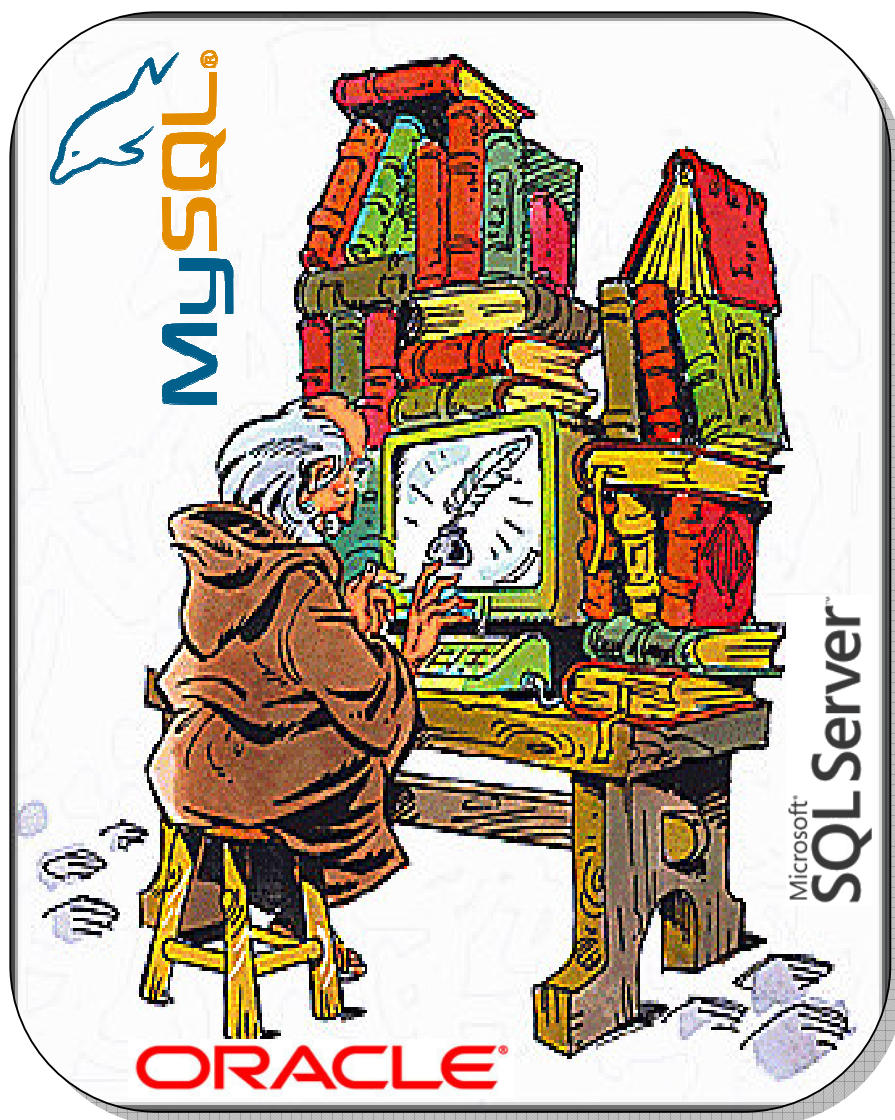


prof. Nebojša Lukić dipl.ing.el.



# BAZE PODATAKA

za III razred elektrotehničke škole

Tehnička škola Mihajlo Pupin

Prvo izdanje

Bijeljina, avgust 2007.

# Sadržaj

## I

### 1 UVOD U BAZE PODATAKA

- 1.1 Osnovni pojmovi vezani uz baze podataka
  - 1.1.1 Baza podataka, DBMS, model podataka
  - 1.1.2 Ciljevi uvođenja baza podataka
  - 1.1.3 Arhitektura baze podataka
  - 1.1.4 Jezici za rad s bazama podataka
  - 1.1.5 Poznati softverski paketi za rad s bazama podataka
- 1.2 Životni ciklus baze podataka
  - 1.2.1 Analiza potreba
  - 1.2.2 Modeliranje podataka
  - 1.2.3 Implementacija
  - 1.2.4 Testiranje
  - 1.2.5 Održavanje

### 2 MODELOVANJE PODATAKA

- 2.1 Modelovanje entiteta i veza
  - 2.1.1 Entiteti i atributi
  - 2.1.2 Veze
  - 2.1.3 Prikaz ER-šeme pomoću dijagrama
  - 2.1.4 Složenije veze
- 2.2 Relacioni model
  - 2.2.1 Relacija, atribut, n-torka, ključ
  - 2.2.2 Pretvaranje ER-šeme u relaciju
  - 2.2.3 Relacioni model, mrežni i hijerarhijski
- 2.3 Normalizacija relacione šeme
  - 2.3.1 Funkcionalna zavisnost
  - 2.3.2 Druga normalna forma
  - 2.3.3 Treća normalna forma
  - 2.3.4 Boyce-Codd -ova normalna forma
  - 2.3.5 Višeznačna zavisnost i četvrta normalna forma
  - 2.3.6 Razlozi zbog kojih se može odustati od normalizacije

### 3 JEZICI ZA RELACIONE BAZE PODATAKA

- 3.1 Relaciona algebra
  - 3.1.1 Skupovni operatori
  - 3.1.2 Selekcija
  - 3.1.3 Projekcija
  - 3.1.4 Kartezijev produkt
  - 3.1.5 Prirodni spoj
  - 3.1.6 Theta-spoj
  - 3.1.7 Delenje
  - 3.1.8 Spoljni spoj
- 3.2 Relacioni račun

3.3 Jezik SQL

3.3.1 Postavljanje upita

3.3.2 Ažuriranje relacija

3.4 Optimizacija upita

3.4.1 Odnos između relacione algebre i računa

3.4.2 Osnovna pravila za optimizaciju

## **II**

### **4 MS ACCESS**

4.1 Uvod

4.2 Tabele (Tables)

4.3 Forme (Forms)

4.4 Upiti (Querys)

4.5 Izveštaji (Reports)

# I

## 1 UVOD U BAZE PODATAKA

### 1.1 Osnovni pojmovi vezani uz baze podataka

Baze podataka predstavljaju viši nivo rada s podacima u odnosu na klasične programske jezike. Reč je o tehnologiji koja je nastala s namerom da se uklone slabosti tradicionalne “automatske obrade podataka”. Ta tehnologija osigurala je veću produktivnost, kvalitet i pouzdanost u razvoju aplikacija koje se svode na pohranjivanje i pretraživanje podataka u računar.

#### 1.1.1 Baza podataka, DBMS, model podataka

**Baza podataka** je skup međusobno povezanih podataka, pohranjenih u spoljnoj memoriji računara. Podaci su istovremeno dostupni raznim korisnicima i aplikacionim programima. Ubacivanje, promena, brisanje i čitanje podataka obavlja se posredstvom zajedničkog softvera. Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka; umesto toga koriste logičku strukturu baze.

**Sistem za upravljanje bazom podataka** (Data Base Management System - DBMS) je server baze podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom. Takođe, on obavlja u ime klijenata sve operacije s podacima. Dalje, on je u stanju podržati razne baze, od kojih svaka može imati svoju logičku strukturu, no u skladu s istim modelom. Isto tako, brine se za sigurnost podataka, te automatizuje administrativne poslove s bazom. Podaci u bazi su logički organizovani u skladu s nekim **modelom podataka**. Model podataka je skup pravila koja određuju kako može izgledati logička struktura baze. Model čini osnovu za koncipiranje, projektovanje i implementaciju baze. Dosadašnji DBMS -ovi obično su podržavali neki od sledećih modela:

**Relacioni model** - zasnovan na matematičkom pojmu relacije. I podaci i veze među podacima prikazuju se “pravougaonim” tabelama.

**Mrežni model** - baza je predložena usmerenim grafom. Čvorovi su tipovi zapisa, a lukovi definišu veze među tipovima zapisa.

**Hijerarhijski model** - specijalni slučaj mrežnog. Baza je predložena jednim stablom ili skupom stabala. Čvorovi su tipovi zapisa, a hijerarhijski odnos “nadređeni-podređeni” izražava veze među tipovima zapisa.

**Objektni model** - inspirisan je objektno-orijentisanim programskim jezicima. Baza je skup trajno pohranjenih objekata koji se sastoje od svojih internih podataka i “metoda” (operacija) za rukovanje s tim podacima. Svaki objekt pripada nekoj klasi. Između klasa

se uspostavljaju veze nasleđivanja, agregacije, odnosno međusobnog korišćenja operacija.

Od 1980 -tih godina pa sve do današnjih dana prevladava relacioni model.

### 1.1.2 Ciljevi uvođenja baza podataka

Pomenuto je da baze podataka predstavljaju viši nivo rada s podacima u odnosu na klasične programske jezike. Taj viši nivo rada ogleda se u tome što tehnologija baza podataka nastoji ispuniti sledeće ciljeve:

**Fizička nezavisnost podataka.** Razdvaja se logička definicija baze od njene stvarne fizičke građe. Znači, ako se fizička građa promeni (npr, podaci se prepisu u druge datoteke na drugim diskovima), to neće zahtevati promene u postojećim aplikacijama.

**Logička nezavisnost podataka.** Razdvaja se globalna logička definicija cele baze podataka od lokalne logičke definicije za jednu aplikaciju. Znači, ako se logička definicija promeni (npr, uvede se novi zapis ili veza), to neće zahtevati promene u postojećim aplikacijama. Lokalna logička definicija obično se svodi na izdvajanje samo nekih elemenata iz globalne definicije, uz neke jednostavne transformacije tih elemenata.

**Fleksibilnost pristupa podacima.** U starijim mrežnim i hijerarhijskim bazama, staze pristupanja podacima bile su unapred definisane, dakle korisnik je mogao pretraživati podatke jedino onim redosledom koji je bio predviđen u vreme projektovanja i implementiranja baze. Danas se zahteva da korisnik može slobodno prebirati po podacima, te po svom nahođenju uspostavljati veze među podacima. Ovaj zahtev zadovoljavaju jedino relacione baze.

**Istovremeni pristup do podataka.** Baza mora omogućiti da veći broj korisnika istovremeno koristi iste podatke. Pritom ti korisnici ne smeju ometati jedan drugoga, te svaki od njih treba da ima utisak da sam radi s bazom.

**Čuvanje integriteta.** Nastoji se automatski sačuvati korektnost i konzistencija podataka, i to u situaciji kad postoje greške u aplikacijama, te konfliktne istovremene aktivnosti korisnika.

**Mogućnost oporavka nakon kvara.** Mora postojati pouzdana zaštita baze u slučaju kvara hardvera ili grešaka u radu sistemskog softvera.

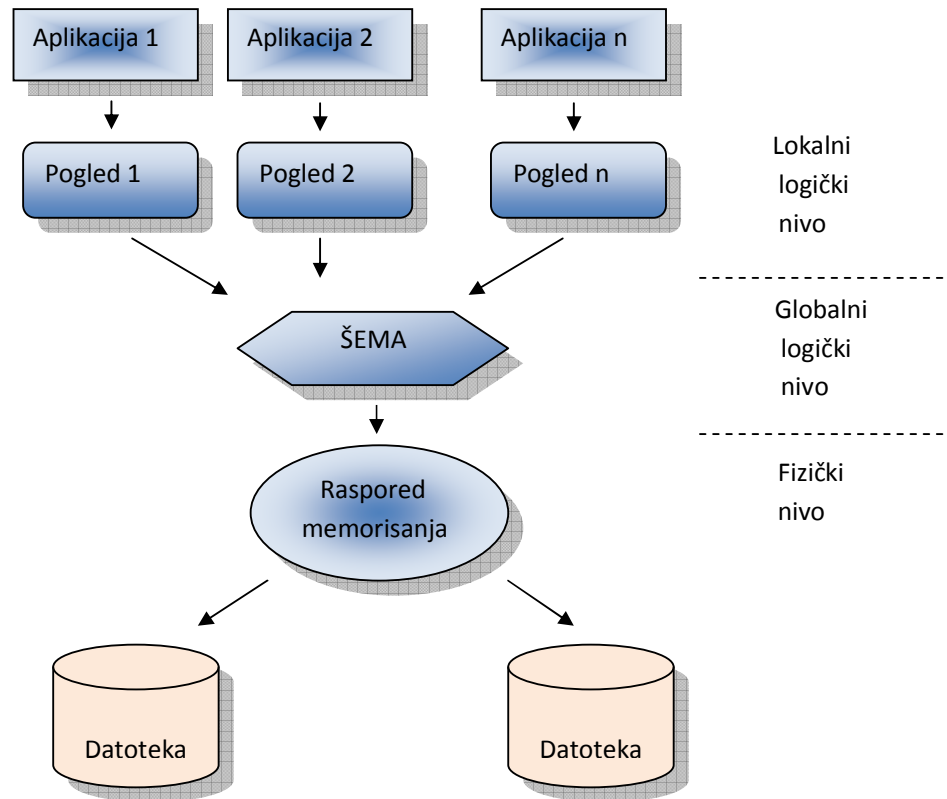
**Zaštita od neovlašćenog korišćenja.** Mora postojati mogućnost da se korisnicima ograniče prava korišćenja baze, dakle da se svakom korisniku regulišu ovlašćenja šta on može a šta ne raditi s podacima.

**Zadovoljavajuća brzina pristupa.** Operacije s podacima moraju se odvijati dovoljno brzo, u skladu s potrebama određene aplikacije. Na brzinu pristupa može se uticati odabirom pogodnih fizičkih struktura podataka, te izborom pogodnih algoritama za pretraživanje.

**Mogućnost podešavanja i kontrole.** Velika baza zahteva stalnu brigu: praćenje performansi, menjanje parametara u fizičkoj građi, rutinsko pohranjivanje rezervnih kopija podataka, regulisanje ovlašćenja korisnika. Takođe, svrha baze se vremenom menja, pa povremeno treba podesiti i logičku strukturu. Ovakvi poslovi moraju se obavljati centralizovano. Odgovorna osoba zove se **administrator** baze podataka. Administratoru treba da stoje na raspolaganju razni alati i pomagala.

### 1.1.3 Arhitektura baze podataka

Arhitektura baze podataka sastoji se od tri “sloja” i interfejsa (sučelja) među slojevima, kao što je prikazano na slici 1.1. Reč je o tri nivoa apstrakcije.



Slika 1.1 Arhitektura baze podataka

**Fizički nivo** odnosi se na fizički prikaz i raspored podataka na jedinicama spoljne memorije. To je aspekt kojeg vide samo sistemski programeri (oni koji su razvili DBMS). Sam fizički nivo može se dalje podeliti na više pod-nivoa apstrakcije, od sasvim konkretnih staza i cilindara na disku, do već donekle apstraktnih pojmova datoteke i zapisa kakve susrećemo u klasičnim programskim jezicima.

**Raspored memorisanja** opisuje kako se elementi logičke definicije baze preslikavaju na fizičke uređaje.

**Globalni logički nivo** odnosi se na logičku strukturu cele baze. To je aspekt kojeg vidi projektant baze odnosno njen administrator. Zapis logičke definicije naziva se **šema** (engleski schema). Šema je tekst ili dijagram koji definiše logičku strukturu baze, i u skladu je sa zadatim modelom. Dakle imenuju se i definišu svi tipovi podataka i veze među tim tipovima, u skladu s pravilima korišćenog modela. Takođe, šema uvodi i ograničenja kojima se čuva integritet podataka.

**Lokalni logički nivo** odnosi se na logičku predodžbu o delu baze kojeg koristi pojedina aplikacija. To je aspekt kojeg vidi korisnik ili aplikacioni programer. Zapis jedne lokalne logičke definicije zove se **pogled** (engleski view) ili pod-šema. To je tekst ili dijagram kojim se imenuju i definišu svi lokalni tipovi podataka i veze među tim tipovima, opet u skladu s pravilima korišćenog modela. Takođe, pogled zadaje preslikavanje kojim se iz globalnih podataka i veza izvode lokalni.

Za stvaranje baze podataka potrebno je zadati samo šemu i poglede. DBMS tada automatski generiše potrebni raspored memorisanja i fizičku bazu. Administrator može samo donekle uticati na fizičku građu baze, podešavanjem njemu dostupnih parametara. Programi i korisnici ne pristupaju direktno fizičkoj bazi, već dobijaju ili pohranjuju podatke posredstvom DBMS -a. Komunikacija programa odnosno korisnika s DBMS -om obavlja se na lokalnom logičkom nivou.

#### 1.1.4 Jezici za rad s bazama podataka

Komunikacija korisnika odnosno aplikacionog programa i DBMS -a odvija se pomoću posebnih jezika. Ti jezici tradicionalno se dele na sledeće kategorije:

**Jezik za opis podataka** (Data Description Language - DDL). Služi projektantu baze ili administratoru u svrhu zapisivanja šeme ili pogleda. Dakle tim jezikom definišemo podatke i veze među podacima, i to na logičkom nivou. Koji put postoji posebna varijanta jezika za šemu, a posebna za poglede. Naredbe DDL obično podsećaju na naredbe za definisanje složenih tipova podataka u jezicima poput COBOL, PL/I, C, Pascal.

**Jezik za manipulisanje podacima** (Data Manipulation Language - DML). Služi programeru za uspostavljanje veze između aplikacionog programa i baze. Naredbe DML omogućuju "manevrisanje" po bazi, te jednostavne operacije kao što su upis, promena, brisanje ili čitanje zapisa. U nekim softverskim paketima, DML je zapravo biblioteka potprograma: "naredba" u DML svodi se na poziv potprograma. U drugim paketima zaista se radi o posebnom jeziku: programer tada piše program u kojem su izmešane naredbe dvaju jezika, pa takav program treba prevoditi s dva prevodioca (DML-precompiler, obični compiler).

**Jezik za postavljanje upita** (Query Language - QL). Služi neposrednom korisniku za interaktivno pretraživanje baze. To je jezik koji podseća na govorni (engleski) jezik. Naredbe su neproceduralne, dakle takve da samo specifikuju rezultat koga želimo dobiti, a ne i postupak za dobijanje rezultata.

Ovakva podela na tri jezika danas je već prilično zastarela. Naime, kod relacionih baza postoji tendencija da se sva tri jezika objedine u jedan sveobuhvatni. Primer takvog **integrisanog** jezika za relacione baze je SQL: on služi za definisanje podataka, manipulisanje i pretraživanje. Integrisani jezik se može koristiti interaktivno (preko on-line interpretera) ili se on može pojavljivati uklopljen u aplikacione programe.

Naglasimo da gore spomenute vrste jezika nisu programski jezici. Dakle ti jezici su nam nužni da bi se povezali s bazom, no oni nam nisu dovoljni za razvoj aplikacija koje će nešto raditi s podacima iz baze.

Tradicionalni način razvoja aplikacija koje rade s bazom je korišćenje klasičnih programskih jezika (COBOL, PL/I, C, Pascal . . . ) s ugneždenim DML-naredbama. U 80-tim godinama 20. veka bili su dosta popularni i tzv. **jezici 4. generacije** (4-th Generation

Languages - 4GL): reč je o jezicima koji su bili namenjeni isključivo za rad s bazama, te su zato u tom kontekstu bili produktivniji od klasičnih programskih jezika opšte namene. Problem s jezicima 4. generacije je bio u njihovoj nestandardnosti: svaki od njih je u pravilu bio deo nekog određenog softverskog paketa za baze podataka, te se nije mogao koristiti izvan tog paketa (baze).

U današnje vreme, aplikacije se najčešće razvijaju u standardnim **objektno orijentisanim** programskim jezicima (Java, C++, . . .). Za interakcije s bazom koriste se unapred pripremljene klase objekata. Ovakva tehnika je dovoljno produktivna zbog korišćenja gotovih klasa, a rezultirajući program se lako doteruje, uklapa u veće sisteme ili prenosi s jedne baze na drugu.

### 1.1.5 Poznati softverski paketi za rad s bazama podataka

Baze podataka se u pravilu realizuju korišćenjem nekog od proverenih softverskih paketa. Tabela prikaz 1.1 daje pregled softvera koji u ovom trenutku predstavljaju tehnološki vrh te imaju značajan udeo na svetskom tržištu.

Proizvođač	Produkt	Operativni sistem	Jezici
IBM Corporation	DB2	Linux, UNIX (razni), MS Windows NT/2000/XP, VMS, MVS, VM, OS/400	SQL, COBOL, Java, . . .
Oracle Corporation	Oracle	MS Windows (razni), Mac OS, UNIX (razni), Linux i drugi	SQL, Java i drugi
IBM Corporation (prije : Informix Software Inc.)	Informix	UNIX (razni), Linux, MS Windows NT/2000/XP	SQL, Java i drugi
Microsoft	MS SQL Server	MS Windows NT/2000/XP	SQL, C++, . . .
MySQL AB	MySQL	Linux, UNIX (razni), MS Windows (razni), Mac OS	SQL, C, PHP, . . .
Sybase Inc.	Sybase SQL Server	MS Windows NT/2000, OS/2, Mac, UNIX (razni), UNIXWare	SQL, COBOL, . . .
Hewlett Packard Co.	Allbase/SQL	UNIX (HP-UX)	SQL, 4GL, C, . . .
Cincom Systems Inc.	Supra	MS Windows NT/2000, Linux, UNIX (razni), VMS, MVS, VM	SQL, COBOL, . . .
Microsoft Corporation	MS Access	MS Windows (razni)	Access Basic, SQL



Tabela 1.1 Softverski paketi za rad sa bazama

Gotovo svi današnji softverski paketi podržavaju relacioni model i SQL. Svaki od njih sadrži svoj DBMS, uobičajene klijente (npr interaktivni interpreter SQL), te biblioteke i alate za razvoj aplikacija. Svaki paket isporučuje se u verzijama za razne računarske platforme (operacione sisteme). Konkurencija među proizvođačima softvera za baze podataka je izuzetno velika, tako da je poslednjih godina često dolazilo do njihovog nestanka, spajanja ili preuzimanja. Lista relevantnih softverskih paketa zato je svake godine sve kraća. Jedino osveženje predstavlja nedavna pojava public-domain softvera poput MySQL.

## 1.2 Životni ciklus baze podataka

Uvođenje baze podataka u neko preduzeće ili ustanovu predstavlja složeni zadatak koji zahteva timski rad stručnjaka raznih profila. To je projekt koji se može podeliti u pet faza: analiza potreba, modelovanje podataka, implementacija, testiranje i održavanje.

### 1.2.1 Analiza potreba

Proučavaju se tokovi informacija u preduzeću. Uočavaju se **podaci** koje treba memorisati i veze među njima. U velikom preduzećima, gde postoje razne grupe korisnika, pojaviće se razni “pogledi” na podatke. Te poglede treba uskladiti tako da se eliminiše redundanca i nekonzistentnost. Na primer, treba u raznim pogledima prepoznati sinonime i homonime, te uskladiti terminologiju.

Analiza potreba takođe treba da obuhvati analizu **transakcija** (operacija) koje će se obavljati nad bazom podataka, budući da to može isto imati uticaja na sadržaj i konačni oblik baze. Važno je proceniti frekvenciju i opseg pojedinih transakcija, te zahteve na performanse.

Rezultat analize je dokument (pisan neformalno u prirodnom jeziku) koji se zove **specifikacija potreba**.

### 1.2.2 Modelovanje podataka

Različiti pogledi na podatke, otkriveni u fazi analize, sintetizuju se u jednu celinu - globalnu šemu. Precizno se utvrđuju tipovi podataka. Šema se dalje doteruje (“normalizuje”) tako da zadovolji neke zahteve kvaliteta. Takođe, šema se prilagođava ograničenjima koje postavlja zadati model podataka, te se dodatno modifikuje da bi bolje mogla udovoljiti zahtevima na performanse. Na kraju se iz šeme izvode pogledi (pod-šeme) za pojedine aplikacije (grupe korisnika).

### 1.2.3 Implementacija

Na osnovu šeme i pod-šema, te uz pomoć dostupnog DBMS-a, fizički se realizuje baza podataka na računaru. U DBMS-u obično postoje parametri kojima se može uticati na fizičku organizaciju baze. Parametri se podešavaju tako da se osigura efikasan rad

najvažnijih transakcija. Razvija se skup programa koji realizuju pojedine transakcije te pokrivaju potrebe raznih aplikacija. Baza se inicijalno puni podacima.

#### 1.2.4 Testiranje

Korisnici eksperimentalno rade s bazom i proveravaju da li ona zadovoljava sve zahteve. Nastoje se otkriti greške koje su se mogle potkrasti u svakoj od faza razvoja: dakle u analizi potreba, modelovanju podataka, implementaciji. Greške u ranijim fazama imaju teže posledice. Na primer, greška u analizi potreba uzrokuje da transakcije možda korektno rade, no ne ono što korisnicima treba već nešto drugo.

Dobro bi bilo kad bi takve propuste otkrili pre implementacije. Zato se u novije vreme, pre prave implementacije, razvijaju i približni **prototipovi** baze podataka, te se oni pokazuju korisnicima.

Jeftinu izradu prototipova omogućuju jezici 4. generacije i objektno-orjentisani jezici.

#### 1.2.5 Održavanje

Odvija se u vreme kad je baza već ušla u redovnu upotrebu. Sastoji se od sledećeg:

- ✚ popravak grešaka koje nisu bile otkrivene u fazi testiranja,
- ✚ uvođenje promena zbog novih zahteva korisnika,
- ✚ podešavanje parametara u DBMS u svrhu poboljšavanja performansi.

Održavanje zahteva da se stalno prati rad s bazom, i to tako da to praćenje ne ometa korisnike. Administratoru baze podataka treba da stoje na raspolaganju odgovarajući alati (utility programi).

## 2 MODELOVANJE PODATAKA

### 2.1 Modelovanje entiteta i veza

Kako oblikovati šemu za bazu podataka, usklađenu s pravilima relacionog modela? U stvarnim situacijama dosta je teško direktno pogoditi relacionu šemu. Zato se služmo jednom pomoćnom fazom koja se zove **modelovanje entiteta i veza** (Entity-Relationship Modelling). Reč je o oblikovanju jedne manje precizne, konceptualne šeme, koja predstavlja apstrakciju realnog sveta.

Ta tzv. ER-šema se dalje, više-manje automatski, pretvara u relacionu. Modelovanje entiteta i veza zahteva da se svet posmatra preko tri kategorije:

- ☒ **entiteti** : objekti ili događaji koji su nam od interesa;
- ☒ **veze** : odnosi među entitetima koji su nam od interesa;
- ☒ **atributi** : svojstva entiteta i veza koja su nam od interesa.

#### 2.1.1 Entiteti i atributi

**Entitet** je nešto o čemu želimo pamtit i podatke, nešto što je u stanju postojati ili ne postojati, te se može identifikovati. Entitet može biti objekt ili biće (na primer kuća, student, auto), odnosno događaj ili pojava (na primer utakmica, praznik, servisiranje auta).

Entitet je opisan **atributima** (na primer atributi kuće su: adresa, broj spratova, boja fasade, . . . ).

Ukoliko neki atribut i sam zahteva svoje atribute, tada ga radije treba smatrati novim entitetom (na primer model auta). Isto pravilo vredi i ako atribut može istovremeno imati više vrednosti (na primer kvar koji je popravljen pri servisiranju auta).

Ime entiteta, zajedno sa pripadnim atributima, zapravo određuje **tip** entiteta. Može postojati mnogo **primeraka** (pojava, instanci) entiteta zadatog tipa (na primer *STUDENT* je tip čiji primerci su Petrović Petar, Marković Marko, . . . ).

**Kandidat za ključ** je atribut, ili skup atributa, čije vrednosti jednoznačno određuju primerak entiteta zadatog tipa. Dakle, ne mogu postojati dva različita primerka entiteta istog tipa s istim vrednostima kandidata za ključ. (Na primer za tip entiteta *AUTO*, kandidat za ključ je atribut *REG\_BROJ*). Ukoliko jedan tip entiteta ima više kandidata za ključ, tada biramo jednog od njih i proglašavamo ga **primarnim ključem**. (Na primer primarni ključ za tip entiteta *STUDENT* mogao bi biti atribut *BROJ\_INDEKSA*.)

#### 2.1.2 Veze

**Veze** se uspostavljaju između dva ili više tipova entiteta (na primer veza *IGRA\_ZA* između tipova entiteta *IGRAČ* i *TIM* ). Zapravo je reč o imenovanoj binarnoj ili *k*-narnoj

relaciji između primeraka entiteta zadatih tipova. Za sada ćemo se ograničiti na veze između tačno dva tipa entiteta.

**Funkcionalnost** veze može biti:

- a) **Jedan-ka-jedan** (1 : 1). Jedan primerak prvog tipa entiteta može biti u vezi s najviše jednim primerkom drugog tipa entiteta, te takođe jedan primerak drugog tipa može biti u vezi s najviše jednim primerkom prvog tipa. Na primer veza *JE\_ŠEF* između tipova entiteta RADNIK i ORGANIZACIONA\_JEDINICA.
- b) **Jedan-ka-mnogo** (1 : N). Jedan primerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primeraka drugog tipa entiteta, no jedan primerak drugog tipa može biti u vezi s najviše jednim primerkom prvog tipa. Na primer veza *JE\_ČLAN* između tipova entiteta ODELJENJE i UČENIK.
- c) **Mnogo-ka-mnogo** (M : N). Jedan primerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primeraka drugog tipa entiteta, te takođe jedan primerak drugog tipa može biti u vezi s 0, 1 ili više primeraka prvog tipa. Na primer veza *POLAŽE* između tipova entiteta STUDENT i ISPIT.

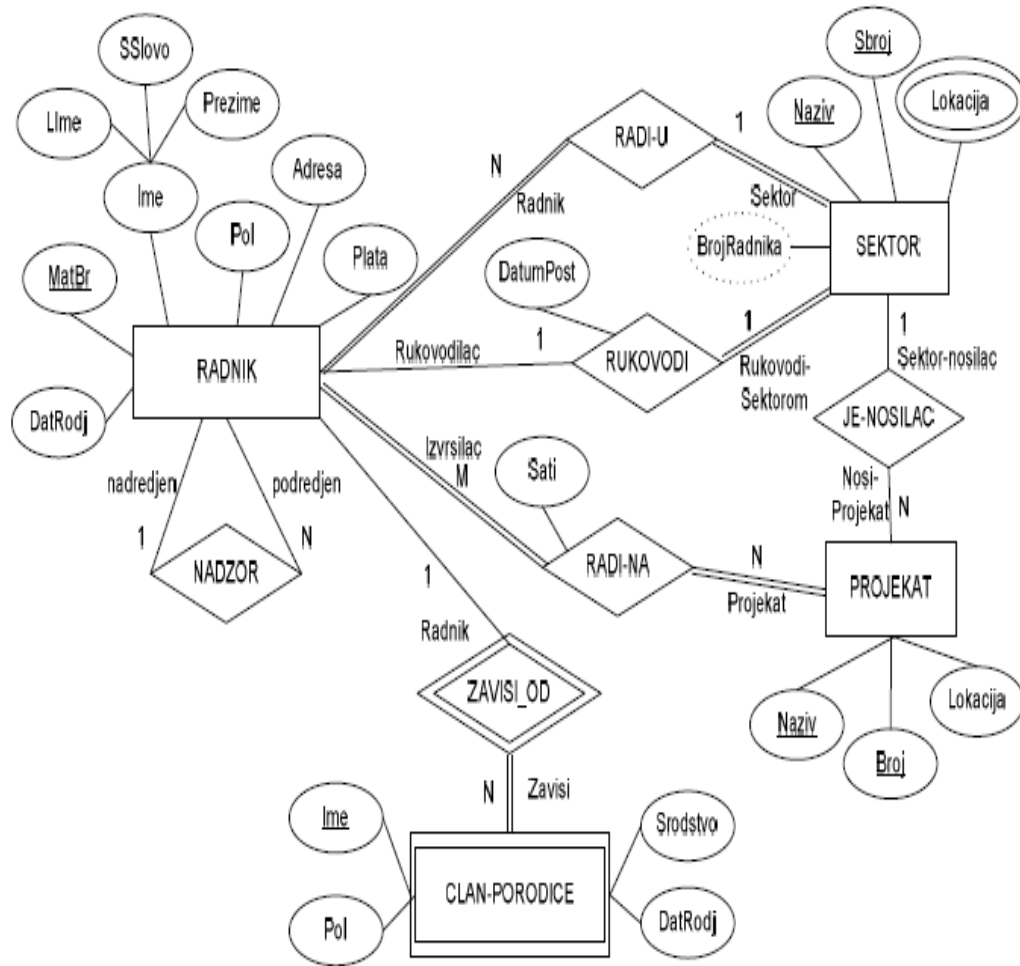
Veza može imati i svoje atribute koje ne možemo pripisati ni jednom od tipova entiteta (na primer veza *UPISAO* može imati atribut *DATUM UPISA*).

Ako svaki primerak entiteta nekog tipa mora sudelovati u zadatoj vezi, tada kažemo da tip entiteta ima **obavezno članstvo** u toj vezi. Inače tip entiteta ima neobavezno članstvo. (Na primer između tipova entiteta NASTAVNIK i PREDMET zadata je veza *PREDAJE*, koja ima funkcionalnost (M : N). PREDMET ima obavezno članstvo u vezi, jer svaki predmet mora neko predavati, a NASTAVNIK nema obavezno članstvo – može biti bibliotekar ili sl.)

### 2.1.3 Prikaz ER-šeme pomoću dijagrama

Običaj je da se ER-šema nacrtava kao dijagram u kojem pravougaonici predstavljaju tipove entiteta, a rombovi veze. Veze su povezane potezima s odgovarajućim tipovima entiteta. Imena tipova entiteta i veza, te funkcionalnost veza, uneseni su u dijagram. Posebno se prilaže lista atributa za svaki entitet odnosno vezu. U toj listi možemo specifikovati obaveznost članstva u vezama.

Kao primer, pogledajmo dijagram na Slici 2.1 koji prikazuje bazu podataka PREDUZEĆE. Pravougaonicima su predstavljeni entiteti, rombovima veze, a u eliptične oblike upisani su nazivi atributa. Podvučeni atributi su ključevi. Obavezno članstvo je naznačeno dupliranim potezom, a slabi tip entiteta dvostrukim pravougaonikom.



Slika 2.1 Baza podataka PREDUZEĆE

#### 2.1.4 Složenije veze

U stvarnim situacijama pojavljuju se i složenije veze od onih koje smo do sada posmatrali. Neke od njih su:

**Involuirana veza** povezuje jedan tip entiteta s tim istim tipom. Dakle reč je o binarnoj relaciji između raznih primeraka entiteta istog tipa. Funkcionalnost takve veze opet može biti (1 : 1), (1 : N), odnosno (M : N). Na slici 2.1 se može uočiti ovakva veza NADZOR.

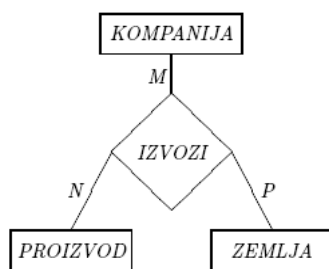
**Pod-tipovi (slabi tipovi entiteta)** Tip entiteta  $E_1$  je podtip tipa entiteta  $E_2$  ako je svaki primjerak od  $E_1$  takođe i primjerak od  $E_2$ .  $E_1$  nasleđuje sve atribute od  $E_2$ , no  $E_1$  može imati i dodatne atribute. Situaciju opisujemo pomoću specijalne (1 : 1) veze  $JE$  (engleski  $IS\_A$ ) ili  $ZAVISI\_OD$  koja se može pojaviti više puta unutar ER-šeme. Na slici 2.1 slabi tip entiteta je ČLAN-PORODICE.

**Ternarne veze** uspostavljaju se između tri tipa entiteta. Znači reč je o ternarnoj relaciji između primeraka triju tipova entiteta. Postoje brojne mogućnosti za funkcionalnost ternarne veze, na primer (N : M : P), (1 : N : M), (1 : 1 : N) ili čak (1 : 1 : 1).

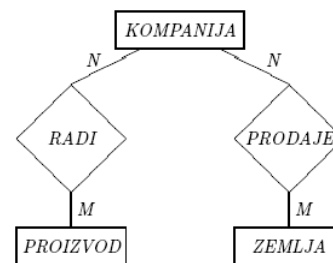
Primer ternarne veze sa Slike 2.2 odnosi se na podatke o kompanijama, proizvodima koje one proizvode i zemljama u koje one izvoze svoje proizvode. Funkcionalnost ove veze je mnogo-ka-mnogo-ka-mnogo, dakle ( $N : M : P$ ), jer na primer za zadani par (kompanija, proizvod) postoji mnogo zemalja u koje ta kompanija izvozi taj proizvod, itd.

Ternarnu vezu uvodimo samo onda kad se ona ne može rastaviti na dve binarne. Uzmimo da u primeru sa Slike 2.2 vredi pravilo: ako kompanija izvozi u neku zemlju, tada ona odmah izvozi sve svoje proizvode u tu zemlju. Uz ovo pravilo, razmatrana ternarna veza može se zameniti s dve binarne, u skladu s dijagramom na Slici 2.3.

ER model dovoljno je jednostavan da ga ljudi različitih struka mogu razumeti. Zato ER šema služi za komunikaciju projektanta baze podataka i korisnika, i to u najranijoj fazi razvoja baze. Postojeći DBMS ne mogu direktno implementirati ER šemu, već zahtevaju da se ona detaljnije razradi, te modifikuje u skladu s pravilima relacionog, mrežnog, odnosno hijerarhijskog modela.



Slika 2.2 Ternarna veza



Slika 2.3 Razbijanje ternarne na dve binarne veze

## 2.2 Relacioni model

**Relacioni model** bio je teoretski zasnovan još krajem 60-tih godina 20. veka, u radovima E.F.Codd-a. Model se dugo pojavljivao samo u akademskim raspravama i knjigama. Prve realizacije na računaru bile su suviše spore i neefikasne. Zahvaljujući intenzivnom istraživanju, te napretku samih računara, efikasnost relacionih baza postepeno se poboljšavala. Sredinom 80-tih godina 20. veka relacioni model je postao prevladavajući. I danas većina DBMS koristi taj model.

### 2.2.1 Relacija, atribut, n-torka, ključ

Relacioni model zahteva da se baza podataka sastoji od skupa pravougaonih tabela - tzv. **relacija**. Svaka relacija ima svoje ime po kojem je razlikujemo od ostalih u istoj bazi. Jedna kolona relacije obično sadrži vrednost jednog atributa (za entitet ili vezu) - zato kolonu poistovećujemo s **atributom** i obratno. Atribut ima svoje ime po kojem ga razlikujemo od ostalih u istoj relaciji. Vrednosti jednog atributa su podaci istog tipa. Dakle, definisan je skup dozvoljenih vrednosti za atribut, koji se zove **domen** atributa.

Vrednost atributa mora biti jednostruka i jednostavna (ne da se rastavi na delove). Pod nekim uslovima tolerišemo situaciju da vrednost atributa nedostaje (nije upisana). Jedan red relacije (tabele) obično predstavlja jedan primerak entiteta, ili beleži vezu između dva ili više primeraka. Red nazivamo **n-torka**. U jednoj relaciji ne smeju postojati dve jednake n-torke. Broj atributa je **stepen** relacije, a broj n-torki je **kardinalnost** relacije.

Kao primer, navedimo u Tabelarnom prikazu 2.1 relaciju *AUTO*, s atributima *REG\_BROJ*, *PROIZVOĐAČ*, *MODEL*, *GODINA*. Relacija sadrži podatke o automobilima koji se nalaze na popravku u nekoj radionici.

<b>AUTO</b>			
<i>REG_BROJ</i>	<i>PROIZVOĐAČ</i>	<i>MODEL</i>	<i>GODINA</i>
ZID654	Ford	Fiesta	1997
BXI930	Volkswagen	Golf	1996
COI453	Nissan	Primera	1997
ZXI675	Ford	Escort	1995
RST786	Fiat	Punto	1993
TXI521	Ford	Orion	1995
HCY675	Volkswagen	Jetta	1997

Tabelarni prikaz 2.1: Relacija s podacima o automobilima.

Relacija ne propisuje nikakav redosled svojih n-torki i atributa. Dakle, permutiranjem redova i kolona tabele dobijamo drukčiji zapis iste relacije.

Uvedena terminologija potiče iz matematike. Naime, neka je  $R$  relacija stepena  $n$  i neka su domeni njenih atributa redom  $D_1, D_2, \dots, D_n$ . Tada se  $R$  može interpretirati kao podskup Kartezijevog produkta  $D_1 \times D_2 \times \dots \times D_n$ . Znači, naš pojam relacije odgovara matematičkom pojmu  $n$ -narne relacije. U komercijalnim DBMS, umesto "matematičkih" termina (relacija, n-torka, atribut), češće se koriste neposredni termini (tabela - *table*, red - *row*, kolona - *column*) ili termini iz tradicionalnih programskih jezika (datoteka, zapis, polje).

**Ključ**  $K$  relacije  $R$  je podskup atributa od  $R$  koji ima sledeća "vremenski nezavisna" svojstva:

1. Vrednosti atributa iz  $K$  jednoznačno određuju n-torku u  $R$ . Dakle ne mogu u  $R$  postojati dve n-torke s istim vrednostima atributa iz  $K$ .
2. Ako izbacimo iz  $K$  bilo koji atribut, tada se narušava svojstvo 1.

Budući da su sve n-torke u  $R$  međusobno različite,  $K$  uvek postoji. Naime, skup svih atributa zadovoljava svojstvo 1. Izbacivanjem suvišnih atributa doći ćemo do podskupa koji zadovoljava i svojstvo 2.

Dešava se da relacija ima više kandidata za ključ. Tada jedan on njih proglašavamo **primarnim ključem**. Atributi koji sastavljaju primarni ključ zovu se **primarni atributi**. Vrednost primarnog atributa ne sme ni u jednoj n-torki ostati neupisana.

Građu relacije kratko opisujemo tzv. **šemom relacije**, koja se sastoji od imena relacije i popisa imena atributa u zagradama. Primarni atributi su podvučeni. Na primer, za relaciju o automobilima (Tabelarni prikaz 2.1), šema izgleda ovako:

*AUTO ( REG BROJ, PROIZVOĐAČ, MODEL, GODINA ) .*

## 2.2.2 Pretvaranje ER-šeme u relaciju

Pri pretvaranju ER-šeme u relaciju provodimo sedam sukcesivnih koraka:

### 1. preslikavanje regularnih tipova entiteta

za svaki regularni tip entiteta E u ER šemi kreira se relacija R koja sadrži sve proste attribute i sve proste komponente svih složenih atributa iz E. Jedan od ključnih atributa iz E se uzima za primarni ključ relacije R. Ako je ključni atribut u E složen, tada se njegov skup prostih atributa uzima zajedno kao primarni ključ u R. Za ER model sa slike 2.1 to su RADNIK, SEKTOR i PROJEKAT.

**RADNIK** (LIME, SSLOVO, PREZIME, MATBR, DATRODJ, POL, PLATA,  
ADRESA)

**SEKTOR** (NAZIV, SBROJ)

**PROJEKAT**(NAZIV, LOKPR, BROJPR)

### 2. preslikavanje slabih tipova entiteta

za svaki slabi tip entiteta S u ER šemi čiji je vlasnik tip entiteta E kreira se relacija R koja sadrži sve proste attribute iz S i sve proste komponente svih složenih atributa iz S. Relacija R kao spoljašnji ključ sadrži sve attribute primarnog ključa relacije tipa entiteta E. Za primarni ključ relacije R uzima se kombinacija primarnog ključa vlasnika E i parcijalnog ključa slabog tipa entiteta S ukoliko on postoji. Za model sa slike 2.1 to je CLAN-PORODICE.

**CLAN-PORODICE** (MATBRBRAD, IME, POL, SRODSTVO, DATROĐ)

### 3. preslikavanje veza 1:1

za svaki binarni tip veza V(1:1) u ER šemi identifikuju se relacije S i T koje odgovaraju tipovima entiteta koji participiraju u vezi V. Bira se jedna od ove dve relacije, recimo S i u nju se kao spoljašnji ključ uključuje primarni ključ relacije T. Za relaciju S treba birati tip entiteta koji obavezno učestvuje u vezi V. Kao attribute relacije S treba uključiti sve proste attribute i sve proste komponente složenih atributa tipa veze V.

Alternativno rešenje je formiranje zajedničke relacije R za tipove entiteta S i T u koju se uključuju svi atributi. Rešenje je dobro ako oba tipa entiteta obavezno učestvuju u vezi V i kada ne učestvuju u nekoj drugoj vezi sem V. Za primer sa slike 2.1 to je veza RUKOVODI i rešenje je da relacija SEKTOR bude sledećeg oblika

**SEKTOR** (NAZIV, SBROJ, MATBROJ, DATPOST)

### 4. preslikavanje veza 1:N

za svaki binarni tip veze V(1:N) u ER šemi identifikuje se relacija S koja učestvuje na N strani. U S se kao spoljašnji ključ uključuje primarni ključ relacije T koja predstavlja tip entiteta koji učestvuje na 1 strani. Kao attribute relacije S treba



uključiti i sve attribute veze V. Za naš primer to su veze NADZOR, RADI-U, JE-NOSILAC pa bi odgovarajuće relacije bile:

**RADNIK** (**LIME**, **SSLOVO**, **PREZIME**, **MATBR**, **DATRODJ**, **POL**, **PLATA**,  
**ADRESA**, **MATBROJ**, **BRSEK**)  
**PROJEKAT**(**NAZIV**, **LOKPR**, **BROJPR**, **BRS**)

**5. preslikavanje veza M:N**

za svaki binarni tip veze V(M:N) u ER šemi kreira se nova relacija S. U S se kao spoljašnji ključevi uključuju primarni ključevi relacija koje predstavljaju tipove entiteta koji učestvuju u V. Kombinacija ovih spoljašnjih ključeva formira primarni ključ relacije S. U relaciju S se takođe uključuju i svi atributi veze V.

**RADI-NA** (**MBR**, **BRPR**, **SATI**)

**6. preslikavanje viševrednosnih atributa**

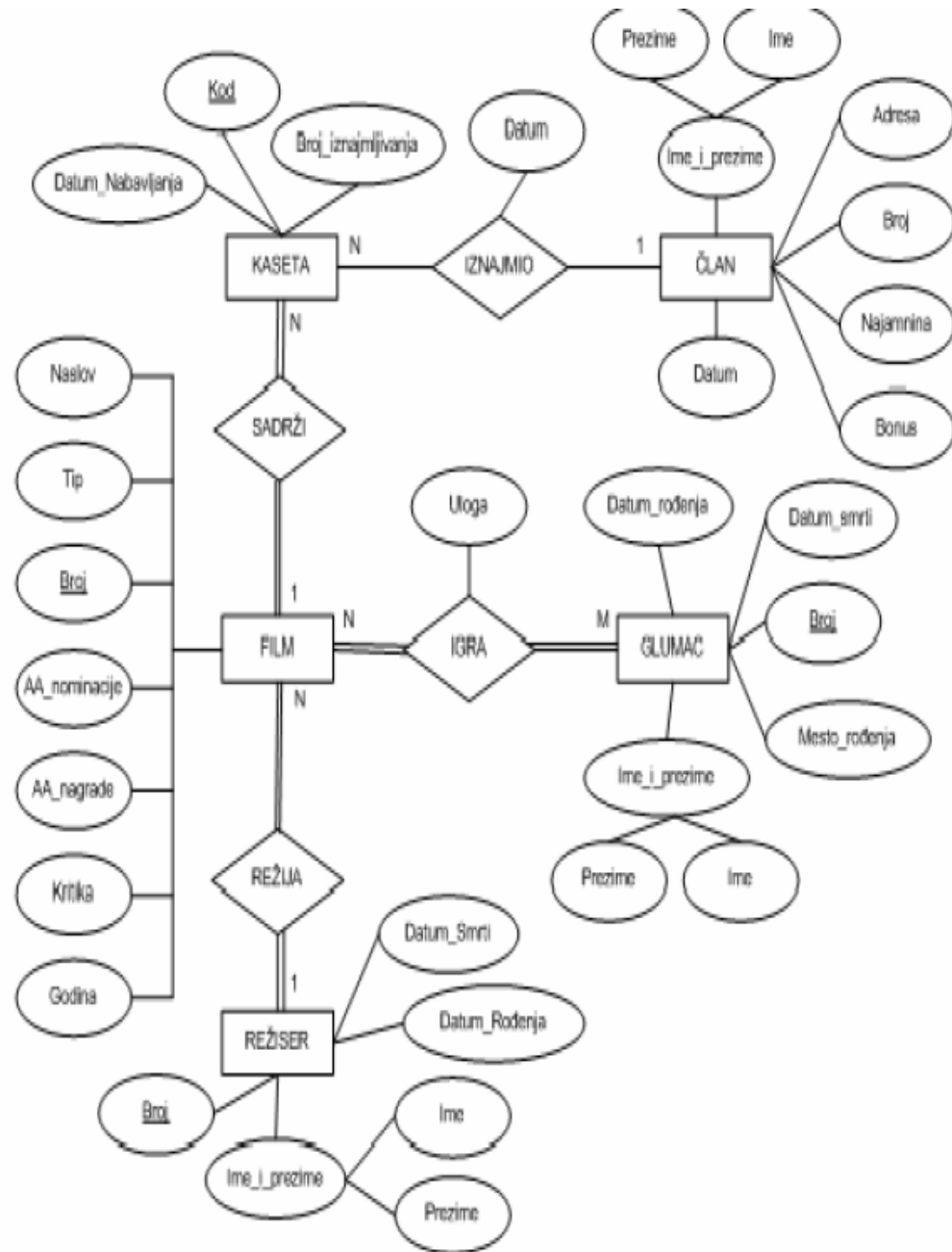
za viševrednosni atribut A tipa entiteta E ili tipa veze V kreira se nova relacija R koja sadrži atribut A i primarni ključ K relacije kojom je predstavljen tip entiteta E ili tip veze V. Za primarni ključ relacije R bira se kombinacija A i K. Ako je viševrednosni atribut složen uključuju se sve njegove proste komponente.

**LOK-SEKTOR** (**BRS**, **LOKACIJA**)

**7. preslikavanje n-arnih veza**

za svaki n-arni tip veze V ,  $n > 2$ , kreira se nova relacija S. U S se kao spoljašnji ključevi uključuju primarni ključevi relacija koje predstavljaju tipove entiteta koji participiraju u V. Kombinacija ovih spoljašnjih ključeva formira primarni ključ relacije S. U S se takođe uključuju i svi atributi n-arnog tipa veze. Ako neki tip entiteta E učestvuje u V sa ograničenjem (min, max) i ako je max=1, tada se kao primarni ključ relacije S može uzeti onaj spoljašnji ključni atribut kojim se relacija S referencira na relaciju kojom je predstavljen tip entiteta E.

Za vežbu prevesti u relcionu šemu ER dijagram kojim je predstavljena baza podataka video kluba:



Rešenje:

FILM							
BROJ	NASLOV	TIP	AA_NOM	AA_NAG	KRITIKA	GODINA	REŽISER
REŽISER							
BROJ	IME	PREZIME	DATUM_ROĐ	DATUM_SM			
ČLAN							
BROJ	IME	PREZIME	ADRESA	NAJAM	BONUS	DATUM	
GLUMAC							
BROJ	IME	PREZIME	DATUM_ROĐ	MESTO_ROĐ	DATUM_SM		
KASETA							
KOD	DATUM_NABAV	FILM	BROJ_IJNAJM	ČLAN	DATUM_IJNAJM		
IGRA							
BR_FILM	BR_GLUMAC	ULOGA					

### 2.2.3 Relacioni model, mrežni i hijerarhijski

Mrežni i hijerarhijski model su prilično slični. Ustvari, hijerarhijski model možemo smatrati specijalnom vrstom mrežnog. S druge strane, relacioni model se po svom pristupu bitno razlikuje od ostala dva.

Osnovna razlika je u načinu prikazivanja veza među entitetima, te načinu korišćenja tih veza.

✎ U mrežnom ili hijerarhijskom modelu moguće je direktno prikazati vezu. Doduše, postoje ograničenja na funkcionalnost veze, te na konfiguraciju svih veza u šemi. Veza se “materijalizuje” pomoću pointera, tj. u jednom zapisu piše adresa drugog (vezanog) zapisa. Mrežni odnosno hijerarhijski DML omogućuje samo jednostavne operacije s jednim zapisom (upis, promena, brisanje, čitanje), te “manevrisanje” kroz šemu putem veza (dohvat prvog zapisa, koji je u zadatoj vezi sa zadatim zapisom, dohvat idućeg zapisa, . . . ). Ovakav pristup ima svoje prednosti i mane. Prednost je da je rad s bazom u tehničkom pogledu brz i efikasan. Mana je da korisnik može upotrebiti samo one veze koje su predviđene šemom pa su u skladu s time i materijalizovane.

✎ U relacionom modelu veze su samo implicitno naznačene time što se isti ili sličan atribut pojavljuje u više relacija. Veza nije materijalizovana, već se dinamički uspostavlja za vreme rada s podacima, uporedbom vrednosti atributa u n-torkama raznih relacija. Relacioni DML, osim jednostavnih operacija s jednom relacijom, mora omogućiti slobodno kombinovanje podataka iz raznih relacija. I ovaj pristup ima svoje prednosti i mane. Mana je da se veza svaki put mora iznova uspostavljati; potrebno je pretraživanje podataka, a to troši vreme. Prednost je da korisnik može uspostaviti i one veze koje nisu bile predviđene u fazi modelovanja podataka. Šta više, kao kriterijum za povezivanje, osim jednakosti za vrednost atributa, mogu poslužiti i razni drugi (složeniji) kriterijumi. To još više povećava fleksibilnost korišćenja baze.

Iz upravo rečenog vidi se da je u relacionom modelu težište bačeno na dinamički aspekt (manje memorisanja a više manipulisanja). Zato upotrebljivost relacionog DBMS -a bitno zavisi o izražajnim mogućnostima njegovog jezika za rad s podacima. Poželjno je takođe da taj jezik bude u što većoj meri neproceduralan i razumljiv neposrednim korisnicima. Njih treba smatrati sastavnim delom relacionog modela.

## 2.3 Normalizacija relacione šeme

Relaciona šema, dobijena iz ER-šeme na osnovu prethodnih uputa, može sadržavati nedorečenosti koje treba otkloniti pre implementacije. Proces daljnjeg doterivanja šeme zove se **normalizacija**.

Teorija normalizacije zasnovana je na pojmu normalnih formi. Relacije dobijene u skladu s uputstvom za prevođenje u relacionu šemu morale bi u najmanju ruku biti u **prvoj normalnoj formi** (1NF). Naime, relacija je u 1NF ako je vrednost svakog atributa jednostruka i nedeljiva - to svojstvo već je bilo uključeno u našu definiciju relacije. U svojim radovima (1970-1974. godina) E.F. Codd je najpre definisao **drugu** i **treću normalnu formu** (2NF, 3NF), a zatim i poboljšanu varijantu 3NF koja se zove **Boyce-Coddova normalna forma** (BCNF). R. Fagin je 1977. i 1979. uveo **četvrtu** i **petu**

**normalnu formu** (4NF, 5NF). U praksi je lako naići na relacije koje odstupaju od 2NF, 3NF, BCNF, no vrlo retko se susreću relacije u BCNF koje nisu u 4NF i 5NF. Zato su “više” normalne forme prvenstveno od teorijskog značaja.

Teorija normalizacije nije ništa drugo nego formalizacija nekih intuitivno prihvatljivih principa o “zdravom” oblikovanju šeme. Ukoliko već na početku dobro uočimo sve potrebne entitete, attribute i veze, tada nam nikakva daljnja normalizacija neće biti potrebna. No ako je polazna relaciona šema bila loše oblikovana, tada će postupak normalizacije ispraviti te greške.

### 2.3.1 Funkcionalna zavisnost

Za zadanu relaciju  $R$ , atribut  $B$  od  $R$  je **funkcionalno zavisan** o atributu  $A$  od  $R$  (oznaka:  $A \rightarrow B$ ) ako vrednost od  $A$  jednoznačno određuje vrednost od  $B$ . Dakle ako u isto vreme postoje u  $R$  dve n-torke s jednakom vrednošću  $A$ , tada te n-torke moraju imati jednaku vrednost  $B$ . Analogna definicija primenjuje se i za slučaj kad su  $A$  i  $B$  složeni atributi (dakle skupovi atributa).

Kao primer, razmotrimo sledeću (loše oblikovanu) relaciju:

IZVEŠTAJ ( BR\_INDEKSA, BR\_ISPITA, NASLOV\_ISPITA, IME\_NASTAVNIKA,  
BR\_SOBE\_NASTAVNIKA, OCENA ) .

Pretpostavimo da svaki ispit ima jednog nastavnika, a svaki nastavnik jednu sobu. Navodimo neke od funkcionalnih zavisnosti:

$(BR\_INDEKSA, BR\_ISPITA) \rightarrow OCENA$  ,  
 $BR\_ISPITA \rightarrow NASLOV\_ISPITA$  ,  
 $BR\_ISPITA \rightarrow IME\_NASTAVNIKA$  ,  
 $BR\_ISPITA \rightarrow BR\_SOBE\_NASTAVNIKA$  ,  
 $IME\_NASTAVNIKA \rightarrow BR\_SOBE\_NASTAVNIKA$  .

Za zadanu relaciju  $R$ , atribut  $B$  od  $R$  je **potpuno funkcionalno zavisan** o (složenom) atributu  $A$  od  $R$  ako vredi:  $B$  je funkcionalno zavisan o  $A$ , no  $B$  nije funkcionalno zavisan ni o jednom pravom podskupu od  $A$ .

Svaki atribut relacije je funkcionalno zavisan o ključu, no ta zavisnost ne mora biti potpuna. Na primer  $OCENA$  je potpuno funkcionalno zavisna o primarnom ključu ( $BR\_INDEKSA, BR\_ISPITA$ ). S druge strane,  $NASLOV\_ISPITA$ ,  $IME\_NASTAVNIKA$  i  $BR\_SOBE\_NASTAVNIKA$  su **parcijalno** zavisni o ključu, budući da su zavisni samo o  $BR\_ISPITA$ , a ne i o  $BR\_INDEKSA$ .

Za  $BR\_SOBE\_NASTAVNIKA$  se kaže da je **tranzitivno** zavisan o  $BR\_ISPITA$ , budući da je zavisan o  $IME\_NASTAVNIKA$ , koji je opet zavisan o  $BR\_ISPITA$ .

Parcijalne i tranzitivne zavisnosti mogu uzrokovati probleme kod manipulisanja s podacima, pa ih je poželjno ukloniti.

### 2.3.2 Druga normalna forma

Relacija je u **drugoj normalnoj formi** (2NF) ako je u 1NF i ako je svaki ne-primarni atribut potpuno funkcionalno zavisan o primarnom ključu. Prethodno definisana relacija *IZVEŠTAJ* nije u 2NF, i to dovodi do anomalija:

- ✎ Ako u bazu želimo uneti podatke o novom ispitu, to ne možemo učiniti sve dok bar jedan student ne upiše taj ispit (naime ne smemo imati praznu vrednost primarnog atributa *BR\_INDEKSA*). Slično, ako želimo uneti podatke o novom nastavniku i njegovoj sobi, to ne možemo učiniti dok nastavnika ne zadužimo s bar jednim ispitom i dok bar jedan student ne upiše taj ispit.
- ✎ Ako npr želimo promeniti naslov ispita 361 iz “Linearna algebra” u “Vektorski prostori”, tada moramo naći sve n-torke koje sadrže tu vrednost za *BR\_ISPITA*, te promeniti vrednost za *NASLOV\_ISPITA* u svim takvim n-torkama. Biće onoliko promena koliko ima studenata koji su upisali ispit 361. Ako zaboravimo izvršiti neku od promena, imaćemo kontradiktorne podatke.
- ✎ Pretpostavimo da svi studenti koji su upisali ispit 361 odustanu od tog ispita. Ako shodno tome obrišemo odgovarajuće n-torke, iz baze će nestati svi podaci o ispitu 361.

Ove anomalije rešavamo svođenjem relacije u 2NF. Polaznu relaciju razbijamo u dve, tako da iz stare relacije prebacimo u novu sve one attribute koji su parcijalno zavisni o ključu:

IZVEŠTAJ ( *BR\_INDEKSA*, *BR\_ISPITA*, *OCENA* ) ,  
ISPIT ( *BR\_ISPITA*, *NASLOV\_ISPITA*, *IME\_NASTAVNIKA*, *BR\_SOB\_E\_NASTAVNIKA* ) .

Obe relacije su sada u 2NF. No relacija *ISPIT* zahteva dalju normalizaciju; naime u njoj još uvek postoji tzv. tranzitivna zavisnost, gde srednji atribut nije kandidat za ključ:

*BR\_ISPITA* → *IME\_NASTAVNIKA* → *BR\_SOB\_E\_NASTAVNIKA* .

Uzmimo još jedan primer za prevođenje relacije u 2NF. Neka je data relacija *R* sa sledećim sadržajem:

<i>R</i>	<i>I_SIF</i>	<i>K_SIF</i>	<i>NAZIV</i>
	i1	k1	Prosveta
	i3	k2	Dečje novine
	i3	k3	Dečje novine
	i4	k1	Matica srpska
	i5	k1	Prosveta
	i5	k2	Prosveta

Gde je *I\_SIF* šifra izdavača, *K\_SIF* šifra knjige. Neka u relaciji *R* važi funkcionalna zavisnost *I\_SIF* → *NAZIV*, i neka je to jedina funkcionalna zavisnost u toj relaciji. U relaciji *R* jedini ključ, pa i primarni ključ jeste par atributa (*I\_SIF*, *K\_SIF*).

Anomalija unošenja u relaciju *R* ogleda se u nemogućnosti unošenja podataka o šifri i nazivu izdavača dok se ne unese i podatak o bar jednoj knjizi koju taj izdavač izdaje (atribut *K\_SIF* ne može imati nedostajuću vrednost s obzirom da je deo primarnog ključa).

Anomalija brisanja ogleda se u činjenici da se brisanjem informacije o izdanju neke knjige može izbrisati, bez takve namere, i informacija o nazivu odgovarajućeg izdavača. Tako, isključenje knjige k1 iz izdanja izdavača i4 može se izvršiti samo brisanjem trojke (i4, k1, Matica srpska), čime se briše i informacija o nazivu izdavača sa šifrom i4.

Anomalija ažuriranja ogleda se u nemogućnosti nezavisnog ažuriranja vrednosti atributa NAZIV (izdavača). Na primer, ako je potrebno promeniti naziv izdavača sa šifrom i5, sa “Prosveta” na “Nova Prosveta”, onda se taj podatak ne može promeniti (“ažurirati”) u jednoj trojci relacije, već se to mora učiniti u svakoj trojci koja se odnosi na izdavača sa šifrom i5, tj. onoliko puta koliko izdanja ima taj izdavač.

Uzrok anomalija koje ispoljava relacija  $R$  je u činjenici da su tom relacijom predstavljeni jedan tip entiteta IZDAVAČ i jedan odnos između tipa entiteta IZDAVAČ i tipa entiteta KNJIGA. Entitet IZDAVAČ može se predstaviti relacijom IZDAVAČ sa atributima I\_SIF, NAZIV, itd, a odnos – relacijom IZDANJA sa atributima I\_SIF, K\_SIF, itd. Ovaj semantički uzrok anomalija sintaksno se može prepoznati u funkcionalnoj zavisnosti atributa NAZIV od atributa I\_SIF koji je pravi podskup primarnog ključa. Ovakva funkcionalna zavisnost zove se parcijalna, njeno postojanje čini da relacija  $R$  nije u drugoj normalnoj formi, a anomalije koje prouzrokuje eliminišu se pravilom po kome se relacija  $R$  zamenjuje svojim projekcijama na parove atributa (I\_SIF, NAZIV), (I\_SIF, K\_SIF):

$$R [I\_SIF, NAZIV] = IZDAVAC$$

I_SIF	NAZIV
i1	Prosveta
i3	Dečje novine
i4	Matica srpska
i5	Prosveta

$$R [I\_SIF, K\_SIF] = IZDANJA$$

I_SIF	K_SIF
i1	k1
i3	k2
i3	k3
i4	k1
i5	k1
i5	k2

Dobijene relacije nemaju pomenute anomalije, a njihovim prirodnim spajanjem po atributu I\_SIF dobija se prvobitna relacija  $R$ . Dakle, dekompozicija relacije  $R$  u dve projekcije IZDAVAC i IZDANJA je korektna, bez gubitka informacije, što je prvi i obavezni uslov pri bilo kojoj transformaciji podataka.

**Definicija:** Za funkcionalnu zavisnost  $X \rightarrow Y$  kaže se da je *potpuna* ako za svaki pravi podskup  $X'$  od  $X$  važi  $X' \rightarrow Y$ . Ako za neki skup  $X' \subsetneq X$  važi  $X' \rightarrow Y$ , funkcionalna zavisnost  $X \rightarrow Y$  zove se *parcijalna*.

**Definicija:** Atribut  $A$  relacije  $R$  je *sporedan* u toj relaciji ako ne učestvuje ni u jednom ključu te relacije; u suprotnom je atribut  $A$  *ključni* atribut. Neka je  $X$  skup svih sporednih atributa relacije  $R$ . Relacija  $R$  je u *drugoj normalnoj formi*, u oznaci 2NF, ako svaki atribut iz  $X$  potpuno funkcionalno zavisi od svakog ključa relacije  $R$ .

**Teorema (2NF normalizacija):** Ako relacija  $R(A_1, \dots, A_n)$  nije u 2NF, onda postoji dekompozicija relacije  $R$  u skup njenih projekcija koje su u 2NF, a iz kojih se operacijom prirodnog spajanja može ponovo dobiti relacija  $R$ .

**Algoritam 2NF normalizacije**

Ulaz: relacija  $R$  sa atributima  $\{A_1, \dots, A_i\}$  i skupom FZ  $F$ ;

Izlaz: skup  $R_1, \dots, R_i$  ( $i \geq 1$ ) projekcija relacije  $R$ , koje su sve u 2NF i za koje  
 važi: ako je  $i = 1$ , onda je  $R_1 \equiv R$ ; ako je  $i > 1$ , onda je  $R = R_1 * R_2 * \dots * R_i$ ;

```

BEGIN
   $i := 1$ ;
  WHILE relacija  $R$  nije u 2NF DO
    BEGIN
      uočiti parcijalnu FZ  $X \rightarrow A$ , ( $X \subseteq Atr(R)$ ;  $A \in Atr(R)$ )
        sporednog atributa  $A$  od ključa  $X$ ;
      neka je  $X = X'X''$ , gde je  $X' \rightarrow A$  potpuna FZ;
      neka je  $Z = Atr(R) \setminus (X \cup \{A\})$ ;
      zameniti relaciju  $R$  njenim projekcijama  $R[XZ]; R[X'A]$ ;
       $R_i := R[X'A]$ ;
       $i := i + 1$ ;
       $R := R[XZ]$ 
    END;
   $R_i := R$ 
END.

```

**2.3.3 Treća normalna forma**

Relacija je u **trećoj normalnoj formi** (3NF) ako je u 2NF i ako ne sadrži tranzitivne zavisnosti. Preciznije, relacija  $R$  je u 3NF ako za svaku funkcionalnu zavisnost  $X \rightarrow A$  u  $R$ , takvu da  $A$  nije u  $X$ , vredi:  $X$  sadrži ključ za  $R$  ili je  $A$  primarni atribut.

Pre navedena relacija ISPIT nije u 3NF jer imamo zavisnost  $IME\_NASTAVNIKA \rightarrow BR\_SOBE\_NASTAVNIKA$ , i pritom  $IME\_NASTAVNIKA$  nije ključ, a  $BR\_SOBE\_NASTAVNIKA$  nije primarni atribut. Ova tranzitivna zavisnost može dovesti do anomalija:

- ✎ Ne možemo uneti podatke o novom nastavniku i njegovoj sobi, sve dok ga nismo zadužili s ispitom.
- ✎ Da bi promenili broj sobe nastavnika, moramo izvršiti promenu u svakoj n-torki koja odgovara nekom ispitu kojeg predaje taj nastavnik.
- ✎ Ako nastavnik (privremeno) ne predaje ni jedan ispit, tada iz baze nestaju svi podaci o tom nastavniku i njegovoj sobi.

Da bi relaciju ISPIT prebacili u 3NF, razbijamo je u dve, i time prekidamo tranzitivnu zavisnost. Konačna relaciona šema koja zamenjuje polaznu relaciju *IZVEŠTAJ* izgleda ovako:

IZVEŠTAJ ( BR\_INDEKSA, BR\_ISPITA, OCENA ) ,  
ISPIT ( BR\_ISPITA, NASLOV\_ISPITA, IME\_NASTAVNIKA ) ,  
NASTAVNIK ( IME\_NASTAVNIKA, BR\_SOBENASTAVNIKA ) .

Relacije su sada do kraja normalizovane. Primetimo da bi konačnu šemu odmah dobili da smo u fazi modelovanja entiteta postupili ispravno, te studente, ispite i nastavnike odmah prikazali posebnim tipovima entiteta.

Razmotrimo dodatno ovaj problem. Kriterijum koji definiše drugu normalnu formu nije dovoljno strog, pa relacija koja je u 2NF i dalje može da ispoljava anomalije unošenja, brisanja i ažuriranja. Tada su anomalije posledica drugih nepoželjnih oblika funkcionalnih zavisnosti koje treba otkloniti dekompozicijom relacije bez gubljenja informacije, u skup projekcija koje su u tzv. trećoj normalnoj formi.

Primer: Neka je relacijom  $S$  ( $P\_SIF$ ,  $I\_SIF$ ,  $DRZAVA$ ) predstavljen tip entiteta IZDAVAČ (sa atributima  $I\_SIF$  i  $DRZAVA$ ) i odnos tog tipa entiteta sa tipom entiteta PISAC koji je predstavljen jedinstvenim identifikatorom  $P\_SIF$ . Neka je u relaciji  $S$  primarni ključ atribut  $P\_SIF$ , i neka u njoj važe FZ  $P\_SIF \rightarrow I\_SIF$  i  $I\_SIF \rightarrow DRZAVA$ , dok  $I\_SIF \not\rightarrow P\_SIF$  i  $DRZAVA \not\rightarrow I\_SIF$  (jedan pisac može imati samo jednog izdavača, dok jedan izdavač može izdavati dela većeg broja pisaca). Iz ovih funkcionalnih zavisnosti proističu sve ranije pomenute anomalije. Na primer, ne može se uneti informacija o državi u kojoj je registrovan novi izdavač dok se ne unese šifra bar jednog pisca čija dela taj izdavač izdaje; brisanjem poslednjeg pisca čija dela izdaje neki izdavač gubi se i informacija o državi u kojoj je izdavač registrovan; promena države datog izdavača mora se izvršiti u svakoj trojci koja se odnosi na nekog pisca tog izdavača. Sintaksno gledano, anomalije su posledica tzv. tranzitivne zavisnosti sporednog atributa  $DRZAVA$  od ključnog atributa  $P\_SIF$ , a mogu se otkloniti zamenom relacije  $S$  njenim projekcijama  $S[P\_SIF, I\_SIF]$ ,  $S[I\_SIF, DRZAVA]$ .

Za novodobijene relacije važi da se njihovim spajanjem može dobiti relacija  $S$  (zbog FZ  $I\_SIF \rightarrow DRZAVA$ ), pri čemu u njima nema tranzitivne zavisnosti pa su u tzv. trećoj normalnoj formi.

**Definicija:** Neka su  $X, Y, Z$  skupovi atributa relacije  $R(A_1, \dots, A_n)$ . Kaže se da skup atributa  $Z$  *tranzitivno zavisi od*  $X$  ako važi  $X \rightarrow Y$ ,  $Y \rightarrow Z$ .

Relacija  $R$  je u *trećoj normalnoj formi* (u oznaci 3NF) ako je u 1NF i nijedan njen sporedni atribut ne zavisi tranzitivno ni od jednog njenog ključa.

Lako se pokazuje da, ako je relacija  $R$  u 3NF, onda je ona i u 2NF. Naime, ako relacija  $R$  nije u 2NF, onda postoji sporedni atribut  $A_i$  koji je parcijalno zavis od ključa  $X$ , ali je onda sporedni atribut  $A_i$  i tranzitivno zavis od ključa  $X$  jer postoji takav skup  $X' \subset X$  da je  $X \rightarrow X'$ ,  $X' \rightarrow A_i$ . Dakle, relacija  $R$  tada nije ni u 3NF.

**Teorema (3NF normalizacija):** Ako relacija  $R(A_1, \dots, A_n)$  nije u 3NF, onda postoji dekompozicija relacije  $R$  u skup njenih projekcija koje jesu u 3NF, koje čuvaju sve polazne FZ relacije  $R$ , a iz kojih se operacijom prirodnog spajanja može ponovo dobiti relacija  $R$ .



**Algoritam 3NF normalizacije**

Ulaz: relacija  $R$  sa atributima  $\{A_1, \dots, A_n\}$  i skupom FZ  $F$ ;

Izlaz: skup  $R_1, \dots, R_m$  ( $m \geq 1$ ) projekcija relacije  $R$ , koje su sve u 3NF i za koje važi: ako je  $m = 1$ , onda je  $R_1 \equiv R$ ; ako je  $m > 1$ , onda je  $R = R_1 * R_2 * \dots * R_m$ ;

```

BEGIN
  i := 1;
  WHILE relacija R nije u 3NF, DO
    BEGIN
      uočiti tranzitivnu zavisnost  $X \rightarrow Y$ ;  $Y \rightarrow A$ ;  $Y \not\rightarrow X$  sporednog atributa
      A od ključa X, za koju je  $Y \rightarrow A$  potpuna funkcionalna zavisnost;
      neka je  $Z = Atr(R) \setminus (X \cup \{A\})$ ;
      zameniti relaciju R projekcijama  $R[YA]$ ;  $R[XZ]$ ;
       $R_i := R[YA]$ ;
       $i := i + 1$ ;
       $R := R[XZ]$ 
    END;
   $R_i := R$ 
END.

```

**2.3.4 Boyce-Codd-ova normalna forma**

**Determinanta** je atribut (ili kombinacija atributa) o kojem je neki drugi atribut potpuno funkcionalno zavisn. Relacija je u **Boyce-Codd-ovoj normalnoj formi** (BCNF) ako je svaka njena determinanta ujedno i kandidat za ključ.

Očito je relacija koja je u BCNF takođe i u 2NF i 3NF. No postoje relacije koje su u 3NF, no nisu u BCNF. Primer za to možemo konstruisati tako da gledamo relaciju u kojoj postoje dva kandidata za ključ, oba ključa su složena, i preklapaju se u bar jednom atributu.

Na primer neka na fakultetu jedan ispit predaje više nastavnika, ali svaki nastavnik predaje samo jedan ispit. Svaki student upisuje više ispita, no ima samo jednog nastavnika za zadani ispit.

Situacija se može opisati relacijom:

$UPISAO ( \underline{BR\_INDEKSA}, \underline{IME\_NASTAVNIKA}, BR\_ISPITA ) .$

Relacija nije ni u 2NF, jer postoji parcijalna zavisnost:

$IME\_NASTAVNIKA \rightarrow BR\_ISPITA.$

No mi možemo drukčije izabrati primarni ključ:

$UPISAO ( BR\_INDEKSA, BR\_ISPITA, \underline{IME\_NASTAVNIKA} ) .$

Sad je relacija u 3NF, no ne i u BCNF jer postoji zavisnost:

$$IME\_NASTAVNIKA \rightarrow BR\_ISPITA$$

i pritom *IME\_NASTAVNIKA* nije kandidat za ključ.

Zbog odstupanja od BCNF nastaju slične anomalije kao kod odstupanja od 3NF. Ne možemo evidentirati činjenicu da zadati nastavnik predaje zadati ispit, sve dok bar jedan student ne upiše taj ispit kod tog nastavnika. Takođe, veza nastavnika i ispita je zapisana s velikom redundancijom, onoliko puta koliko ima studenata, što otežava ažuriranje. Ako svi studenti odustanu, briše se evidencija da zadati nastavnik predaje zadati ispit.

Rešenje problema je, kao i pre, razbijanje relacije na dve. Prekida se nepoželjna funkcionalna zavisnost.

$$\begin{aligned} &KLASA ( BR\_INDEKSA, IME\_NASTAVNIKA ), \\ &PREDAJE ( IME\_NASTAVNIKA, BR\_ISPITA ). \end{aligned}$$

Sad su obe relacije u BCNF.

Problemi s relacijom *UPISAO* izbegli bi se da smo već u fazi modelovanja entiteta i veza uočili da zapravo imamo posla s dve nezavisne binarne veze:

☒ ( $N : M$ ) veza između *STUDENT* i *NASTAVNIK*,

☒ ( $1 : N$ ) veza između *ISPIT* i *NASTAVNIK*.

Ternarna veza je tada suvišna.

Na problem možemo gledati i ovako. Anomalije unošenja, brisanja i ažuriranja javljaju se i kod nekih relacija koje su u 3NF, kada mogu biti posledica tranzitivne zavisnosti ključnih atributa od ključa.

Sledeći primer ilustruje takav slučaj.

Primer: Neka u relaciji  $R(I\_SIF, NAZIV, K\_SIF, TIRAZ)$  važe funkcionalne zavisnosti  $\{I\_SIF, K\_SIF\} \rightarrow TIRAZ$ ,  $I\_SIF \rightarrow NAZIV$ ,  $NAZIV \rightarrow I\_SIF$ ,  $\{NAZIV, K\_SIF\} \rightarrow TIRAZ$ . Jedan ključ relacije  $R$  je par atributa  $(I\_SIF, K\_SIF)$ , a drugi ključ (kandidat za ključ) je par atributa  $(NAZIV, K\_SIF)$ . Relacija  $R$  je u 3NF, ali ispoljava anomalije. Na primer, promena naziva izdavača sa datom šifrom mora se izvršiti u onoliko  $n$ -torki koliko knjiga taj izdavač izdaje. Anomalije su posledica tranzitivne zavisnosti ključnog atributa *NAZIV* od ključa  $(I\_SIF, K\_SIF)$  (tj. funkcionalne zavisnosti  $I\_SIF \rightarrow NAZIV$  na čijoj levoj strani nije ceo ključ). Zamenom relacije  $R$  njenim projekcijama  $R [ I\_SIF, NAZIV ]$ ,  $R [ I\_SIF, K\_SIF, TIRAZ ]$  uklanjanju se anomalije, a njihovim spajanjem dobija se relacija  $R$ . Dobijene projekcije su u tzv. Boyce-Coddovoj normalnoj formi.

**Definicija:** Relacija  $R (A_1, \dots, A_n)$  je u *Boyce-Codd-ovoj normalnoj formi* (u oznaci BCNF) ako egzistencija FZ  $X \rightarrow Y$ , gde je  $X, Y \subseteq \{A_1, \dots, A_n\}$  i  $X \cap Y = \emptyset$ , povlači egzistenciju FZ  $X \rightarrow A_i$ , za svako  $i = 1, 2, \dots, n$ .

Sintaksna interpretacija definicije BCNF je sledeća: sve FZ u relaciji koja je u BCNF jesu posledice ključa. Semantička interpretacija je da nijedan entitet nije sadržan, kao pravi

podskup, u datoj relaciji; takav entitet bi se odrazio postojanjem FZ  $X \rightarrow Y$ , gde  $X$  nije ključ relacije. Ako relacija sadrži takav entitet, njega treba izdvojiti u postupku projektovanja, što se postiže dekompozicijom relacije u niz projekcija koje su u BCNF.

**Teorema (BCNF normalizacija):** *Ako relacija  $R(A_1, \dots, A_n)$  nije u BCNF, onda postoji dekompozicija relacije  $R$  u skup njenih projekcija koje su u BCNF, a iz kojih se operacijom prirodnog spajanja može ponovo dobiti relacija  $R$ .*

#### Algoritam BCNF normalizacije

Ulaz: relacija  $R$  sa atributima  $\{A_1, \dots, A_n\}$  i skupom FZ  $F$ ;

Izlaz: skup  $R_1, \dots, R_m$  ( $m \geq 1$ ) projekcija relacije  $R$ , koje su sve u BCNF i za koje važi: ako je  $m = 1$ , onda je  $R_1 \equiv R$ ; ako je  $m > 1$ , onda je  $R = R_1 * R_2 * \dots * R_m$ ;

```
BEGIN
  IF relacija  $R$  nije u BCNF
  THEN
    BEGIN
      uočiti funkcionalnu zavisnost  $X \rightarrow Y$ , u kojoj leva strana  $X$  nije
        ključ relacije  $R$  i  $X \cap Y = \emptyset$ ;
      neka je  $Z = Atr(R) \setminus XY$ ;
      zameniti relaciju  $R$  njenim projekcijama  $R[XY]$ ,  $R[XZ]$ ;
      ponoviti rekurzivno algoritam za relaciju  $R[XY]$ ;
      ponoviti rekurzivno algoritam za relaciju  $R[XZ]$ 
    END
  ELSE
    izdati relaciju  $R$  na izlazu;
  END.
```

#### 2.3.5 Višeznačna zavisnost i četvrta normalna forma

Četvrtu normalnu formu najlakše je opisati pomoću primera. Posmatrajmo relaciju koja prikazuje vezu između kompanija, proizvoda i zemalja:

IZVOZI ( KOMPANIJA, PROIZVOD, ZEMLJA ) .

Jedna  $n$ -torka označava da zadata kompanija zadati proizvod izvozi u zadatu zemlju. Relacija može u jednom trenutku izgledati kao u sledećoj tabeli. Lagano je proveriti da je relacija u BCNF.

<i>IZVOZI</i>		
<i>KOMPANIJA</i>	<i>PROIZVOD</i>	<i>ZEMLJA</i>
IBM	Desktop	Francuska
IBM	Desktop	Italija
IBM	Desktop	Velika Britanija
IBM	Mainframe	Francuska
IBM	Mainframe	Italija
IBM	Mainframe	Velika Britanija
HP	Desktop	Francuska
HP	Desktop	Španjolska
HP	Desktop	Irska
HP	Server	Francuska
HP	Server	Španjolska
HP	Server	Irska
Fujitsu	Mainframe	Italija
Fujitsu	Mainframe	Francuska

U nastavku uzimamo da vredi pravilo: čim kompanija izvozi u neku zemlju, ona odmah izvozi sve svoje proizvode u tu zemlju. Tada relacija očito sadrži veliku dozu redundance. Na primer, da bi dodali novi proizvod IBM-a, moramo upisati n-torku za svaku zemlju u koju IBM izvozi. Slično, ako HP počne izvoziti u Kinu, moraće se ubaciti posebna n-torka za svaki njegov proizvod.

Redundanca će se eliminisati ako zamenimo polaznu relaciju *IZVOZI* s dve manje relacije *RADI* i *PRODAJE*:

$RADI ( \underline{KOMPANIJA}, \underline{PROIZVOD} ) ,$   
 $PRODAJE ( \underline{KOMPANIJA}, \underline{ZEMLJA} ) .$

Podacima iz prethodne tabele tada odgovaraju podaci sledeće tabele:

<i>RADI</i>		<i>PRODAJE</i>	
<i>KOMPANIJA</i>	<i>PROIZVOD</i>	<i>KOMPANIJA</i>	<i>ZEMLJA</i>
IBM	Desktop	IBM	Francuska
IBM	Mainframe	IBM	Italija
HP	Desktop	IBM	Velika Britanija
HP	Server	HP	Francuska
Fujitsu	Mainframe	HP	Španjolska
		HP	Irska
		Fujitsu	Italija
		Fujitsu	Francuska

Dosadašnja pravila normalizacije nam ne pomažu da eliminišemo redundancu u relaciji *IZVOZI*. To je zato što redundanca nije bila uzrokovana funkcionalnim zavisnostima, već tzv. višeznačnim zavisnostima:

$KOMPANIJA \rightarrow \rightarrow PROIZVOD ,$   
 $KOMPANIJA \rightarrow \rightarrow ZEMLJA .$

Neka je zadata relacija  $R(A,B,C)$ . **Višeznačna zavisnost**  $A \rightarrow \rightarrow B$  vredi ako: skup  $B$ -vrednosti koje se u  $R$  pojavljuju uz zadati par ( $A$ -vrednost,  $C$ -vrednost) zavisi samo o  $A$ -vrednosti, a ne i o  $C$ -vrednosti.

Ista definicija primenjuje se i kad su  $A, B$  i  $C$  složeni atributi (dakle skupovi atributa).

U gornjem primeru, skup zemalja u koje zadata kompanija izvozi zadati proizvod zavisi samo o kompaniji a ne i o proizvodu. Slično, skup proizvoda koje zadata kompanija izvozi u zadatu zemlju zavisi samo o kompaniji a ne o zemlji.

Relacija  $R$  je u **četvrtoj normalnoj formi** (4NF) ako vredi: kad god postoji višeznačna zavisnost u  $R$ , na primer  $A \rightarrow\rightarrow B$ , tada su svi atributi od  $R$  funkcionalno zavisni o  $A$ . Ekvivalentno,  $R$  je u 4NF ako je u BCNF i sve višeznačne zavisnosti u  $R$  su zapravo funkcionalne zavisnosti.

U našoj relaciji *IZVOZI*, ni jedna od uočenih višeznačnih zavisnosti nije funkcionalna zavisnost. Znači, *IZVOZI* nije u 4NF i zato je treba rastaviti na *RADI* i *PRODAJE* (koje jesu u 4NF).

Odstupanje od 4NF opet možemo tumačiti kao grešku u modelovanju entiteta i veza. Posmatrana relacija *IZVOZI* nastala je zbog pokušaja da se odnos triju tipova entiteta *KOMPANIJA*, *PROIZVOD*, *ZEMLJA* tretira kao ternarna veza. Zapravo se ovde radi o dve nezavisne binarne veze.

Postoje, naravno, i prave ternarne veze. Na primer ako skup proizvoda koje zadata kompanija izvozi varira od zemlje do zemlje, tada pre uočene višeznačne zavisnosti ne vrede, relacija *IZVOZI* je u 4NF i ne možemo je rastaviti na dve manje relacije.

### 2.3.6 Razlozi zbog kojih se može odustati od normalizacije

Za većinu praktičnih primera dovoljno je relacije normalizovati do 3NF. Koji put je potrebno neku relaciju i dalje normalizovati do BCNF ili 4NF. Peta normalna forma, koja se takođe navodi u literaturi, nije od praktičnog značaja.

Postoje razlozi zbog kojih ponekad možemo odustati od pune normalizacije. Pogledajmo dva takva razloga.

**Složeni atribut**. Dešava se da nekoliko atributa u relaciji čine celinu koja se u aplikacijama nikad ne rastavlja na sastavne delove. Na primer, posmatrajmo relaciju  
KUPAC ( PREZIME IME, POŠTANSKI\_BROJ, GRAD, ULICA ).

Strogo govoreći, *GRAD* je funkcionalno zavisn o *POŠTANSKI\_BROJ*, pa relacija nije u 3NF. No mi znamo da *POŠTANSKI\_BROJ*, *GRAD* i *ULICA* čine celinu koja se zove adresa. Budući da se podaci iz adrese koriste i ažuriraju "u paketu", ne može doći do pre pominjanih anomalija.

Nije preporučljivo razbijati ovu relaciju na dve.

**Efikasno čitanje podataka**. Normalizacijom se velike relacije razbijaju na mnogo manjih. Kod čitanja je često potrebno podatke iz malih relacija ponovo sastaviti u veće n-torke. Uspostavljanje veza među podacima u manjim relacijama traje znatno duže nego čitanje podataka koji su već povezani i upisani u jednu veliku relaciju.

Projektant baze podataka treba da proceni kada treba provesti normalizaciju do kraja a kada ne.

Za tu procenu je važno razumevanje značenja podataka i načina kako će se oni koristiti.

### 3 JEZICI ZA RELACIONE BAZE PODATAKA

#### 3.1 Relaciona algebra

Relacionu algebru uveo je E.F. Codd u svojim radovima iz 70-tih godina 20. veka. Reč je o teorijskoj (matematičkoj) notaciji, a ne o praktičnom jeziku kojeg bi ljudi zaista neposredno koristili. Zato i ne postoji standardna sintaksa. Relaciona algebra se svodi na izvrednjavanje algebarskih izraza, građenih od relacija te unarnih i binarnih operatora (čiji operandi su relacija a rezultat je opet relacija). Svaki algebarski izraz predstavlja jedan upit (pretraživanje).

Kao primer će služiti pojednostavnjena (engleska) verzija baze o fakultetu:

STUDENT ( SNO, SNAME, LEVEL ) ,  
 COURSE ( CNO, TITLE, LNAME ) ,  
 REPORT ( SNO, CNO, MARK ) ,  
 LECTURER ( LNAME, ROOMNO ) .

STUDENT			COURSE		
SNO	SNAME	LEVEL	CNO	TITLE	LNAME
876543	Jones	2	216	Database Systems	Black
864532	Burns	1	312	Software Engineering	Welsh
856434	Cairns	3	251	Numerical Analysis	Quinn
876421	Hughes	2	121	Compilers	Holt

REPORT			LECTURER	
SNO	CNO	MARK	LNAME	ROOMNO
876543	216	82	Black	1017
864532	216	75	Welsh	1024
864532	312	71	Holt	2014
856434	121	49	Quinn	1010
876421	312	39		
876543	251	70		
864532	251	69		
864532	121	78		

Relacija *STUDENT* popisuje studente, *COURSE* nabraja ispite, a *LECTURER* sadrži podatke o nastavnicima. Student je jednoznačno određen svojim brojem iz indeksa *SNO*, ispit je jednoznačno određen svojom šifrom *CNO*, a nastavnici se razlikuju po svojim prezimenima *LNAME*. Za svakog studenta pamtimo njegovo ime *SNAME* i godinu studija *LEVEL*. Za ispit se pamti njegov naslov *TITLE* i ime (jednog) nastavnika koji ga predaje *LNAME*. Za svakog nastavnika poznat nam je broj njegove sobe *ROOMNO*. Relacija *REPORT* beleži koji je student upisao koji ispit i koju ocenu *MARK* je dobio.

### 3.1.1 Skupovni operatori

Relacije su ustvari skupovi  $n$ -torki. Zato na njih možemo primenjivati uobičajene skupovne operatore.

Neka  $R$  i  $S$  označavaju relacije. Tada je:

$R \text{ union } S$  . . . skup  $n$ -torki koje su u  $R$  ili u  $S$  (ili u obe relacije).

$R \text{ intersect } S$  . . . skup  $n$ -torki koje su u  $R$  i takođe u  $S$ .

$R \text{ minus } S$  . . . skup  $n$ -torki koje su u  $R$  no nisu u  $S$ .

Da bi se operatori mogli primeniti, relacije  $R$  i  $S$  moraju biti "kompatibilne", to jest moraju imati isti stepen i iste attribute (ista imena i tipove). Primitimo da uvek vredi:

$R \text{ intersect } S = R \text{ minus } (R \text{ minus } S)$  .

Znači, **intersect** nije nužan.

Za primer, posmatramo relaciju *NEW STUDENT*, građenu kao *STUDENT*, u kojoj se nalaze  $n$ -torke popisane u sledećem primeru:

<i>NEW_STUDENT</i>		
<i>SNO</i>	<i>SNAME</i>	<i>LEVEL</i>
876342	Smith	3
865698	Turner	2
875923	Murphy	2
856434	Cairns	3
871290	Noble	1

Tada primenom skupovnih operatora dobijamo rezultate iz sledećeg primera:

*STUDENT union NEW\_STUDENT*

<i>SNO</i>	<i>SNAME</i>	<i>LEVEL</i>
876543	Jones	2
864532	Burns	1
856434	Cairns	3
876421	Hughes	2
876342	Smith	3
865698	Turner	2
875923	Murphy	2
871290	Noble	1

*STUDENT intersect NEW\_STUDENT*

<i>SNO</i>	<i>SNAME</i>	<i>LEVEL</i>
856434	Cairns	3

*STUDENT minus NEW\_STUDENT*

<i>SNO</i>	<i>SNAME</i>	<i>LEVEL</i>
876543	Jones	2
864532	Burns	1
876421	Hughes	2

### 3.1.2 Selekcija

Selekcija je unarni operator koji izvlači iz relacije one n-torke koje zadovoljavaju zadati booleov uslov. Selekciju na relaciji  $R$  u skladu s booleovim uslovom  $B$  označavamo s  $R$  **where**  $B$ . Uslov  $B$  je formula koja se sastoji od:

- ☞ operandi koji su ili konstante ili atributi,
- ☞ operatora za upoređivanje  $=, <, >, \leq, \geq, \neq$
- ☞ logičkih operatora **and, or, not**.

Navešćemo nekoliko primera od kojih svaki sadrži jedan upit sa selekcijom i odgovarajući izraz u relacionoj algebri. Izračunate vrednosti izraza (odgovori na upite) vide se u ilustraciji.

**Upit 1:** Pronađi sve studente na stepenu (godini) 1.

$RESULT1 := STUDENT$  **where**  $LEVEL = 1$  .

**Upit 2:** Pronađi sve ispite s naslovom 'Database Systems'.

$RESULT2 := COURSE$  **where**  $TITLE = 'Database Systems'$  .

**Upit 3:** Pronađi sve studente koji su iz ispita 216 dobili ocenu veću od 70.

$RESULT3 := REPORT$  **where**  $((CNO=216) \text{ and } (MARK>70))$  .

<i>RESULT1</i>			<i>RESULT2</i>		
<i>SNO</i>	<i>SNAME</i>	<i>LEVEL</i>	<i>CNO</i>	<i>TITLE</i>	<i>LNAME</i>
864532	Burns	1	216	Database Systems	Black

<i>RESULT3</i>		
<i>SNO</i>	<i>CNO</i>	<i>MARK</i>
876543	216	82
864532	216	75

### 3.1.3 Projekcija

Projekcija je unarni operator koji iz relacije izvlači zadate attribute, s time da se u rezultirajućoj relaciji eliminišu n-torke duplikati. Projekciju relacije  $R$  na njene attribute  $A1, A2, \dots, Am$  označavaćemo s  $R[A1, A2, \dots, Am]$ . Smatraćemo da projekcija ima viši prioritet od ostalih operacija.

Navešćemo dva primera koji se sastoje od upita s projekcijom i odgovarajućeg izraza u relacionoj algebri. Izračunate vrednosti izraza (odgovori na upite) vide se u ilustraciji.

**Upit 1:** Pronađi brojeve soba od svih nastavnika.

$RESULT1 := LECTURER[ROOMNO]$  .

**Upit 2:** Pronađi ime nastavnika koji predaje kolegij 312.

$RESULT2 := ( COURSE$  **where**  $CNO = 312 ) [LNAME]$  .



RESULT1		RESULT2	
ROOMNO		LNAME	
1017			
1024			
2014		Welsh	
1010			

### 3.1.4 Kartezijev produkt

Neka su  $R$  i  $S$  relacije stepena  $n1$  odnosno  $n2$ . Tada algebarski izraz  $R$  **times**  $S$  daje Kartezijev produkt od  $R$  i  $S$ , dakle skup svih  $(n1 + n2)$ -torki čijih prvih  $n1$  komponenti čine  $n1$ -torku u  $R$ , a zadnjih  $n2$  komponenti čine  $n2$ -torku u  $S$ .

Atribut u  $R$  **times**  $S$  ima isto ime kao odgovarajući atribut u  $R$  odnosno  $S$ , s time da se po potrebi to ime proširuje imenom polazne relacije (slično kao za komponentu zapisa u C-u).

Kao primer korištenja Kartezijevog produkta, ispisaćemo za svakog studenta sve ispite koje on nije upisao:

$ALL\ COMB := STUDENT[SNO] \text{ times } COURSE[CNO],$   
 $DO\ NOT\ TAKE := ALL\ COMB \text{ minus } REPORT[SNO, CNO].$

Drugi primer pronalazi sve parove studenata koji su na istom stepenu (godini). Za to nam je potrebno napraviti Kartezijev produkt relacije  $STUDENT$  sa samom sobom. Da ne bi došlo do pometnje s imenima atributa, specijalnim operatorom **aliases** uvodimo "pseudonim" (drugo ime, nadimak) za relaciju  $STUDENT$ :

$TEMP \text{ aliases } STUDENT,$   
 $PAIRS := ( (TEMP \text{ times } STUDENT)$   
 $\quad \text{where } ( (TEMP.LEVEL = STUDENT.LEVEL)$   
 $\quad \text{and } (TEMP.SNO < STUDENT.SNO)) ) [TEMP.SNAME, STUDENT.SNAME].$

DO_NOT_TAKE		PAIRS	
SNO	CNO	TEMP.SNAME	STUDENT.SNAME
876543	312		
876543	121		
856434	216		
856434	312		
856434	251		
876421	216		
876421	251		
876421	121	Hughes	Jones

### 3.1.5 Prirodni spoj

Prirodni spoj (natural join) je binarni operator primenjiv na dve relacije  $R$  i  $S$  koje imaju bar jedan zajednički atribut.  $R \text{ join } S$  sastoji se od svih  $n$ -torki dobijenih spajanjem jedne  $n$ -torke iz  $R$  s jednom  $n$ -torkom iz  $S$  koja ima iste vrednosti zajedničkih atributa. U rezultirajućoj relaciji zajednički atribut se pojavljuje samo jednom.

Primer za prirodni spoj vidi se u ilustraciji. Pretpostavlja se da su zajednički atributi dvaju relacija tačno oni koji imaju ista imena.

Prirodni spoj omogućuje nam da u našoj fakultetskoj bazi podataka odgovorimo na složenije upite koji zahtevaju uspostavljanje veza između podataka u raznim tabelama.

$R$			$S$		
$A$	$B$	$C$	$B$	$C$	$D$
$a_1$	$b_1$	$c_1$	$b_1$	$c_1$	$d_1$
$a_2$	$b_1$	$c_1$	$b_1$	$c_1$	$d_2$
$a_3$	$b_2$	$c_2$	$b_2$	$c_3$	$d_3$
$a_4$	$b_2$	$c_3$			

$R \text{ join } S$			
$A$	$B$	$C$	$D$
$a_1$	$b_1$	$c_1$	$d_1$
$a_1$	$b_1$	$c_1$	$d_2$
$a_2$	$b_1$	$c_1$	$d_1$
$a_2$	$b_1$	$c_1$	$d_2$
$a_4$	$b_2$	$c_3$	$d_3$

**Upit 1:** Pronađi imena svih studenata koji su upisali ispit 251.

$RESULT1 := ( (REPORT \text{ where } CNO = 251) \text{ join } STUDENT ) [SNAME] .$

**Upit 2:** Pronađi broj sobe nastavnika koji predaje ispit 351.

$RESULT2 := ( (COURSE \text{ where } CNO = 351) \text{ join } LECTURER ) [ROOMNO] .$

**Upit 3:** Pronađi imena nastavnika koji predaju ispite koje je upisao bar jedan student na stepenu (godini) 2.

$RESULT3 := ( ( (STUDENT \text{ where } LEVEL=2) \text{ join } REPORT ) \text{ join } COURSE ) [LNAME] .$

$STUDENT \text{ where } LEVEL = 2$			$(STUDENT \text{ where } LEVEL=2) \text{ join } REPORT$				
$SNO$	$SNAME$	$LEVEL$	$SNO$	$SNAME$	$LEVEL$	$CNO$	$MARK$
876543	Jones	2	876543	Jones	2	216	82
876421	Hughes	2	876543	Jones	2	251	70
			876421	Hughes	2	312	39

$( (STUDENT \text{ where } LEVEL=2) \text{ join } REPORT ) \text{ join } COURSE$							$RESULT3$
$SNO$	$SNAME$	$LEVEL$	$CNO$	$MARK$	$TITLE$	$LNAME$	$LNAME$
876543	Jones	2	216	82	Database Systems	Black	Black
876543	Jones	2	251	70	Numerical Analysis	Quinn	Quinn
876421	Hughes	2	312	39	Software Engineering	Welsh	Welsh

Prirodni spoj uvek se može izraziti preko ostalih operatora. Na primer, za relacije  $R(A,B,C)$  i  $S(C,D)$  vredi:

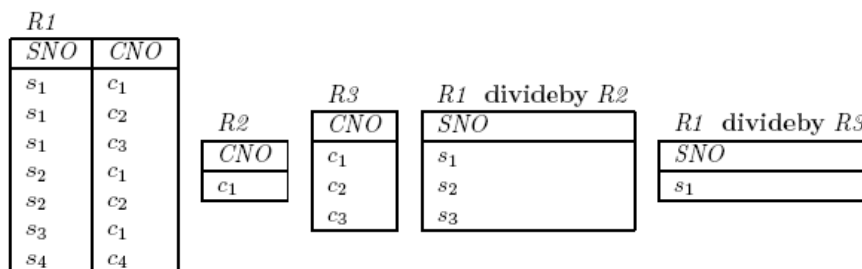
$$R \text{ join } S = ( (R \text{ times } S) \text{ where } R.C = S.C ) [A,B,C,D].$$

### 3.1.6 Teta-spoj

Teta-spoj relacija  $R$  i  $S$ , pisano  $R \text{ join } (A \theta B) S$ , je skup onih  $n$ -torki Kartezijevog produkta  $R$  sa  $S$  za koje je predikat  $R.A \theta S.B$  istina. Simbol  $\theta$  predstavlja jedan od operatora za uporedbu ( $=$ ,  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $\neq$ ). Teta-spoj se uvek može izraziti pomoću Kartezijevog produkta i selekcije. Prirodni spoj se može smatrati specijalnim slučajem teta-spoja.

### 3.1.7 Deljenje

Neka je  $R$  relacija stepena  $n$ , a  $S$  relacija stepena  $m$ , i neka se svi atributi od  $S$  pojavljuju i u  $R$ . Rezultat deljenja  $R$  sa  $S$ , oznakom  $R \text{ divideby } S$ , je skup svih  $(n - m)$ -torki ( $x$ ) takvih da se  $n$ -torke  $(x, y)$  pojavljuju u  $R$  za sve  $m$ -torke ( $y$ ) u  $S$ . Ovde  $x$  i  $y$  predstavljaju grupu od jedne ili više vrednosti atributa.



**Upit 1:** Pronađi imena studenata koji su upisali sve ispite.

$RESULT1 := ( (REPORT[SNO,CNO] \text{ divideby } COURSE[CNO]) \text{ join } STUDENT ) [SNAME]$ .

Za naše konkretne podatke, jedini student koji zadovoljava upit je Burns.

**Upit 2:** Pronađi brojeve onih studenata koji su upisali barem one ispite koje je upisao student s brojem 856434 (i možda još neke ispite).

$RESULT2 := REPORT [SNO,CNO] \text{ divideby } (REPORT \text{ where } SNO=856434)[CNO]$ .

Kao odgovor dobijamo brojeve studenata 856434, 864532.

Operator deljenja predstavlja način implementiranja univerzalne kvantifikacije  $\forall$  u relacionoj algebri. Deljenje se takođe može izraziti pomoću prethodno opisanih operatora. Na primer, ako imamo relacije  $R(A,B,C,D)$  i  $S(C,D)$ , tada je:

$R \text{ divideby } S = R[A,B] \text{ minus } ( (R[A,B] \text{ times } S) \text{ minus } R ) [A,B]$ .

### 3.1.8 Spoljni spoj

Spoljni spoj (outer join) je binarni operator vrlo sličan prirodnom spoju. Primenjiv je pod istim uslovima i daje kao rezultat relaciju s istom građom (šemom). No sadržaj od  $R \text{ outerjoin } S$  je nešto bogatiji nego sadržaj  $R \text{ join } S$ . Naime, pored svih  $n$ -torki iz  $R \text{ join } S$ , relacija  $R \text{ outerjoin } S$  sadrži i sve "nesparene" (nespojene)  $n$ -torke iz  $R$  odnosno  $S$  (time da su te nesparene  $n$ -torke na odgovarajući način proširene *null* vrednostima).

<i>R</i>			<i>S</i>		
<i>A</i>	<i>B</i>	<i>C</i>	<i>B</i>	<i>C</i>	<i>D</i>
<i>a</i> <sub>1</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>d</i> <sub>1</sub>
<i>a</i> <sub>2</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>d</i> <sub>2</sub>
<i>a</i> <sub>3</sub>	<i>b</i> <sub>2</sub>	<i>c</i> <sub>2</sub>	<i>b</i> <sub>2</sub>	<i>c</i> <sub>3</sub>	<i>d</i> <sub>3</sub>
<i>a</i> <sub>4</sub>	<i>b</i> <sub>2</sub>	<i>c</i> <sub>3</sub>			

<i>R outerjoin S</i>			
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
<i>a</i> <sub>1</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>d</i> <sub>1</sub>
<i>a</i> <sub>1</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>d</i> <sub>2</sub>
<i>a</i> <sub>2</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>d</i> <sub>1</sub>
<i>a</i> <sub>2</sub>	<i>b</i> <sub>1</sub>	<i>c</i> <sub>1</sub>	<i>d</i> <sub>2</sub>
<i>a</i> <sub>4</sub>	<i>b</i> <sub>2</sub>	<i>c</i> <sub>3</sub>	<i>d</i> <sub>3</sub>
<i>a</i> <sub>3</sub>	<i>b</i> <sub>2</sub>	<i>c</i> <sub>2</sub>	—

Spoljni spoj obično se koristi za traženje podataka koji ne zadovoljavaju neki uslov. Na primer:

**Upit 1:** Pronađi imena studenata koji nisu upisali ni jedan ispit.

*RESULT1* := ( (*STUDENT outerjoin REPORT*) **where** (*CNO* is null) ) [*SNAME*].

## 3.2 Relacioni račun

Relacioni račun takođe je bio predložen od E.F. Codd-a, kao alternativa relacionoj algebri. Reč je o matematičkoj notaciji zasnovanoj na predikatnom računu. Upit se izražava tako da zadamo predikat kojeg *n*-torke moraju zadovoljavati.

Postoje dve vrste relacionog računa: račun orjentisan na *n*-torke (gde su osnovni objekti *n*-torke) i račun orjentisan na domene (gde su osnovni objekti vrednosti iz domena za atribut).

## 3.3 Jezik SQL

Structured Query Language (SQL) razvio je IBM u sklopu projekta “System R” (Chamberlin i drugi, kasne 70-te godine 20. veka). Jezik se postepeno usavršavao, a njegova doterana varijanta pojavljuje se u današnjem IBM-ovom relacionom DBMS-u zvanom DB2. Druge softverske kuće (na primer Oracle Corporation) ugradile su SQL u svoje DBMS-e, te ga time učinile vrlo popularnim i dostupnim na svim važnijim računarskim platformama. Preostale kuće (INGRES Corporation, DEC, . . . ) koje su razvijale svoje jezike, bile su prisiljene da se prilagode SQL-u. Zbog pojave raznih “dijalekata” donesen je ISO/ANSI standard za SQL (zadnja verzija 1998. godine).

SQL je uglavnom zasnovan na relacionom računu, s time da je matematička notacija zamenjena ključnim rečima nalik na govorni engleski jezik. No lagano se realizuju i sve operacije iz relacione algebre.

Osim postavljanja upita, jezik takođe omogućuje: definisanje relacija, ažuriranje relacija (upis, promena, brisanje *n*-torki), sortiranje i formatiranje ispisa, neke aritmetičke operacije s podacima, definisanje “pogleda” (virtuelnih relacija izvedenih iz

postojećih), uticaj na fizičku građu baze (na primer stvaranje tzv. indeksa), te kontrolu sigurnosti. Sada ćemo navesti samo nekoliko primera.

### 3.3.1 Postavljanje upita

Upit se postavlja fleksibilnom naredbom **SELECT** (nije isto što i operacija selekcije u relacionoj algebri). Rezultat upita se shvata kao nova privremena relacija, izvedena iz stalnih (po tome je SQL sličan relacionoj algebri).

Upit 1: Pronađi brojeve i imena svih studenata na stepenu 1.

```
SELECT SNO, SNAME  
FROM STUDENT  
WHERE LEVEL = 1;
```

ili (ukoliko želimo uzlazno sortirani ispis)

```
SELECT SNO, SNAME  
FROM STUDENT  
WHERE LEVEL = 1  
ORDER BY SNAME;
```

Upit 2: Pronađi brojeve i imena studenata koji su upisali ispit 121.

```
SELECT STUDENT.SNO, STUDENT.SNAME  
FROM STUDENT, REPORT  
WHERE STUDENT.SNO = REPORT.SNO  
AND REPORT.CNO = 121;
```

Ovde smo morali proširiti imena atributa da bi izbegli dvoznačnost. Vidimo kako SQL **SELECT** naredba zamenjuje prirodni spoj iz relacione algebre. Štaviše, to je mogao biti i theta-spoj.

Drugo rešenje za isti upit glasi:

```
SELECT SNO, SNAME  
FROM STUDENT  
WHERE SNO IN  
(SELECT SNO  
FROM REPORT  
WHERE CNO = 121);
```

Upit 3: Pronađi brojeve i imena studenata koji su upisali bar jedan ispit kojeg predaje Quinn.

```
SELECT SNO, SNAME  
FROM STUDENT  
WHERE SNO IN
```

```
(SELECT SNO
FROM REPORT
WHERE CNO IN
(SELECT CNO
FROM COURSE
WHERE LNAME = 'Quinn'));
```

Drugo rešenje za isti upit glasi:

```
SELECT STUDENT.SNO, STUDENT.SNAME
FROM STUDENT, REPORT, COURSE
WHERE STUDENT.SNO = REPORT.SNO
AND REPORT.CNO = COURSE.CNO
AND COURSE.LNAME = 'Quinn';
```

Upit 4: Pronađi sve parove brojeva studenata takve da su odgovarajući studenti na istom stepenu.

```
SELECT TEMP1.SNO, TEMP2.SNO
FROM STUDENT TEMP1, STUDENT TEMP2
WHERE TEMP1.SNO < TEMP2.SNO
AND TEMP1.LEVEL = TEMP2.LEVEL;
```

Ovde smo uveli drugo ime (alias) za relaciju *STUDENT*.

Upit 5: Pronađi sve podatke o studentima koji nisu upisali ispit 121.

```
SELECT *
FROM STUDENT
WHERE SNO NOT IN
(SELECT SNO
FROM REPORT
WHERE CNO = 121);
```

Znak \* ovde označava sve attribute relacije.

Upit 6: Pronađi brojeve onih studenata koji su upisali sve kolegije.

Budući da SQL nema univerzalni kvantifikator ali ima egzistencijalni, upit moramo ovako preformulisati: "Pronađi brojeve onih studenata za koje ne postoji ispit kojeg oni nisu upisali".

```
SELECT SNO
FROM STUDENT
WHERE NOT EXISTS
(SELECT *
FROM COURSE
```

```
WHERE NOT EXISTS  
(SELECT *  
FROM REPORT  
WHERE REPORT.SNO = STUDENT.SNO  
AND REPORT.CNO = COURSE.CNO));
```

Ovde je **EXISTS (SELECT . . . )** istina ukoliko je rezultat uključene **SELECT** naredbe neprazan.

### 3.3.2 Ažuriranje relacija

Upis, promena, odnosno brisanje n-torke u relaciji postiže se naredbom **INSERT**, **UPDATE**, odnosno **DELETE**. Sve tri naredbe građene su po analogiji sa **SELECT**.

Primer 1: Upiši u relaciju *STUDENT* novu n-torku sa sledećim vrednostima atributa:  
*SNO = 867520* , *SNAME = 'Smith'* , *LEVEL = 2*.

```
INSERT  
INTO STUDENT  
VALUES (867520, 'Smith', 2);
```

Primer 2: Promeni nastavnika ispita 251 iz Quinn u Black.

```
UPDATE COURSE  
SET LNAME = 'Black'  
WHERE CNO = 251;
```

Ova naredba menja **svaku** n-torku u kojoj je *CNO = 251*.

Primer 3: Briši sve studente na stepenu 3.

```
DELETE  
FROM STUDENT  
WHERE LEVEL = 3;
```

## 3.4 Optimizacija upita

Jezici za relacione baze podataka daju korisniku veliku slobodu u postavljanju upita. Teret efikasnog odgovaranja na te raznolike upite prebačen je na DBMS. Naime, odgovor na jedan te isti upit obično se može dobiti na razne načine, a zadatak DBMS-a je da odabere najefikasniji način. Odabir dobrog postupka za odgovaranje zove se optimizacija upita. Moderni DBMS provodi optimizaciju na dva nivoa:

**viši (logički) nivo** : preformulisanje algebarskog izraza u oblik koji je ekvivalentan polaznom, ali je pogodniji sa stanovišta izvrednjavanja;

**niži (fizički) nivo** : odabir dobrog algoritma za izvednjavanje svake od osnovnih operacija u algebarskom izrazu. Pritom se nastoji iskoristiti eventualno prisustvo pomoćnih struktura podataka (indeksi i slično).

### 3.4.1 Odnos između relacione algebre i računa

Svaki upit zapisan u relacionoj algebri može se zameniti ekvivalentnim upitom zapisanim u relacionom računu. Strogi dokaz ove tvrdnje može se naći u literaturi. Mi kao ilustraciju navodimo ekvivalente za "bitne" algebarske operacije:

$$\begin{aligned} R1 \text{ union } R2 &\dots \{t \mid R1(t) \text{ or } R2(t)\}, \\ R1 \text{ minus } R2 &\dots \{t \mid R1(t) \text{ and not } R2(t)\}, \\ R1 \text{ times } R2 &\dots \{ \langle t, r \rangle \mid R1(t) \text{ and } R2(r) \}, \\ R1 \text{ where } f(X) &\dots \{t \mid R1(t) \text{ and } f(t.X)\}, \\ R1[X] &\dots \{t.X \mid R1(t)\}. \end{aligned}$$

Ovdje  $\langle t, r \rangle$  znači kombinaciju  $n$ -torki  $t$  i  $r$ . E.F. Codd je u svom članku iz 1972. godine dokazao i obratnu tvrdnju, tj. da se svaki upit zapisan pomoću relacionog računa može formulisati i pomoću relacione algebre. Štaviše, Codd je izložio **redukциони algoritam** kojim se izraz u računu pretvara u izraz u algebri.

Praktični jezici poput SQL kombinuju elemente računa i algebre. Stoga je očigledno da se upiti zapisani u tim jezicima takođe mogu preformulisati u relacionu algebru.

Zahvaljujući ovim rezultatima, prilikom razmatranja optimizacije možemo bez gubitka opštosti smatrati da je upit već zapisan u relacionoj algebri. Naime, ako to nije tako, tada bi prvi korak optimizacije bila pretvorba upita u algebarski izraz.

### 3.4.2 Osnovna pravila za optimizaciju

Značajna ušteda vremena postiže se menjanjem redosleda operacija, u cilju da se što pre smanji veličina relacija s kojima radimo.

**Kombinovanje selekcija** . Očito vredi ekvivalencija:

$(R \text{ where } B1) \text{ where } B2 = R \text{ where } (B1 \text{ and } B2).$

Ovime smanjujemo potrebno vreme ukoliko se obe selekcije odvijaju podjednako "sporo" (tj. pregledom cele relacije). Ako se jedna relacija odvija brzo (na primer zahvaljujući prisustvu pomoćnih fizičkih struktura podataka) a druga sporo, tada se kombinovanje ne isplati, jer će rezultirajuća selekcija takođe biti spora. Znači, odluka šta je bolje zavisi o fizičkoj građi baze podataka.

**Izvlačenje selekcije ispred prirodnog spoja odnosno Kartezijevog produkta** . Ako uslov  $B$  sadrži samo atribut od  $R$ , a ne one od  $S$ , tada vredi:

$(R \text{ join } S) \text{ where } B = (R \text{ where } B) \text{ join } S,$

$(R \text{ times } S) \text{ where } B = (R \text{ where } B) \text{ times } S.$

Ova transformacija može znatno smanjiti broj  $n$ -torki koje ulaze u prirodni spoj odnosno Kartezijev produkt. Opštije, ako  $B$  rastavimo na  $B = BR \text{ and } BS \text{ and } BC \text{ and } B0$ , gde  $BR$  sadrži samo atribut od  $R$ ,  $BS$  sadrži samo atribut od  $S$ ,  $BC$  sadrži zajedničke atribut od  $R$  i  $S$ ,  $B0$  predstavlja ostatak od  $B$ , tada vredi:

$(R \text{ join } S) \text{ where } B = ( (R \text{ where } (BR \text{ and } BC)) \text{ join } (S \text{ where } (BS \text{ and } BC)) ) \text{ where } B0.$

Slična ekvivalencija može se napisati i za operator **times**.



Na primer, želimo naći brojeve svih studenata na stepenu 1 koji su upisali neki ispit kod nastavnika Quinna i dobili ocenu iznad 70 iz tog ispita. Tada se to može izraziti kao:

**RESULT := ( (STUDENT join REPORT join COURSE) where (LEVEL=1 and MARK>70 and LNAME='Quinn') ) [SNO].**

Primenom transformacije dobijamo:

**RESULT := ( ((STUDENT where (LEVEL=1) join (REPORT where MARK>70)) join (COURSE where LNAME='Quinn') ) [SNO] .**

**Izvlačenje selekcije ispred projekcije** . Ako uslov B sadrži samo projicirane attribute X, tada je:

**$R[X] \text{ where } B = (R \text{ where } B) [X]$**  .

Projiciranje može dugo trajati zbog eliminacije "duplikata". Zato je dobro najpre smanjiti broj n-torki selektovanjem. To je posebno preporučljivo onda kad postoje fizička sredstva za brzu selekciju.

**Kombinovanje projekcija** . Ako su X, Y i Z atributi od relacije R, tada vredi:

**$((R[X, Y, Z])[X, Y])[X] = R[X]$**  .

Umesto tri projekcije dovoljna je samo jedna. U ovom jednostavnom slučaju, ta jednakost je očigledna. No u dugačkim i komplikovanim izrazima nije lako uočiti redundantno projiciranje.

**Izvlačenje projekcije ispred prirodnog spoja** . Moramo paziti da projiciranjem ne izbacimo zajednički atribut iz relacija pre nego što je bio obavljen prirodni spoj. Ako X označava zajedničke attribute od R i S, tada je:

**$(R \text{ join } S)[X] = R[X] \text{ join } S[X]$**  .

No pravilo ne vredi za proizvoljni skup atributa X. Neka su AR atributi od R, AS atributi od S, AC = AR  $\cap$  AS zajednički atributi od R i S. Tada vredi:

**$(R \text{ join } S)[X] = ( R[(X \cap AR) \cup AC] \text{ join } S[(X \cap AS) \cup AC] ) [X]$**  .

Opet se smanjuje broj n-torki koje ulaze u prirodni spoj. Zahvat ne mora uvek biti koristan, jer projekcija može sprečiti efikasnu implementaciju prirodnog spoja. Na primer, projekcija može stvoriti privremenu relaciju na koju nisu primenjive postojeće pomoćne fizičke strukture. Znači, odluka šta je bolje opet zavisi o fizičkoj građi.

**Optimizacija skupovnih operatora** . Koji put su od koristi sledeća pravila:

**$(R \text{ union } S) \text{ where } B = (R \text{ where } B) \text{ union } (S \text{ where } B)$**  ,

**$(R \text{ minus } S) \text{ where } B = (R \text{ where } B) \text{ minus } (S \text{ where } B)$**  ,

**$(R \text{ union } S)[X] = R[X] \text{ union } S[X]$**  ,

**$(R \text{ minus } S)[X] = R[X] \text{ minus } S[X]$**  ,

**$(R \text{ where } B1)[X] \text{ union } (R \text{ where } B2)[X] = (R \text{ where } (B1 \text{ or } B2)) [X]$**  .

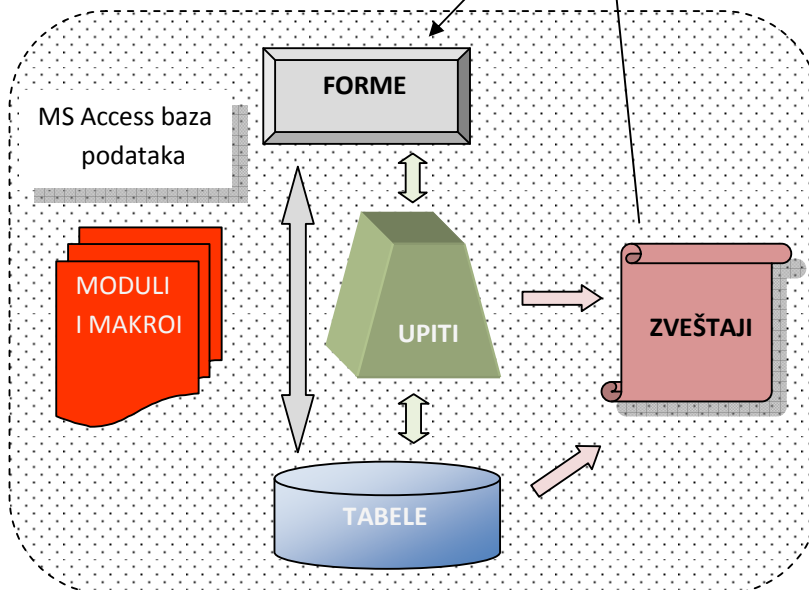
Predzadnja jednakost vredi pod pretpostavkom da X sadrži ključne attribute od R (a time i od S). Transformacijama se nastoji smanjiti broj n-torki koje sudeluju u operacijama **union** i **minus**. Operator **intersect** je specijalni slučaj od **join**, pa za njega vrede ista pravila kao za **join**.

## II

### 4 MS ACCESS

#### 4.1 Uvod

Jedan od softverskih paketa na tržištu namenjen kreiranju baza podataka je MS Access, program iz grupe MS Office. Baza podataka kreirana u ovom programu sastoji se od elemenata koji su prikazani na donjoj ilustraciji. Najvažniji deo posla je definicija **tabela**, tj pažljivo prevođenje relacione šeme opisan u prethodnom poglavlju u tabele u postavljaju „temelji“ za izgradnju baze vršila manipulacija podacima na način ws okruženju, sledeći korak je definicija prozora). No ovo je jedan od načina pristupa pruža naročitu fleksibilnost. Drugi način je koji se mogu definisati i upotrebom SQL jezika (pogledati prethodno poglavlje).



Da bi se dobili trajni ( štampani ) rezultati rada sa bazom, potrebno je kreirati odgovarajuće **izveštaje** ( reports ).

Ukoliko su pak zahtevi obrade podataka složeni i ne mogu se obaviti definicijom prethodnih elemenata potrebno je preći na programiranje **modula**.

Pri tome se koristi VBA (Visual Basic for Applications). No to već izlazi iz okvira programa predmeta Baze podataka za III razred. Prema tome u nastavku ćemo se pozabaviti redom sledećim elementima Access-a: tabelama, formama, upitima i izveštajima. Ilustracije su urađene u Access –u 2007.

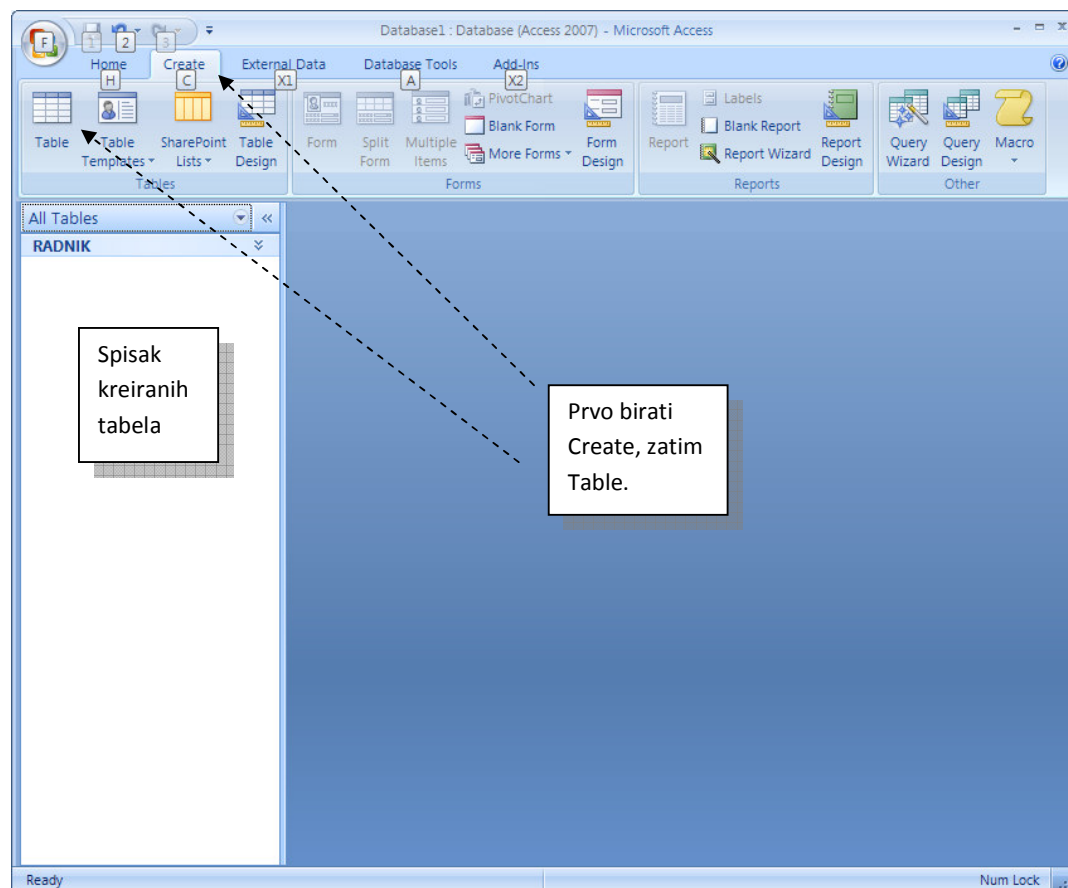
## 4.2 Tabele

Tabela u Access-u predstavlja relaciju iz relacionog modela baza podataka. Jedan red tabele predstavlja jednu n-torku ili jednu pojavu relacije, što odgovara jednoj pojavi tipa entiteta. Kolona u toj tabeli ili polje (field) u terminologiji Access-a odgovara jednom atributu tipa entiteta. Definicija tabele se u krajnjoj liniji i svodi na to da se definiše lista field-ova jedne tabele i da se svako polje (field) okarakterise nizom osobina. Neke od tih osobina je neophodno odrediti, neke se mogu i „preskočiti“.

Kada se definišu sve tabele na osnovu relacione šeme, potrebno je još uspostaviti i veze (relationships) među tabelama koje su u vezi. Preciznije, naznačava se preko kojih polja su tabele u vezi, što omogućava Access-u da se između ostalog brine i o integritetu (tačnosti) baze podataka.

### 4.2.1 Definicija polja tebele

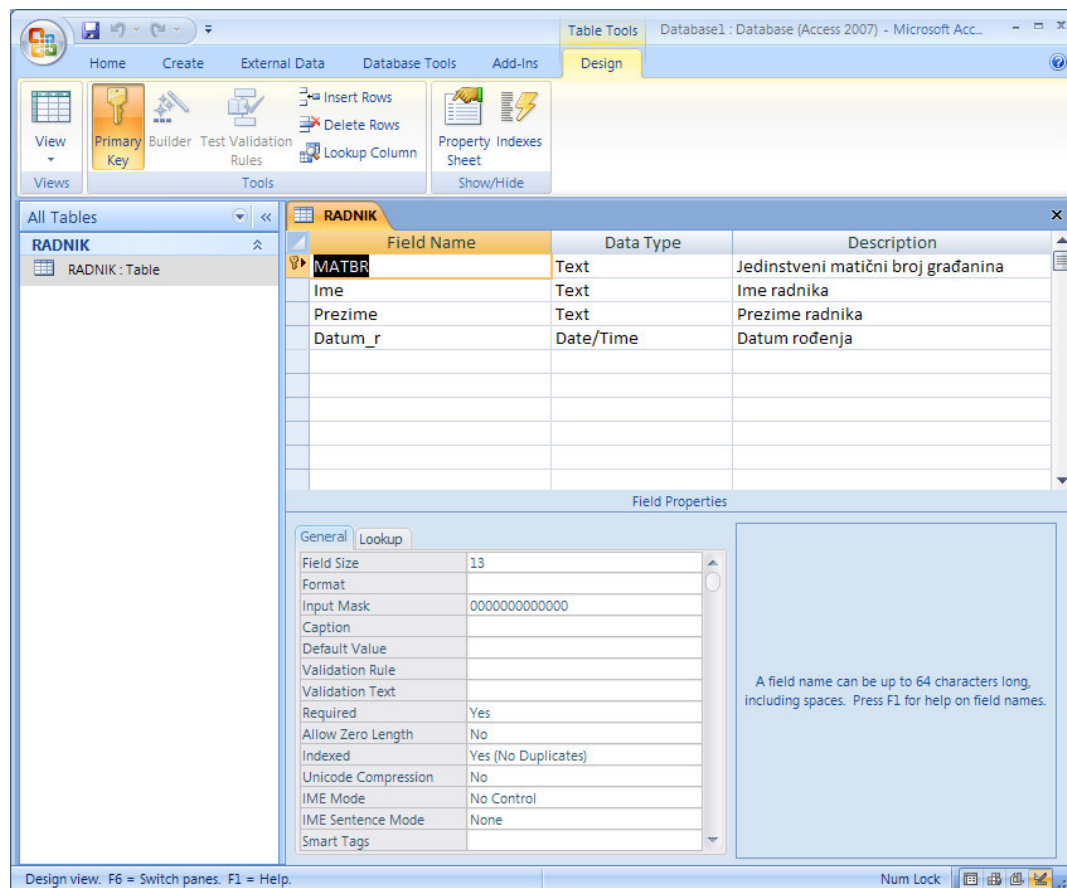
Da bi pristupili kreiranju tabele potrebno je odabrati **tab** (karticu) *Create*, a zatim alat za definisanje tabele *Table*.



Nakon toga pojaviće se odgovarajući alat u centralnoj radnoj površini pomoću koga se može izvršiti definicija polja (drugim rečima atributa). Potrebno je odrediti ime polja (Field Name), tip podatka koji će biti memorisan (Data Type) i opciono opis tog polja

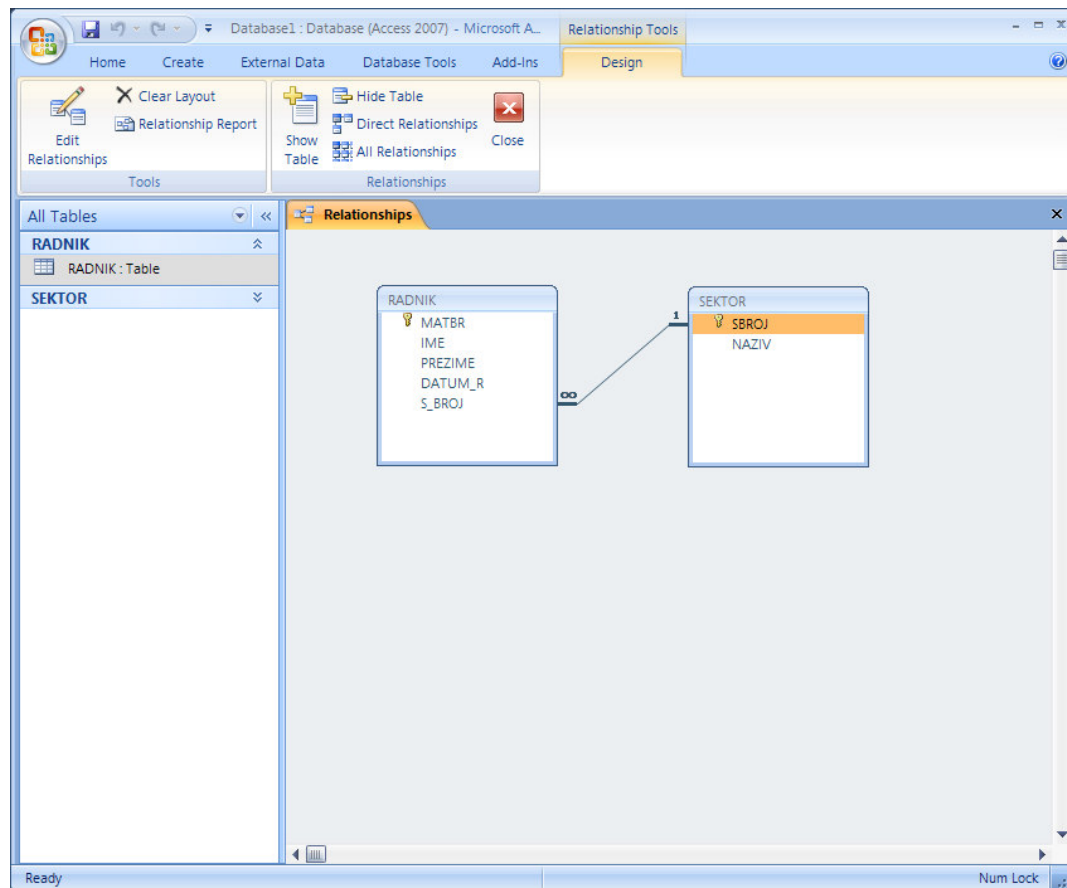
radi pojašnjenja korisniku koji bude radio sa bazom (Description). (Taj opis će se pojaviti u statusnoj liniji forme nad datom tabelom kada se uđe u ažuriranje vrednosti tog polja). Na sledećoj slici vidi se definisanje (u Design View -u) tabele RADNIK, tj. njenih polja. Obrat ćemo pažnju na par neophodnih aktivnosti:

- ✎ prilikom određivanja Data type (tekst, broj, datum, ...) u donjem delu prozora nalazi se nekoliko dodatnih stvari koje je potrebno „pretresti“. Najvažnije su sledeće:
  - Field Size – kolika je veličina polja, npr ako je tip text onda se ovde zadaje broj slova
  - Required – da li je obavezan unos ovog podatka
  - Indexed – indeksiranje tabele po ovom polju radi ubrzanja rada; pri tome se duplikati mogu dozvoliti ili ne (npr ako je to primarni ključ – Primary Key)
- ✎ jedno ili više polja mora biti proglašeno za primarni ključ. Indikator je sličica ključića pored imena polja; to se radi tako da se selektuju željena polja a zatim se klikne na ekranki dugmić Primary Key.
- ✎ tabelu treba imenovati recimo prilikom snimanja ili naknadno uraditi Rename.



## 4.2.2 Kreiranje relacija

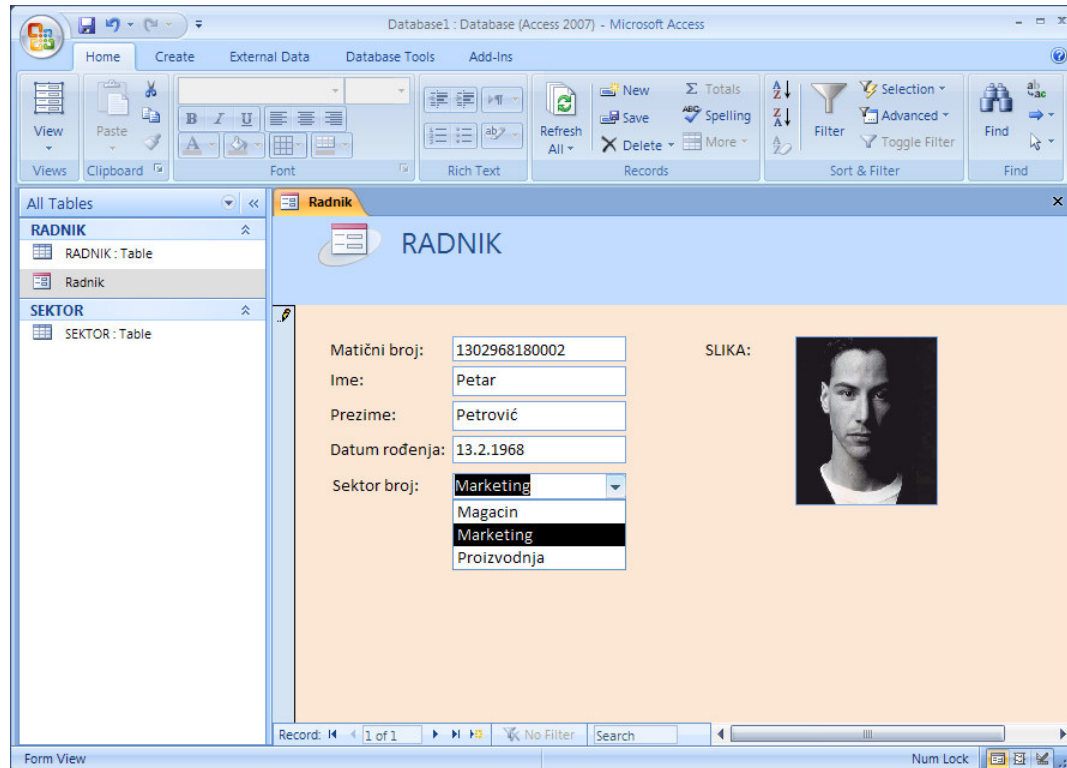
Nakon kreiranja potrebnih tabela prelazimo na sledeći korak, a to je uspostavljanje veza među tabelama, tačnije među poljima iz tabela preko kojih su one povezane. Da bi smo to uradili biramo tab (karticu) Database Tools, a zatim Relationships. Sledeći korak je da odaberemo tabele koje treba povezati, a nakon toga pomoću miša tehnikom prevlačenja ćemo uspostaviti vezu. Za donju sliku aktivnost je bila sledeća – postaviti pokazivač miša nad polje SBROJ u tabeli SEKTOR, a zatim prevući do polja S\_BROJ u tabeli RADNIK. Otpustiti taster miša. U malom prozoru koji se pojavi treba čekirati opciju *Enforce Referential Integrity*. Pojaviće se grafički prikaz veze na kojoj je označena i funkcionalnost: 1 na strani SEKTOR, a ∞ (što znači N) na strani RADNIK.



Pošto smo uspostavili *Referential Integrity* neće biti moguće napraviti grešku sledeće vrste – da unosimo podatke za radnika i da mu u polje S\_BROJ upišemo vrednost koja ne postoji već uneta u tabelu SEKTOR. Znači ne mora više programer da se brine o tome kao nekada kada su se aplikacije za baze podataka pisale u npr Clipper –u. Takođe programer ne mora više da misli o indeksima (mehanizam koji omogućava znatno brži rad sa tabelama podataka) i njihovoj ispravnosti niti o silnim DBF (tabele, data base file) fajlovima, pošto je u Access –u sve to objedinjeno – čitava baza sa svim svojim tabelama, indeksima, upitima itd je sada u samo jednom fajlu na disku.

### 4.3 Forme

Forme (prozori, formulari, obrasci) su Acces –ovi objekti kojima se povećava konfor i efikasnost rada sa bazom, na taj način što će korisnik pred sobom imati prozor sa kontrolama na koje je već navikao koristeći Windows operativni sistem:



Iforme se mogu kreirati, kao i većina drugih stvari u Access –u, pomoću Wizarada ali ćemo mi koristiti drugačiji pristup, a to je korišćenje Design View alata. Pre nego što kreiramo i najprostiju formu mala napomena:

*Forma (prozor) je okvir, kontejner koji obuhvata grafičke objekte odnosno **kontrole** koje su svojstvene Windows operativnom sistemu i koje se mogu povezati sa memorisanim podacima u tabelama. Time se olakšava rad sa kreiranom bazom širokom krugu korisnika.*



Čarobnjak (wizard)  
koji olakšava  
definiciju kontrola  
kao što su npr

Komandni taster  
(dugme)

Lista

*Sa ove ilustracije se vidi da na formu možemo ugraditi poznate objekte kao što su komandni dugmići, liste, labele itd.*

### 4.3.1 kreiranje proste forme

Pod prostom formom možemo podrazumevati onu koja je napravljena nad jednom tabelom. Najčešće na takvoj formi ništa osim editabilnih polja povezanih sa poljima tabele i nije potrebno kreirati. Naime Access sam kreira dugmiće za navigaciju – napred, nazad, na početak, na kraj, dodavanje novog.



Prema tome treba uraditi sledeće. Pokrenuti Form Design čime će se startovati pravljenje nove forme (na početku je prazna). Na tu praznu površinu se mogu dodavati kontrole, a ono što je nama zanimljivo je da forma sadrži text box –ove (pozicije koje sadrže podatke iz tabele i koje možemo uređivati, npr ime radnika, prezime itd). Naravno ti text box –ovi moraju biti vezani za polja tabele. To postizemo na sledeći način. Kliknemo na crni kvadratić u gornjem levom uglu forme da bismo dobili Properties prozor. Biramo Data tab. Postavljamo se u Record Source polje i upisujemo SQL rečenicu `SELECT * FROM ime_tabele;` (naprimer `SELECT * FROM SEKTOR;` ako pravimo formu za tabelu SEKTOR). Zatvorimo Properties prozor. Sada će u liniji toolbar –a biti dostupno dugme Field List



I nakon klika na njega dobijamo prozorčić sa listom polja iz potrebne tabele. Sve što sada treba uraditi je da se prevuku potrebna polja na formu. Preostaje da se promeni Caption (tekst koji piše ispred editabilnog polja; naime nakon prevlačenja ovde će pisati naziv polja iz tabele npr MATBR ali je bolje to prepraviti u Matični broj).

Postoje situacije kada se i ova prosta forma mora malo doraditi. Uzmimo sledeći primer – pravimo formu za prijem robe od nekog dobavljača. U njoj se nalazi i šifra dobavljača. No kako dobavljača ima puno, ne mogu se napamet znati sve šifre, pa bi bilo zgodno da se umesto upisa šifre, ovde nađe padajuća lista (Combo Box) u kojoj će biti svi dobavljači. Tada bi prostim klikom na potrebnog, njegova šifra automatski bila upisana. Upravo to što nam treba i realizuje se jednostavno – izbrišemo Text Box za unos šifre dobavljača sa forme, a iz ToolBox –a (pazeći da je uključen Wizard) biramo Combo Box. Zatim ga „nacrtamo“ na formi. Kada se otpusti taster miša, pokrenuće se Wizard. U nekoliko narednih koraka ćemo birati opcije koje će omogućiti da se iz tabele dobavljača, šifra dobavljača ugradi u potreno polje u tabeli prijema robe.

Sledeći primer bi bio da se popunjavaju podaci iz neke tabele koja ima dosta polja, s tim da su neka važna a neka manje važna pa se i ne moraju popuniti. Tada bi bilo zgodno ugraditi Tab Control kao na sledećoj ilustraciji za formu nad tabelom RADNIK. Kada „nacrtamo“ tu kontrolu na formi, onda prostom tehnikom Cut – Paste sa forme preselimo na Tab Control željene Text Box –ove.

Ostali podaci Slika

DATUM\_R:

ADRESA:

OPSTINA:

TEL:

#### 4.3.2 kreiranje složene forme (sa podformom)

Nekada zahtev za kreiranje forme može da bude složen, tako da zahteva da na formi budu podaci iz dve ili više tabela/upita, a klasičan primer je forma kojom se pravi neka vrsta fakture. Naime kupac je kupio nekoliko različitih proizvoda. Postoje podaci koji su vezani za sam čin kupovine (naziv kupca, datum kupovine, iznos itd) – zaglavlje fakture postoje podaci o specifikaciji te kupovine (naziv prvog proizvoda, cena, količina, iznos pa naziv drugog proizvoda itd). Očito bi najpogodnije bilo da se u jednom delu forme (ili tačnije u glavnoj formi) popunjavaju podaci iz zaglavlja, a da se u drugom delu (ili tačnije podformi) popuni specifikacija.

Ili drugi primer. Ako se vratimo u prvo poglavlje ove knjige i podsetimo se relacije šeme preduzeća videćemo da postoji slabi tip entiteta CLAN\_PORODICE, koji zavisi od regularnog tipa entiteta RADNIK. Znači bilo bi dobro da glavna forma RADNIK sadrži podformu kojom će se ažurirati tabela CLAN\_PORODICE.

Preduzeće

**RADNICI**

Matični broj:

Ime:

Prezime:

Stručna sprema:

Sektor broj:

Ostali podaci Slika

DATUM\_R:

ADRESA:

OPSTINA:

TEL:

Članovi porodice:

	matični broj	ime	prezime	datum rođenja
▶	<input type="text" value="10"/>	<input type="text" value="Raja"/>	<input type="text"/>	<input type="text"/>
	<input type="text" value="10"/>	<input type="text" value="Vlaja"/>	<input type="text"/>	<input type="text"/>
*	<input type="text" value="10"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Record:  of 2


Record:  of 3



Šta uraditi da se dobije takva forma? Postoje različiti načini a jedan od njih je sledeći. Prvo posebno napraviti glavnu formu kao da pravimo prostu formu. Zatim napraviti novu formu koja će kasnije biti ugrađena u glavnu kao podforma. Da vidimo kako se pravi ta druga.

Krenemo kao sa pravljenjem proste forme. Na formu prevučemo sva polja iz Field List – a koja su nam potrebna, a onda izbrišemo sve Caption –e. Zatim Text Box –ove postavimo u jedan red pri vrhu forme (pogledati sekciju Detail na sledećoj slici)

Ako ne ide od ruke onda se može izvršiti selekcija svih tih Text Box –ova i pokrenuti naredba *Format > Align > Top*. Sada izabrati *View > Form Header/Footer* i na formi će se pojaviti nove dve sekcije Form Header i Form Footer. Form Footer se može mišem podići da se ne vidi jer nam neće biti potreban. U sekciji Form Header treba dodati nekoliko labela (tačnije onoliko koliko ima polja tabele) i upisati sadržaj koji jasno asocira na to šta predstavlja kolona tabele.

Kada se ovo obavi treba klinuti na  da bi dobili Properties forme, a onda svojstvo *Default View* postaviti na *Continuous Forms* da bi bili prikazani svi slogovi iz tabele CLAN\_PORODICE. Snimiti ovu formu pod nekim imenom i zatvoriti je, a otvoriti onu koja treba da bude glavna. Sada iz ToolBox –a birati kontrolu *Subform/Subreport* i „nacrtati“ pravougaonu oblast u donjem delu forme. Ako je pre ovoga bio uključen Wizard on će voditi aktivnost do kraja. Ako nije onda u Properties –u za podformu treba uraditi sledeće

Naziv forme koja je „malo pre“ napravljena

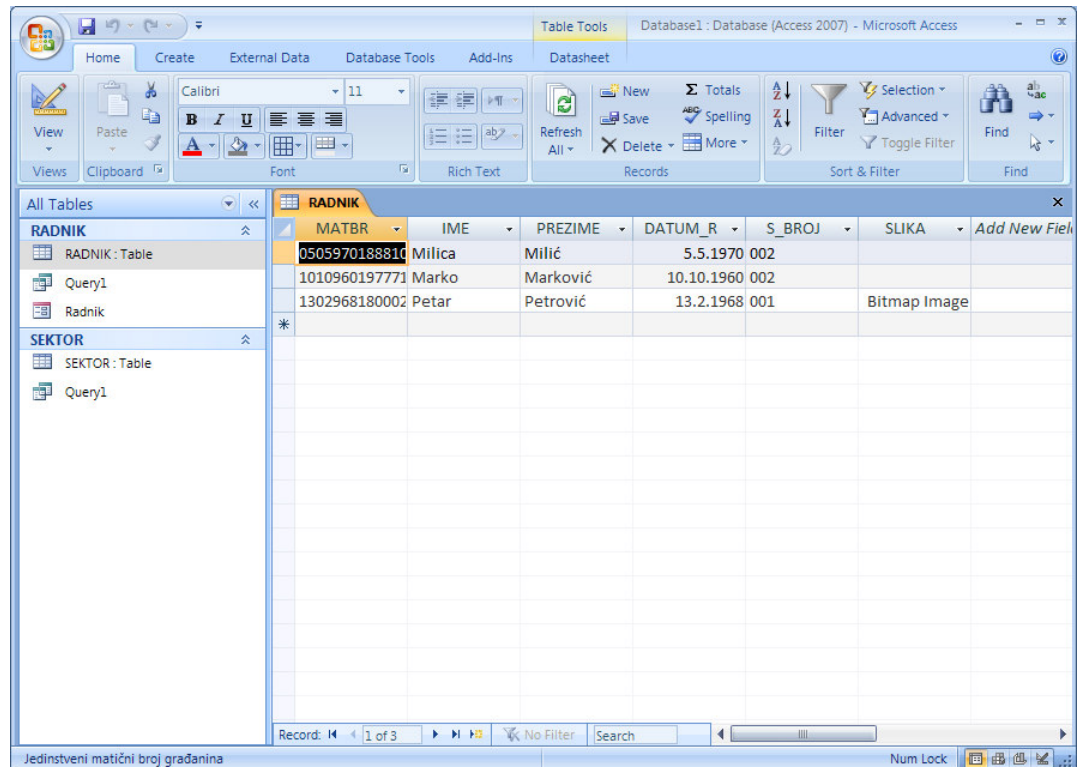
polje iz tabele slabog entiteta

polje (primarni ključ) iz tabele „jakog“ entiteta

Ovim podešavanjem u podformi neće biti prikazivani svi slogovi tabele CLAN\_PORODICE, već samo oni vezani za tekućeg radnika u glavnoj formi !

## 4.4 Upiti

Da bi bilo jednostavnije razumevanje upita pratićemo jedan primer. Koristićemo dve tabele – RADNIK u kojoj su podaci sa donje slike i SEKTOR u kojoj se nalaze podaci šifru i naziv sektora.

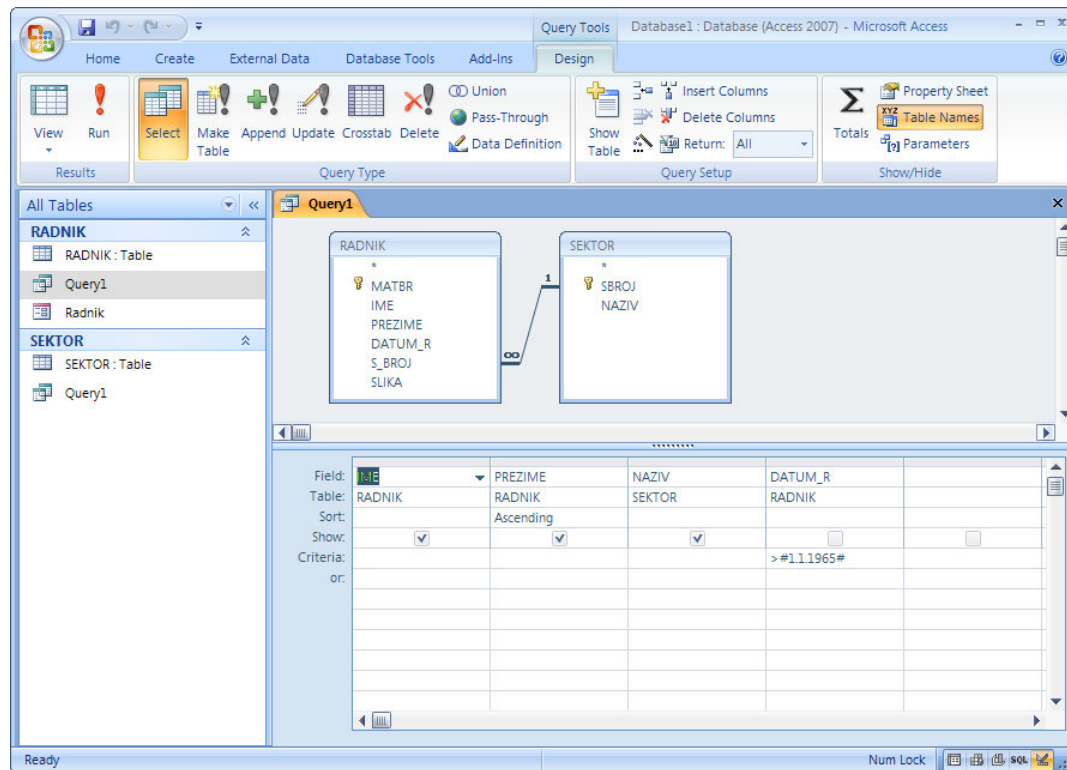


Pretpostavićemo dalje da nas interesuje odgovor na sledeće pitanje: izlistati spisak radnika, rođenih nakon 01.01.1965. sa nazivima sektora u kojima rade, sortirano po prezimenu radnika.

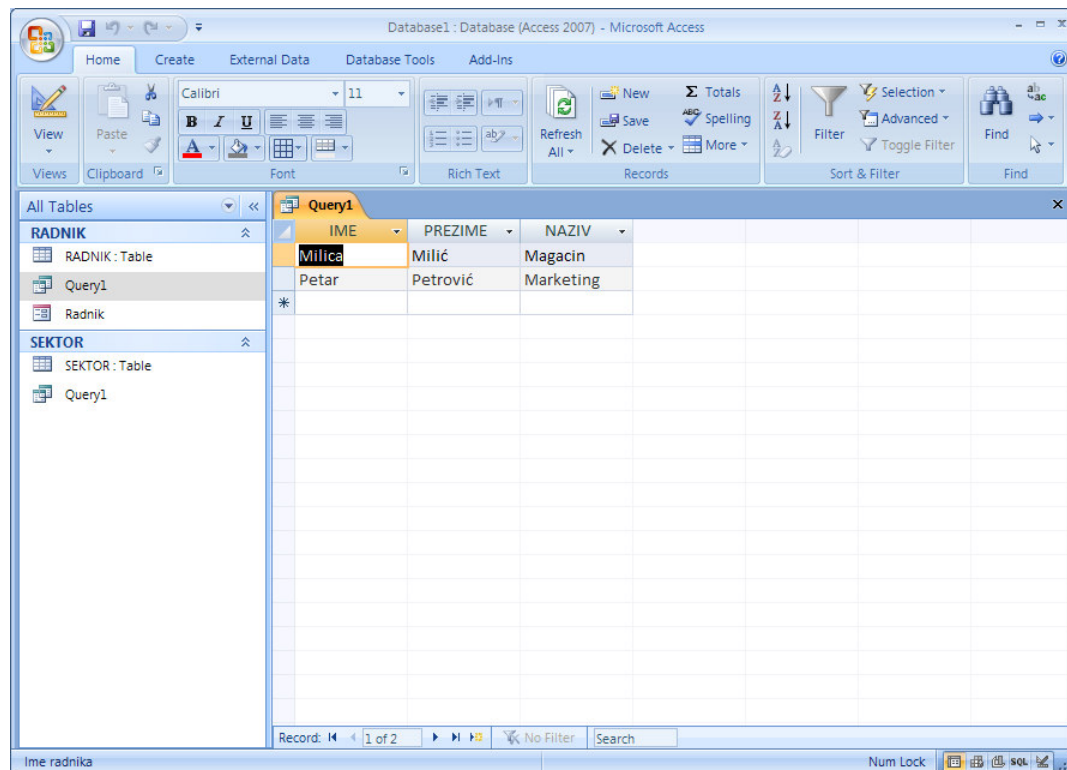
Da bismo dobili odgovor moramo postaviti odgovarajući upit (Query). Upit se može postaviti tako što će se napisati odgovarajuća SQL rečenica ili tako što će se koristiti Wizard. Da vidimo kako to ide preko Wizarada.

U tab –u Create postoji dugme Query Wizard. Kliknuvši na njega pokreće se alat tj proces u kome prvo treba da izaberemo tabele iz kojih ćemo „izvlačiti“ podatke, a kad to uradimo sledi određivanje tabele u donjem delu prozora, kao na sledećoj slici.

U prvom redu *Field* biramo redom polja iz tabela koja nam trebaju. U drugom redu *Table* su zapisane tabele iz kojih su ta polja. U trećem redu *Sort* biramo da li ćemo i po kom polju da radimo sortiranje (koje može biti rastuće i opadajuće; kako je slovo a „manje vrenosti“ nego b ako hoćemo uređenje po abecedi onda ćemo ostaviti *Ascending*). Četvrti red *Show* služi da biramo da li će polje biti prikazano u rezultatu upita ili ne. Ako nećemo – poništimo ček znak. Dolazimo do završnog red *Criteria* koji odgovara WHERE klauzuli u SQL upitu. Uslovom koji ovde zadamo vršimo izdvajanje (filtraciju) samo onih slogova iz tabele koji zadovoljavaju uslov. Kada smo završili sa određivanjem uslova, snimićemo upit pod nekim imenom.



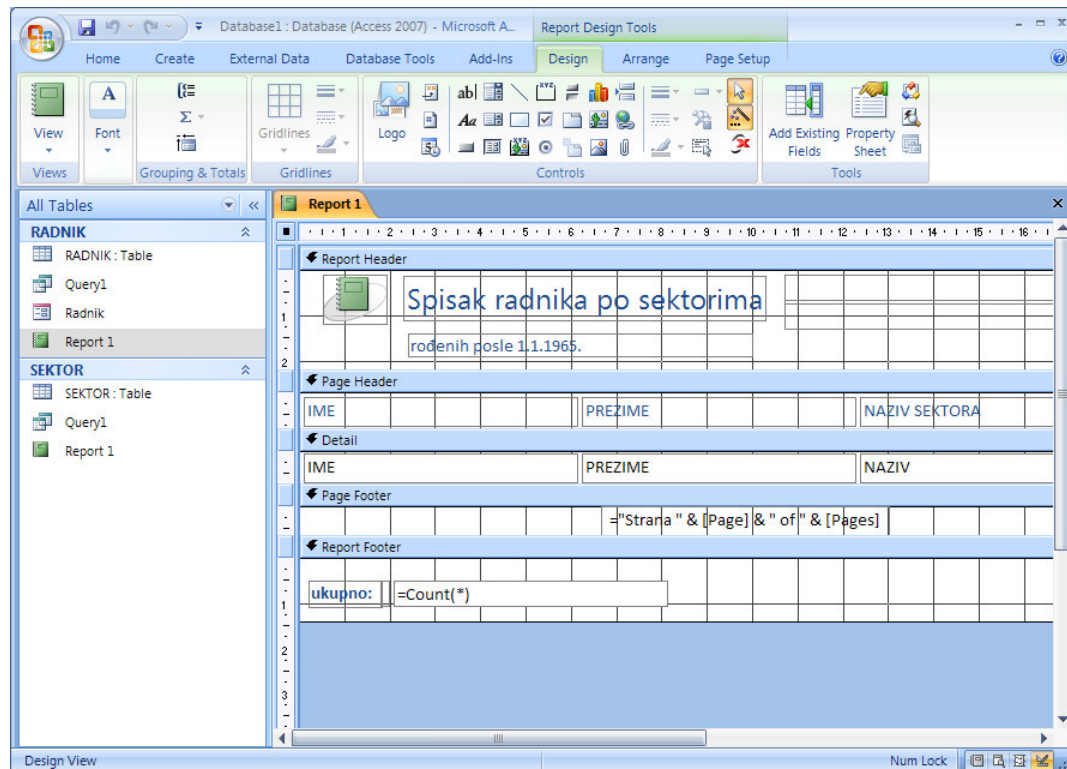
Sada preostaje da se upit pokrene (open). Rezultat je kao na sledećoj slici.



## 4.5 Izveštaji

Izveštaj je završni cilj obrade podataka. Kada se kreira realna baza podataka potrebno je vrlo pažljivo notirati koji su to tipski izveštaji koji će biti potrebni rukovodstvu firme i koji će se jednostavno dobijati klikom na taster. Naravno to ne isključuje mogućnost pravljenja dodatnih izveštaja u hodu.

Po svojoj prirodi izveštaj se pravi nad nekim prethodno definisanim upitom kojim je dobijen odgovor na željeno pitanje. Da bi smo dobili izveštaj nad upitom koji smo prethodno razmatrali potrebno je selektovati upit, a onda odabirom tab –a *Create* klinkuti na dugme *Report*. Dobićemo prozor kao na sledećoj slici. Ako hoćemo možemo izmeniti Caption –e labela ili dodati nove labela.



Nakon izvršenih intervencija izveštaj treba snimiti pod nekim imenom. Nakon toga se može pokrenuti pa bi reakcija Access –a bila kao na sledećoj slici. Ako je potrebno izveštaj se može pregledati (Print Preview) i štampati (Print).

Na kraju, kada se kreiraju svi potrebni elementi baze podataka – preostaje da se kreira jedna forma koja neće biti naslonjena ni na jednu tabelu/upit već će služiti kao glavna (main) forma u koju će se ugraditi meni-sistem ili komandni dugmići (command button -i) kojima će se otvarati forme, pokretati izveštaji, završavati rad sa aplikacijom itd.

Na kraju treba još napomenuti da se svi odgovori vezani za Access mogu pronaći osim u velikom broju knjiga dostupnih na tržištu, između ostalog i u Help sistemu same aplikacije.

