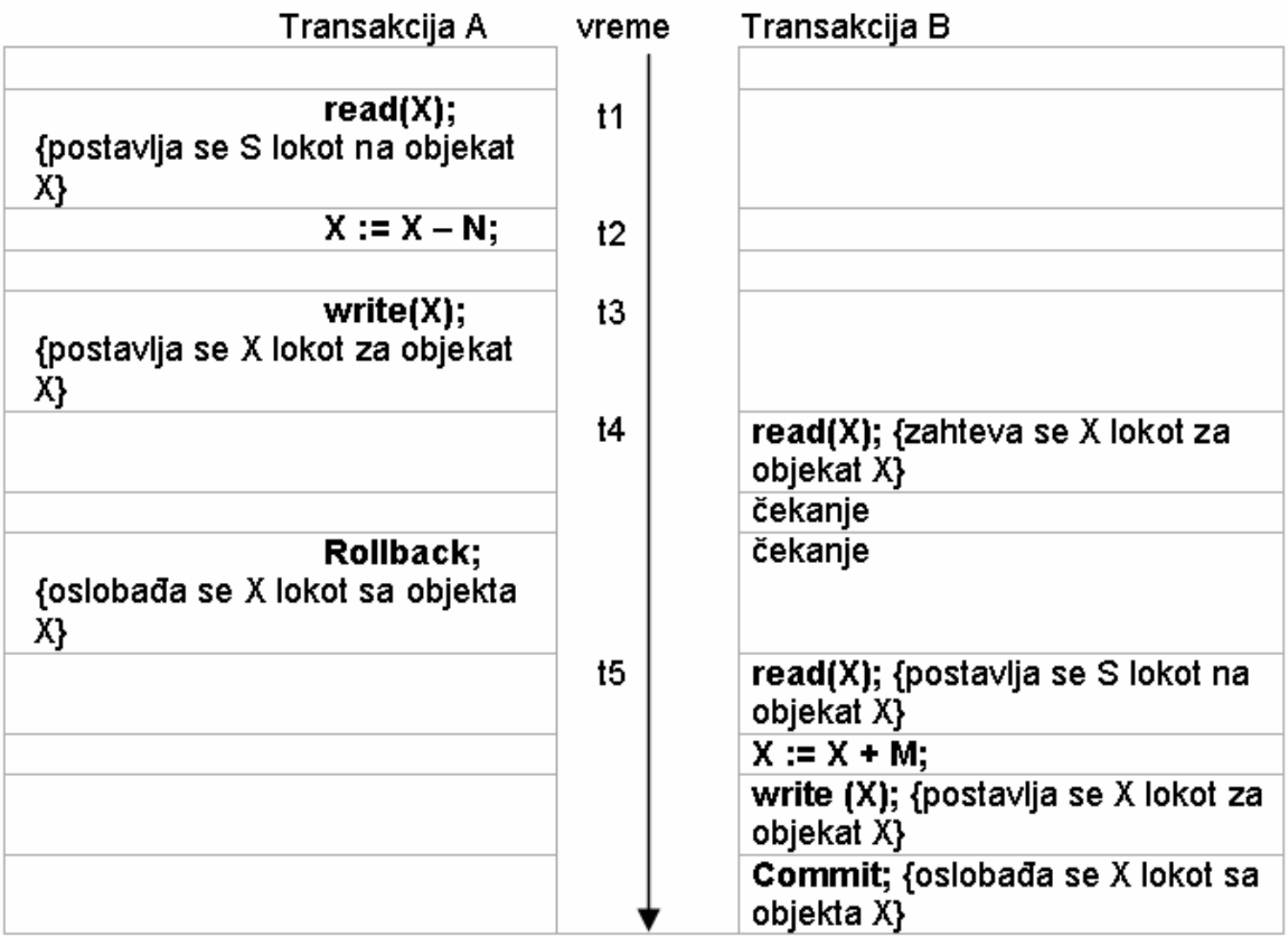


Slika 11.8. "Mrtvi čvor" umesto gubljenje rezultata ažuriranja

Dvofazni protokol zaključavanja

- ***Pre nego što operiše sa nekim objektom baze podataka, transakcija mora da postavi lokot na njega;***
- ***Posle oslobađanja nekog lokota, transakcija ne može više postaviti lokot ni na jedan objekat baze.***

- ***Može se dokazati teorema da ako u nekom skupu sve transakcije poštuju dvofazni protokol zaključavanja, taj skup se uvek serijabilno izvršava.***



Slika 11.9. Rešenje problema nepotvrđenih promena

Vremensko označavanje (Timestemping)

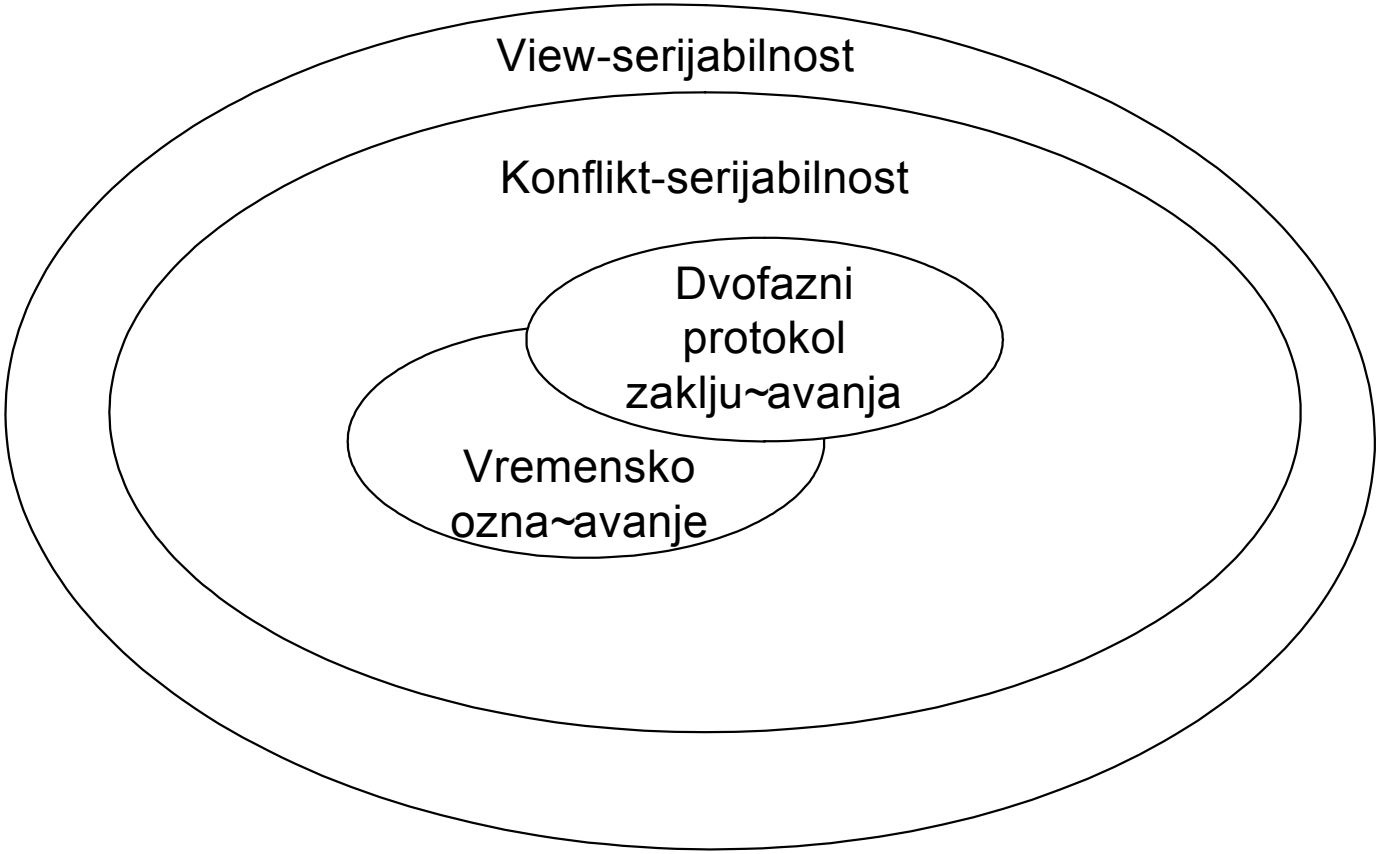
Svakom objektu baze podataka pridružuju se dve oznake:

- RMAX, najveći identifikator transakcije koja je uspešno izvršila čitanje posmatranog objekta i
- UMAX, najveći identifikator transakcije koja je uspešno izvršila ažuriranje posmatranog objekta.

Algoritam vremenskog označavanja

```
read(R)
 if t >= UMAX
then { operacija se prihvata}
RMAX := MAX(t, RMAX);
else { konflikt}
restart T;
write(R) { zajedno sa COMMIT}
 if t >= UMAX and t >= RMAX
then { operacija se prihvata}
UMAX := t;
else { konflikt}
restart T;
```

Nivoi izolovanosti transakcija



Načini narušavanja serijabilnosti izvršenja skupa transakcija:

- ***Prljavo čitanje ("dirty read").*** Ovako izvršenje skupa transakcija je već ranije objašnjeno. Ako transakcija T1 ažurira neki podatak u bazi, zatim transakcija T2 pročita taj podatak, a nakon toga se transakcija T1 završi sa ROLLBACK, tada je očigledno transakcija T2 pročitala nepostojeći podatak, odnosno izvršila "prljavo čitanje".

- ***Neponovljivo čitanje (Nonrepeatable read).*** U jednoj transakciji, ponovno čitanje istih podataka mora dati isti rezultat. Naprotiv, ako transakcija T1 pročita neki podatak baze, zatim ga transakcija T2 promeni, ponovno čitanje istog podatka u transakciji T1 neće dati isti rezultat.

- ***Fantomsko čitanje (Fantoms).*** Ako transakcija T1 preuzme upitom skup n-torki koje zadovoljavaju zadati uslov, a posle toga transakcija T2 doda novu n-torku koja zadovoljava isti uslov, novo izvršenje istog upita u transakciji T1 sadržaće i novu "fantomsku" n-torku.

Odnos nivoa izolovanosti i ponašanje izvršenja skupa transakcija

Ponašanje / Nivo izolovanosti	PRLJAVO ČITANJE	NEPONOVLJIVO ČITANJE	FANTOMSKO ČITANJE
READ UNCOMMITTED	DA	DA	DA
READ COMMITTED	NE	DA	DA
REPEATABLE READ	NE	NE	DA
SERIALIZABLE	NE	NE	NE

Upravljanje zaključavanjem

Osnovu upravljanja zaključavanjem čine tri procedure ***menadžera lokota***:

(1) r_lock – postavljanje ekskluzivnog lokota,

(2) w_lock – postavljanje deljivog lokota i

(3) unlock – oslobađanje zaključanog elementa baze podataka.

```
r_lock (T,X, errcode, timeout)
w_lock (T,X, errcode, timeout)
ulock (T,X)
```

T - identifikator transakcije,

X – elemenat baze podataka koji se zaključava,

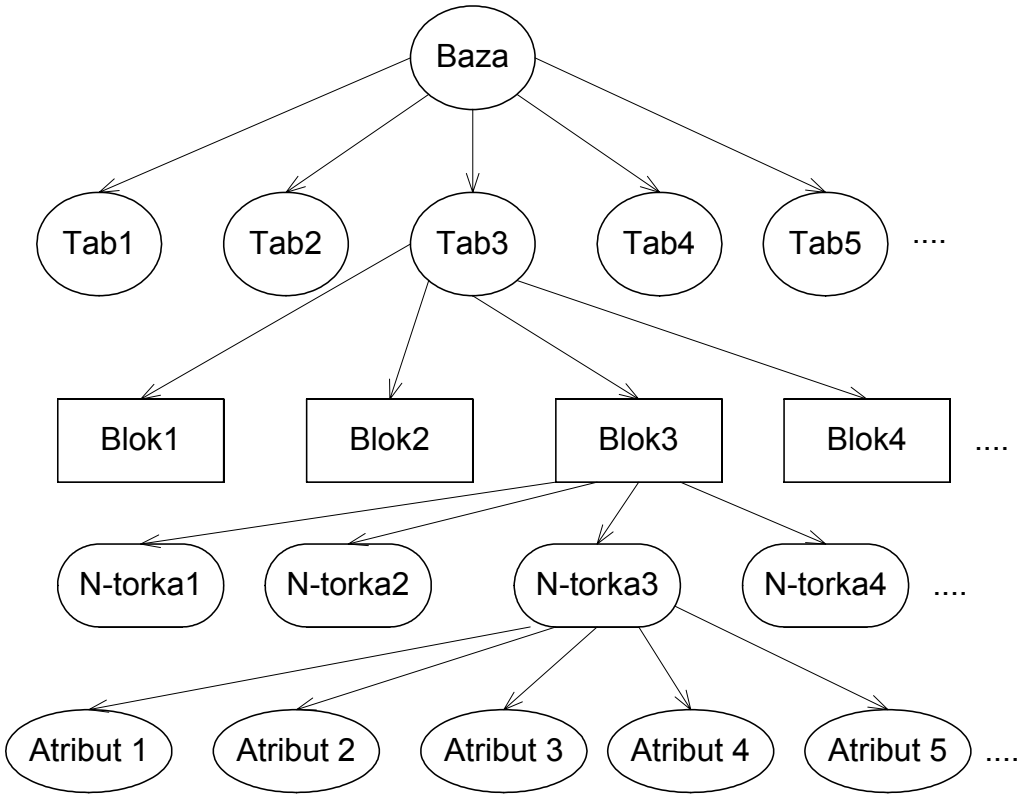
errcode – daje kod statusa zahteva (0-zahtev zadovoljen, ≠0 - nezadovoljen)

timeout – maksimalni interval vremena koji transakcija “čeka” da bi dobila lokot nad objektom X.

Granularnost zaključavanja i hijerarhijsko zaključavanje

Pored ekskluzivnog lokota (XL) i deljivog lokota (SL) za ovaj protokol se definišu i sledeće vrste lokota:

- **ISL** – nameravani deljivi lokot: transakcija namerava da postavi deljivi lokotna elemenat baze podataka;
- **IXL** - nameravani ekskluzivni lokot: transakcija namerava da postavi ekskluzivni lokot na elementa baze podataka;
- **SIXL** – nameravano deljivo-ekskluzivno zaključavanje: transakcija namerava da postavi deljivi lokot na elemenat baze podataka X i ekskluzivni lokot na neki elemenat na nižem nivou granularnosti koji je deo elementa X.



Granularnost zaključavanja i hijerarhijsko zaključavanje

U hijerarhijskom protokolu zaključavanje se obavlja po sledećim pravilima:

1. Lokoti se zaključavaju počev od vrha, niz hijerarhiju elemenata;
2. Lokoti se oslobađaju počev od elementa sa najmanjom glanularnošću, naviše, uz hijerarhiju elemenata.
3. Da bi se postavio SL ili ISL lokot na neki elemenat X, transakcija mora prethodno da postavi ISL ili IXL lokot na element “roditelj” elementu X u stablu;
4. Da bi zahtevala IXL, XI ili SIXL lokot na elemenat X, transakcija mora prethodno da postavi SIXL ili IXL lokot na element “roditelj” elementu X u stablu;
5. Matrica kompatibilnosti ovi lokota za transakcije A i B data je na slici

B A	ISL	IXL	SL	SIXL	XL
ISL	DA	DA	DA	DA	NE
IXL	DA	DA	NE	NE	NE
SL	DA	NE	DA	NE	NE
SIXL	DA	NE	NE	NE	NE
XL	NE	NE	NE	NE	NE

Hijerarhijsko zaključavanje indeksnih struktura

Ako bi se na B-stablima primenjivale standardne vrste lokota (ekskluzivni i deljivi) i standardni protokoli zaključavanja (Dvofazni protokol zaključavanja) tako da se celo B-stablo zaključava, jer se pretpostavlja da će biti potrebno i njegovo ažuriranje, paralelizam u izvršenju skupa transakcija bi se gotovo potpuno eliminisao.

Zbog toga se definišu specijalni protokoli za hijerarhijske, odnosno indeksne strukture. Jedan takav protokol je:

- Transakcija može postaviti svoj prvi lokot na bilo koji čvor stabla;
- Sledeći lokoti se mogu dobiti samo ako je data transakcija postavila lokot i na čvor “roditelj”;
- Lokoti se mogu osloboditi u bilo kom trenutku;
- Transakcija ne može ponovo da postavi lokot na elemenat sa koga je skinula lokot, čak i ako još uvek drži lokot na čvoru “roditelju”.

“Živi” i “mrtvi” lokoti

Kao što je u nekim od prethodnih primera pokazano, u toku izvršenja skupa transakcija, moguće je da dve ili više transakcija formiraju “mrtvi lokot”, odnosno da lokoti koje su one postavile na objekte baze podataka sve njih dovode u stanje čekanja.

Pored pojma “mrtvog lokota”, ponekad se definiše i pojam “živog lokota”. To je situacija u kojoj je neka transakcija stalno u stanju čekanja na neki objekat baze podataka zbog toga što druge transakcije uvek pre nje postave lokot na taj objekat. Problem “živog lokota” jednostavno se rešava uvođenjem nekog redosleda zaključavanja objekta (na primer FIFO).

“Živi” i “mrtvi” lokoti - 2

Za razrešavanje problema mrtvih lokota koriste se tri tehnike:

1. ***Prekidanje transakcije posle isteka intervala vremena*** predviđenog za čekanje na lokot za neki elemenat. Ovo pravilo koristi parametar “timeout” koji menadžer lokota dodeljuje transakciji pri pokušaju zaključavanja nekog objekta. Kada ovo vreme istekne transakcija se poništava i ponovo startuje, očekujući da tada neće proizvesti “mrtvi lokot”.

“Živi” i “mrtvi” lokoti - 3

2. ***Prevencija lokota***. Mogu se definisati različiti protokoli koji će sprečiti da do “mrtvog čvora” dođe. Jedan od takvih protokola se zasniva na uređenju elemenata baze podataka. Na primer, blokovi se mogu urediti po njihovim adresama na spoljnoj memoriji (A1 < A2 < A3 < ...< An). Ako se zahteva da svaka transakcija zaključa elemente u njihovom definisanom redosledu, očigledno je da do “mrtvog čvora” ne može da dođe. Na primer ako T1 zahteva A1 pa A2, transakcija T2 neće moći da traži lokote u redosledu A2 pa A1, koji bi doveo do “mrtvog čvora”, jer ovakvo zaključavanje nije po definisanom protokolu. Drugi protokol prevencije zasniva se na dodeljivanju, za ovaj protokol specifične, vremenske oznake transakciji. Ako je transakcija koja zahteva lokot starija (manja vremenska oznaka) od transakcije koja drži lokot na elementu baze, dozvoljava joj se čekanje na dobijanje lokota, u protivnom se prekida. Moguće je primeniti i obrnuti uslov prekidanja.

“Živi” i “mrtvi” lokoti - 4

3. ***Detekcija “mrtvog čvora*.** Dozvoljava da “mrtvog lokota” dođe, pa se tada neka od transakcija koja ga je izazvala "ubije", njeni efekti na bazu podataka poništе, a ona sama, eventualno, ponovo startuje, nadajući se da tada neće izazvati mrtvi lokot. Za otkrivanje “mrtvih lokota” u sistemu koristi se tzv. ***graf čekanja*** (Wait-For graf) čiji su čvorovi transakcije T u izvršenju, a grana Ti - Tj postoji ako transakcija Ti zahteva neki objekat koji je transakcija Tj zaključala. Na Slici 11.15 prikazan je jedan graf čekanja. Transakcija T1 čeka na objekat koji je zaključala transakcija T2, ova čeka na objekat koji je zaključala T3, a T3 čeka na objekat koji je zaključala T1. Ispitivanje mrtvih lokota se može vršiti bilo svaki put kada neka transakcija prelazi u stanje "čekaj", bilo periodično, ili se može smatrati da je mrtvi lokot nastao ako transakcija ništa nije uradila u predefinisanom periodu vremena. Očigledno je da se nalaženje “mrtvog lokota” svodi na nalaženje ciklusa u grafu čekanja. Kriterijumi za izbor transakcije koja će se poništiti ("žrtve") mogu biti različiti, na primer, transakcija koja je poslednja startovala ili transakcija koja zahteva najmanje lokota. "Ubijanje" transakcije i poništavanje njenih efekata oslobađa i sve lokote koje je ona postavila.

Dugačke transakcije

Prethodno diskutovane metode upravljanja izvršenjem skupa transakcija pogodne su za poslovne sisteme sa jednostavnim kratkim transakcijama, kada je prihvatljivo da jedna transakcija drži zaključanim neki elemenat baze podataka za neko relativno kratko vreme, reda veličine desetinke sekunde, sekunde ili čak i minuta, zavisno od vrste aplikacija.

Međutim, postoje aplikacije u kojima su transakcije mnogo duže, reda desetine minuta, sati ili čak nekoliko dana. Prikazani mehanizmi zaključavanja u kome bi neka transakcija ovako dugo ekskluzivno zaključala elemente baze podataka koje koristi je, očigledno, neprihvatljivo.

Primeri aplikativnih sistema sa dugim transakcijama:

- Neke transakcije u poslovnim aplikacijama (npr. ukupno stanje računa banke)
- Projektni sistemi (CAE, CAD, CASE)
- Workflow sistemi

Sigurnost baze podataka

Sigurnost baze podataka

Pod sigurnošću baze podataka podrazumeva se zaštita baze od neovlašćenog pristupa.

Osnovni koncepti sigurnosti baze podataka

Sigurnost celokupnog sistema. Jedan od problema sigurnosti je i zaštita celokupnog sistema, odnosno sprečavanje neovlašćenoj osobi da pristupi sistemu bilo sa ciljem da dobije neke informacije ili da ošteti deo ili celu bazu podataka.

Model sigurnosti baze podataka. Osnovni model sigurnosti baze podataka se može definisati preko skupa ***autorizacija*** odnosno skupa četvorki,

Autorizacija: <korisnik, objekat, operacija, uslov>

Osnovni koncepti sigurnosti baze podataka

Podsistem sigurnosti (security subsystem) ili ***podsistem autorizacije (autorization subsystem)*** je deo SUBP-a koji treba da omogući da se model sigurnosti uskladišti i da se operacije nad bazom podataka obavljaju na osnovu definisanog modela.

Diskrecioni mehanizam sigurnosti (Discretionary mechanism) podržava osnovni model u kome se pojedinim korisnicima daju specifična prava pristupa (privilegije) za različite objekte.

Mehanizam ovlašćenja (mandatory mechanism) podržava specifičan model sigurnosti u kome je, s jedne strane svakom objektu baze podataka dodeljen neki stepen “tajnosti”, a sa druge korisnicima dato pravo da pristupe objektima definisanog stepena tajnosti.

Osnovni koncepti sigurnosti baze podataka

Statističke baze podataka se, sa tačke gledišta sigurnosti, posebno analiziraju. Mehanizam sigurnosti statističkih baza mora da onemogući da se iz zbirnih podataka izvuku zaključci o vrednostima pojedinačnih podataka za neki entitet.

Enkripcija podataka. Da bi se osigurali posebno osetljivi podaci u bazi podataka primenjuje se enkripcija podataka, specifičan način kodiranja podataka koji je veoma teško "razbiti".

Registar i revizija korisnikovih akcija (Audit Trail). Neophodno je voditi poseban log koji se naziva "Registar korisnikovih akcija" i povremeno vršiti pregled (reviziju) ovog registra. Zapis u registru sadrži: (1) izvorni tekst korisnikovog zahteva, (2) identifikator terminala sa kojeg je zahtev postavljen, (3) identifikator korisnika koji je postavio zahtev, (4) datum i vreme, (5) relacije, n-torke i atributi kojima je pristupljeno, (6) novu i staru vrednost

Diskrecioni mehanizam sigurnosti

SQL standard podržava samo diskrecioni mehanizam sigurnosti preko naredbe GRANT za kreiranje autorizacije i naredbe REVOKE kojom se autorizacija ukida. Sintaksa GRANT naredbe je:

```
GRANT < lista privilegija>
ON <objekat baze>
TO < lista identifikatora korisnika>
[WITH GRANT OPTION];
```

SQL ne podržava postavljanje uslova u autorizaciji. Taj nedostatak se za tabele može rešiti prethodnim definisanjem pogleda sa datim uslovom, nad kojim se primenjuje prikazani GRANT iskaz.

Diskrecioni mehanizam sigurnosti

Privilegija data nekom korisniku se može opozvati korišćenjem naredbe REVOKE koja ima sledeću sintaksu:

```
REVOKE [GRANT OPTION FOR] <lista privilegija>
          ON <objekat baze>
          FROM<lista identifikatora    korisnika>
          <opcija>;
```

Primer

Neka je Miloš vlasnik šeme StudentskiIS koja ima sledeće tabele:

Student (BI, Ime, Starost, Semestar, Smer)

Predmet ([Pred, NazivPred, BrojČas)

Miloš daje privilegiju SELECT i INSERT Ani za tabelu Student, a Aci privilegije SELECT i UPDATE za tabelu Predmet, dozvoljavajući im da te privilegije prenesu i na druge korisnike.

```
GRANT SELECT, INSERT ON Student TO Ana
WITH GRANT OPTION;
GRANT SELECT, UPDATE ON Predmet TO Aca;
WITH GRANT OPTION;
```

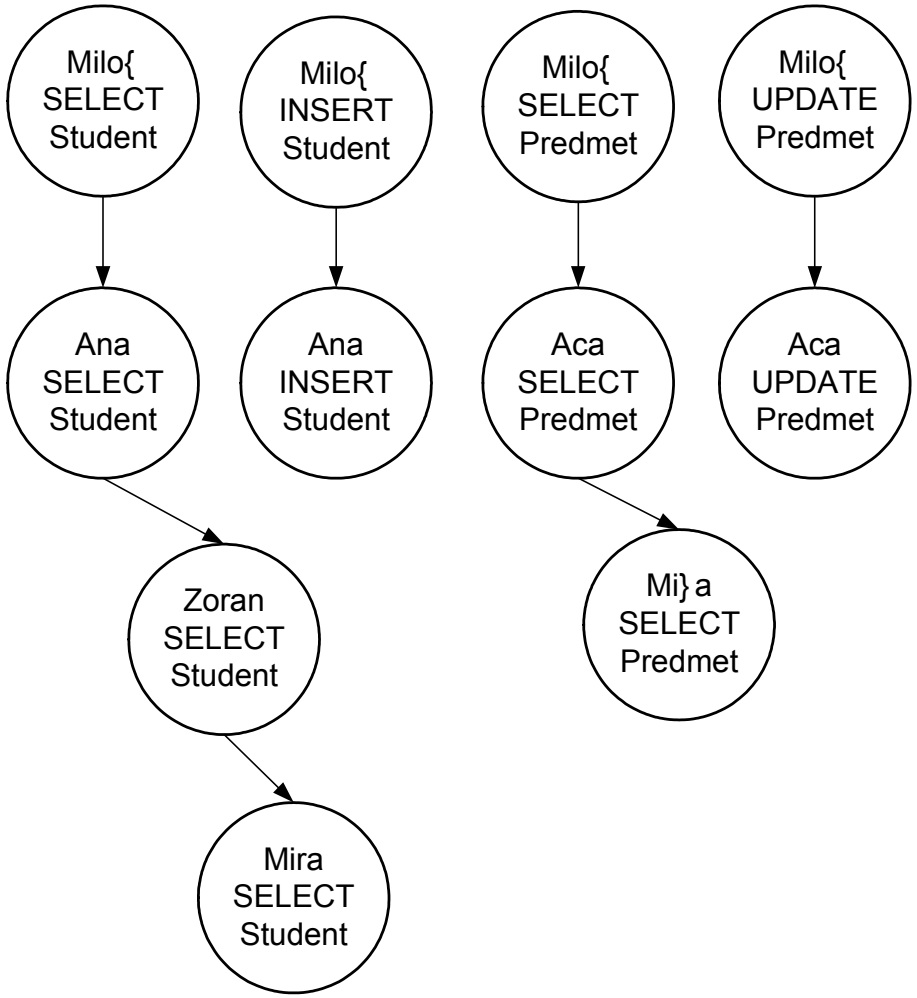
Primer

Ana prenosi privilegiju SELECT za tabele Student na Zorana sa opcijom prenosa na druge, a ovaj prenosi privilegiju SELECT za tabelu Student na Miru. Aca prenosi privilegiju SELECT za tabelu Predmet na Miću.

```
GRANT SELECT Student TO Zoran
WITH GRANT OPTION;
GRANT SELECT Student TO Mira;
GRANT SELECT Predmet TO Mića;
```

Davanje i prenos privilegija prikazaćemo preko tzv. “Grant” dijagrama (grafa). Čvor ovoga grafa predstavlja jednu privilegiju korisnika, a usmerena grana prikazuje prenos privilegija na drugog korisnika.

Primer



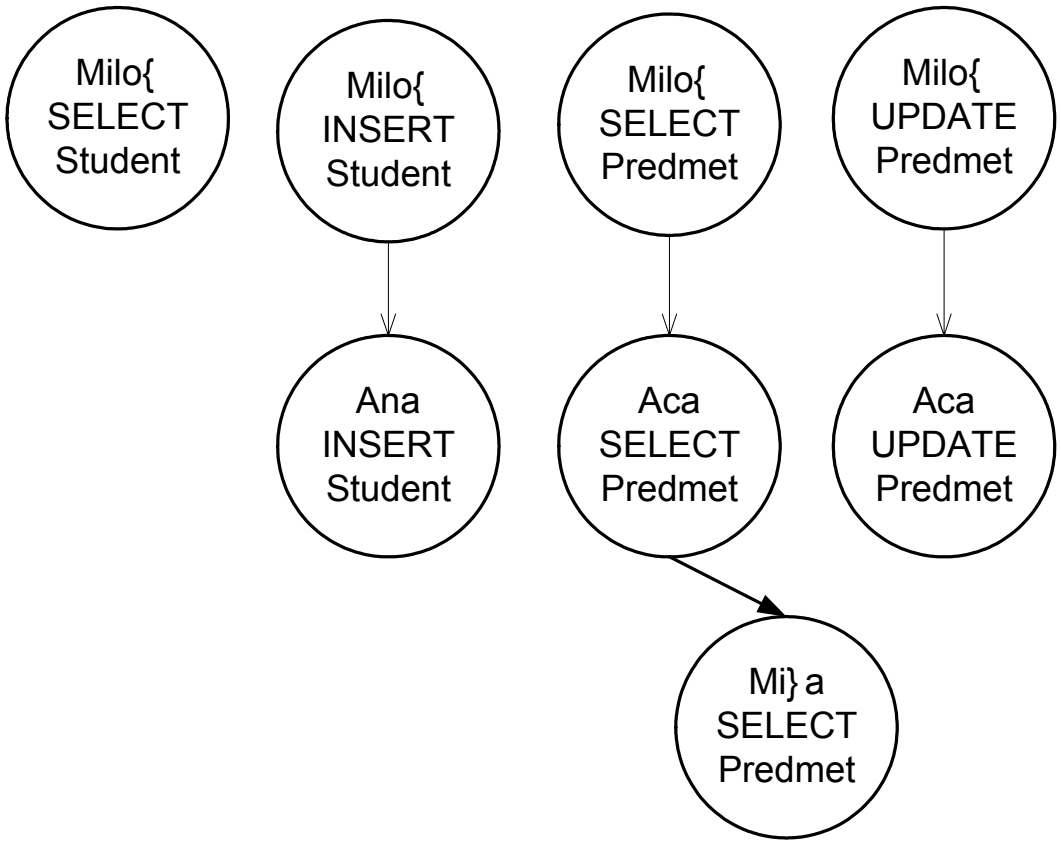
Grant dijagram za prikazani primer

Primer

Na sledecoj slici prikazan je Grant dijagram za isti primer, posle povlačenja privilegija Zoranu i Ani za SELECT Student i Aci za SELECT Predmet preko naredbi:

```
REVOKE GRANT SELECT ON Student
      FROM Ana
      CASCADES;
REVOKE GRANT SELECT ON Predmet
      FROM Aca
      RESTRICT;
```

Primer



Grant dijagram posle povla~enja privilegija

Katalog baze podataka

Katalog baze podataka

Svaki SUBP mora da poseduje svoj ***katalog*** ili ***rečnik podataka***.

Pored termina ***katalog***, koriste se i termini ***rečnik podataka (data dictionary)***, ***direktorijum podataka (data directory***), ***repozitorijum podataka (data repository)*** ili ***enciklopedija.***

Uobičajena je podela na **aktivne** i **pasivne** rečnike podataka.

SQL okru`enje i katalog baze podataka

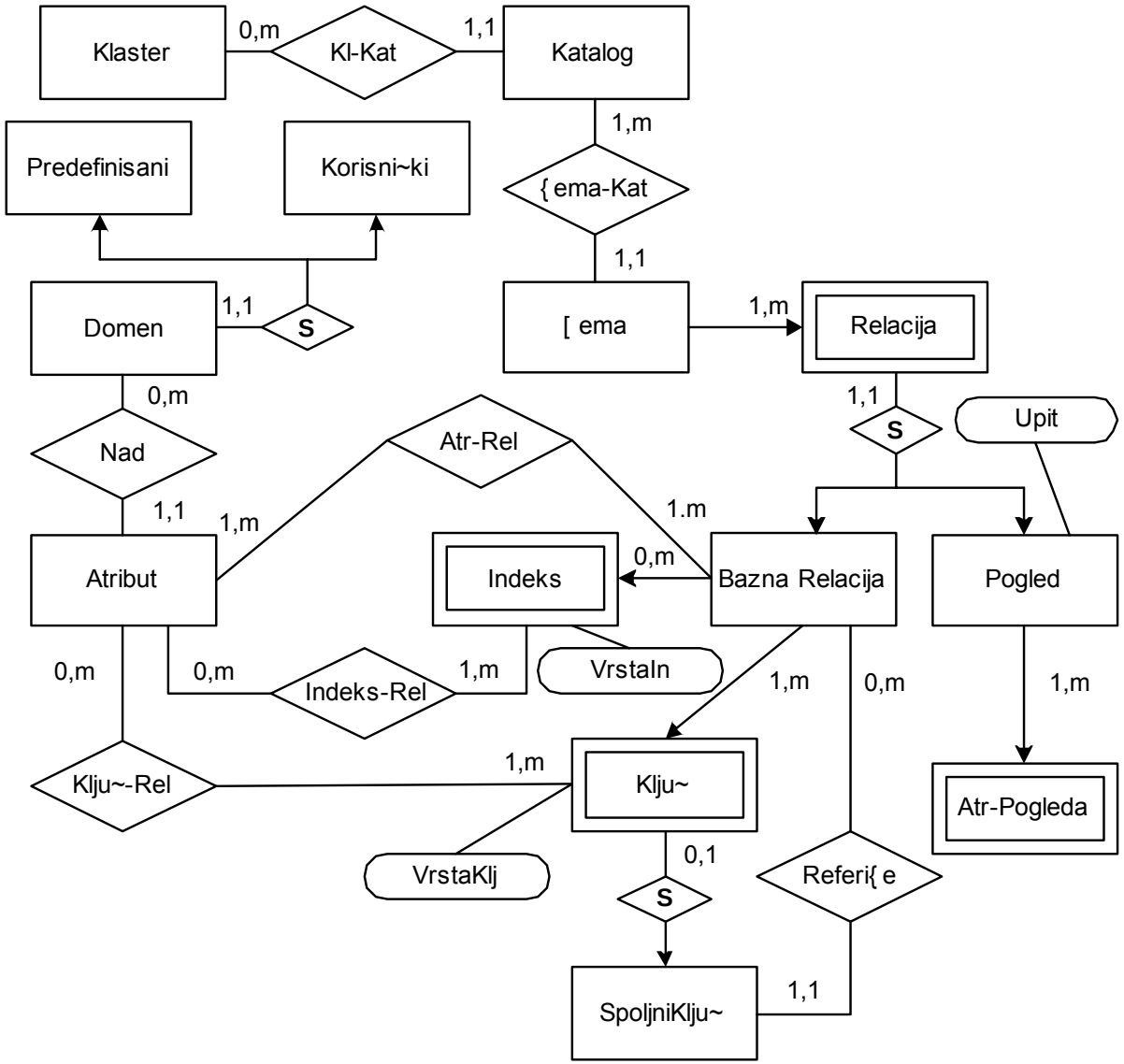
Pod SQL okruženjem podrazumeva se okvir u kome se mogu čuvati podaci i operisati sa njima.

SQL okruženje predstavlja u osnovi neki relacioni SUBP.

SQL okruženje definiše i sledeće:

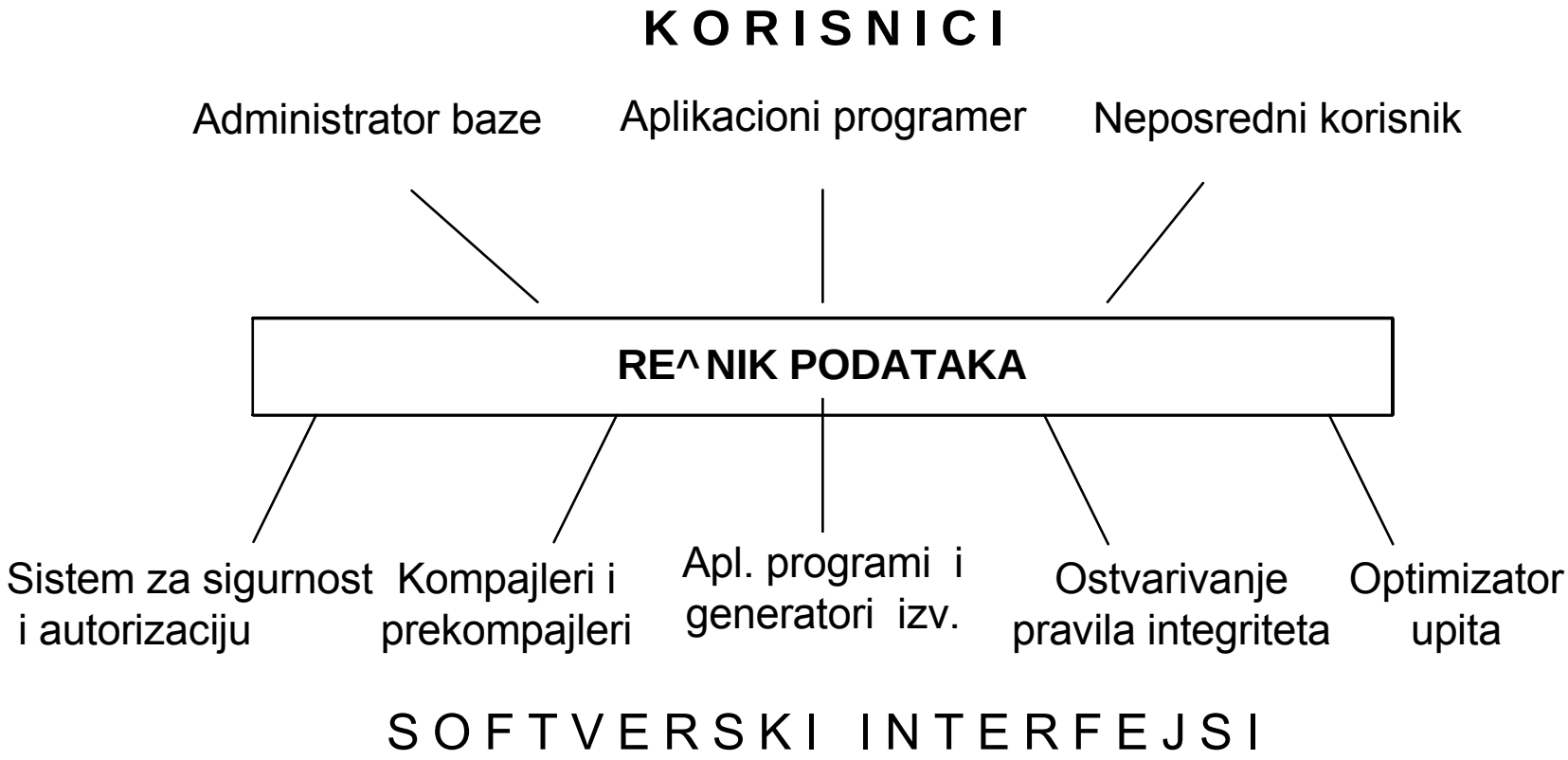
- ***Šema*** je kolekcija tabela, pogleda, domena, tvrdnji (assertions)
- ***Katalog*** je kolekcija šema. U katalogu postoji i informaciona šema (INFORMATION_SCHEME), katalog u užem smislu reči, u kojoj se daju informacije o svim šemama u katalogu.
- ***Klaster*** je kolekcija kataloga.

SQL okru`enje i katalog baze podataka



Model objekti veze za relacioni katalog

SQL okru`enje i katalog baze podataka



Korisnici i softverski interfejsi za rečnik podataka

Primer rečnika podataka za Oracle SUBP

Dat je primer kreiranja jedne Oracle baze, a zatim slede neki upiti nad katalogom te šeme i njihovi rezultati.

Naziv šeme je MINIB.

Prikazane su relacije o studentima, predmetima, prijavama i profesorima i pogled u kome se daju položeni ispiti za svakog studenta.

```
CREATE TABLE MINIB.STUDENT (
BRIND       VARCHAR2(6) NOT NULL UNIQUE,
IME         VARCHAR2(15),
PREZIME     VARCHAR2(30),
CONSTRAINT BROJ_IND PRIMARY KEY (BRIND));

CREATE TABLE MINIB.PREDMET (
SIFRAPRED   INTEGER NOT NULL UNIQUE,
NAZIV       VARCHAR2(30),
SEMESTAR    INTEGER,
CONSTRAINT KLJUC PRIMARY KEY (SIFRAPRED));
```

Primer rečnika podataka za Oracle SUBP

```
CREATE TABLE MINIB.PRIJAVA (
BRIND     VARCHAR2(6),
SIFRAPRED  INTEGER,
DATUM      DATE NOT NULL,
OCENA      INTEGER NOT NULL,
CONSTRAINT PR PRIMARY KEY (SIFRAPRED,BRIND),
CONSTRAINT PRED FOREIGN KEY(SIFRAPRED) REFERENCES
    MINIB.PREDMET(SIFRAPRED),
CONSTRAINT STD FOREIGN KEY(BRIND) REFERENCES
  MINIB.STUDENT(BRIND));

CREATE TABLE MINIB.PROFESOR (
SIFRAPROF  INTEGER NOT NULL,
IME        VARCHAR2(15),
PREZIME    VARCHAR2(30),
SIFRAPRED  INTEGER,
CONSTRAINT KLJUCPROF PRIMARY KEY (SIFRAPROF),
CONSTRAINT PREDAJE FOREIGN KEY(SIFRAPRED) REFERENCES
     MINIB.PREDMET(SIFRAPRED));
```

Primer rečnika podataka za Oracle SUBP

```
 CREATE VIEW POLOZIO AS
SELSELECT STUDENT.IME, STUDENT.PREZIME, PREDMET.NAZIV,
   PRIJAVA.DATUM,   PRIJAVA.OCENA
FROM   STUDENT,PREDMET,PRIJAVA
WHERE  PRIJAVA.BRIND = STUDENT.BRIND
AND    PRIJAVA.SIFRAPRED = PREDMET.SIFRAPRED;
```

Prikaz svih tabela vlasnika MINIB:

```
SQL>   SELECT *
       FROM   ALL_CATALOG
       WHERE  OWNER = 'MINIB';

OWNER         TABLE_NAME    TABLE_TYPE
------------  ----------------    ----------------
MINIB         POLOZIO             VIEW
MINIB         PREDMET             TABLE
MINIB         PRIJAVA       TABLE
MINIB         PROFESOR      TABLE
MINIB         STUDENT             TABLE
```

Primer rečnika podataka za Oracle SUBP

Za objektno-relacione konstrukcije ORACLE katalog poseduje USER_TYPES, USER_OBJECT_TABLES i USER_DEPENDECIES tabele. Osnovne kolone tabele USER_TYPES su:

- TYPE_NAME – naziv tipa
- ATTRIBUTES – broj atributa u tipu
- METHODS – broj metoda u tipu

Osnovne kolone u tabeli USER_DEPENDENCIES koja pokazuje kako je jedan objektni tip definisan preko drugog su:

- NAME – naziv zavisnog objekta šeme
- TYPE - tip zavisnog objekta (TYPE, TABLE i slično)
- REFERENCED_NAME – naziv nadređenog objekta
- REFERENCED_TYPE – tip nadređenog objekta (TYPE, TABLE i slično)
- DEPENDENCY_TYPE – vrsta zavisnosti (‘HARD’, ‘REF’).

Primer rečnika podataka za Oracle SUBP

Rezultat slede{eg upita predstavlja podatke o svim kolonama jedne tabele. Kolone NUM_DISTINCT (broj različitih vrednosti), LOW_VALUE (najmanja vrednost), HIGH_VALUE (najveća vrednost) daju značajne podatke za optimizator upita. Ove kolone se ne ažuriraju direktno sa ažuriranjem odgovarajuće tabele već tek kada se postavi zahtev preko Oracle iskaza

```
ANALYZE TABLE STUDENT
COMPUTE STATISTICS;
```

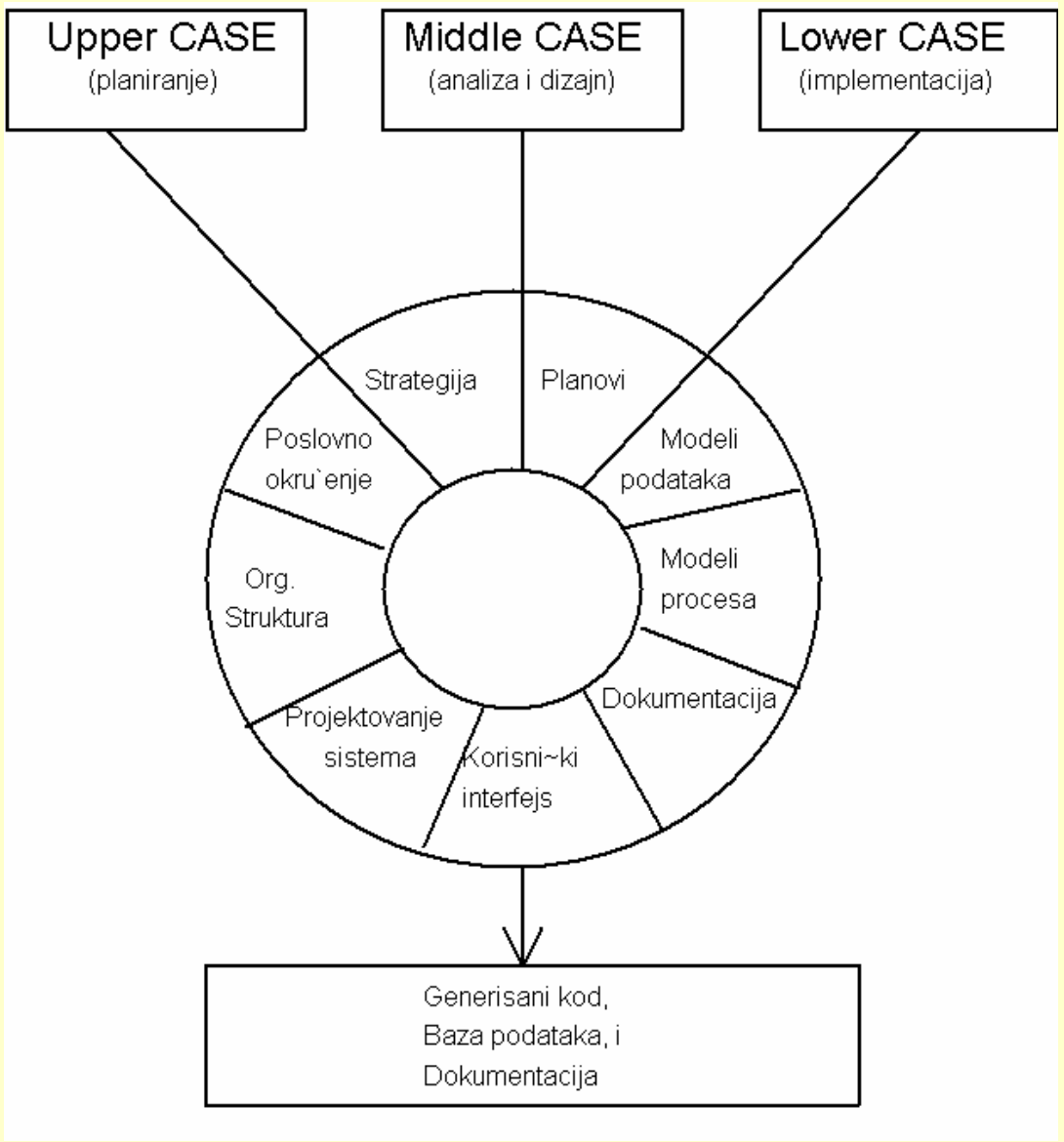
Prikaz svih informacija o jednoj tabeli:

```
SELECT COLUMN_NAME, DATA_TYPE, DATA_LENGTH,   NUM_DISTINCT,
        HIGH_VALUE, LOW_VALUE
FROM  USER_TAB_COLUMNS
WHERE    TABLE_NAME = 'STUDENT';
```

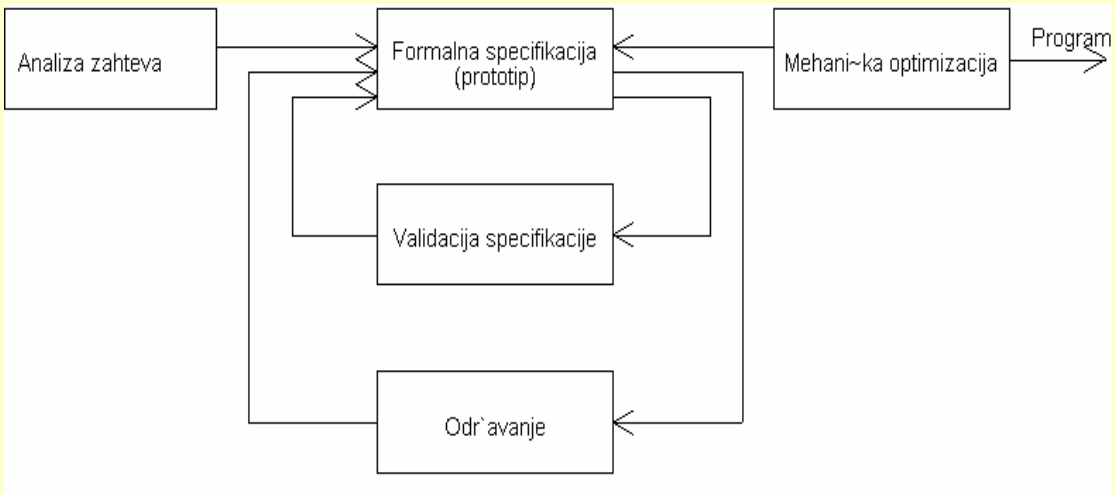
```
COLUMN_NAME   DATA_TYPE DATA_LENGTH   NUM_DISTINCT  HIGH_VALUE LOW_VALUE
-----------   --------- -----------   ------------ ---------- --------
BRIND         VARCHAR2  6             5          3530312F3935 3133372F3936
IME           VARCHAR2  15            4          564C414441   414E41
PREZIME       VARCHAR2  30            5          56554B4F564943 4A41524943
```

CASE alati

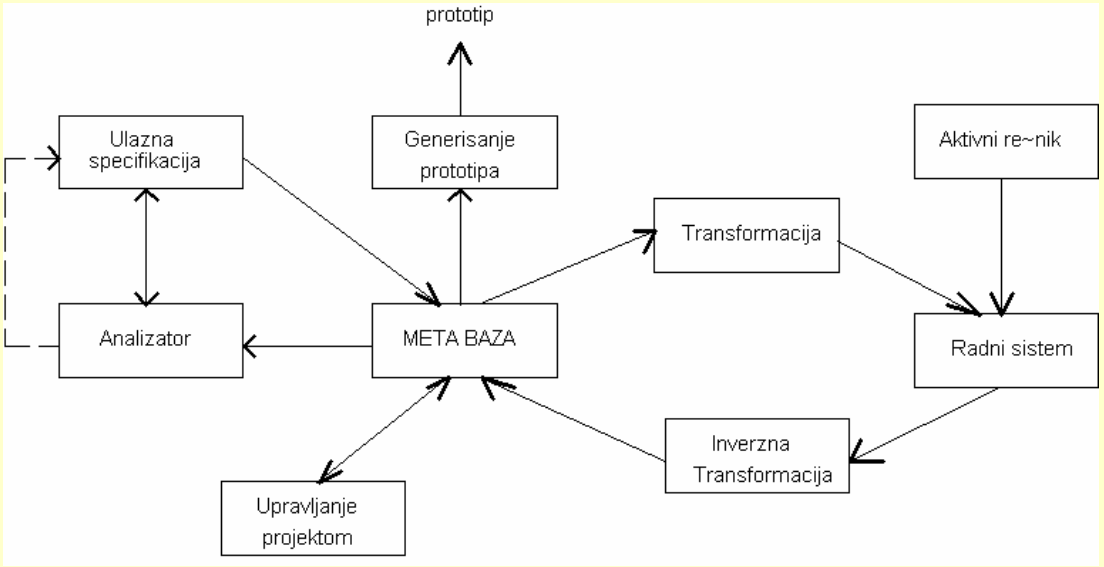
Podela CASE alata po fazama zivotnog ciklusa



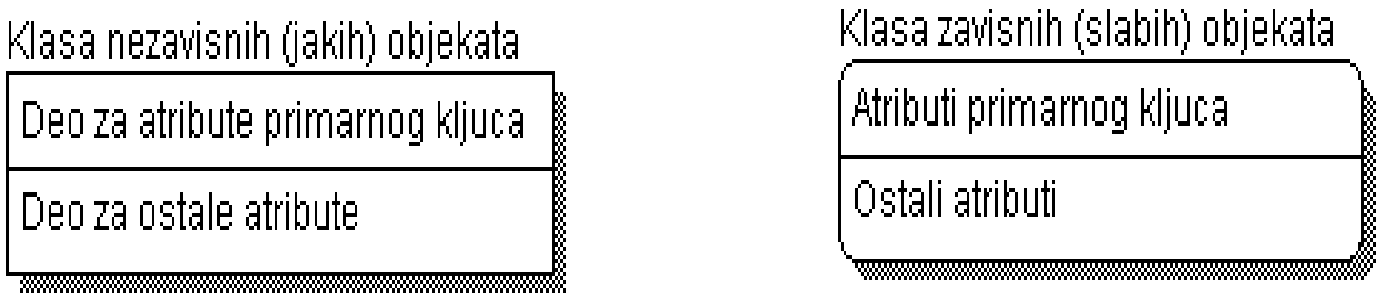
Operacioni (transformacioni) razvoj softvera



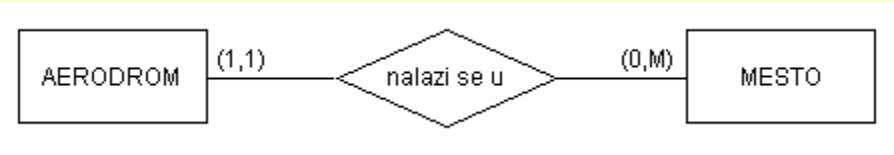
Idealna arhitektura CASE sistema

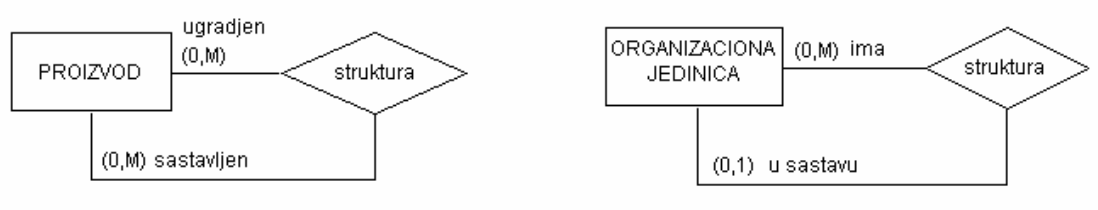


VERZIJE MOV-a: IDEF1x standard

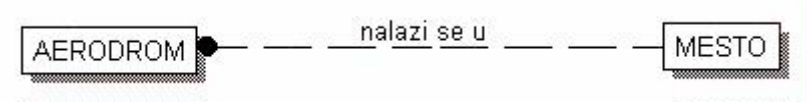


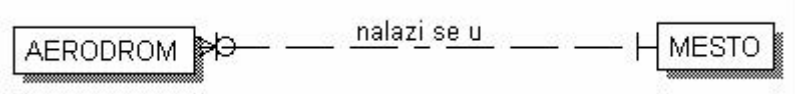
Veze po PMOV sintaksi

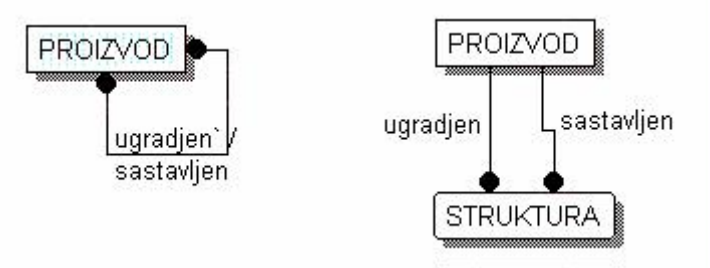




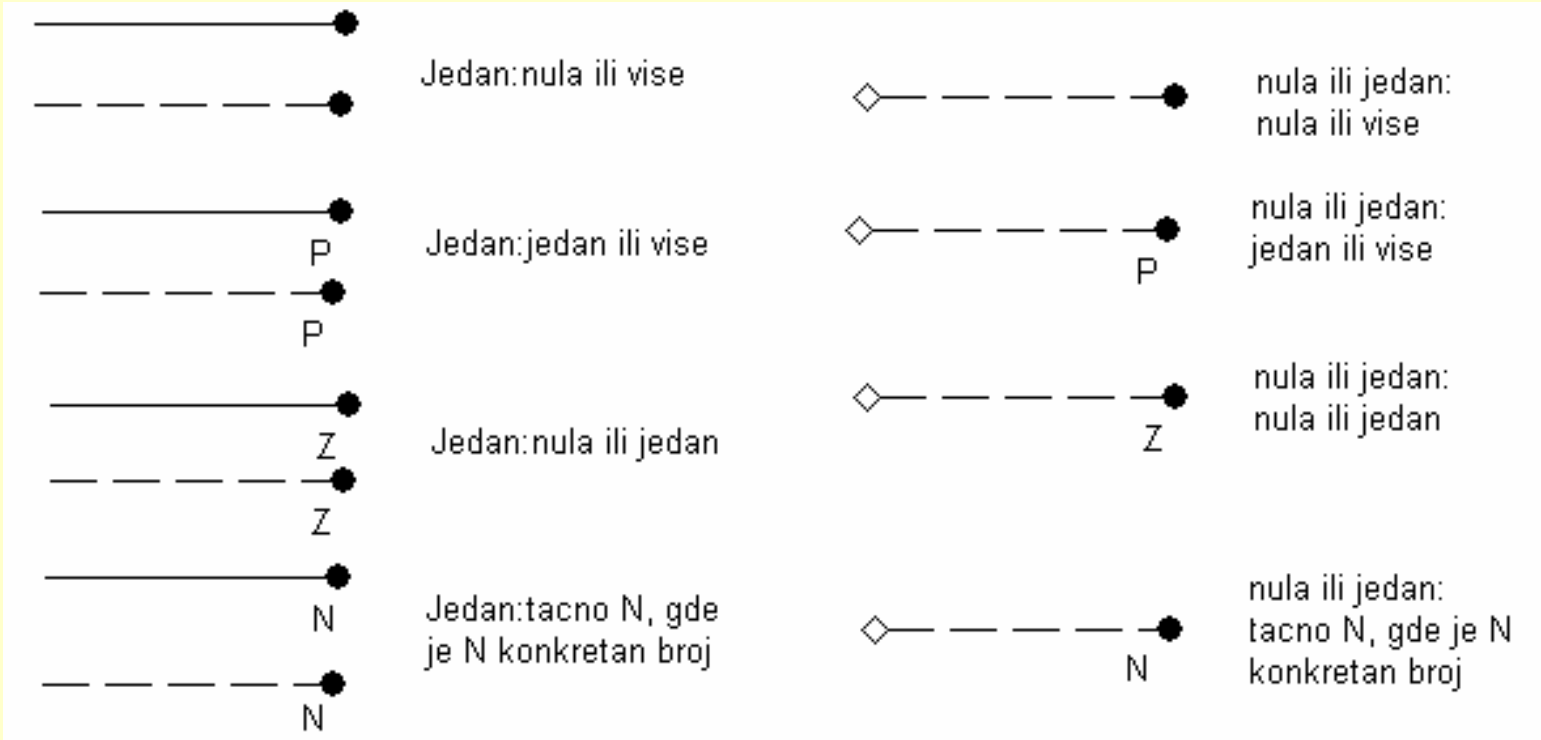
Veze po IDEF1x i IE standardu



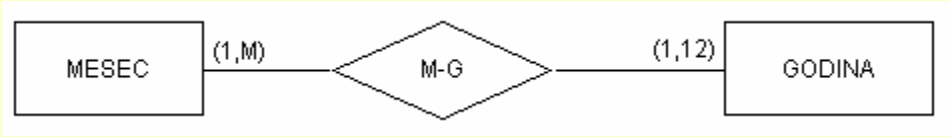


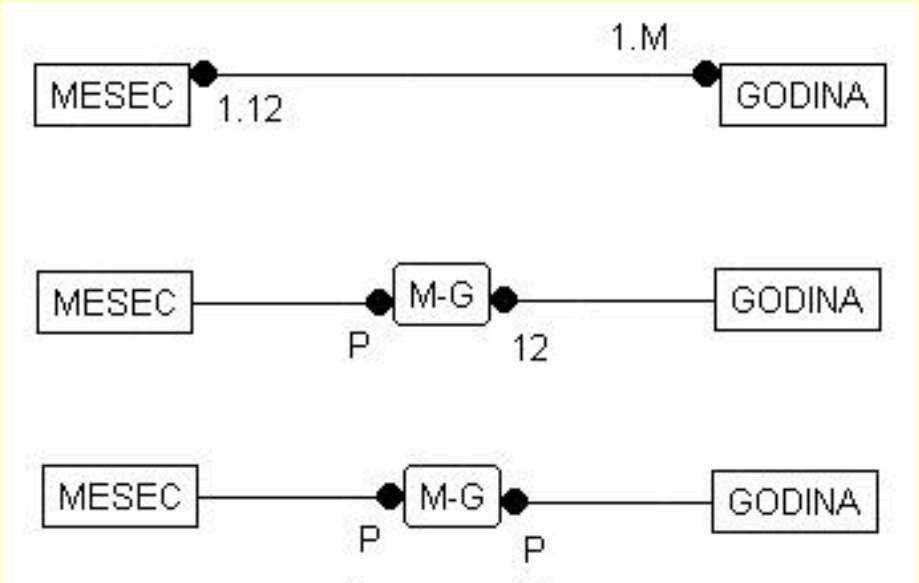


VERZIJE MOV-a: IDEF1x standard

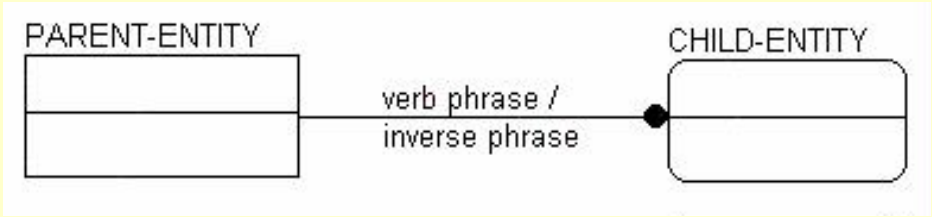


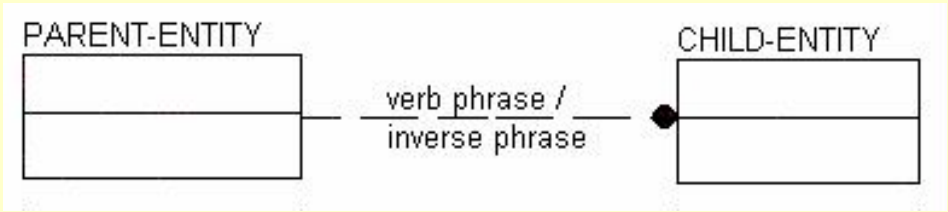
VERZIJE MOV-a: IDEF1x standard

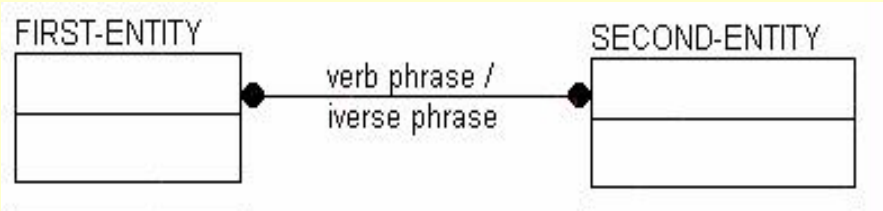




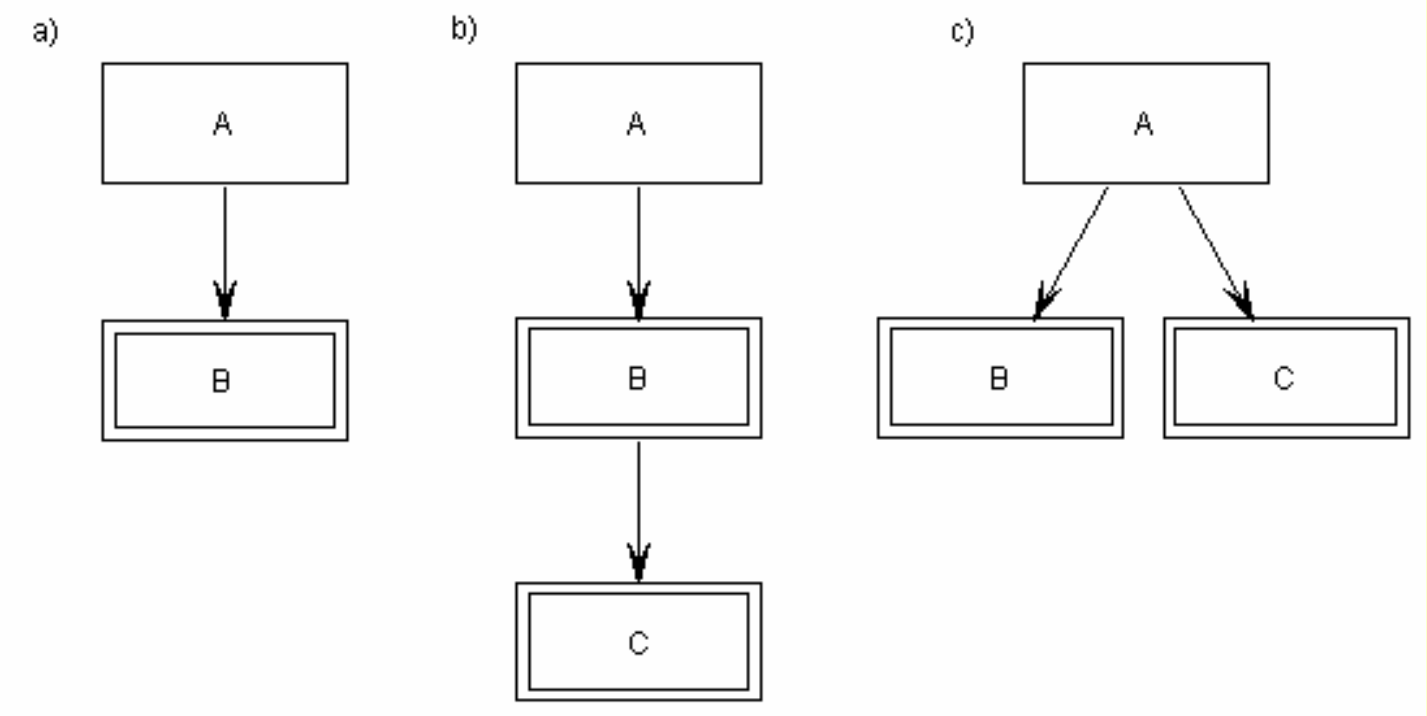
a) jedan:vise identifikujuca veza
b) jedan:vise neidentifikujuca veza
c) nespecificirana veza



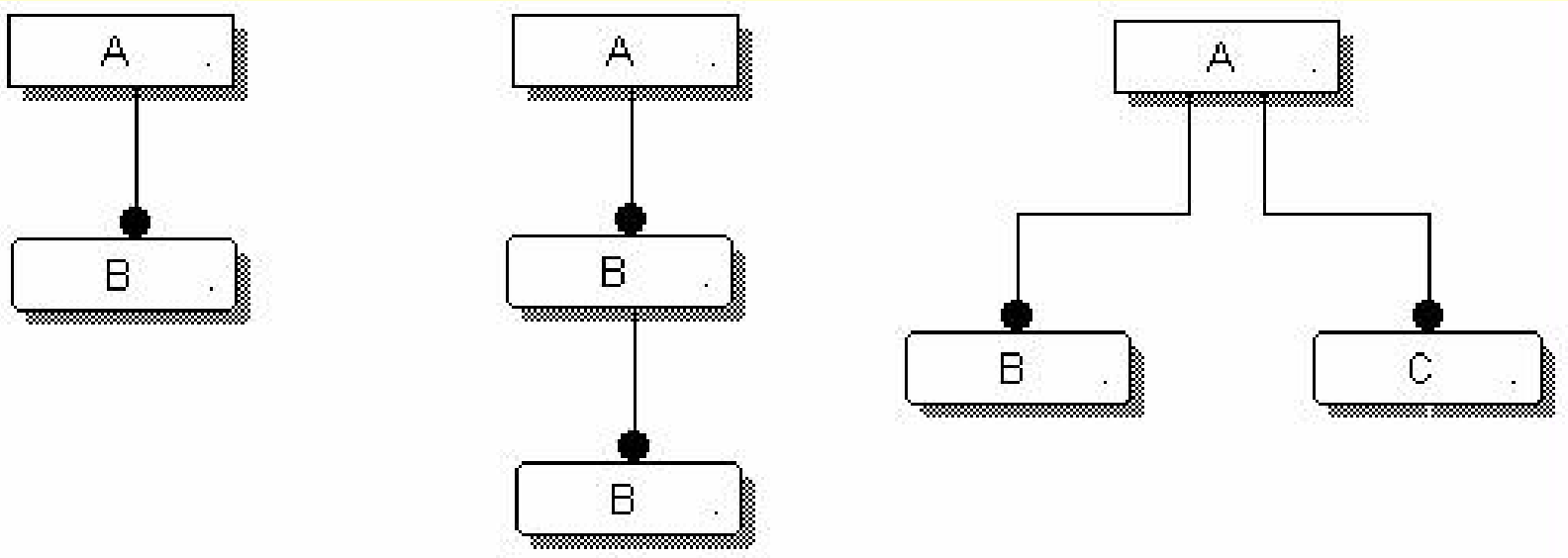




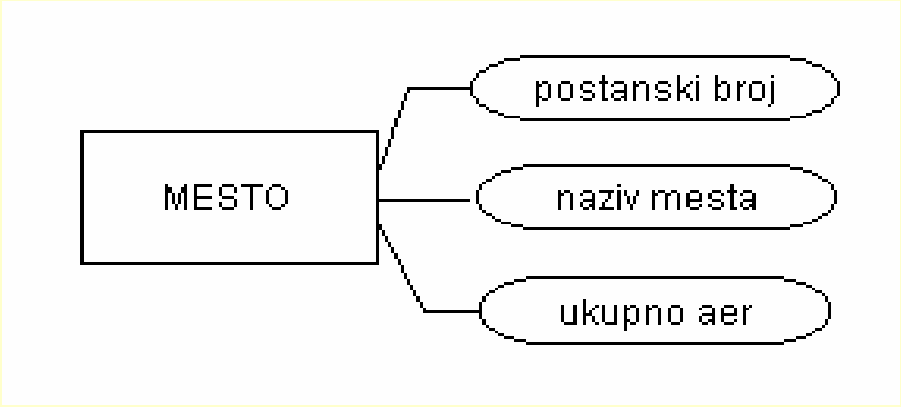
Objekti po PMOV sintaksi



Objekti po IDEF1x standardu

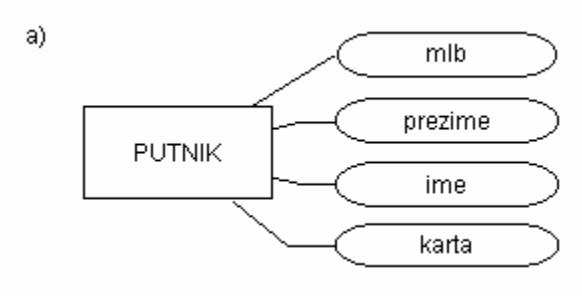


Atributi i domeni

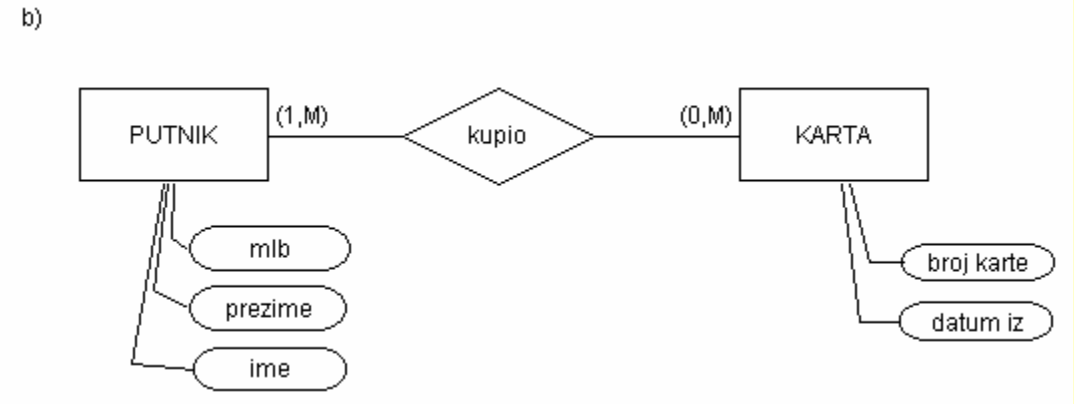


Viseznacni atributi

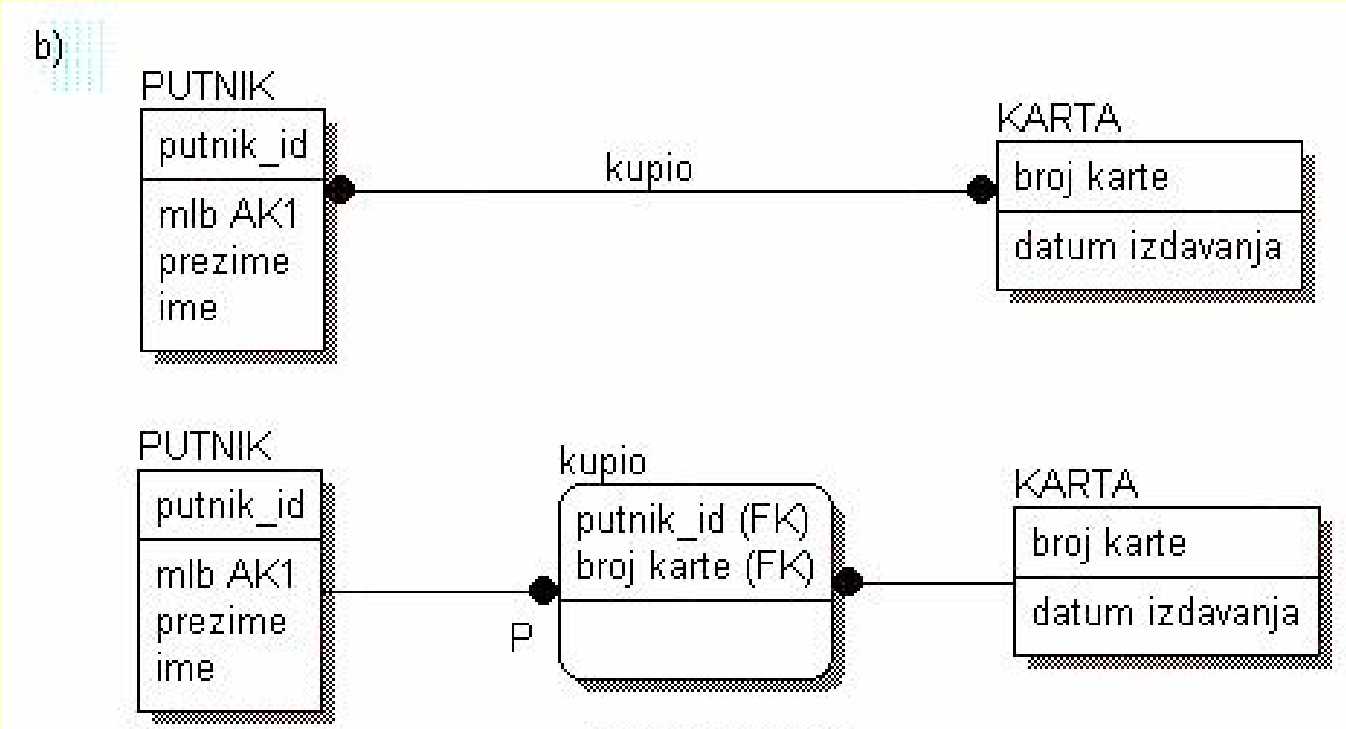
a)



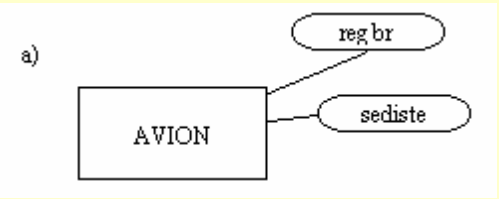
b)

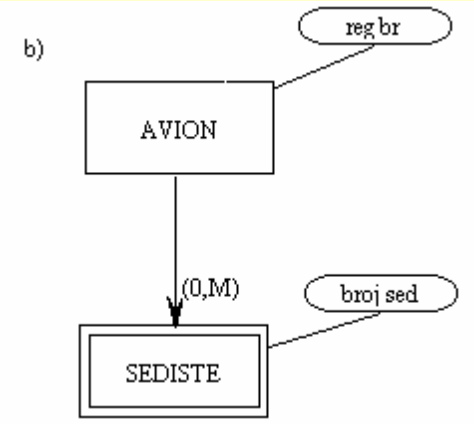


Viseznacni atributi

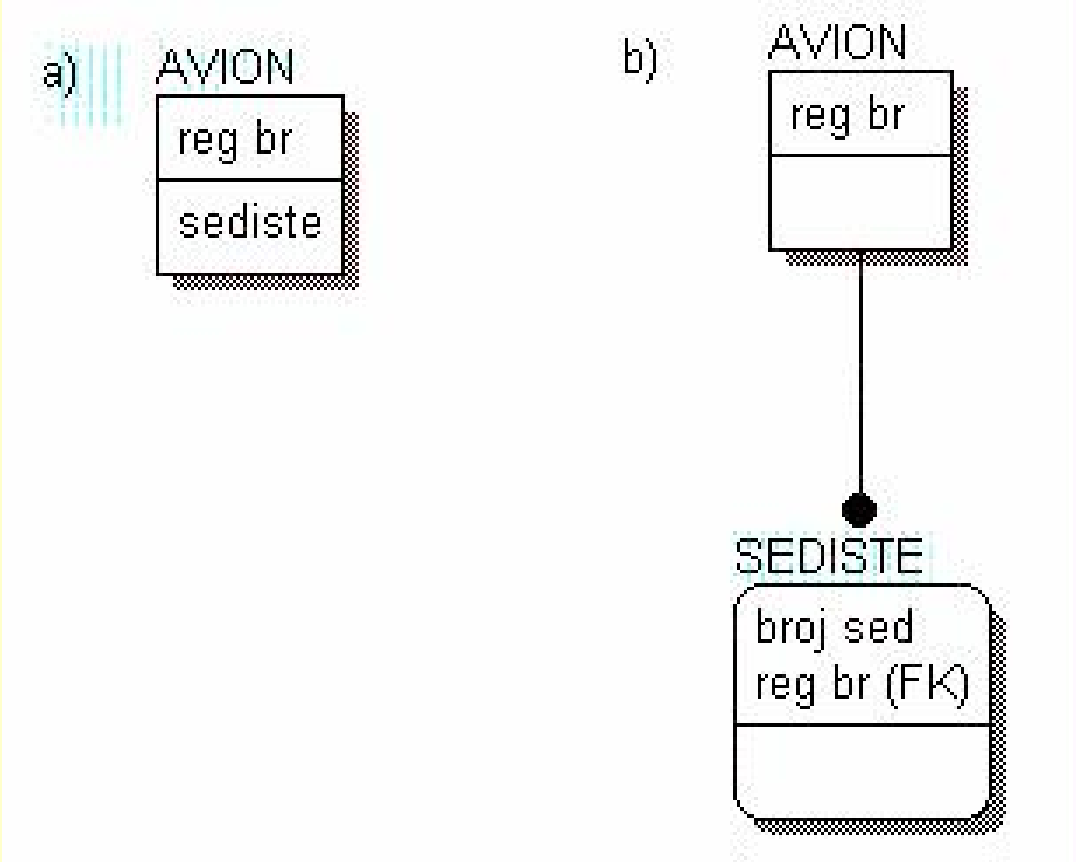


Viseznacni atributi

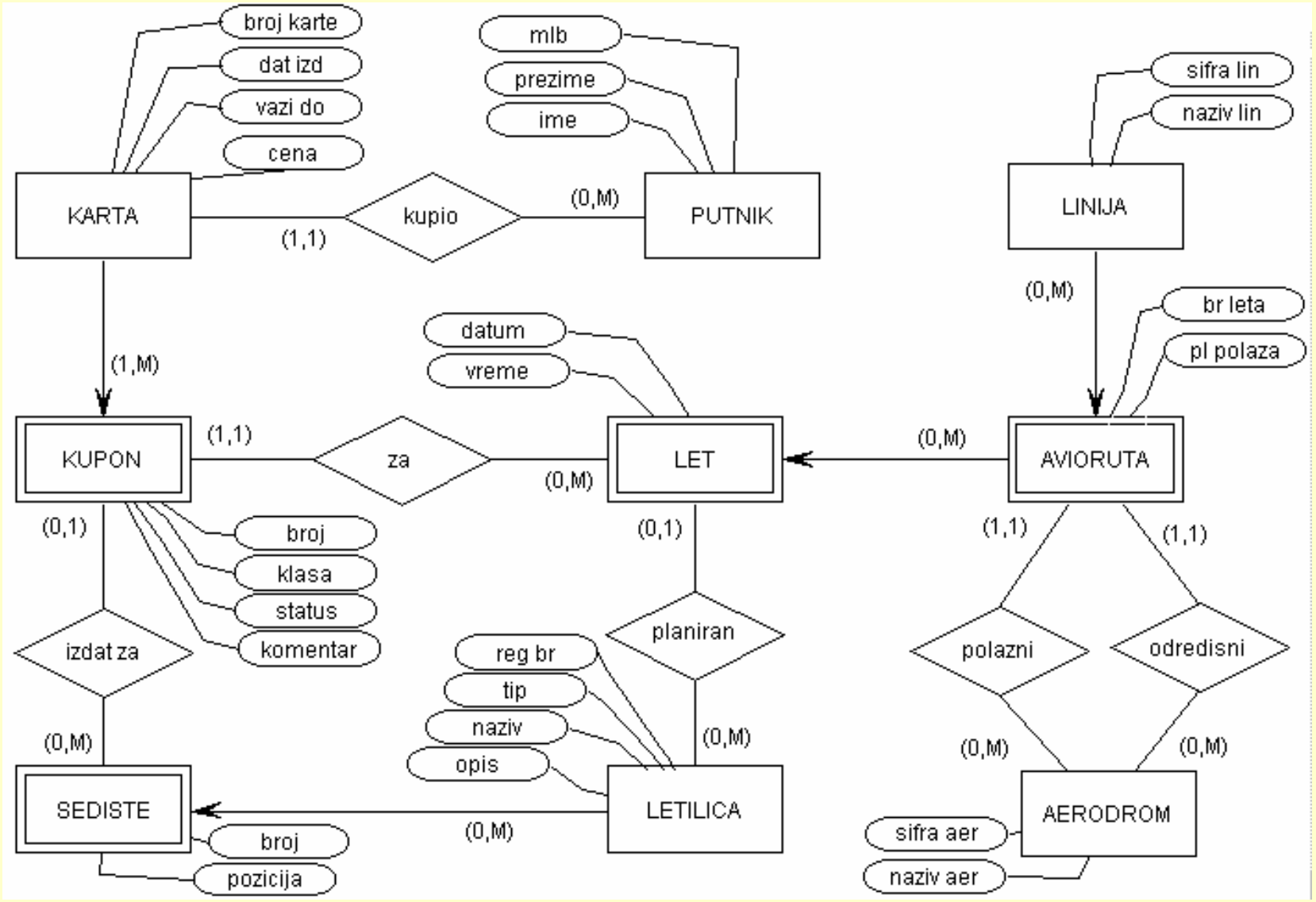


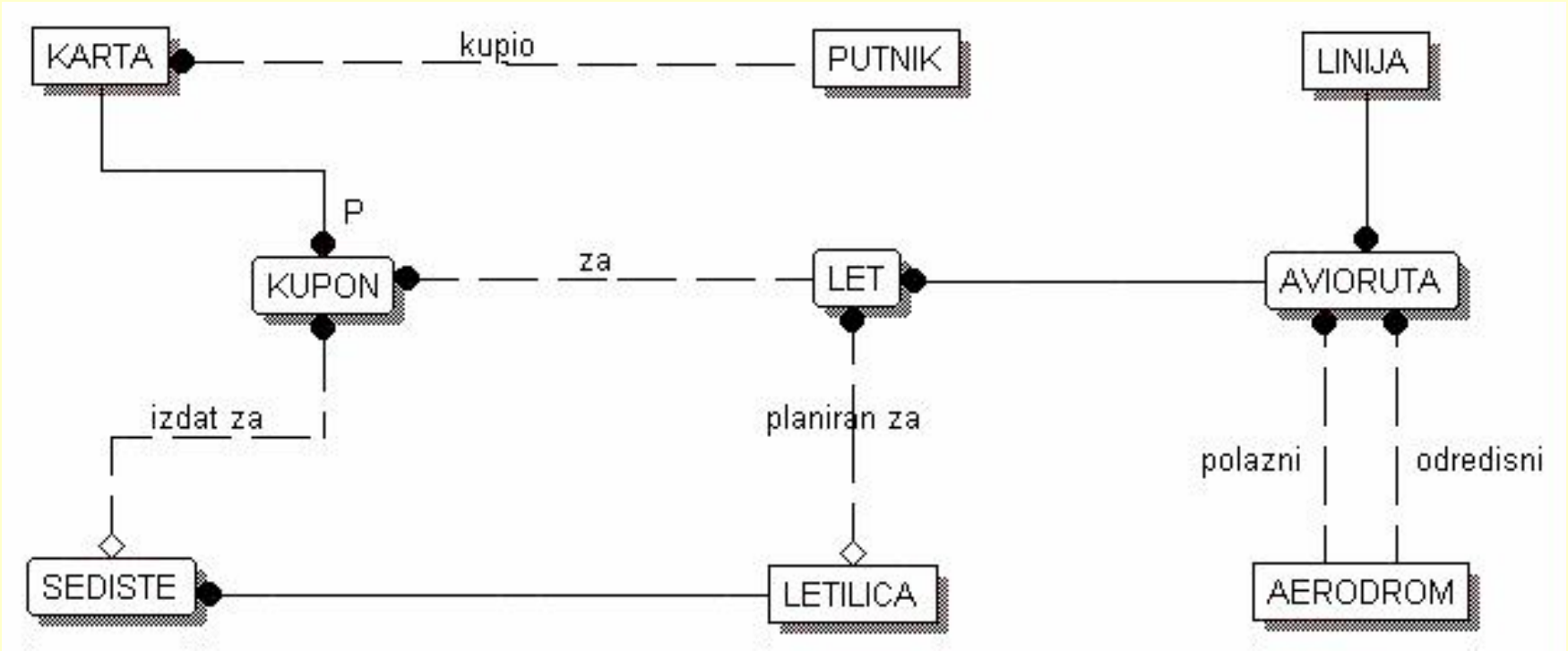


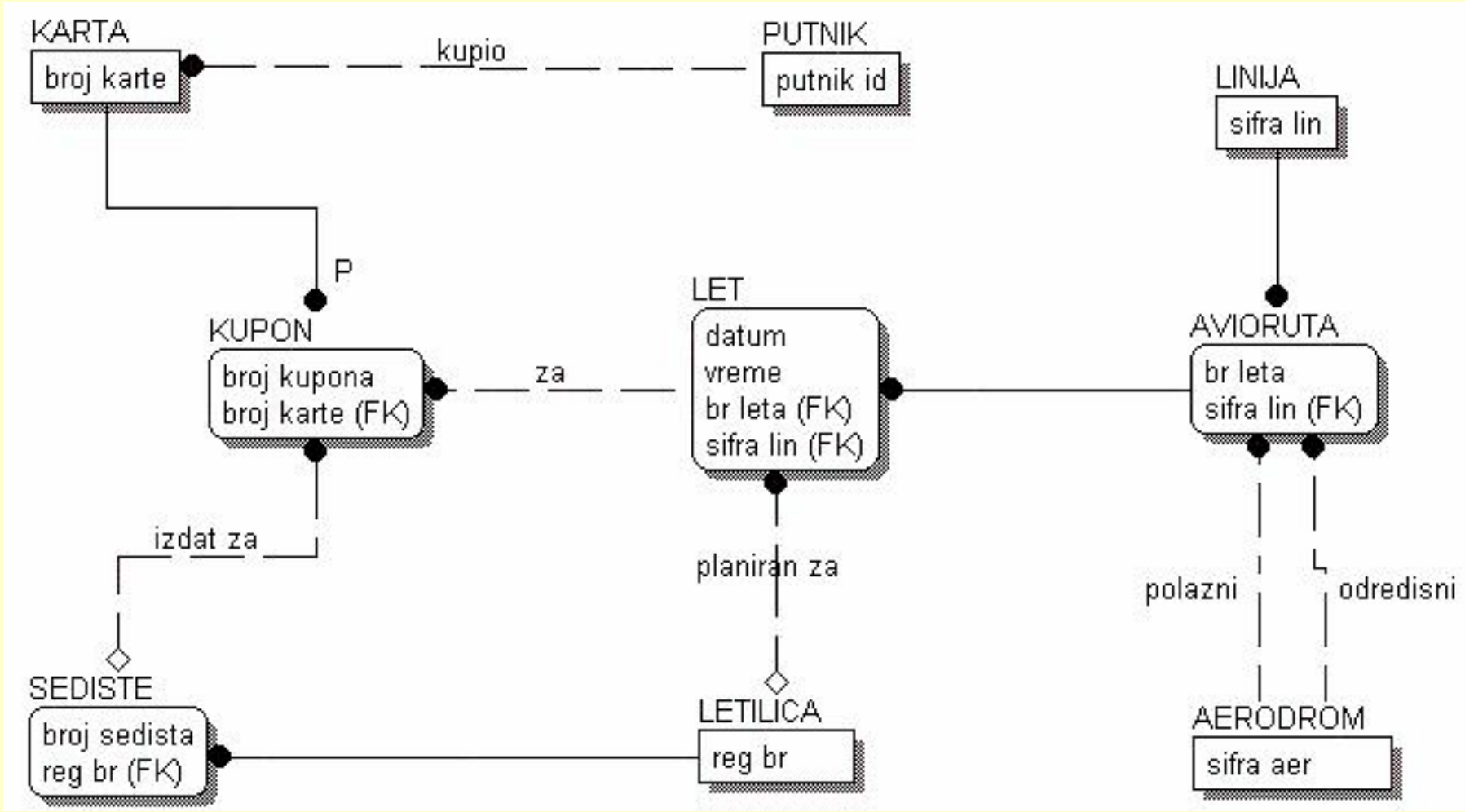
Viseznacni atributi

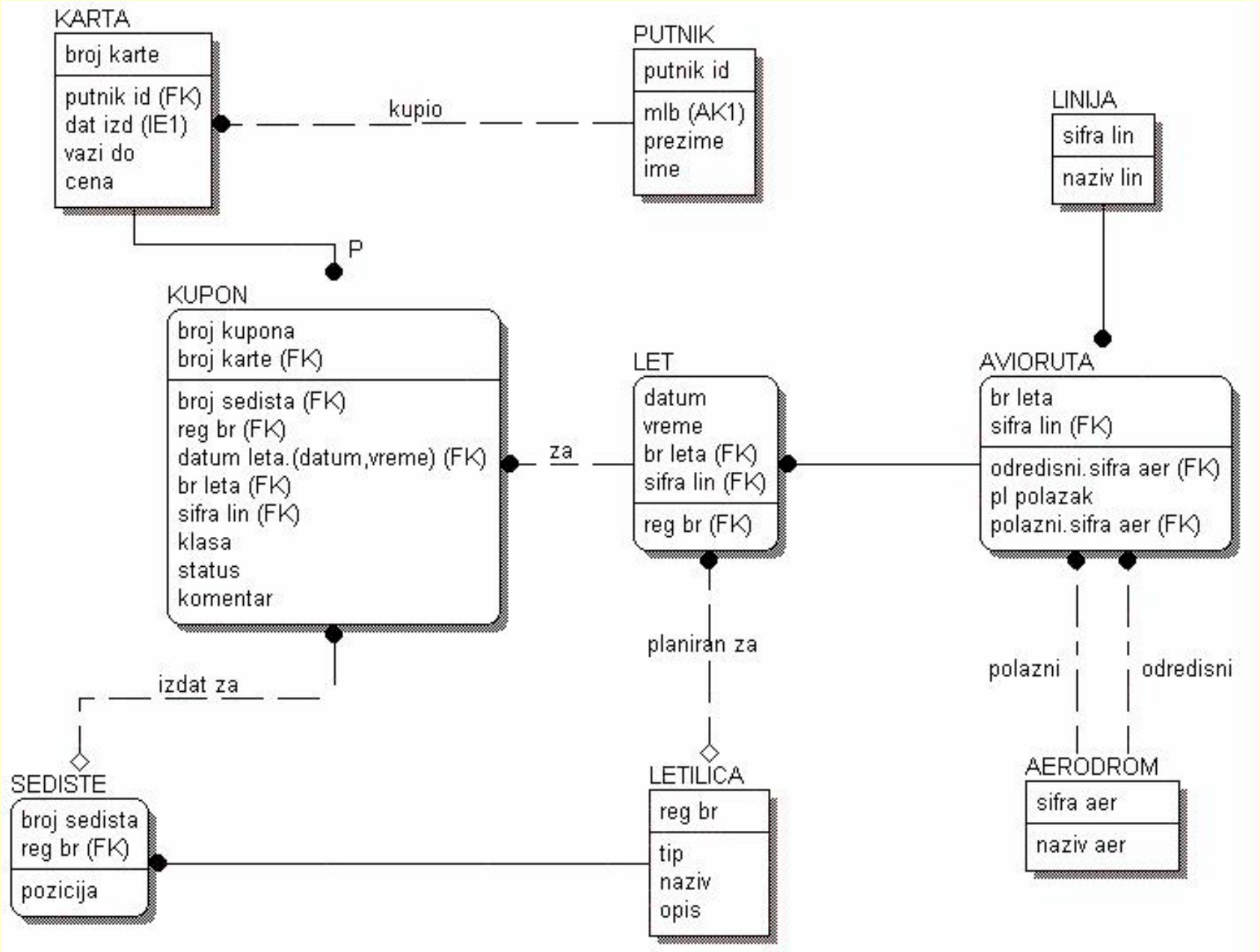


Primer 1: *Avionska karta za jednu standardnu avio-liniju mo`e biti sastavljena od vi{e kupona. Jedna linija mo`e da uklju~i vi{e letova na relaciji izmedju mesta polaska i mesta krajnjeg odredi{ta. Svaki avion obi~no ima nekoliko letova u toku dana (let je identifikovan preko datuma i vremena poletanja aviona). Karta sadr`i podatke o avionskoj liniji, prezimenu i imenu putnika, mestu polazi{ta, mestu krajnjeg odredi{ta, datumu izdavanja, roku va`enja i ceni. Kuponi karte sadr`e identi~ne podatke i podatke o pojedina~nim letovima izmedu polazi{ta i krajnjeg odredi{ta: mesto poletanja, mesto sletanja, osnovni podaci o avionu. broj leta, klasa sedi{ta, datum i vreme poletanja*.

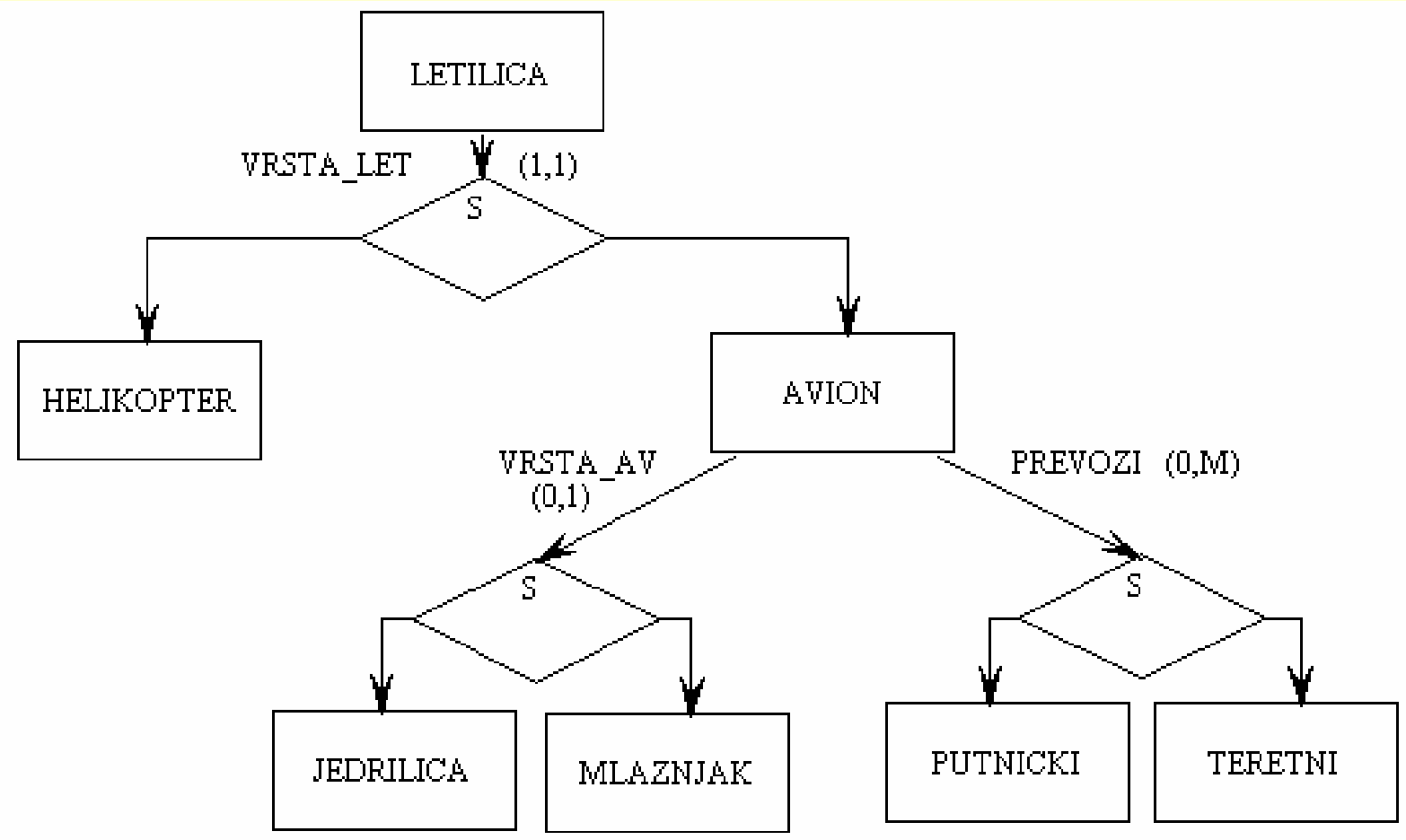

broj karte
dat izd
vazi do
cena
KARTA
kupio
(1,1)
(0,M)
PUTNIK
mlb
prezime
ime
sifra lin
naziv lin
LINIJA
(0,M)
br leta
pl polaza
(1,M)
datum
vreme
KUPON
(1,1)
za
(0,M)
LET
(0,M)
AVIORUTA
(0,1)
broj
klasa
status
komentar
(0,1)
(1,1)
(1,1)
izdat za
planiran
polazni
odredisni
reg br
tip
naziv
opis
(0,M)
(0,M)
(0,M)
SEDISTE
broj
pozicija
(0,M)
LETILICA
AERODROM
sifra aer
naziv aer


KARTA
kupio
PUTNIK
LINIJA
P
KUPON
za
LET
AVIORUTA
izdat za
planiran za
polazni
odredisni
SEDISTE
LETILICA
AERODROM

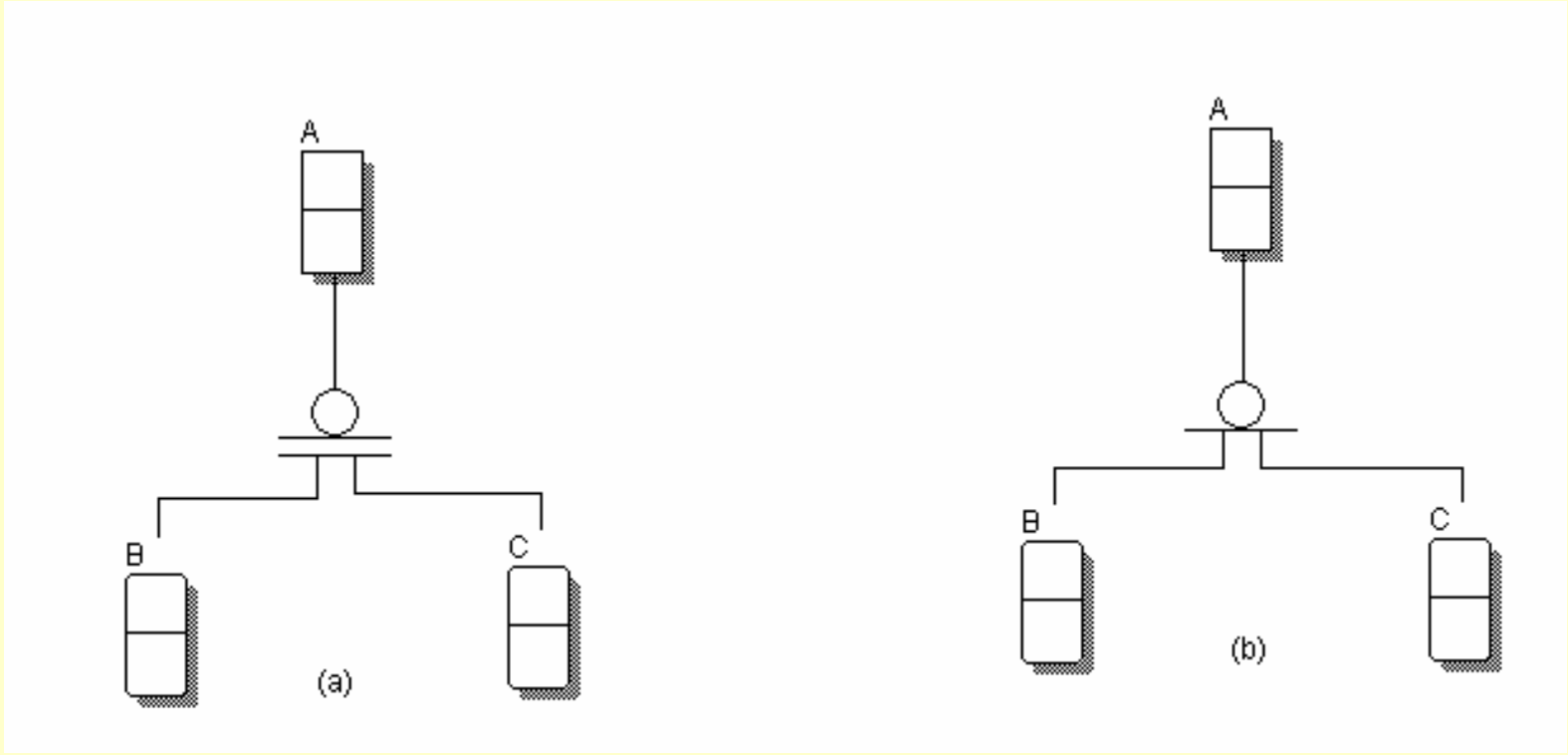

KARTA
broj karte
kupio
PUTNIK
putnik id
LINIJA
sifra lin
P
KUPON
broj kupona
broj karte (FK)
za
LET
datum
vreme
br leta (FK)
sifra lin (FK)
AVIORUTA
br leta
sifra lin (FK)
izdat za
planiran za
polazni
odredisni
SEDISTE
broj sedista
reg br (FK)
LETILICA
reg br
AERODROM
sifra aer


KARTA
broj karte
putnik id (FK)
dat izd (IE1)
vazi do
cena
kupio
PUTNIK
putnik id
mlb (AK1)
prezime
ime
LINIJA
sifra lin
naziv lin
P
KUPON
broj kupona
broj karte (FK)
broj sedista (FK)
reg br (FK)
datum leta.(datum,vreme) (FK)
br leta (FK)
sifra lin (FK)
klasa
status
komentar
za
LET
datum
vreme
br leta (FK)
sifra lin (FK)
reg br (FK)
AVIORUTA
br leta
sifra lin (FK)
odredisni.sifra aer (FK)
pl polazak
polazni.sifra aer (FK)
planiran za
polazni
odredisni
izdat za
SEDISTE
broj sedista
reg br (FK)
pozicija
LETILICA
reg br
tip
naziv
opis
AERODROM
sifra aer
naziv aer

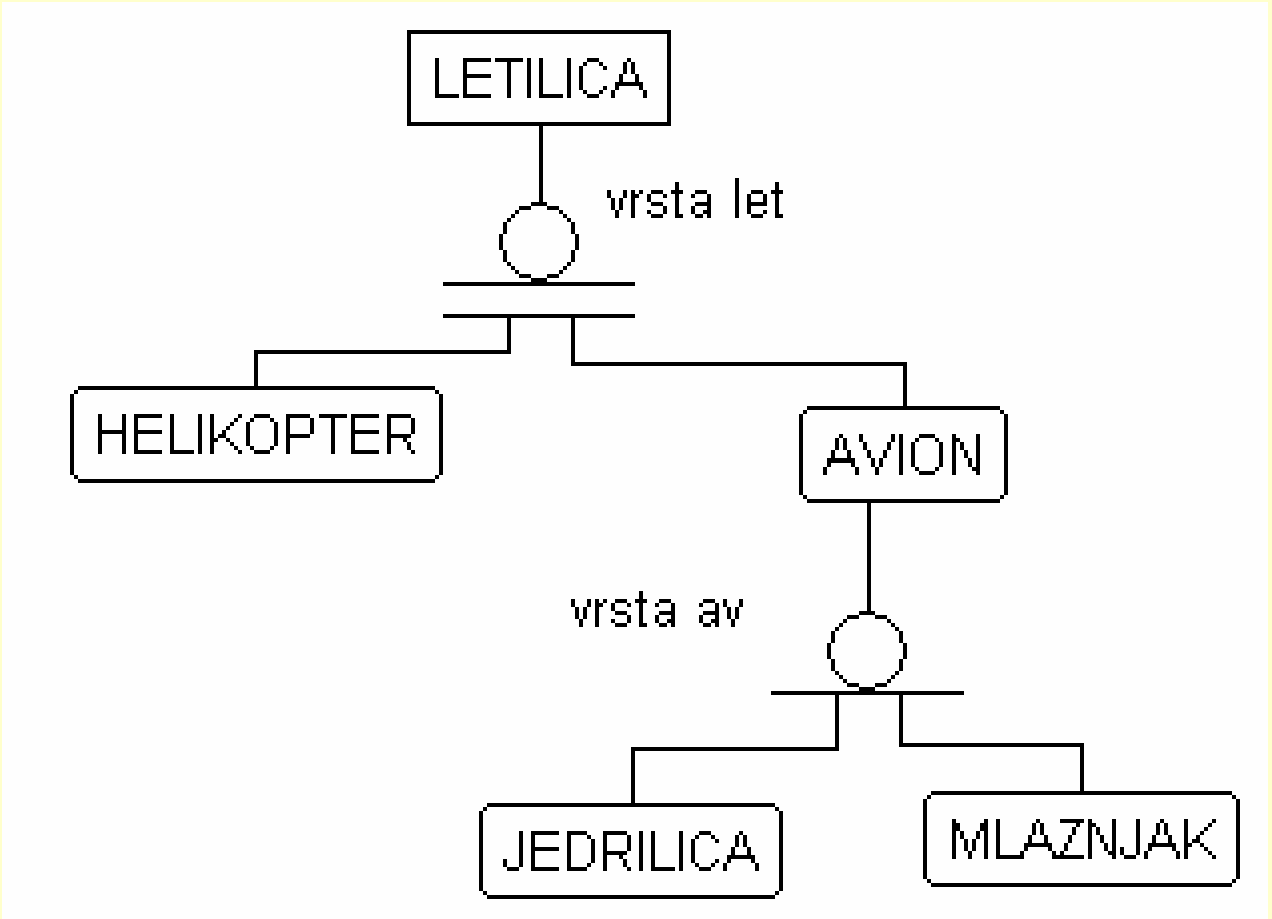
Generalizacija i specijalizacija



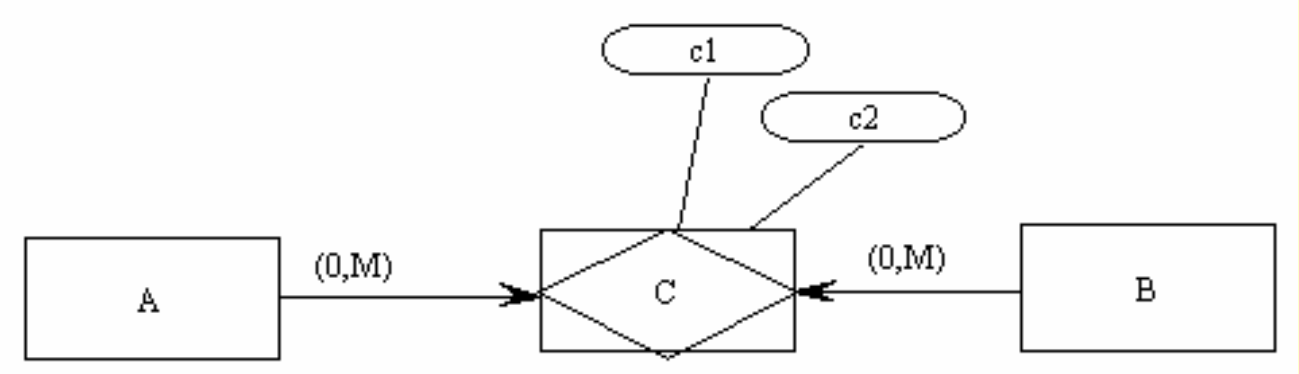
VERZIJE MOV-a: IDEF1x standard

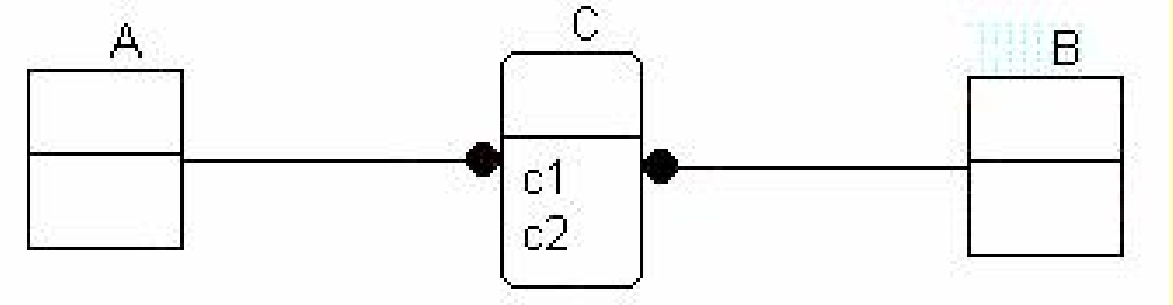


VERZIJE MOV-a: IDEF1x standard



Agregacija





VERZIJE MOV-a: IDEF1x standard