

Kolegij:	<b>BAZE PODATAKA 1</b> , ak.god. 2005/06 god.
Nositelj predmeta:	Dr. sc. Hrvoje Dujmić, doc..
Predavač:	Mr. sc. Renco Marasović
Asistenti:	Željko Agić, dipl.inž. Toni Jakovčević, dipl.inž.
Sadržaj kolegija:	Koncepti baza podataka. DBMS-Sustav baza podataka. Implementacijski modeli baza podataka. Entiteti i atributi. Relacije i funkcionalnost veza. Tipovi relacija. E-R model. Relacijski model. Normalizacija, normalne forme. Relacijska algebra - operatori, ključevi. Integritet baze, referencijalni integritet. Indeksiranje. SQL. Transakcije: obrada zahtjeva, blokiranje pristupa, kontrolne točke, oporavak od pogreški. Modeliranje događaja - okidači. Kontrola višestrukog pristupa. Sigurnost i dozvola pristupa. Projektiranje baza. Distribuirane baze. Sustavi korisnika i poslužnika.
Preporučena literatura:	<ul style="list-style-type: none"><li>• <b>P. O'Neil: "Database - Principles, Programming, Performance", Morgan Kaufmann Publishers, 1994.</b></li><li>• <b>C.J. Date: "An Introduction to Database Systems I", Addison-Wesley Publishing Company, 1990.</b></li><li>• <b>J.L. Harrington: "Relational Database Design Clearly Explained", Morgan Kaufmann Publishers,</b></li></ul>

**1998.**

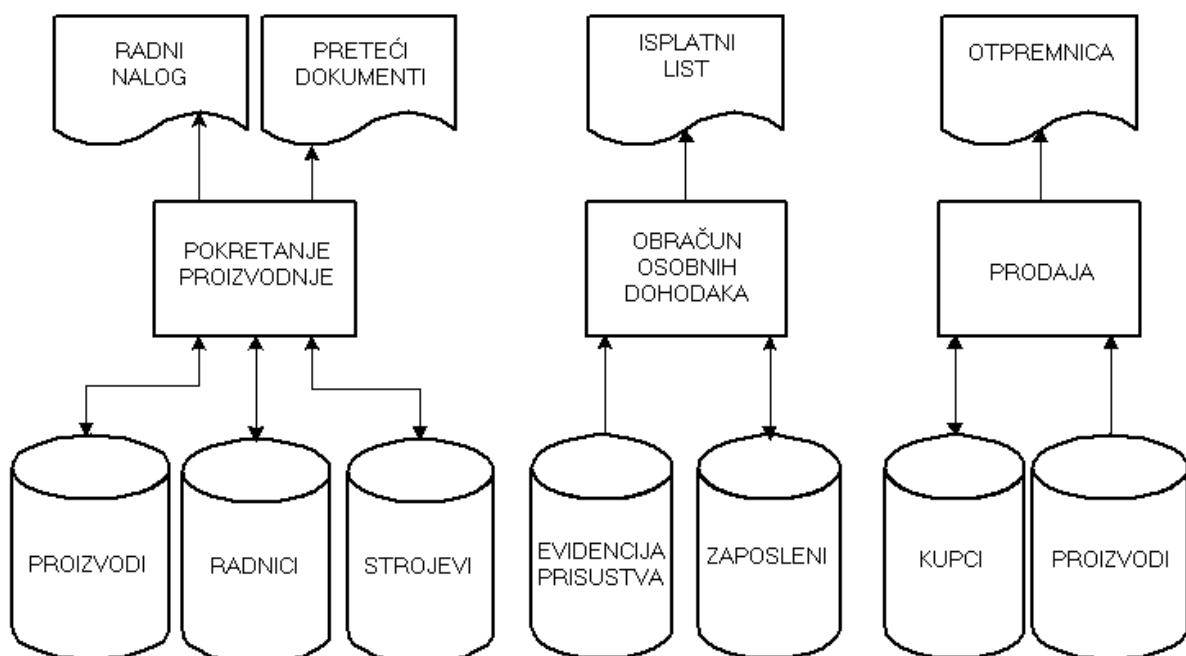
- T.J. Teorey: "Database Modeling & Design", Morgan Kaufmann Publishers, 1999.
- R. Vujnović: "SQL i relacijski model podataka", Znak, Zagreb, 1995.
- "Microsoft SQL server, Database Developer's Companion", Microsoft Press, 1996
- Paul Cassel, Craig Eddy i Jon Price: "Naučite Microsoft Access 2002 za 21 dan", Miš, Zagreb 2002.

# 1. OSNOVNO O BAZAMA PODATAKA

## 1.1. Problem klasične obrade podataka (File model)

Nedostatke klasične (datotečne) obrade podataka najbolje ćemo uočiti na jednom primjeru iz života. Pogledajmo izdvojeni skup "aplikacija" u informacijskom sustavu neke proizvodno-trgovačke radne organizacije koja koristi *klasičnu datotečnu obradu podataka (eng. file model)*.

**Informacijskim sustavom** zajednički nazivamo sve programe, podatke i informacije koji se obrađuju na računalu. Svaka aplikacija je razvijana posebno, koristi svoje "privatne" datoteke sa fizičkom strukturom podataka pogodnom upravo za tu aplikaciju.



**Slika 1.** Klasična obrada podataka. Nezavisne, "privatne" datoteke za svaku aplikaciju

Osnovni nedostaci ovakve obrade podataka su:

- Neminovna *redundancija podataka*, odnosno višestruko pamćenje istih podataka.

Npr., isti podaci o proizvodima se, u nekoj proizvodnoj radnoj organizaciji pamte i nekoliko desetaka puta, ili isti podaci o građanima u nekom komunalnom informacijskom sustavu i nekoliko stotina puta. Višestruko skladištenje istih podataka dovodi do nepremostivih problema pri njihovom ažuriranju (održavanju). Kada se neki podatak promijeni, to se mora učiniti na svim mjestima na kojima se on čuva. To, ne samo da značajno povećava troškove obrade podataka, nego je organizacijski praktično neizvodljivo, pa se, u raznim

izvještajima, pojavljuju različite verzije istog podatka.

- *Zavisnost programa od organizacije podataka.*

Programi su zavisni i od logičke i od fizičke strukture podataka. **Fizička struktura podataka** definira način spremanja podataka na medijima vanjske memorije.

**Logička struktura** je struktura podataka koja koristi programer. U klasičnim datotekama razlika fizičke i logičke strukture podataka je mala.

Zavisnost programa od logičke strukture se očituje u tome što program zavisi, npr., od naziva i redoslijeda polja u zapisu, što ubacivanje novog polja u zapis ili bilo kakvo drugo restrukturiranje zapisa, koje ne mijenja sadržaj podataka koje program koristi, ipak zahtijeva i izmjenu samog programa.

Fizička zavisnost se očituje u tome što program zavisi od izabrane fizičke organizacije datoteke i izabrane metode pristupa, načina sortiranja i slično.

U osnovi i *logička i fizička organizacija podataka su prilagođene konkretnom programu*. Novi zahtjevi za informacijama, iz podataka koji već postoje u datotekama, mogli bi zahtijevati izmjenu fizičke i logičke strukture podataka, a to bi zahtijevalo izmjene u ranije razvijenim programima. Umjesto toga, obično se za novi program stvara nova datoteka, a to dalje povećava redundanciju podataka.

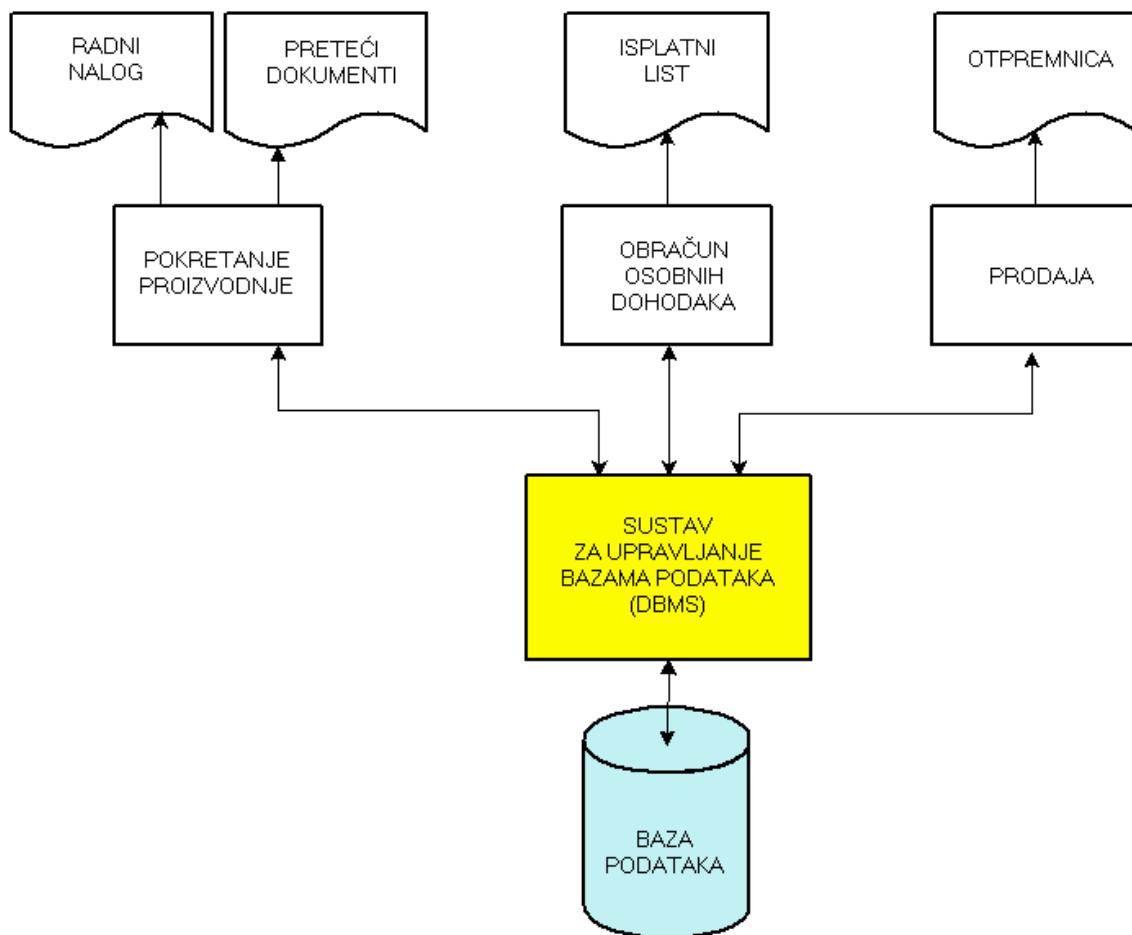
- *Niska produktivnost u razvoju informacijskog sustava (IS)*

Strukturiranje podataka u nezavisan skup datoteka je jedan od uzroka veoma niske produktivnosti u razvoju informacijskog sustava. Npr., čak i kada postoje svi podaci koji se zahtijevaju u nekoj aplikaciji, ali se oni nalaze u različitim datotekama, na raznim medijima, sa različitim fizičkim organizacijama, pa zadovoljenje i nekog jednostavnog zahtjeva iziskuje značajne napore programera. Nad strukturon podataka predstavljenom velikim brojem nezavisnih datoteka veoma je teško razviti softverske alate za brz i učinkovit razvoj aplikacija i za neposrednu

komunikaciju korisnika sa sustavom.

## 1.2. Baza podataka i sustav za upravljanje bazama podataka

Prethodno nabrojeni razlozi uvjetovali su značajne tehnološke izmjene u upravljanju podacima. Od sredine šezdesetih godina do danas, izuzetno se brzo razvijaju *sustavi za upravljanje bazom podataka (SUBP)* ili eng. *database management systems (DBMS)*.



**Slika 2.** Sustav za upravljanje bazama podataka

**Baza podataka** je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim programima. *Umjesto da su razbacani po nezavisnim datotekama, podaci su organizirani u jedinstvenu strukturu baze podataka*. Podaci postaju jedinstveni resurs sustava i njima se mora upravljati na jedinstven način, onako kako se upravlja i sa svim drugim vitalnim resursima poslovnih sustava.

Dodavanje, izmjena, brisanje i čitanje podataka obavlja se posredstvom **zajedničkog softvera**. Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka, već se referenciraju na logičku strukturu baze podataka. *Sustav za upravljanje bazom podataka (Data Base Management System - DBMS) je poslužitelj (server) baze podataka*. On oblikuje fizički prikaz baze u skladu s traženom logičkom

strukturom. Također, on obavlja u ime klijenata sve operacije s podacima. Isto tako, brine se za sigurnost podataka, te automatizira administrativne poslove s bazom.

Baze podataka predstavljaju višu razinu rada s podacima u odnosu na klasične programske jezike. Ta viša razina rada očituje se u tome što tehnologija baza podataka nastoji (i u velikoj mjeri uspijeva) ispuniti sljedeće ciljeve:

- Spremanje podataka sa *minimalnom mogućom redundancijom*
- *Fizička nezavisnost podataka*. Razdvaja se logička definicija baze od njene stvarne fizičke grade. Znači, ako se fizička građa promijeni (na primjer, podaci se prepišu u druge datoteke na drugim diskovima), to neće zahtijevati promjene u postojećim aplikacijama.
- *Logička nezavisnost podataka*. Razdvaja se globalna logička definicija cijele baze podataka od lokalne logičke definicije za jednu aplikaciju. Znači, ako se logička definicija promijeni (na primjer uvede se novi zapis ili veza), to neće zahtijevati promjene 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*. Zahtijeva se da korisnik može slobodno prebirati po podacima, te po svom nahođenju uspostavljati veze među podacima. Ovaj zahtjev zadovoljavaju jedino relacijske baze.
- *Istovremeni pristup do podataka*. Baza mora omogućiti da veći broj korisnika istovremeno koristi iste podatke. Pritom ti korisnici ne smiju ometati jedan drugoga, te svaki od njih treba imati dojam 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 neovlaštenog korištenja*. Mora postojati mogućnost da se korisnicima ograniče prava korištenja baze, dakle da se svakom korisniku reguliraju ovlaštenja što on smije a što ne smije 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 utjecati

odabirom pogodnih fizičkih struktura podataka, te izborom pogodnih algoritama za pretraživanje.

- **Mogućnost podešavanja i kontrole.** Velika baza zahtijeva stalnu brigu: praćenje performansi, mijenjanje parametara u fizičkoj gradi, rutinsko pohranjivanje rezervnih kopija podataka, reguliranje ovlaštenja korisnika. Također, svrha baze se vremenom mijenja, pa povremeno treba podesiti i logičku strukturu.

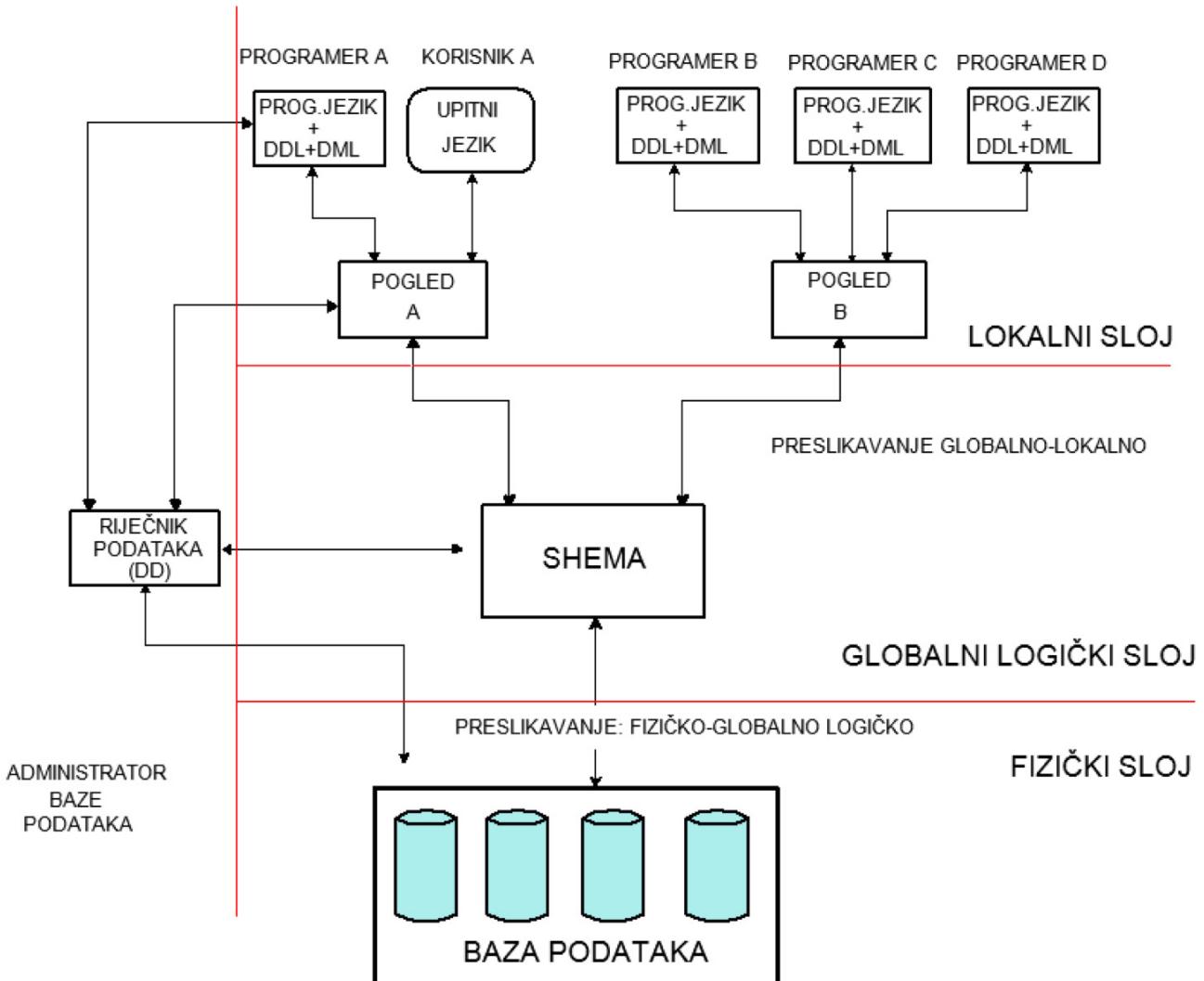
Ovakvi poslovi moraju se obavljati centralizirano.

### 1.3. Kako se ostvaruje fizička i logička nezavisnost podataka?

Da bi lakše objasnili kako je ostvarena fizička i logička nezavisnost podataka, sustav za upravljanje bazama podataka moramo promatrati kao višeslojnu arhitekturu sa strogo definiranim sučeljima između slojeva, kao što je prikazano na Slici 3. Uočimo postojanje tri sloja:

- **Fizički sloj** odnosi se na fizički prikaz i raspored podataka na jedinicama vanjske memorije. To je aspekt kojeg vide samo **sistem-programeri** (oni koji su razvili DBMS). Sam fizički sloj može se dalje podijeliti na više pod-razina apstrakcije, od sasvim konkretnih staza i cilindara na disku, do već donekle apstraktnih pojmoveva datoteke i zapisa kakve susrećemo u klasičnim programskim jezicima. Raspored pohranjivanja opisuje kako se elementi logičke definicije baze preslikavaju na fizičke uređaje.
- **Globalni logički sloj** odnosi se na logičku strukturu cijele baze. To je aspekt kojeg vidi **projektant baze** odnosno njen **administrator**. Zapis logičke definicije naziva se **shema** (*eng. schema*). Shema je tekst ili dijagram koji definira logičku strukturu baze, i u skladu je sa zadanim modelom (koji je implementiran u DBMS-u). Dakle imenuju se i definiraju svi tipovi podataka i veze među tim tipovima, u skladu s pravilima korištenog modela. Također, shema uvodi i ograničenja kojim se čuva integritet podataka.
- **Lokalni logički sloj** odnosi se na logičku predodžbu o dijelu baze kojeg koristi pojedina aplikacija. To je aspekt kojeg vidi **korisnik** ili **aplikacijski programer**. Zapis jedne lokalne logičke definicije zove se **pogled** (*engleski view*) ili **podshema**. To je tekst ili dijagram kojim se imenuju i definiraju svi lokalni tipovi podataka i veze među tim tipovima, opet u skladu s pravilima korištenog modela. Također, pogled zadaje **preslikavanje kojim se iz globalnih podataka i veza izvode lokalni**.

Ako se promijeni fizička struktura baze podataka, nije neophodno mijenjati globalnu logičku strukturu (shemu baze podataka), već samo preslikavanje sheme baze podataka u fizičku razinu. Isto tako, ako se promijeni struktura sheme, pogledi ostaju nepromijenjeni, mijenja se samo način preslikavanja sheme u pogled. Time se ostvaruje logička i fizička nezavisnost programa od podataka.



Slika 3. Arhitektura sustava za upravljanje bazom podataka

Na fizičkom sloju, preko jezika koji se naziva **jezik za definiciju podataka** (*Data Definition Language - DDL*), specificira se fizička organizacija podataka, odnosno metode pristupa.

Programer razvija svoju aplikaciju nad njemu pogodnom logičkom strukturu koja predstavlja njegov **"pogled"** na cjelokupnu bazu podataka. On za to koristi neki uobičajeni programski jezik (C++, Basic, Javu ili neki drugi), u koji se ugrađuju **naredbe DDL-a** i naredbe **jezika za rukovanje podacima u bazi podataka** (*Data Manipulation Language - DML*). Naredbe DML omogućuju "manevriranje" po bazi, te jednostavne operacije kao što su čitanje, upis, promjenu ili brisanje 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 izmiješane naredbe dvaju jezika, pa takav program treba prevoditi s dva prevodioca (DML-precompiler, obični compiler).

Za korisnike neprofesionalce obično postoji i neki neproceduralni *upitni jezik (Query Language)*. On služi za interaktivno pretraživanje baze. To je jezik koji podsjeća na govorni (engleski) jezik. Naredbe su neproceduralne, dakle takve da samo specificiraju rezultat kojeg želimo dobiti, a ne i postupak za dobivanje rezultata.



Jezik za definiciju podataka (Data Definition Language - DDL) i jezik za rukovanje podacima u bazi podataka (Data Manipulation Language - DML) 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.

Ovakva podjela na tri jezika danas je već prilično zastarjela. Naime, kod relacijskih baza postoji tendencija da se *sva tri jezika objedine u jedan sveobuhvatni*. Primjer takvog integriranog jezika za relacijske baze je *SQL: on služi za definiranje, manipuliranje i pretraživanje podataka*.

Integrirani jezik se može koristiti interaktivno (preko on-line interpretera) ili se on može pojavljivati uklopljen u aplikacijske programe.

U 80-tim godinama 20. stoljeća bili su dosta popularni i tzv. jezici 4. generacije (4-th Generation Languages - 4GL): riječ je o jezicima koji su bili namijenjeni isključivo za rad s bazama, te su zato u tom kontekstu bili produktivniji od klasičnih programskih jezika opće namjene. Problem s jezicima 4. generacije je bio u njihovoј nestandardnosti: svaki od njih je u pravilu bio dio nekog određenog softverskog paketa za baze podataka, te se nije mogao koristiti izvan tog paketa (baze).

U današnje vrijeme, aplikacije se najčešće razvijaju u popularnim objektno orijentiranim programskim jezicima (Java, C++, . . . ). Za interakcije s bazom koriste se unaprijed pripremljene klase objekata.

Ovakva tehnika je dovoljno produktivna zbog korištenja gotovih klasa, a rezultirajući program se lako dotjeruje, uklapa u veće sustave ili prenosi s jedne baze na drugu.

Glavna i odgovorna osoba u smislu upravljanja cjelokupnim sustavom baze podataka je *administrator baze podataka*. On upravlja radom cjelokupnog sustava obavljajući sljedeće funkcije:

- Razvoj globalnog sloja, odnosno upravljanje podacima u sustavu
- Izbor fizičke organizacije baze podataka i metoda pristupa

podacima

- Definicija i realizacija preslikavanja između globalni logičkog sloja - fizičkog sloj i globalnog logičkog sloja - lokalnih pogleda, odnosno definiranje ili pomoć u definiranju pogleda na bazu podataka
- Osiguravanje sigurnosti podataka i integriteta baze podataka
- Definiranje strategija oporavka baze podataka poslije mogućih oštećenja (problema)
- Praćenje efikasnosti (performansi) sustava

Administrator baze podataka koristi jezik za definiciju podataka (DDL) za definiciju opće logičke strukture baze podataka.

Da bi lakše obavio svoje zadatke administrator se koristi različitim pomoćnim programima::

- programi za punjenje baze podataka (BP),
- "Dump/restore" rutine,
- programi za fizičku reorganizaciju BP,
- programi za ispis i analizu statistike korištenja podataka,

Ipak, osnovni alat administratora baze podataka je *rječnik (katalog) baze podataka (Data Dictionary)*. *Rječnik baze podataka je baza podataka o bazi podataka (meta baza podataka)* i u njemu su opisani svi objekti sustava (podaci, fizičke i logičke strukture, prava korištenja i slično).

Rječnik podataka se puni prilikom definiranja odgovarajućih objekata u sustavu.

Zbog složenosti navedenih osnovnih i pomoćnih funkcija sustavi za upravljanje bazama podataka su složeni i veoma skupi software-ski sustavi. Međutim prednosti koje pružaju su takve da se i pored visoke cijene gotovo svuda uvode.

## 1.4. Metodologija projektiranja baza podataka

Tehnologija baza podataka omogućila je da se problem razvoja IS odnosu na realni sustav u kome djeluje postavi na slijedeći način:

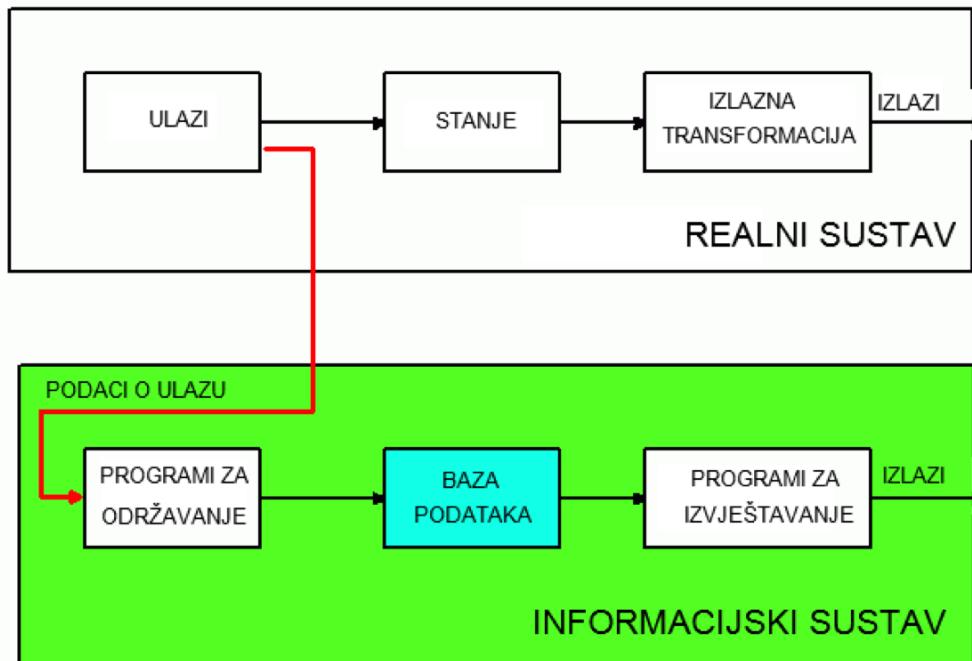
Sustav se najopćenitije definira kao skup objekata (entiteta) i njihovih međusobnih veza (relacija). Objekti u sustavu mogu biti neki fizički objekti, koncepti, događaji i drugo. Objekti se u modelu nekog sustava opisuju preko svojih svojstava (atributa). Djelovanje okoline na sustav opisuje se preko ulaza u sustav, a djelovanje sustava na okolinu preko njegovih izlaza. Dinamičko ponašanje realnog sustava standardno se shematski predstavlja kao na gornjem dijelu slike 10.

Ulazi u sustav mijenjaju stanja sustava. Stanje sustava definira se kao skup informacija o prošlosti i sadašnjosti sustava koji je potreban da bi

se, uz djelovanje budućih poznatih ulaza, mogli odrediti budući izlazi. U stanju sustava sadržana je cijelokupna povijest realnog sustava.

Očigledno je da stanje sustava opisuje osnovne karakteristike sustava. U jednom trenutku vremena ono predstavlja skup objekata sustava, skup njihovih međusobnih veza i skup vrijednosti atributa objekata u upravo tom trenutku.

Izlazna transformacija definira neki način mjerjenja ili promatranja dinamičkog ponašanja realnog sustava i daje, na osnovu stanja sustava, njegove izlaze.



**Slika 12.** Položaj informacijskog sistema u odnosu na realni sistem

Pojasnimo, prethodne postavke primjeru nekog skladište proizvoda kao sustava. Objekti u ovom sustavu su sami proizvodi, njihovi atributi su npr., naziv, osobine, jedinica mjere i količina u skladištu. Veza između objekata je, npr., sastav proizvoda, koji pokazuje od kojih se drugih proizvoda, promatrani proizvod, sastoji. Svaki prijem ili izdavanje proizvoda iz skladišta predstavlja ulaz u sustav (jer mijenja njegova stanja). Osnovna komponenta stanja u sustavu je količina svakog proizvoda u skladištu. Izlaz iz sustava može da bude količina zaliha svakog proizvoda (samo stanje sustava), zatim ukupna vrijednost zaliha, ukupna vrijednost zaliha proizvoda određene vrste i slično. Izlazna transformacija daje ove izlaze na osnovu stanja sustava. Kao što je to na Slici 12 prikazano, informacijski sustav predstavlja model realnog sustava u kome djeluje. Podaci o ulazu (zahtjevnice, primke, otpremnice i sl. za navedeni primjer), preko programa za održavanje (koji modeliraju djelovanje ulaza na sustav), djeluju na bazu podataka (kartica zaliha materijala, za navedeni primjer), koja modelira stanje sustava. Programi za izvještavanje predstavljaju model izlazne transformacije (postupak izračunavanja ukupne vrijednosti zaliha, ...) i daju zahtijevane izlaze.

Osnovu informacijskog sustava čini baza podataka, jer ona predstavlja osnovne, stabilne, sporo promjenjive osobine sustava, objekte u sustavu i njihove međusobne veze. Zato se projekt IS mora bazirati na bazi podataka.

Ako je baza podataka dobar model stanja realnog sustava, ako programi za održavanje dobro modeliraju djelovanje ulaza na stanje realnog sustava, onda će se bilo koja informacija potrebna za upravljanje (izlazi), čak i one koje nismo unaprijed predvidjeli, moći dobiti iz IS.

Ako je informacijski sustav model realnog sustava u kome djeluje, onda se postupak projektiranja IS svodi na neku vrstu modeliranja realnog sustava, a za to su nam neophodna neka intelektualna sredstva (alati) i to:

- *Model podataka* kao intelektualno sredstvo za prikazivanje objekata sustava, njihovih atributa i njihovih međusobnih veza (statičkih karakteristika sustava) preko logičke strukture baze podataka.
- *Model procesa* kao intelektualno sredstvo za opisivanje dinamike sustava, djelovanja ulaza na stanje sustava i izlazne transformacije, preko programa nad definiranim modelom podataka.

U ovom kolegiju baviti ćemo se samo modelima podataka, dok će modeli procesa biti prikazani samo usput, u onoj mjeri u kojoj to bude neophodno za izlaganje procesa projektiranja baza podataka.

**Podatak je neka (kodirana) činjenica iz realnog sustava, on je nosilac informacije.**

**Informacija je protumačeni (interpretirani) podatak.**

Krajnju interpretaciju podataka iz baze podataka daje korisnik IS, preko odgovarajućih programa ili upita. Interpretacija podataka se vrši na osnovu strukture podataka, semantičkih ograničenja na njihove vrijednosti, a preko operacija koje se nad njima mogu izvršiti. Zbog toga svaki model podataka posjeduje tri osnovne komponente:

- *Strukturu modela*, odnosno skup postavki za opis objekata sustava, njihovih atributa i njihovih međusobnih veza
- *Ograničenja - semantička ograničenja vrijednosti podataka* koja se ne mogu predstaviti samom strukturom modela, a koja u svakom trenutku moraju biti zadovoljena. Ova ograničenja se obično nazivaju *pravilima integriteta modela podataka*
- *Operacije nad postavkama strukture*, pod definiranim ograničenjima, preko kojih je moguće opisati dinamiku sustava u modelima procesa

Na osnovu definicije, očigledno je da svaki model podataka treba da zadovolji dva temeljna kriterija:

- mora biti pogodan za modeliranje realnih sustava
- postavke, struktura, ograničenja i operacije koje se dobiju modeliranjem mogu jednostavno implementirati na računalu

Po tome kako zadovoljavaju ove kriterije modeli podataka mogu se svrstati u sljedeće skupine:

- *Prvu generaciju čine standardni programski jezici*, koji se također mogu tretirati kao modeli podataka. Postavke za opisivanje strukture u njima su tipovi podataka kojima raspolažu (prosti i složeni tipovi kao što su zapis (record), vektor, lista, stablo, koji se međusobno ne mogu eksplicitno povezivati na željene načine, pa se zato model podataka u njima predstavlja kao skup nepovezanih datoteka), ograničenja nad ovim tipovima obično nije moguće direktno postaviti, a same operacije nisu dovoljno moćne. Oni nisu dovoljno pogodni za modeliranje realnog sustava (zato je programiranje u njima teško), ali se model realnog sustava opisan pomoću njih može direktno implementirati na računalu.
- *Drugu generaciju čine tri klasična modela baze podataka, hijerarhijski, mrežni i relacijski model*. Ovi modeli posjeduju semantički bogatije koncepte za opis strukture (moguće je definirati način povezivanja zapisa, odnosno bazu podataka kao skup međusobno povezanih podataka) i znatno moćnije (makro) operacije. Zbog toga su pogodniji za opis realnog sustava, a postoje i komercijalno raspoloživi software-i, (SUBP/DBMS), za njihovu direktnu implementaciju na računalu.
- *Treću generaciju čine tzv. semantički bogati modeli podataka i objektni modeli podataka* (Model objekt i veze (entity-releationship), Semantic Data Model (SDM), Prošireni relacijski model, Semantičke mreže i drugi), koji posjeduju semantički bogate koncepte za opis realnog sustava, ali za njih još uvijek ne postoje software-i preko kojih bi se direktno mogli implementirati na računalu.

Imajući u vidu karakteristike generacije modela podataka, zadovoljavajući metodološki pristup projektiranju baza mogao bi biti:

- *koristeći neki model Treće generacije, formirati semantički bogat model realnog sustava koji se analizira*
- *Prevesti dobiveni model u neki od modela Prve ili Druge generacije, u zavisnosti od software-a kojim raspolažemo i*

drugih faktora, i na taj način implementirati bazu podataka na računalu. Postupak prevođenja sa semantički bogatijeg, na semantički siromašniji model lako se može se formalizirati, pa samim tim i automatizirati.

U ovom kolegiju materijalu koristićemo Entity-relationship model (Model objekti - veze), najpopularniji semantički model za prvi korak, a relacijski model baze podataka će biti implementacijski model.

## 1.5. Modeli baza podataka

Znamo da su programski jezici vremenom evoluirali od neproceduralnih do objektno-orientiranih.

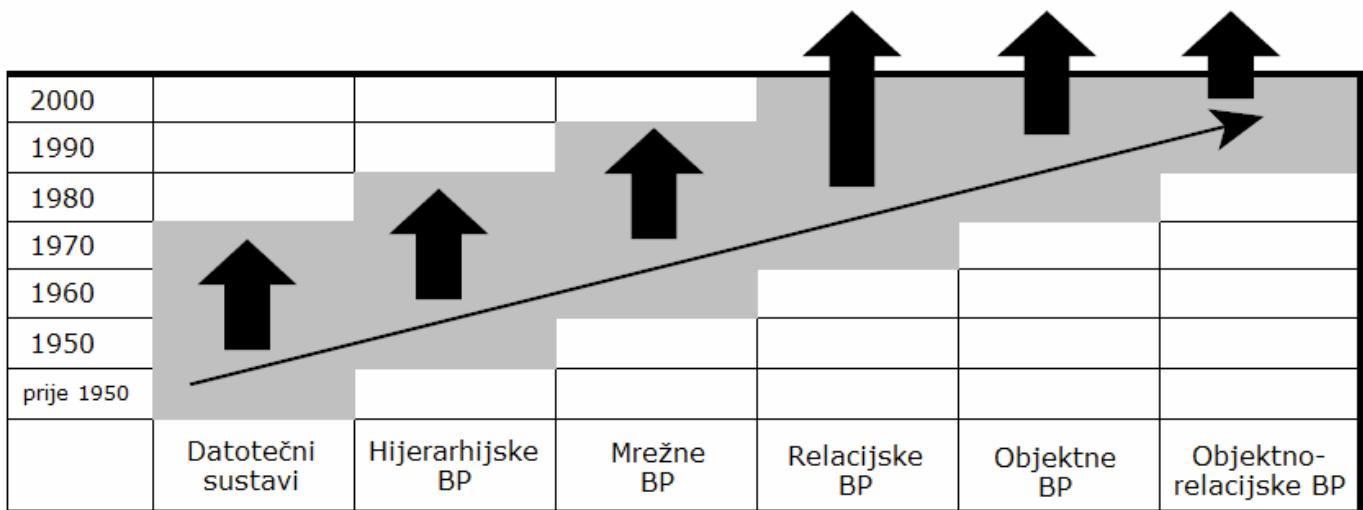
Zbog uočavanih ograničenja u modelima baza podataka, kao što su:

- nedovoljno dobro opisivanje stvarnih veza među podacima
- rast opsega i zahtjeva obrada nad bazom podataka
- promjena karaktera podataka koji se pohranjuju u baze podataka
- povećanje učinkovitosti svih sudionika u obradi podataka (od administratora baze podataka, preko programera do korisnika podataka)
- rastom i većom raspoloživošću hardvera

oni su vremenom evoluirali

Zahvaljujući novim teoretskim postavkama novi su sustavi za upravljanje bazama podataka uspješno su uklanjali trenutna ograničenja. Trend unaprjeđivanja, danas je još naglašeniji.

Na slici 4 prikazani su dosada primjenjivani modeli sustava za upravljanje bazama podataka od njihove pojave do danas.

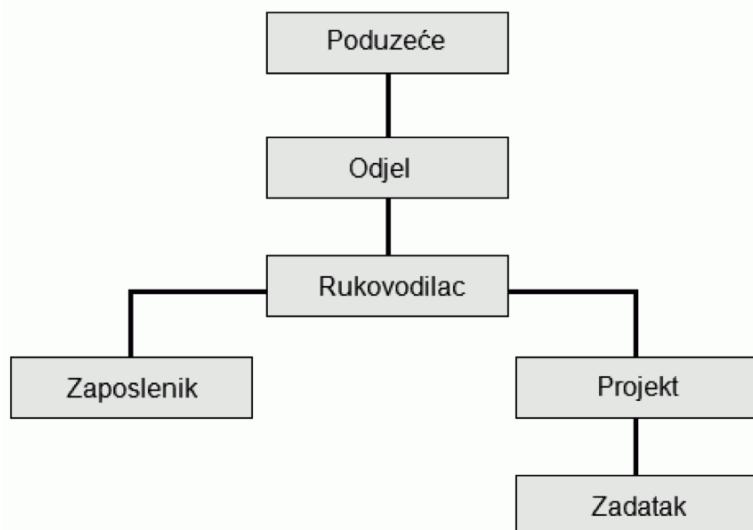


Slika 4. Evolucija modela baza podataka

Pogledajmo karakteristike primjenjivanih modela sustava za upravljanje

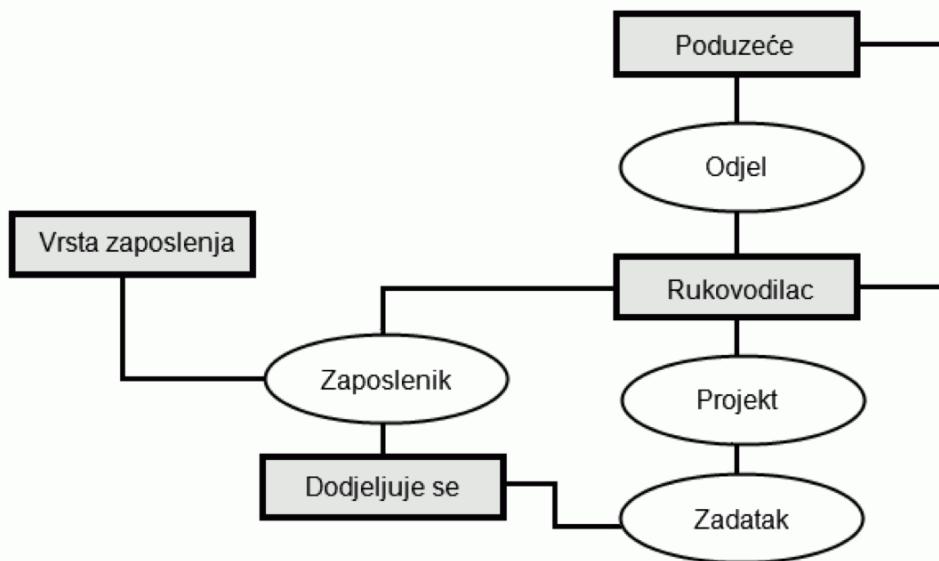
bazama podataka. Tipovi DBMS-ova kronološki su navedeni od starijih prema novijima.

- **Hijerarhijske model baze podataka.** Baza je predložena jednim (izokrenutim) stablom ili skupom (izokrenutih) stabala. Pravokutnicima se označavaju tipovi zapisa. Za svaki par povezanih zapisa u međusobno susjednim razinama utvrđuje se odnos "roditelj-dijete" (roditelj je podatak na višoj razini). Svaki zapis podataka na vršnoj razini je hijerarhijski određen kao korijenski segment. Unutar svakog zapisa dodaju se pokazivači prema zapisu na višoj razini (prema gore) i prema zapisima na nižoj razini (prema dolje) u skladu sa određenjem veza roditelj-dijete. Među zapisima se uspostavlja **fizičko povezivanje**.



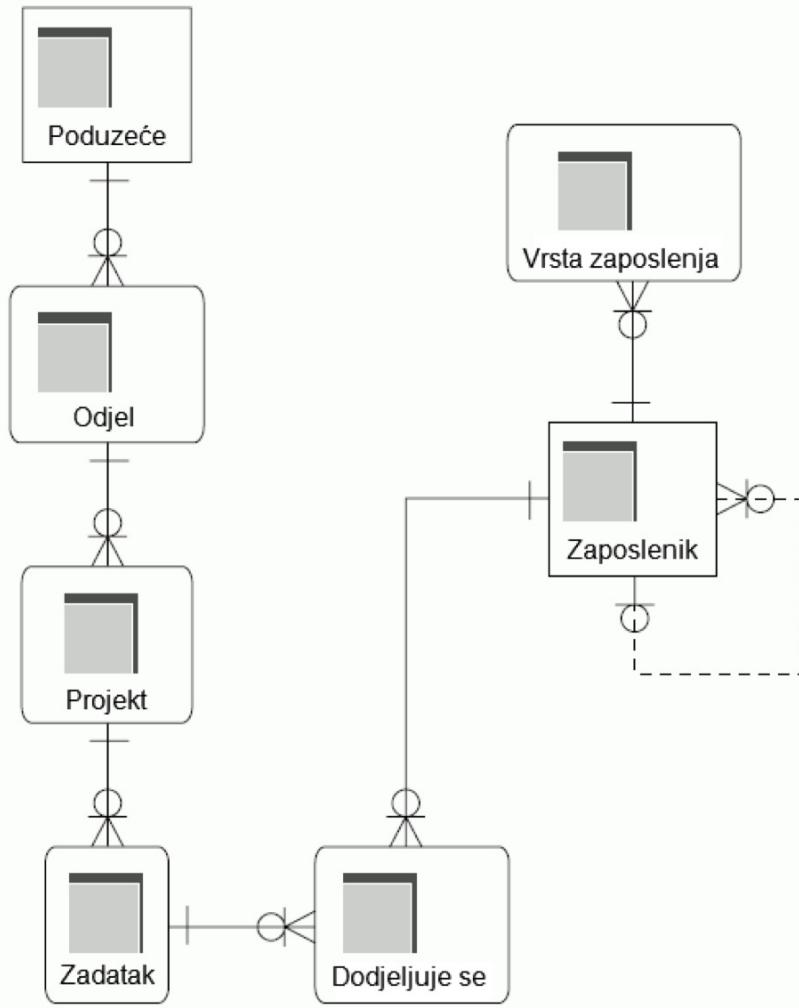
**Slika 5.** Primjer hijerarhijske baze podataka

- **Mrežne model baze podataka.** Veze među zapisima znatno bolje opisuju veze među podacima u realnom svijetu. Baza je predložena usmjerenim grafom. Pravokutnicima se označavaju tipovi zapisa a ovali (elipsoidi) veze među tim zapisima. Međusobno zavisne strukture podataka nemaju strogo utvrđenu hijerarhijsku strukturu. Zapisi se tipiziraju. Tip podataka je struktura podataka koja se može objediniti zahvaljujući zajedničkim osobinama podataka. Uvode se vezni segmenti podataka koji postoje i vezuju se sa njihovim vlasnicima. Ustanavljava se princip višestrukih roditelja. Vezni zapisi mogu imati istovremeno više roditelja. Identično kao i kod hijerarhijskog modela unutar zapisa se dodaju pokazivač, tj. među zapisima se uspostavlja **fizičko povezivanje**.



**Slika 6.** Primjer mrežne baze podataka

- **Relacijski model baze podataka.** Zasnovane su na teoretskim postavkama koje je oblikovao Dr. E.F.Codd 1971 god. Osnova je matematičkom pojmu relacija. Relacija je dvodimenzionalna tablica. Svaki redak tablice odgovara jednom zapisu. Stupaci tablice opisuje elemente (attribute) sa isti smislom (osobinama). I podaci i veze među podacima prikazuju se relacijskim tablicama. Osnovna razlika u odnosu na prethodne modele je nepostojanje fizičke veze među podacima. Veze među podacima su potpuno logičke. Veza između zapisa u relacijskim tabelama uspostavlja se primjenom stranih ključeva. Ključ nekog zapisa je atribut ili skup atributa koji nedvosmisleno određuje sve članove zapisa.



Slika 7. Primjer relacijske baze podataka

PROJEKT_ID	ODJEL_ID	OPIS_PROJEKTA	PROJEKT	ZAVRSETAK	CIJENA
1	1	Projektna dokumentacija Helios	Projekt	01.03.2006	35,000
2	1	Asfaltiranje prilaza		15.02.2007	50,000
3	2	Preuređenje interijera		31.12.2006	25,000,000
4	1	Uređenje prilaznog puta		20.06.2006	250,000

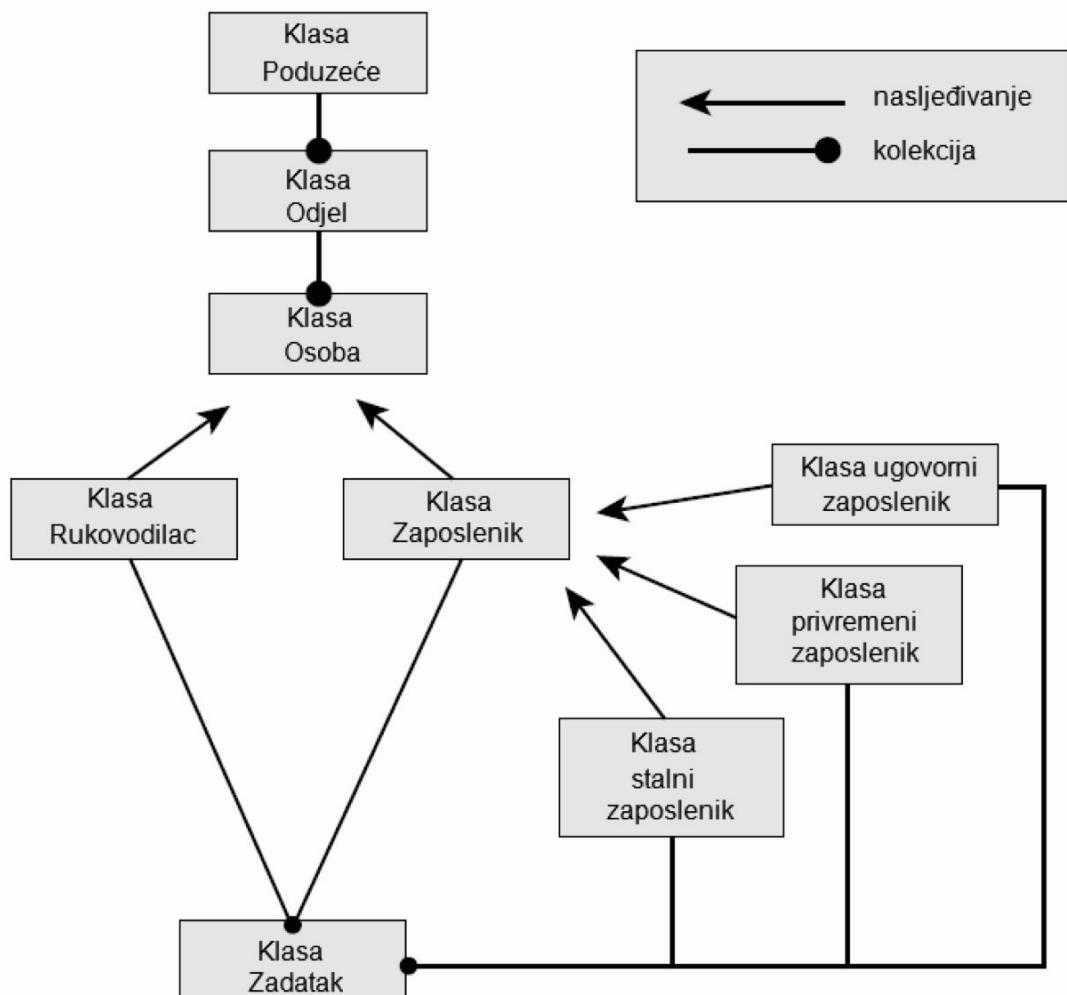
  

ZADATAK_ID	PROJEKT_ID	OPIS_ZADATKA	ZADATAK
1	1	Analiza stanja	
2	1	Geodetska mjerjenja	
3	1	Sondiranje terena	
4	2	Asfalterski radovi	
5	3	Elektičarski radove	
6	3	Vodoinstalaterski radovi	
7	3	Bojadisarski radovi	
8	3	Limarski radovi	
9	3	Drvodjelski radovi	

Slika 8. Primjer uspostavljanje veza među relacijama

- **Objektni model baza podataka.** Inspiriran je objektno-orientiranim 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 nasljeđivanja, agregacije, odnosno međusobnog korištenja operacija.

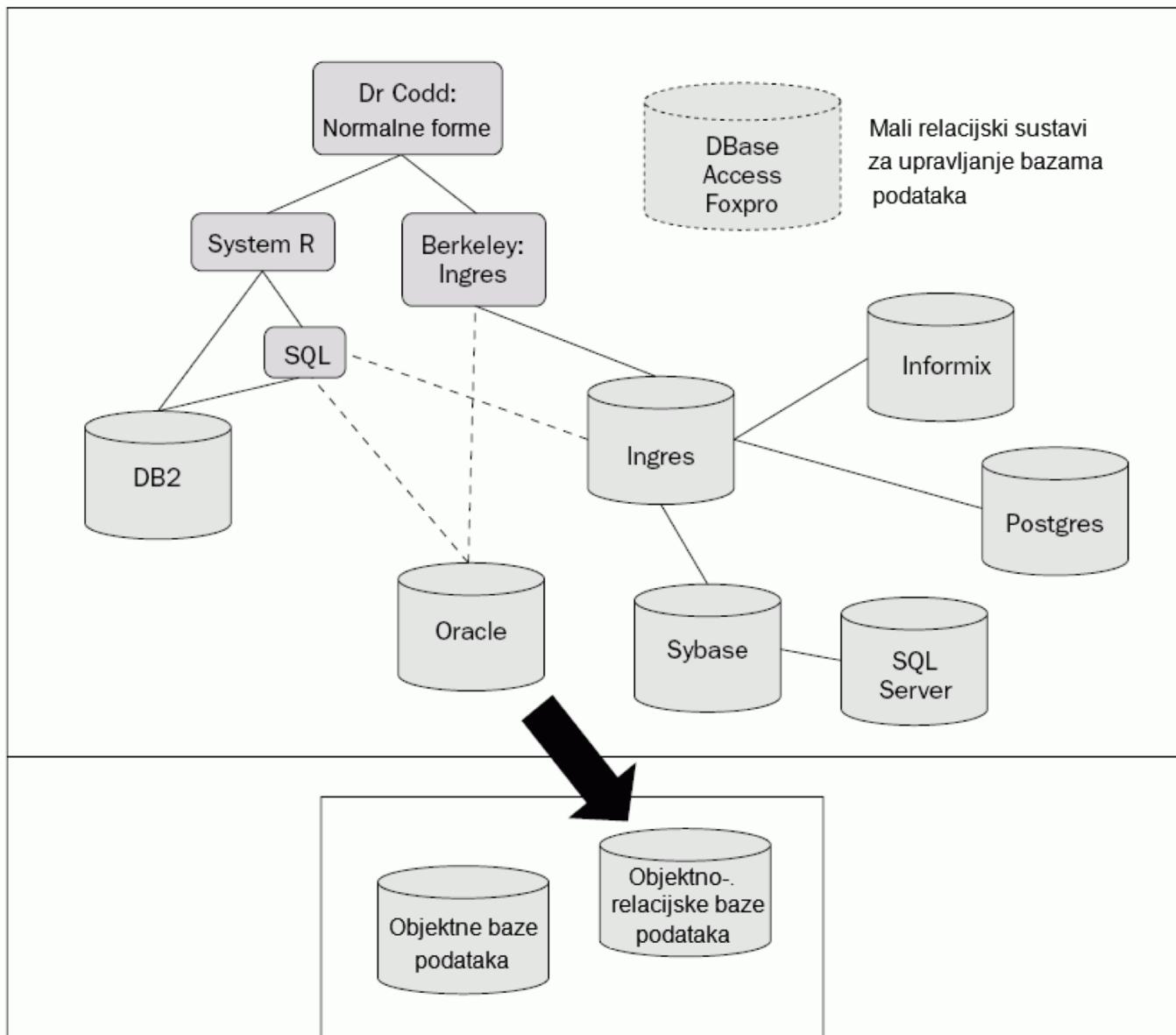


**Slika 10.** Primjer objektne baze podataka

- **Objektno-relacijske model baze podataka**. Otklanjaju probleme objektnih baza podataka. Teoretska podloga već postoji, kasni valjana implementacija.

Hijerarhijski i mrežni model bili su u upotrebi u 60-tim i 70-tim godinama 20. stoljeća. Od 80-tih godina pa sve do današnjih dana prevladava relacijski model. Očekivani prijelaz na objektne, pa kasnije i na objektno-relacijske baze podataka za sada se nije desio (iako je teoretska osnova postoji već duže vrijeme), tako da današnje baze podataka uglavnom još uvijek možemo poistovjetiti s relacijskim bazama.

Sustavi za upravljanje bazama podataka koji podržavaju relacijski implementacijski model prikazani su na slici 11.



**Slika 11.** Povijest relacijskog modela baza podataka.

Uočite postojanje velikog broja sustava za upravljanje bazama podataka različitih proizvođača. Veći broj nezavisnih implementacija uglavnom je posljedica fluktuacije (pretrčavanja) informatičkog kadra među jakom igračima na ovom području.

## 1.6. Tipovi i strukture baza podataka

Ovisno o tome kako je omogućen pristup podacima, načinu raspodjele, obrade i održavanja podataka, razlikujemo:

1. *Centralizirana baza podataka – terminalski pristup.*

Podrazumijeva smještaj podataka na jednom mjestu (središnjem računalu) i terminalski pristup od strane korisnika. Ovakav pristup znači da se svi zahtjevi i obrade podataka vrše na središnjem računalu, na kojem su smješteni i podaci. Korisnik preko terminala, jedino unosi svoje zahtjeve, te dobija prikaz rezultata željenih operacija. Ovakav način

organizacije baze postavlja velike zahtjeve na središnje računalo, koje osim smještaja svih podataka, vrši i sve operacije obrade podataka, njihovog formatiranja i prikaza. Stoga središnje računalo mora imati vrlo veliku procesorsku snagu i visoke performanse.

## 2. *Client- server pristup.*

Client – server struktura podrazumijeva, smještaj podataka na središnjem računalu (server), koje vrši glavninu zahtjeva za manipuliranje i obradu podataka. Korisnici koji pristupaju bazi, taj pristup ostvaruju preko svojih PC računala, koja su mrežom povezana sa serverom. Za razliku od terminalskog pristupa, kod kojeg terminalske stanice nemaju nikakve mogućnosti sudjelovanja u obradi podataka, PC računala na strani klijenta djelomično sudjeluju u obradi podataka. Korisnik preko programskog sučelja formira zahtjev za određenim podacima, koji se proslijeđuje serveru. Serversko računalo prihvata zahtjev, te ga obrađuje, a rezultate te obrade vraća klijentu. Klijentsko računalo prihvata tako obrađene podatke, te ih formira u obliku kojeg definira korisničko sučelje.

## 3. *Paralelna struktura baze podataka*

Paralelna struktura podrazumijeva formiranje baze u okružju računala međusobno povezanih u lokalnu mrežu. Između računala postoji brza veza, ostvarena preko brzih mrežnih kartica (100MB/s i više). Podaci su najčešće smješteni na samo jednom računalu, ali se pri obradi podataka može koristiti procesorska snaga svih računala u mreži. Ovakvim konceptom baze dobiva se mogućnost istovremenog korištenja resursa više računala, pa pojedina računala mogu imati i nešto slabije karakteristike.

## 4. *Distribuirana baza podataka*

Podrazumijeva strukturu baze podataka u kojoj su podaci rašireni na više računala, koja su mrežno povezana. Jednostavno rečeno, distribuirana baza podrazumijeva više lokalnih, međusobno povezanih baza. Pred samim korisnikom je ta "raspršenost" podataka skrivena, te on ima osjećaj da pristupa jednoj središnjoj bazi. U današnjim uvjetima postoji sve veće potreba za realizacijom distribuiranih baza

podataka. Razlozi za to su brojni:

- Mnogi korisnici za koje se rade baze podataka po svojoj prirodi su distribuirani na više lokacija. Uzmimo primjer mnogih multinacionalnih kompanija, koje imaju podružnice diljem svijeta. Svaka podružnica u mjestu u kojem se nalazi formira svoju lokalnu bazu podataka, a sve te baze zatim se povezuju u distribuiranu bazu podataka, koja objedinjava sve lokalne baze. Rukovodstvo kompanije ima mogućnost pristupa u sve baze i nadzora podataka koji se nalaze u bilo kojoj lokalnoj bazi.
- Distribucijom baze podataka povećava se raspoloživost i pouzdanost sustava. U slučaju ispada bilo kojeg računala u mreži, podaci na tom mjestu postaju nedostupni korisnicima, ali su podaci na svim ostalim mjestima sačuvani i dostupni.
- U distribuiranim sustavima se koristi tehnika repliciranja istih podataka na više lokacija u mreži. Zbog obrade manjih baza podataka, brzina obrade na pojedinim mjestima je veća u odnosu na brzinu koju bi imao sustav koji obrađuje veliku centraliziranu bazu.

Da bi se u potpunosti iskoristile prednosti koje pružaju distribuirani sustavi, osim zadatka koji su zajednički sa zadacima centraliziranih sustava, distribuirani sustavi moraju omogućiti:

- pristup udaljenim računalima u mreži te prijenos upita i podataka između računala
- postojanje sistemskog kataloga s podacima o distribuciji podataka u mreži održavanje konzistentnosti podataka koji se repliciraju
- izradu strategije za izvođenje pretraživanja i obrada koje dohvataju podatke iz više lokalnih baza. oporavak

sustava u slučaju ispada pojedinog računala iz mreže.

## 1.7. Životni ciklus baze podataka

Uvođenje baze podataka u neko poduzeće ili ustanovu predstavlja složeni zadatak koji zahtijeva timski rad stručnjaka raznih profila. To je projekt koji se može podijeliti u slijedeće faze:

- **planiranje razvoja.** Projektiranje i izgradnja baza podataka i kupnja odgovarajućeg softvera (sustava za upravljanje bazama podataka) nije jeftina, pa joj se ne smije nikako prići kao igrački grupice informatičara. Odluka da se pristupa izgradnji baze podataka (odnosno izgradnji IS) mora biti dobro promišljena. Mora je slijediti plan realizacije i osigurana financijska sredstva. Ako toga nema i rezultat će biti najvjerojatnije loš.
- **analiza i specifikacija zahtjeva.** Proučavaju se tokovi informacija u poduzeću. Uočavaju se podaci koje treba pohranjivati i veze medu njima. U velikom poduzećima, gdje postoje razne grupe korisnika, pojavit će se razni "pogledi" na podatke. Te poglede treba uskladiti tako da se eliminira redundancija i nekonistentnost. Na primjer, treba u raznim pogledima prepoznati sinonime i homonime, te uskladiti terminologiju. Analiza potreba također treba obuhvatiti analizu transakcija (operacija) koje će se obavljati nad bazom podataka, budući da to može isto imati utjecaja na sadržaj i konačni oblik baze. Važno je procijeniti frekvenciju i opseg pojedinih transakcija, te zahtjeve na performanse. Rezultat analize je dokument (pisan neformalno u prirodnom jeziku) koji se zove specifikacija potreba.
- **modeliranje podataka.** Različiti pogledi na podatke, otkriveni u fazi analize, sintetiziraju se u jednu cjelinu - globalnu shemu. Precizno se utvrđuju tipovi podataka. Shema se dalje dotjeruje ("normalizira") tako da zadovolji neke zahtjeve kvalitete. Također, shema se prilagođava ograničenjima koje postavlja zadani model podataka, te se dodatno modificira da bi bolje mogla udovoljiti zahtjevima na performanse. Na kraju se iz sheme izvode pogledi (pod-scheme) za pojedine aplikacije (grupe korisnika).
- **implementacija.** Na osnovu sheme i pod-shema, te uz pomoć dostupnog DBMS-a, fizički se realizira baza podataka na računalu. U DBMS-u obično postoje parametri kojima se može utjecati na fizičku organizaciju baze. Parametri se podešavaju tako da se osigura efikasan rad najvažnijih transakcija. Razvija se skup programa koji

realiziraju pojedine transakcije te pokrivaju potrebe raznih aplikacija. Baza se inicijalno puni podacima.

- **testiranje.** Korisnici pokusno rade s bazom i provjeravaju da li ona zadovoljava svim zahtjevima. Nastoje se otkriti greške koje su se mogle potkrasti u svakoj od faza razvoja: dakle u analizi potreba, modeliranju podataka, implementaciji. Greške u ranijim fazama imaju teže posljedice. Na primjer, 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 prije implementacije. Zato se u novije vrijeme, prije prave implementacije, razvijaju i približni prototipovi baze podataka, te se oni pokazuju korisnicima. Jeftinu izradu prototipova omogućuju jezici 4. generacije i objektno-orientirani jezici.
- **održavanje.** Odvija se u vrijeme kad je baza već ušla u redovnu upotrebu. Sastoјi se od sljedećeg: popravak grešaka koje nisu bile otkrivene u fazi testiranja; uvođenje promjena zbog novih zahtjeva korisnika; podešavanje parametara u DBMS u svrhu poboljšavanja performansi. Održavanje zahtijeva da se stalno prati rad s bazom, i to tako da to praćenje ne ometa korisnike. Administratoru baze podataka trebaju stajati na raspolaganju odgovarajući alati (utility programi).

Posebnu pažnju treba obratiti na prve tri faze razvoja baza podataka.

Snimanje i analiza zahtjeva posebno su složen zadatak. Informatizirati loše organiziran posao se može ali je dobit ostvarena takvim sustavom sumnjiva. U procesu prikupljanja podataka o nekom sistemu treba se koristiti svim raspoloživim izvorima: postojećim informatičkim rješenjima, znanjem osoblja na svim razinama radne i upravljačke hijerarhije, svim dokumentima koji nastaju unutar poslovnog procesa. Posebno je teško odrediti što je važno a što nije. Ako su prikupljene sve moguće informacije, i dobro analizirane, sve ostale faze u razvoju baze podataka biti će dobro utemeljene, i velika je šansa da će i konačni rezultat biti uspješan.

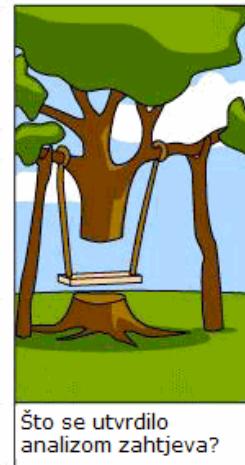
Ilustrirati ćemo problem analize jednim slikovitim (doduše karikiranim) ali izrazito vizualno uspješnim primjerom.



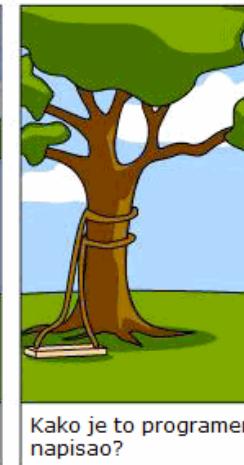
Što je korisnik rekao  
da mu treba?



Kako je to voditelj  
projekta shvatio?



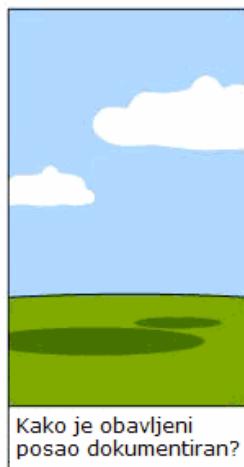
Što se utvrdilo  
analizom zahtjeva?



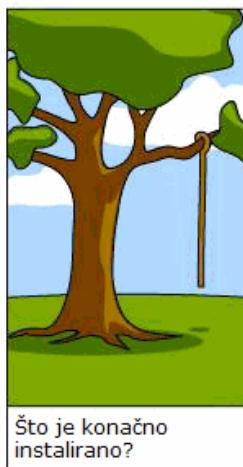
Kako je to programer  
napisao?



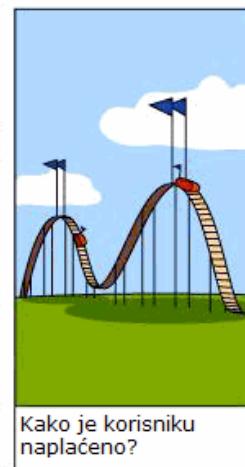
Kako je to opisao  
ugovaratelj posla?



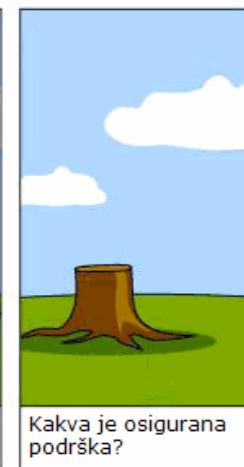
Kako je obavljeni  
posao dokumentiran?



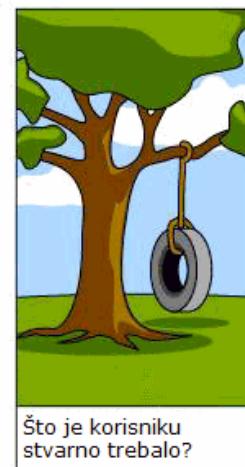
Što je konačno  
instalirano?



Kako je korisniku  
naplaćeno?



Kakva je osigurana  
podrška?



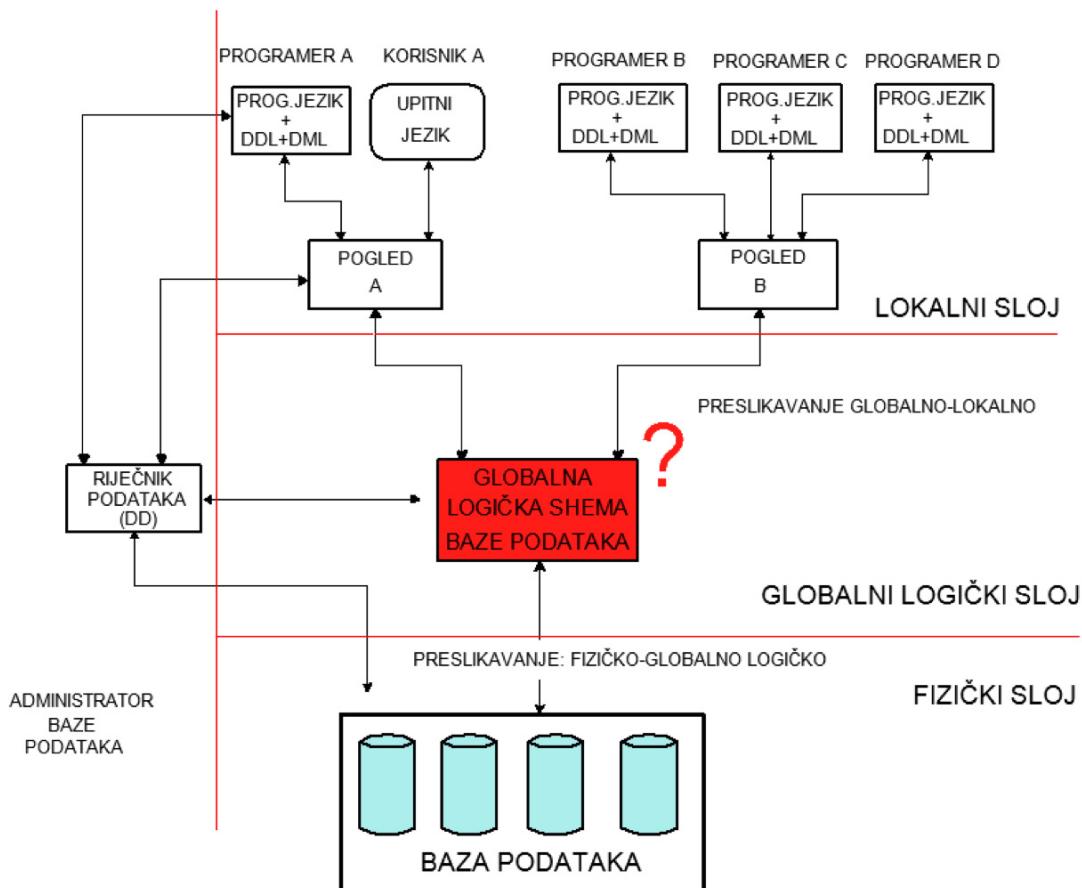
Što je korisniku  
stvarno trebalo?

## 2. MODELIRANJE PODATAKA

### 2.1. Koraci u projektiranju baze podataka

Baza podataka uvijek predstavlja *opis-sliku stvarnog procesa* iz okoline.

Pri tome se u bazu podataka pohranjuju podaci koji su međusobno povezani na različite načine i njihove vrijednosti predstavljaju dio realnog svijeta. Krećemo na zanimljivo putovanje od zamisli do realizacije baze podataka. Prva stepenica na tom putu je izrada globalne logičke sheme baze podataka.



Slika 2.1. Prvi cilj: logička shema baze podataka

Projektiranje baze podataka je proces koji se odvija u sljedećim koracima:

- **Analiza i snimanje zahtjeva.** Prvi korak u projektiranju baze podataka je da razumijete što želite postići, koje podatke treba spremiti u bazu podataka, koje će se obrade i upiti najčešće zahtijevati nad spremljenim podacima, što korisnik želi od baze podataka. Ova se faza obično svodi na prikupljanje i analizu dokumenata koji se koriste u realnom sustavu, uključuje niz opetovanih razgovora s grupama korisnika baze podataka, analize postojećih računalnih rješenja i raspoložive dokumentacije. Postoji nekoliko metodologija i niz softverskih rješenja kojima se prikupljena građa može sistematizirati i organizirano predočiti. U našim primjerima smatrati ćemo da dobro poznajemo što se od nas traži (tj. da smo obavili potrebnu analizu zahtjeva), da znamo sve o podacima sa kojim želimo raditi i njihovoj međusobnim vezama.

- **Početni (konceptualni) model baze podataka.** Prikupljene podatke pokušavamo pretočiti u početni dizajn (skicu) baze podataka. Skica baze podataka izrađuje se primjenom neke prikladne metodologije. Obično se smatra da je konceptualni model krajnja točka u dizajniranju baze podataka. Ovo nije potpuno točno. Konceptualni model je samo aproksimativni opis podataka, dobiven kroz vrlo subjektivno tumačenje informacija prikupljenih u fazi analize zahtjeva. Pažljivija analiza obično će otkriti i ispraviti niz pogrešaka u logičkoj shemi baze podataka, negdje do kraja četvrte faze u dizajniranju baze podataka.

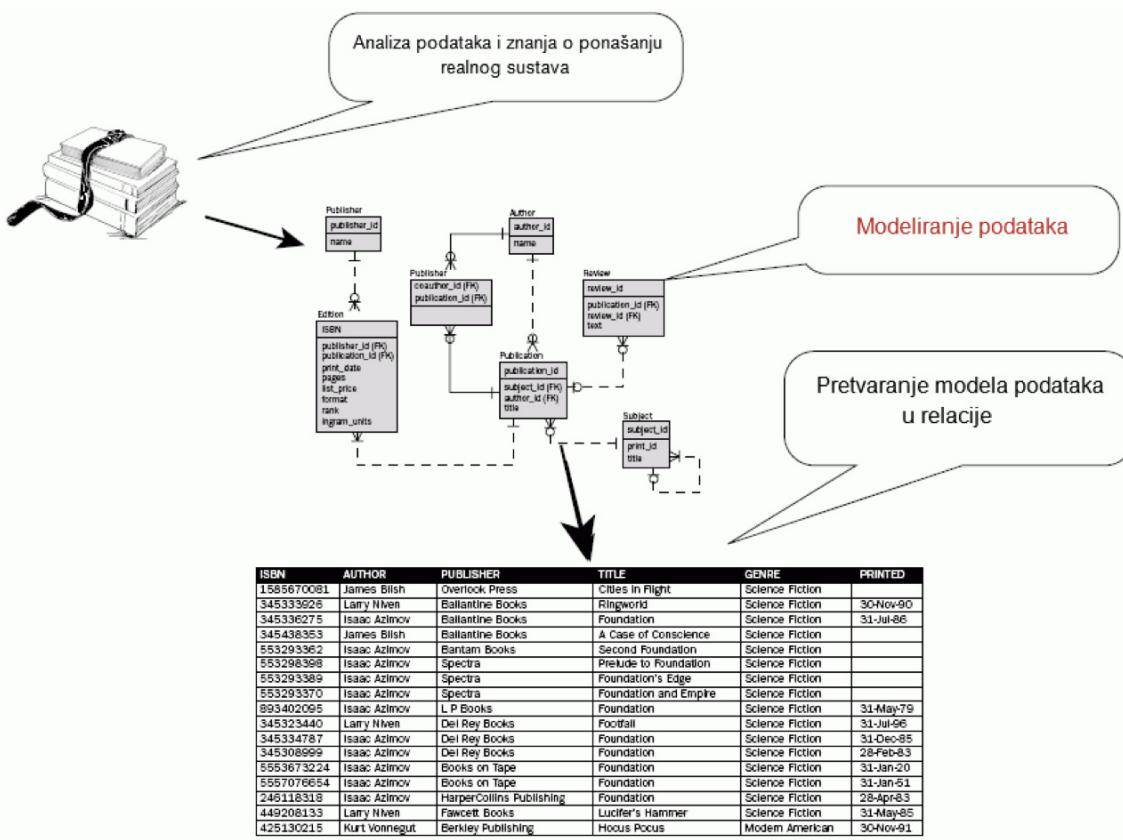
- **Prevođenje konceptualnog modela u početni logički dizajn baze podataka.** U ovoj fazi odabiremo DBMS sustav za implementaciju i konačni dizajn baze podataka.

Pogodnim transformacijama pretvaramo skicu (konceptualni model) iz prethode faze u logičku shemu baze u izabranom DBMS-u.

- **Dorada logičke sheme baze podataka.** U ovoj fazi analiziramo skupove relacija u relacijskoj shemi baze podataka (ako smo, jesmo, odabrali relacijski model DBMS-a), s ciljem da uočimo i otklonimo potencijalne probleme. Za razliku od početne analize zahtjeva koja je izrazito subjektivna, dorađivanje logičke sheme baze podataka je postupak koji se temelji na preciznim teorijskim pravilima. Postupak se naziva proces normalizacije relacijskog modela ili prevođenje relacijskog modela u normalne forme.

- **Fizički dizajn baze podataka.** Logički dizajn je završen, pa krećemo na implementaciju bazu podataka (relacije), gradimo indekse, raspoređujemo datoteke na diskovima. Da bi ostvarili poželjne performanse sustava (brzina odziva) moguće je da se u ovoj fazi redizajniraju pojedine relacije.

- **Uvođenje sigurnosnih zahtjeva i ograničenja.** U ovoj fazi, identificiramo uloge različitih korisnika, i prilagođavamo njihove poglede na bazu podataka. Također se osiguravamo da svaka pojedina grupa korisnika vidi samo ono što joj zaista treba.



Slika 2.2. Koraci u projektiranju baze podataka na primjeru baze podataka o knjigama

Prve tri faze očigledno možemo objediniti pitanjem: **kako oblikovati shemu za bazu podataka uskladenu s pravilima relacijskog modela, na temelju poznavanja ponašanja realnog sustava**. U stvarnim situacijama dosta je teško direktno pogoditi relacijsku shemu. Zato se služimo jednom pomoćnom fazom koja se zove **modeliranje podataka**. Riječ je o oblikovanju jedne manje precizne, konceptualne sheme, koja predstavlja apstrakciju onog dijela realnog svijeta koji nas zanima. Mi ćemo primjeniti metodologiju koja se naziva **metoda entiteti(objekti)-veze** (eng. *Entity-relationship model*). Shema koju nacrtamo naziva se **ER-dijagram**, i on se uspješno, više ili manje automatski, pretvara u relacijsku logičku shemu baze podataka. Zbog izostanka pravog standarda, postoji mnogo varijacija u crtanju ER-dijagrama. Mi ćemo koristiti jednu od njih. Ipak, da bi se lakše snašli u literaturi, u nekim primjerima ilustrirati ćemo i neke druge varijacije ER-dijagrama.

Modeliranje entiteta i veza zahtijeva da se svijet promatra preko tri kategorije:

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

## 2.2. Entiteti i atributi

**Entitet je objekt** u realnom svijetu koji se razlikuje od svih drugi objekata.

Npr. entitet je: Barbie lutka na odjelu igračaka, odjel igračaka, prodavač na odjelu igračaka, adresa stanovanja prodavača na odjelu igračaka.

**Entitet je ono o čemu želimo spremati podatke**, nešto što je u stanju postojati ili ne postojati, te se može identificirati. Entitet može biti objekt ili biće (na primjer kuća, student, auto), odnosno događaj ili pojava (na primjer nogometna utakmica, praznik, servisiranje auta).

Obično se trudimo prepoznati **skupove sličnih entiteta (entity set)** ili **tipove entiteta**.

Neki entiteti mogu se nalaziti istovremenu u dva skupa entiteta. Npr. skup entiteta **Zaposlenici** uključuje entitet **Marko Marković**. Isti entitet se može nalaziti i u skupu entiteta **DAVATELJI KRVI**.

**Entitet je opisan nizom atributa.** Npr. atributi entiteta **ZGRADA** su: adresa, broj katova, boja fasade, . . .

**Ukoliko neki atribut i sam zahtijeva svoje attribute, tada ga radije treba smatrati novim entitetom** (na primjer model automobila u odnosu na automobil).

**Isto pravilo vrijedi i ako atribut može istovremeno imati više vrijednosti** (na primjer kvar koji je popravljen pri servisiranju automobila).

**Svi entiteti unutar jednog skupa entiteta imaju isti skup atributa.**

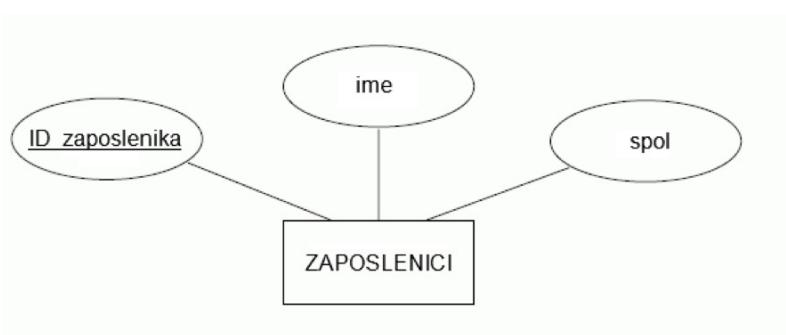
Posjedovanje **istih atributa** je **preduvjet za ustanovljavanje sličnosti entiteta i utvrđivanje pripadnosti istom skupu entiteta**. Izborom atributa izražava se ono što nas u realnom svijetu zanima. Npr. skup entiteta **Zaposlenici** može imati atribute: **ID\_zaposlenika**, **ime**, **matični broj** i **spol**, ali nema atributa: **adresa stanovanja** ili **mjesto za parkiranje** jer nas to u konkretnom slučaju ne zanima.

Za svaki atribut kojim se opisuje entitet, moramo utvrditi **skup dozvoljenih vrijednosti** (odnosno **domenu atributa**). Npr. za skup entiteta ili tip entiteta **Zaposlenik** atribut **ime** je niz od maksimalno 20 znakova. Atribut **ID\_zaposlenika** može sadržavati samo cijele brojeve od 1 do 9999.

Nadalje, za **svaki skup entiteta moramo izabrati ključ**. **Ključ je minimalni set atributa koji jednoznačno određuje jedan entitet u skupu entiteta**.

**Kandidata za ključ može biti više.** U jednom skupu entiteta ne mogu postojati dva različita primjerka entiteta s istim vrijednostima kandidata za ključ. (Na primjer za tip entiteta **AUTOMOBIL**, 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**. (Npr. primarni ključ za tip entiteta **STUDENT** mogao bi biti atribut **BROJ\_INDEKSA**). **Prepostavljamo da svaki skup entiteta ima barem jedan primarni ključ.**

Skup entiteta istog tipa predstavlja se pravokutnikom. Atributi se označavaju ovalima. Imena atributa koji čine primarni ključ se podcrtavaju. Podaci o domeni (skupu vrijednosti) pojedinih atributa ispisuju se uz ime atributa. (Domene vrijednosti često se zbog jasnoće prikaza izostavljaju iz ER-dijagrama.)



Slika 2.3. Skup entiteta Zaposlenici

## 2.3. Veze i skupovi veza

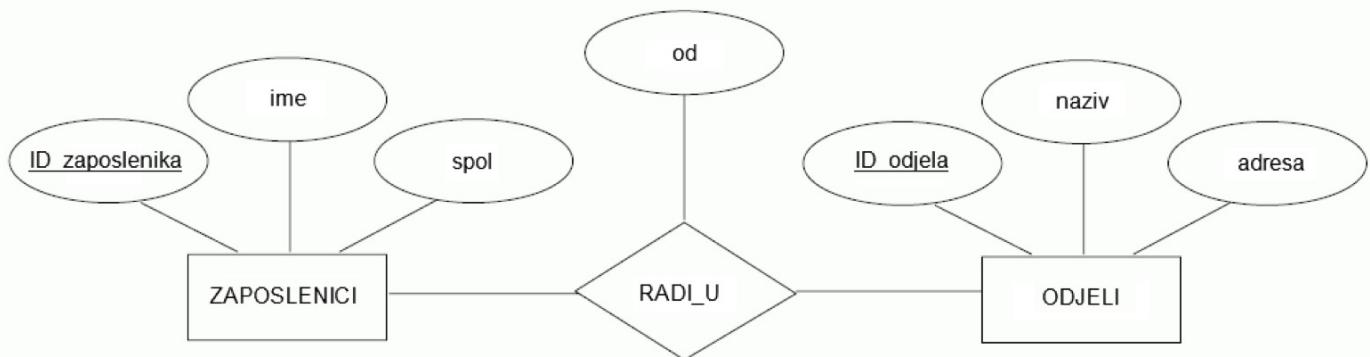
**Veza predstavlja odnos između dva ili više entiteta.** Npr. entiteti **ZAPOSLENIK** i **ODJEL** vezani su na način da **zaposlenik Jure Jurić radi u odjelu prodaje igračaka**.

Slično kao i kod entiteta sve slične veze između sličnih entiteta okupljamo u **skup veza** ili u **tip veze**. Skup veza ili tip veza sa sličnim svojstvima predstavlja se rombovima.

Skup veza možemo predočiti kao skup n-troki:

$$\{(e_1, \dots, e_n) \mid e_1 \in E_1, \dots, e_n \in E_n\}$$

Svaka n-torka označava vezu između entiteta  $e_1$  do  $e_n$ , gdje je  $e_i$  entitet iz skupa entiteta  $E_i$ . Na početku ograničiti ćemo se na veze između točno dva tipa entiteta.

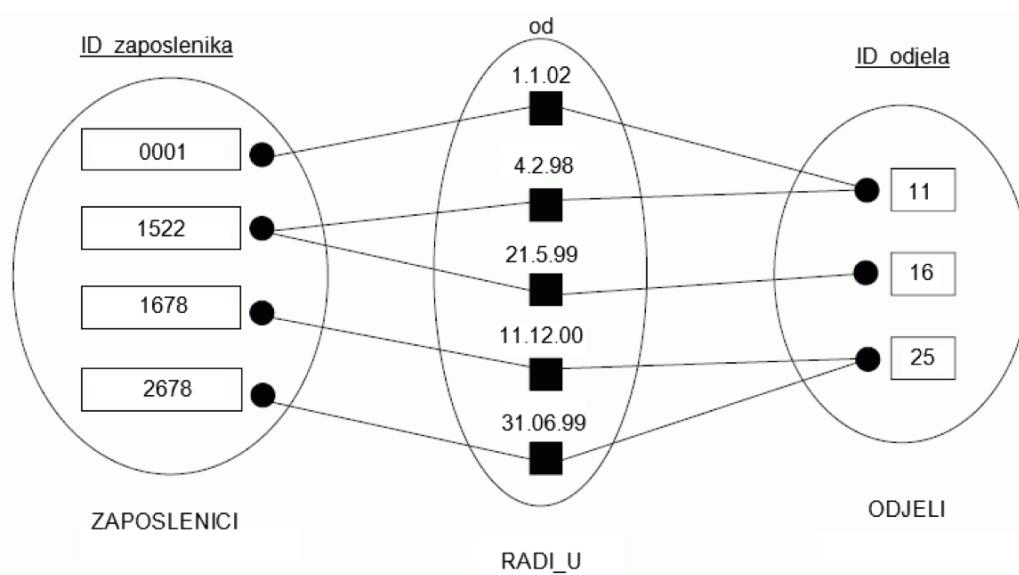


Slika 2.4. Skup veza **RADI\_U**

**Skup veza može također sadržavati opisne atributе.** Opisni atributi opisuju odnos između entiteta a ne same entitete. Npr. **zaposlenik Jure Jurić radi u odjelu prodaje igračaka od siječnja 2002.**

**Svaka veza mora biti jednoznačno određena entitetima koje povezuje, a ne atributima koji je opisuju.** Drugim riječima svaka veza opisuje se isključivo preko ključeva entiteta koje povezuje.

Veze **RADI\_U** skupova entiteta **ZAPOSLENICI** i **ODJELI** prikazane su na slici 2.5.

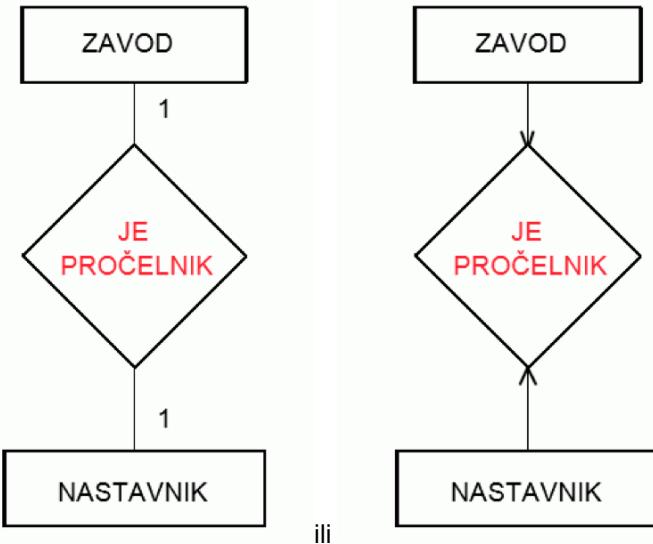


## 2.4. Funkcionalnost veza

Funkcionalnost veze može biti:

- **Jedan-naprama-jedan (1 : 1).** Jedan primjerak prvog tipa entiteta može biti u vezi s najviše jednim primjerkom drugog tipa entiteta. Također jedan primjerak drugog tipa može biti u vezi s najviše jednim primjerkom prvog tipa.

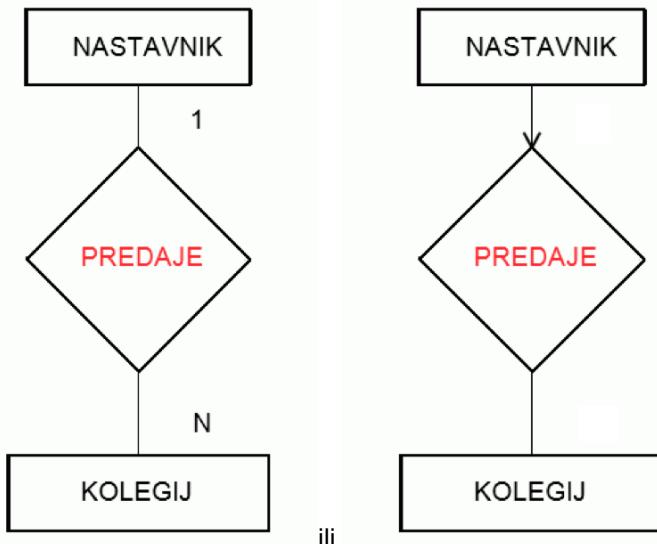
Npr. veza **JE PROČELNIK** između skupova entiteta **NASTAVNIK** i **ZAVOD** (na fakultetu).



Slika 2.6. Veza **JE PROČELNIK** između tipova entiteta **NASTAVNIK** i **ZAVOD** (na fakultetu).

- **Jedan-naprama-mnogo (1 : N).** Jedan primjerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primjera drugog tipa entiteta, no jedan primjerak drugog tipa može biti u vezi s najviše jednim primjerkom prvog tipa.

Npr. veza **PREDAJE** između tipova entiteta **NASTAVNIK** i **KOLEGIJ**.



Slika 2.7. Veza **PREDAJE** između skupova entiteta **NASTAVNIK** i **KOLEGIJ**.

- **Mnogo-naprama-mnogo (M : N).** Jedan primjerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primjera drugog tipa entiteta,

te također jedan primjerak drugog tipa može biti u vezi s 0, 1 ili više primjeraka prvog tipa.

Npr. veza **UPISAO** između tipova entiteta **STUDENT** i **KOLEGIJ**.



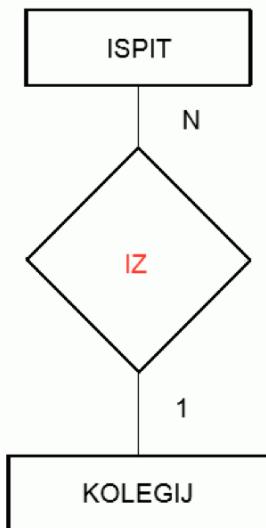
Slika 2.7. Veza **UPISAO** između tipova entiteta **STUDENT** i **KOLEGIJ**.

Funkcionalnost veze najčešće se upisuje kao broj između veze i entiteta na koje se odnosi.

## 2.5. Članstvo entiteta u vezama

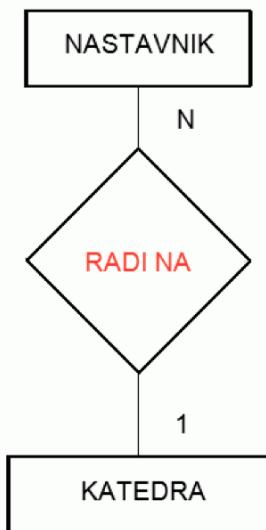
Ako svaki primjerak entiteta nekog tipa mora sudjelovati u zadanoj vezi, tada kažemo da taj **tip entiteta ima obavezno članstvo** u toj vezi.

Npr. između tipova entiteta **ISPIT** i **KOLEGIJ** zadana je veza **IZ**, koja ima funkcionalnost ( $N : 1$ ). **ISPIT** ima obavezno članstvo u vezi **IZ**, jer svaki ispit mora biti iz nekog kolegija.



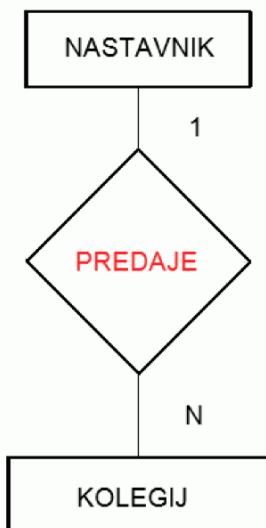
Slika 2.9. Skup entiteta **ISPIT** ima obavezno članstvo u vezi **IZ**

Članstvo entiteta u vezi može biti i neobavezno. Npr. između tipova entiteta **NASTAVNIK** i **KATEDRA** zadana je veza **RADI NA**, koja ima funkcionalnost ( $N : 1$ ). **NASTAVNIK** ima **neobavezno članstvo u vezi RADI NA** jer svaki nastavnik ne mora raditi na nekoj katedri (barem je takvo stanje na FESB-u).



Slika 2.10. Skup entiteta **NASTAVNIK** ima neobavezno članstvo u vezi **RADI NA**

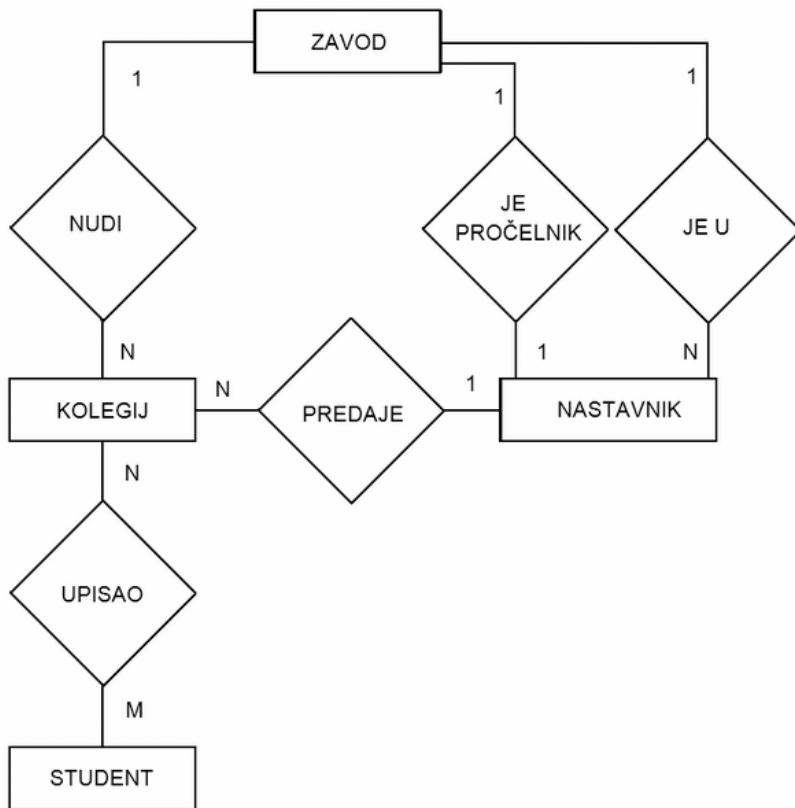
Obavezno članstvo entiteta u nekoj vezi se mora istaknuti. Odluka da li je članstvo obavezno ili neobavezno koji put je stvar dogovora odnosno projektantove odluke (npr. članstvo za entitet **KOLEGIJ** u vezi **PREDAJE**).



Slika 2.11. Postoje li **KOLEGIJI** koji ne predaje ni jedan **NASTAVNIK**

## 2.6. Prikaz ER-modela pomoću dijagrama

Kao primjer jednostavnog cjelovitog ER-dijagrama, pogledajmo dijagram na Slici 2.12 koji prikazuje bazu podataka o fakultetu.



Slika 2.12. ER-dijagram baze podataka o fakultetu

Tipovi entiteta su:

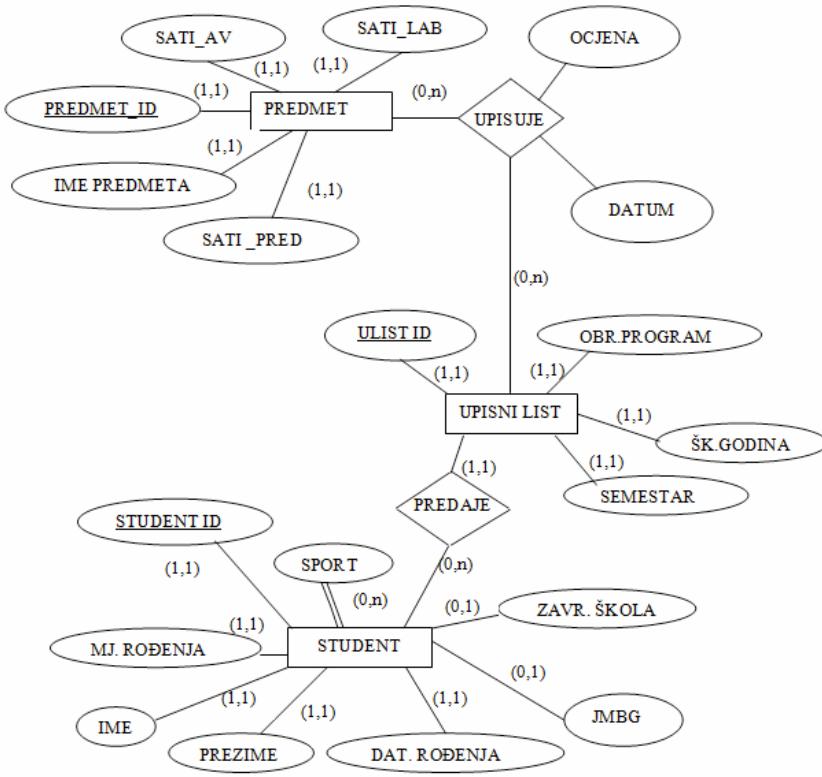
1. **ZAVOD**, s atributima IME ZAVODA, ADRESA, . . .
2. **KOLEGIJ**, s atributima BR KOLEGIJA, NASLOV, SEMESTAR, . . .
3. **STUDENT**, s atributima BR INDEKSA, IME STUDENTA, ADRESA, SPOL, . . .
4. **NASTAVNIK**, s atributima IME NASTAVNIKA (prepostavljamo da je jedinstveno), BR SOBE, . . .

Podvučeni atributi čine primarni ključ.

Veze su:

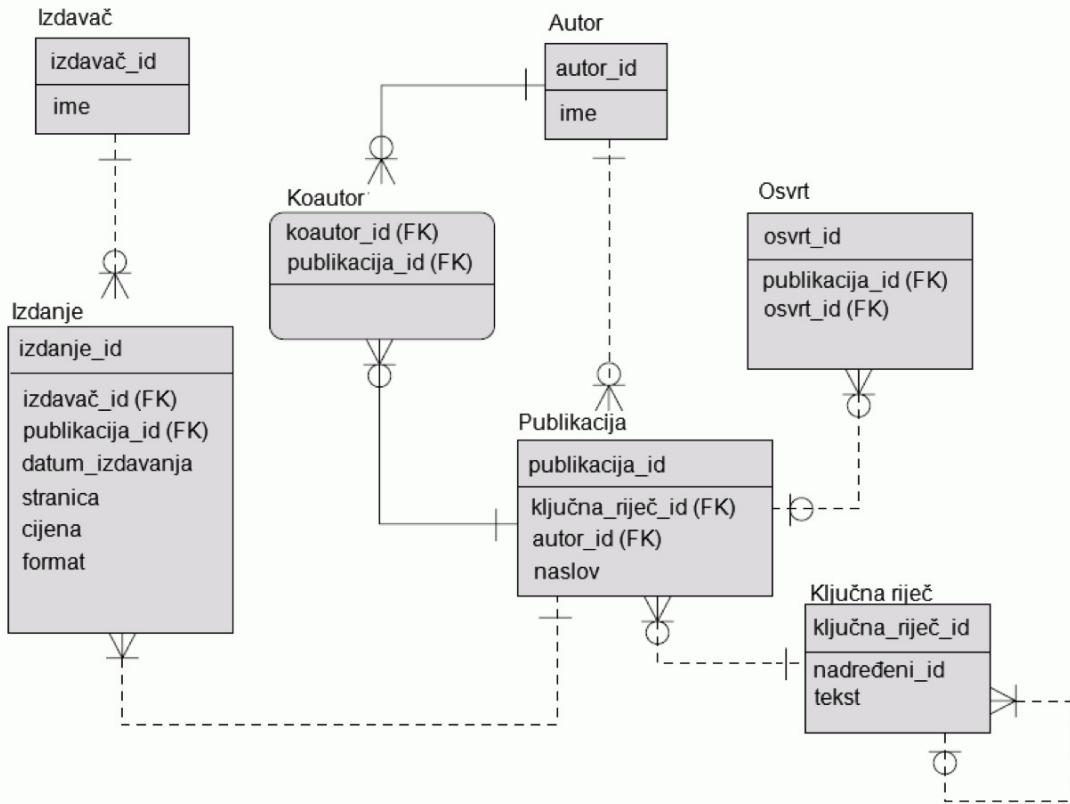
1. **JE PROČELNIK**, bez atributa. **ZAVOD** ima obavezno članstvo.
2. **JE U**, bez atributa. **NASTAVNIK** ima obavezno članstvo.
3. **NUDI**, bez atributa. **KOLEGIJ** ima obavezno članstvo.
4. **UPISAO**, s atributom DATUM UPISA.
5. **PREDAJE**, bez atributa. **KOLEGIJ** ima na primjer obavezno članstvo.

Drugi primjer ER-dijagrama je također model za bazu podataka o fakultetu (malo drugačiji pogled na isti realni sustav). Na slici su ucrtani i atributi. Ovakav dijagram manje pregledan od onog prethodnog.



Slika 2.13. ER-dijagram baze podataka o fakultetu (drugi pogled na dešavanja na fakultetu)

Postoji i treći način za prikazivanje ER-dijagrama, npr. ER-model za spremanje podataka o knjigama.



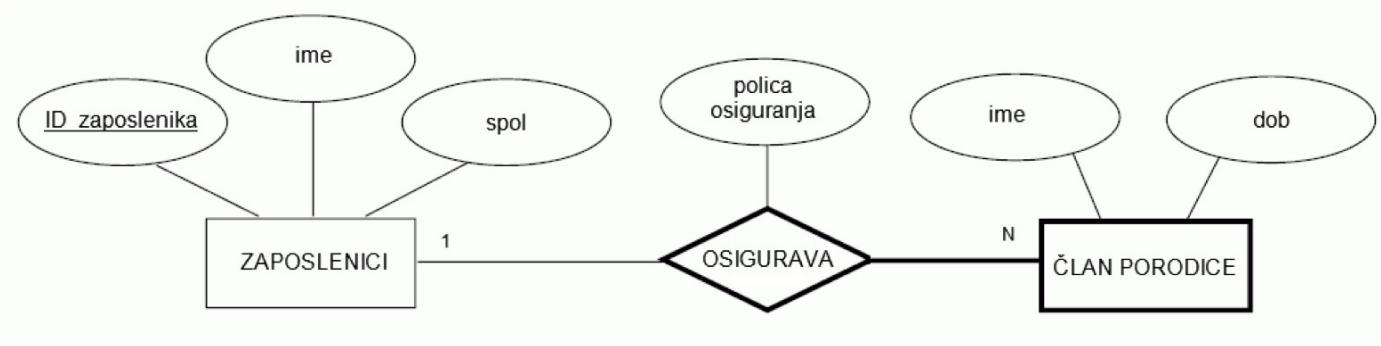
Slika 2.14. ER-dijagram baze podataka o knjigama (treća varijacija)

## 2.7. Složeni oblici u ER-modelima

### 2.7.1. Slabi entiteti

**Slabi entiteti** su entiteti koji nemaju pravog ključa (do sada smo podrazumijevali da unutar skupa entiteta mora postojati barem jedan ključ).

Npr. poduzeće osigurava svoje zaposlenike. Svaki zaposlenik ima svoju policu osiguranja koja pokriva i osiguranje članova njegove obitelji. Ako zaposlenik napusti poduzeće, prestaje se uplaćivati njegova polica osiguranja. Što više polica osiguranja se briše iz baze podataka kao i svi članovi porodice zaposlenika koji su preko nje bili osigurani. Za članove porodice znamo samo ime člana i dob. Atribut ime ne određuje jednoznačno entitet ČLAN PORODICE. Članovi porodice su skup slabih entiteta. Jednoznačnost slabog entiteta postiže se kombiniranjem nekog od njegovih atributa sa ključem nekog drugog entiteta.



Slika 2.15. Primjer slabog entiteta

Skup entiteta iz kojeg uzimamo primarni ključ naziva se **identificirajući vlasnik slabog entiteta (identifying owner)**

Moraju vrijediti sljedeća ograničenja:

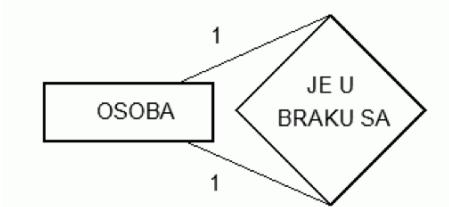
- Identificirajući vlasnik slabog entiteta vezuje se sa slabim entitetom vezom tipa (1:N). Ovakav skup veza naziva se **identificirajući skup veza (identifying relationship set) slabog entiteta**.
- Svi elementi slabog entiteta **imaju obavezno članstvo** u identificirajućem skupu veza.

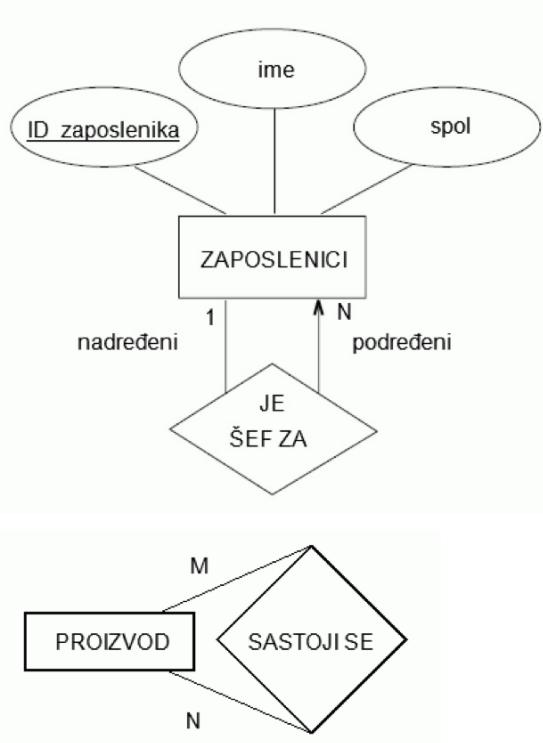
## 2.7.2. Složene veze u ER-modelima

U stvarnim situacijama pojavljuju se i složenije veze od onih koje smo do sada promatrali. Navest ćemo neke od njih.

### 2.7.2.1 . Involuirana veza

**Involuirana veza** povezuje jedan tip entiteta s tim istim tipom. Dakle riječ je o binarnom odnosu između raznih primjeraka entiteta istog tipa. Funkcionalnost takve veze opet može biti (1 : 1), (1: N), odnosno (M : N).





Slika 2.16. Primjeri za involuirane veze

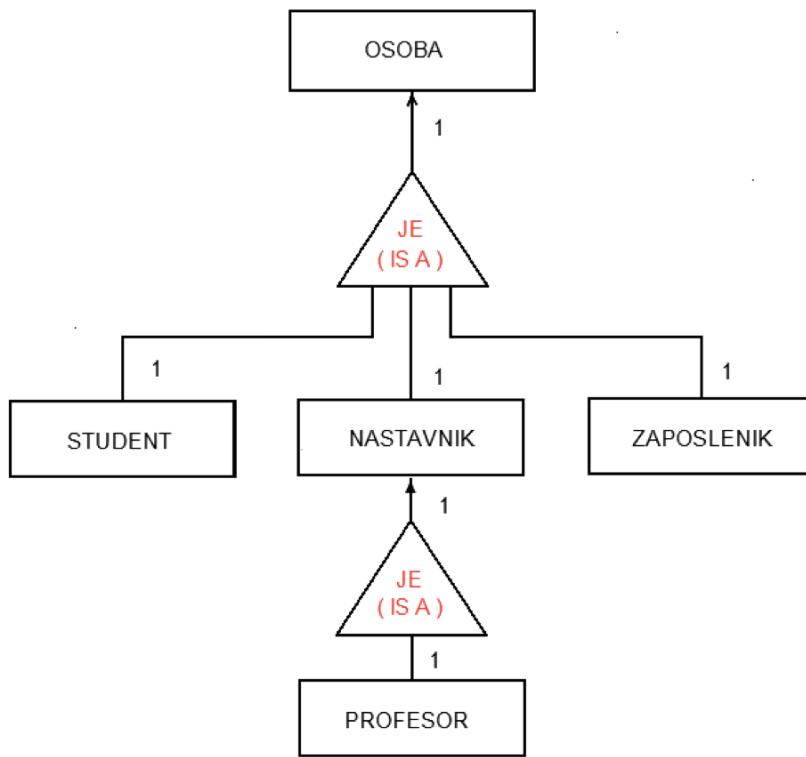
Prvi dijagram na Slici 2.16 napravljen je pod pretpostavkom da su prošli brakovi osobe zaboravljeni, a poligamija zabranjena. Članstvo u vezi **JE U BRAKU SA** je neobavezno.

Drugi dijagram na Slici 2.16 ima ucrtanu strelicu koja pokazuje smjer tumačenja veze **JE ŠEF ZA**. Možemo uzeti da je članstvo u toj vezi neobavezno, ako u realnom svijetu postoji barem jedan suradnik koji nema šefa. Ovakva involvirana veza uvodi *načelo subordinacije*, jedan entitet je **nadređen** a drugi entitet je **podređen**.

Treći dijagram na Slici 2.16 odnosi se na proizvode se proizvode u nekoj tvornici. Pritom se jedan složeniji **PROIZVOD** sastoji od više jednostavnijih. Isti jednostavniji proizvod pojavljuje se u više složenih.

### 2.7.2.2. Pod-tipovi

Tip entiteta  $E_1$  je **podtip** tipa entiteta  $E_2$  ako je svaki primjerak od  $E_1$  također i primjerak od  $E_2$ .  $E_1$  nasljeđuje sve atribute od  $E_2$ , no  $E_1$  može imati i dodatne atribute. Situaciju opisujemo pomoću specijalne (1 : 1) veze **JE** (engleski **IS A**) koja se može pojaviti više puta unutar ER-sheme. Tip veze **IS A** crta se u ER-dijagramu kao trokut okrenut prema nadređenom tipu. Slika 2.17 sadrži primjer ER-sheme s pod-tipovima i nad-tipovima. Riječ je o tipovima entiteta za osobe koje se pojavljuju na fakultetu. **NASTAVNIK** uključuje profesore, docente i asistente.



Slika 2.17. Primjer ER-sheme s pod-tipovima entiteta

Uz pojam podtipova vezana su dva tipa ograničenja:

- **ograničenje preklapanja (Overlap constraints)** razlučuje da li se dva pod-tipa mogu sadržavati isti entitet. Za podtipove student i nastavnik intuitivni odgovor je ne. Za podtipove student i pomoćno osoblje odgovor je da.
- **ograničenje pokrivanja (Covering constraints)** razlučuje da li su svi entiteti iz podtipa nalaze u nad-tipu. Za nad-tip osoba i pod-tip nastavnik intuitivni odgovor je ne. Za nad-tip nastavnik i pod-tip profesor intuitivni odgovor je da.

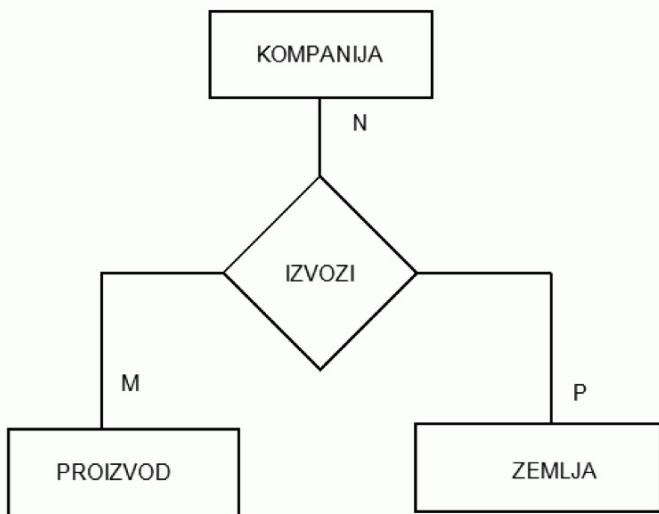
Kada se koriste podtipovi?

Dva su osnovna razloga zašto koristimo podtipove i nadtipove (specijalizaciju ili generalizaciju) :

- Nekim entitetima želimo dodati posebne atribute (posebne atribute dodajemo u podtipovima)
- Želimo izdvojiti skup entiteta koji se učestvuje u nekoj vezi. Skup entiteta Motorna vozila učestvuje u vezi registriran na sa skupom entiteta osobe.

### 2.7.2.3. Ternarne veze

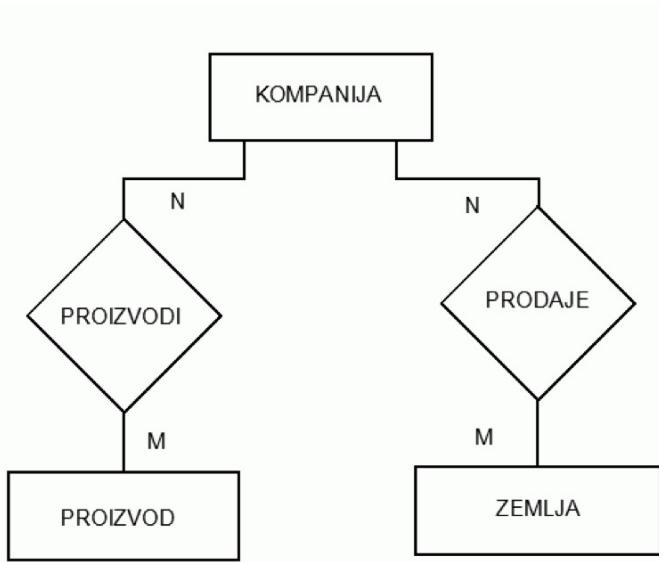
**Ternarne veze uspostavljaju se između tri tipa entiteta.** Znači riječ je o ternarnom odnosu između primjera triju tipova entiteta. Postoje brojne mogućnosti za funkcionalnost ternarne veze, na primjer ( $N : M : P$ ), ( $1 : N : M$ ), ( $1 : 1 : N$ ) ili čak ( $1 : 1 : 1$ ).



Slika 2.18a. Primjer ternarne veze

Primjer ternarne veze sa Slike 2.18a odnosi se na podatke o kompanijama, proizvodima koje one proizvode i zemljama u koje one izvoze svoje proizvode. Funkcionalnost ove veze je mnogo-naprama-mnogo-naprama-mnogo, dakle ( $N : M : P$ ), jer na primjer 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 dvije binarne.** Uzmimo da u primjeru sa Slike 2.18a vrijedi pravilo: ako kompanija izvozi u neku zemlju, tada ona izvozi **sve** svoje proizvode u tu zemlju. Uz ovo pravilo, razmatrana ternarna veza može se zamjeniti s dvije binarne, u skladu s dijagramom na Slici 2.18b.



Slika 2.18b. Primjer rastavljanja ternarne veze na dvije binarne veze

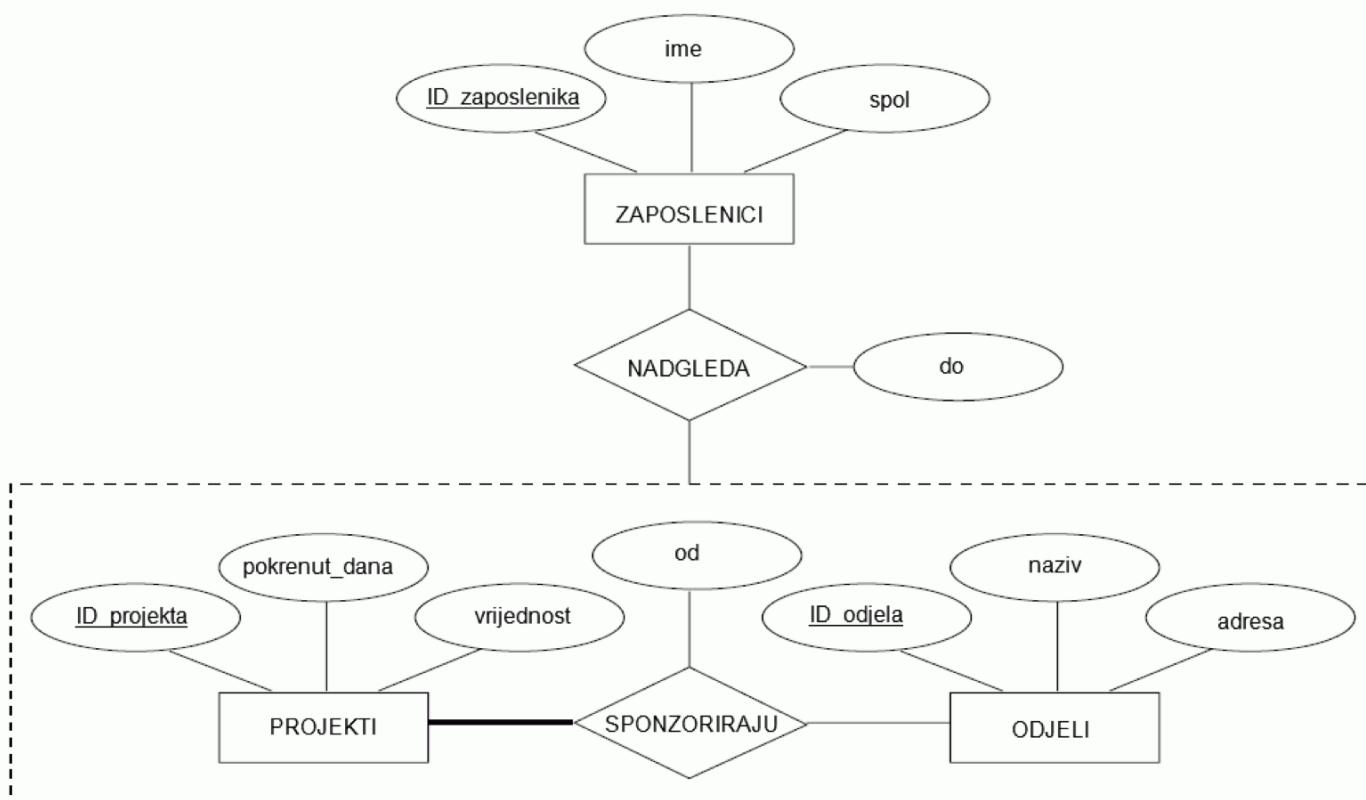
#### 2.7.2.4. Agregacija

Do sada smo kazali da veze predstavljaju odnose između dvaju skupova entiteta (s izuzetkom involviranih veza i ternarnih veza). Katkada trebamo uspostaviti vezu između skupa entiteta i skupa veza.

Npr. neka kompanija sponzorira različite projekte. Svaki pojedini projekt sponzorira jedan ili više odjela te kompanije što je opisano skupom veza

**SPONZORIRA**. Odjel koji sponzorira neki projekt imenuje nekog od zaposlenika da nadzire rad na projektu. Veza nadzire je odnos između entiteta zaposlenik i veze sponzirira.

Da bi ovo mogli ostvariti uvodimo novo svojstvo ER-modela, tzv **agregaciju (Aggregation)**. Agregacija nam omogućuje da naznačimo da je skup veza (označen crtkano) učestvuje u novoj vezi.



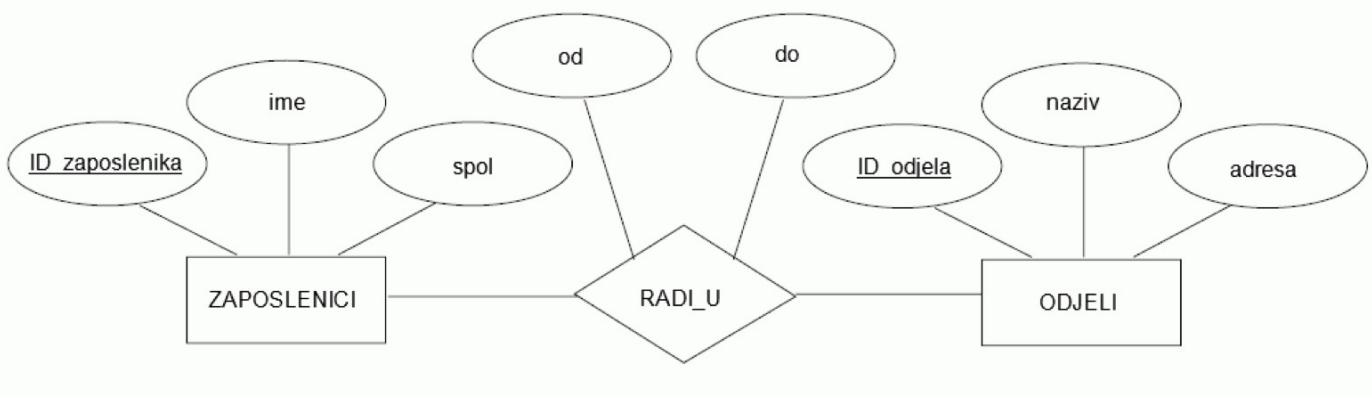
Slika 2.19. Agregacija.

Kada se koristi agregacija? Uvijek kada imamo potrebu uspostaviti vezu sa nekom drugom postojećom vezom. (Često se agregacija može nadomjestiti ternernom vezom.)

## 2.8. Konceptualni dizajn baze podataka primjenom ER-modela

Modeliranje podataka primjenom metode ER obično nas dovodi do niza pitanja:

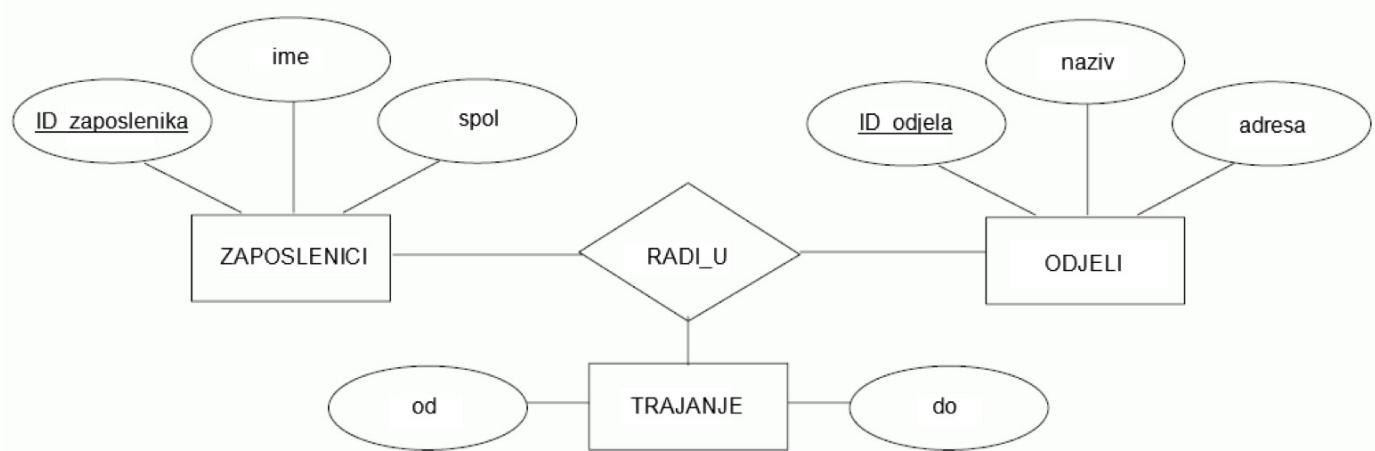
- da li nešto modelirati kao entitet ili atribut
- da li nešto modelirati kao entitet ili vezu
- koje su veze obavezne a koje nisu
- da li primijeniti binarne ili ternarne veze
- da li koristiti agregacije



Slika 2.20. Modele u kojem se pamti povijest radnog odnosa

Odgovor na ova pitanja nije jednostavan. Obično ovisi o konkretnom primjeru koji se rješava. Npr. pogledajmo nešto modificirani primjer skupa veza RADI\_U. Modifikacija se sastoji u tome da je dodan atribut do. Pamti se interval u kojem neki zaposlenik radi u odjelu. Ovisno o tome želimo li pamtitи sve periode u kojima je zaposlenik radio u nekom odjelu ili nas zanima samo zadnji period mijenjati će se i ER-model.

U slučaju da želimo pamtitи sve periode kada je zaposlenik radio u odjelu ER-model izgledati će kao na slici 2.21.



Slika 2.21. Modele u kojem se pamti povijest radnog odnosa

## 2.9. Kako najjednostavnije do modela podataka?

Modeliranje podataka nije jednostavan zadatak. Obično se podaci modeliraju kroz više iteracija. Ipak najjednostavniji pristup je za početak složiti vrlo jednostavnu priču kroz koju ćemo pokušati opisati realni sustav koji želimo modelirati. Npr. natjecanje u Formuli 1.

Priča glasi: svake godine održava se natjecanje (prvenstvo) u Formuli 1.

Natjecanje se sastoji od više utrka. Utrke se voze na trkačkim stazama.

Na utrci se nastupaju vozači u bolidima. Bolidi konstruiraju timovi.

Ponoviti ćemo priču ali tako da u crveno obojimo imenice a u plavo glagole: svake godine održava se **natjecanje (prvenstvo)** u Formuli 1.

**Natjecanje se sastoji od više utrka. Utrke se voze na trkačkim stazama.**

**Na utrci nastupaju vozači u bolidima. Bolidi konstruiraju timovi.**

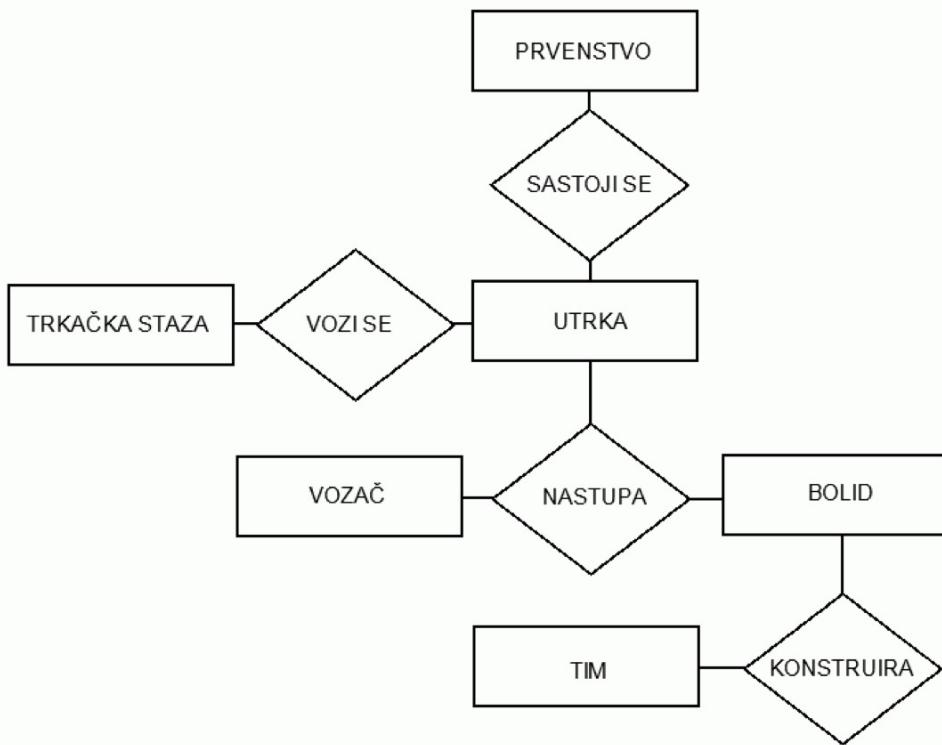
Očito ćemo u našem modelu imati entitete: godišnje natjecanje, utrka, trkačka staza, vozač, bolid i tim. (Slika 2.22)



Slika 2.22 Kandidati za tipove entiteta

Između tipova entiteta potrebno je ucrtati odgovarajuće veze (Slika 2.23).

Kada je i to gotovo treba odrediti atribute entiteta i veza, funkcionalnost veza, obaveznost učestvovanja pojedinih entiteta, uočiti eventualne podtipove i sl.



Slika 2.23 Grubi model podataka za natjecanje u Formuli 1

Ovaj postupak ne osigurava valjanost modela podataka. Ovo je samo način da stvorite početnu sliku o realnom sistemu koji modelirate na temelju koje će vam se otvoriti mnogo pitanja na koja tek treba odgovoriti.

Slijedi niz razgovora, traženja odgovora na pitanja koja vas muče. Nakon toga vraćate se na model popravljate ga. Otvaraju se nova pitanja, novi razgovori, ponovno povratak na model podataka. Postupak traje dok se ne raščiste sva pitanja koja vas muče i dok ne zaključite da model podataka dobro opisuje što se događa u realnom sustavu (ili barem u onom njegovom dijelu koji nas zanima).

## 2.10. Zaključak

ER model dovoljno je jednostavan da ga ljudi različitih struka mogu razumjeti. Zato ER-dijagram služi za komunikaciju projektanta baze podataka i korisnika, i to u najranijoj fazi razvoja baze. Postojeći DBMS ne mogu direktno implementirati ER-dijagram, već zahtijevaju da se ona detaljnije razradi, te modificira u skladu s pravilima izabranog modela baze podataka (*relacijskog*, mrežnog, odnosno hijerarhijskog modela).

## 4. POSTUPAK NORMALIZACIJE I NORMALNE FORME

Relacijska shema, dobivena iz ER-dijagrama na osnovu prethodnih uputa, može sadržavati nedorečenosti koje treba otkloniti prije implementacije. *Proces daljnog dotjerivanja relacijske sheme zove se normalizacija..* Uobičajeno se normalizacija primjenjuje kod dizajniranja logičke strukture relacijskog modela. Stoga ćemo i normalne forme definirati koristeći terminologiju relacijskog modela. Međutim, značaj postupka normalizacije je općenitiji i može se primijeniti i na druge "klasične" modele baze podataka (mrežni i hijerarhijski), kao i za dizajniranje strukture zapisa u obradi podataka zasnovanoj na skupu datoteka. Relacijska terminologija se u tom slučaju jednostavno prevodi u terminologiju tih drugih modela.

Najprije o terminu normalizacija. Otac relacijskog modela E.F.Codd u članku "A Fireside Chat" u časopisu DBMS iz prosinca 1993 napisao je:

"I called it normalization because then President Nixon was talking a lot about normalizing relations with China. I figured that if he could normalize relations, so could I".

Teorija normalizacije nije ništa drugo nego formalizacija nekih intuitivno prihvatljivih principa o "dobrom" oblikovanju sheme (ili popravljanju loše dizajnirane sheme) baze podataka. Ukoliko već na početku dobro uočimo sve potrebne entitete, atribute i veze, tada nam nikakva daljnja normalizacija neće biti potrebna. No ako je polazna relacijska shema bila loše oblikovana, tada će postupak normalizacije ispraviti te greške. Vrijedi napomena da nijedan postupak normalizacije neće u model uvrstiti podatke koje smo u fazi modeliranja podataka propustili.

Proces normalizacije odvija se u etapama. Etape u procesu normalizacije nazivaju se općenito *normalne forme*. U svojim radovima (1970-1974. godina) E.F. Codd je najprije definirao 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), a 1981. i normalnu formu domena i ključeva (DK/NF). U praksi je lako naići na relacije koje odstupaju od 2NF, 3NF, BCNF, no vrlo rijetko se susreću relacije u BCNF koje nisu u 4NF i 5NF. Zato su "više" normalne forme prvenstveno od teorijskog značaja.

*Najopćenitije kazano, dobra je ona struktura baze podataka u kojoj je logička redundancija podataka minimalna.*

Problem redundancije (ponavljanja) podataka najviše se očituje kroz operacije održavanja podataka u bazi podataka. Osnovne operacije održavanja podataka u relacijskoj bazi podataka su:

- dodavanje nove n-torke u relaciju,
- izbacivanje neke n-torke iz relacije i
- izmjena vrijednosti nekog atributa u relaciji (ažuriranje)

Problemi pri izvođenju ovih operacija (potreba da se pri izmjeni jednog atributa, dodavanju ili izbacivanju jedne n-torke sama operacija mora ponavljati više puta, ili čak da se neko logičko dodavanje ne može izvršiti, odnosno da izbacivanje jednog logičkog skupa podataka dovodi do neželjenog izbacivanja drugih podataka) nazivaju se *anomalije u održavanju baze podataka*. Obično se posebno definiraju i analiziraju:

- anomalije u dodavanju n-torke,
- anomalije u izbacivanju n-torke i
- anomalije u ažuriranju.

Loše dizajnirana logička struktura baze podataka ne dovodi samo do anomalija u njenom održavanju nego i u čitanju podataka (u svrhu izvještavanju) iz takve baze podataka. Kvaliteta logičke strukture baze podataka u smislu izvještavanja može se procijeniti primjenom slijedećeg kriterija:

Koliko su teški(laki) zahtjev za izvještajima toliko će ih teško(lako) biti realizirati (drugim riječima ako je realizacija izuzetno složena, vrlo je vjerojatno da logička struktura baze podataka nije najbolje pogodena).

Probleme održavanja i izvještavanja ilustrirati ćemo jednim primjerom. Npr. nismo najbolje modelirali podatke. Dobiveni model pretvorili smo u relacije. Rezultat je nenormalizirana relacija **STUDENT** (prethodno smo rekli da u nenormaliziranoj relaciji postoje atributi čije vrijednosti nisu atomske, odnosno koji se mogu tumačiti kao "grupa sa ponavljanjem".)

## STUDENT

BR_INDEX	IME	SEMESTAR	STUDIJ	VOD_STUDIJA	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	OCJENA
21	Mate	4	Računarstvo	Frane	123	Baze podataka	2
					124	Matematika	4
					267	Fizika	5
77	Ana	7	Elektronika	Jure	678	Polja i valovi	4

					589	Teorija sustava	1
					245	Modeliranje	4
					567	Elektronički skloovi	2
36	Pero	6	Elektronika	Jure	678	Polja i valovi	2
					245	Modeliranje	3
...							

Uočite probleme u održavanju podataka i izvještavanju za ovako strukturiranu relaciju:

- **Anomalije u dodavanju.** Ako je u novom nastavnom planu definiran novi kolegij, ne mogu se ubaciti podaci o tom kolegiju dok ga neki student ne položi. Ili, ako se otvorи neki novi studij, ne mogu se ubaciti podaci o tom studiju dok ga neki student ne upiše.
- **Anomalije u izbacivanju.** Ako je jedan kolegij (**FIZIKA**) položio samo jedan student (**MATE**) i ako se on ispiše sa fakulteta, odnosno ako se izbaci odgovarajuća n-torka, gube se i sve informacije o tom kolegiju. Ako je taj student bio i jedini student na nekom studiju, gube se i sve informacije o tom studiju.
- **Anomalije u ažuriranju.** Ako se promijeni naziv nekog kolegija ili voditelj nekog studija, to se mora učiniti na onoliko mjesta koliko je studenata položilo taj kolegij, odnosno koliko je studenata upisano na dati studij.
- **Problemi u izveštavanju:** Ovakva struktura relacije je veoma pogodna za izvještaj po zahtjevu: "Prikaži listu studenata, svih ispita koje je svaki student položio i njegovu prosječnu ocjenu" (Npr. prijepis indeksa, uvjerenje o položenim ispitima). Odgovarajući program bi bio veoma jednostavan i sveo bi se na listanje date relacije (datoteke). Suprotan zahtjev: "Prikaži listu kolegija, imena svih studenata koji su ga položili i prosječnu ocjenu na kolegiju", za danu strukturu relacije zahtijevao bi znatno složeniji program ili bi se samu relaciju trebalo prestrukturirati u dvije relacije sa istim skupom podataka za dva različita zahtjeva. Time bi se samo povećala, redundancija (ponavljanje) podataka i umnožili problemi održavanja baze podataka.

Postupkom normalizacije logička struktura baze podataka se dovodi u takav oblik (ili, drugim riječima, relacije se dovode u normalne forme) u kojem se izbjegavaju anomalije u održavanju i problemi u izvještavanju.

## 4.1. Prva normalna forma (1NF)

Nenormalizirana relacija dovodi do anomalija u ažuriranju i nije pogodna za sve zahtjeve za izvještavanje koji se realno mogu očekivati.

Relacija R je u Prvoj normalnoj formi (1NF) ako su sve vrijednosti njenih atributa atomske.

Relaciju **STUDENT** možemo normalizirati (svesti na 1NF) ako u svakom retku prikazane tablice, za svaki ispit koji je neki student položio, ponovimo i sve ostale njegove podatke (**BR\_INDEX**, **IME**, **SEMESTAR**, **STUDIJ**, **VOD\_STUDIJA**).

### STUDENT

BR_INDEX	IME	SEMESTAR	STUDIJ	VOD_STUDIJA	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	OCJENA
21	Mate	4	Računarstvo	Frane	123	Baze podataka	2
21	Mate	4	Računarstvo	Frane	124	Matematika	4
21	Mate	4	Računarstvo	Frane	267	Fizika	5
77	Ana	7	Elektronika	Jure	678	Polja i valovi	4
77	Ana	7	Elektronika	Jure	589	Teorija sustava	1
77	Ana	7	Elektronika	Jure	245	Modeliranje	4
77	Ana	7	Elektronika	Jure	567	Elektronički sklopolvi	2
36	Pero	6	Elektronika	Jure	678	Polja i valovi	2
36	Pero	6	Elektronika	Jure	245	Modeliranje	3
...							

Međutim, da bi se smanjila redundancija podataka do koje bi, očigledno, ovakva normalizacija dovela, možemo ovako dobivenu relaciju, rastaviti na dvije relacije, i to:

**STUDENT1** (BR\_INDEX, **IME**, **SEMESTAR**, **STUDIJ**, **VOD\_STUDIJA**)

**PRIJAVA** (BR\_INDEX, ŠIF\_KOLEGIJA, **NAZ\_KOLEGIJA**, **OCJENA**)

### STUDENT1

BR_INDEX	IME	SEMESTAR	STUDIJ	VOD_STUDIJA
21	Mate	4	Računarstvo	Frane

77	Ana	7	Elektronika	Jure
36	Pero	6	Elektronika	Jure
...				

## PRIJAVA

BR_INDEX	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	OCJENA
21	123	Baze podataka	2
21	124	Matematika	4
21	267	Fizika	5
77	678	Polja i valovi	4
77	589	Teorija sustava	1
77	245	Modeliranje	4
77	567	Elektronički sklopovi	2
36	678	Polja i valovi	2
36	245	Modeliranje	3
...			

Pri svođenju nenormalizirane relacije **STUDENT** na prvu normalnu formu rastavili smo ovu relaciju na dvije relacije **STUDENT1** i **PRIJAVA**. Bitno je da proces rastavljanja jedne relacije na dvije ne uzrokuje bilo gubitak podataka ili stvaranje novih podataka koji ne postoje u polaznoj relaciji.

## 4.2. Funkcionalne zavisnosti - osnovne definicije i terminologija

Definicije Druge, Treće i Boyce-Codd-ove normalne forme zasnivaju se na konceptu funkcionalne zavisnosti atributa relacije.

**Funkcionalna zavisnost** se definira na slijedeći način:

Neka je zadana relacija R sa atributima X i Y.

X i Y mogu biti i složeni atributi (Složeni atribut je atribut koji se sastoji od više jednostavnih atributa)

*Atribut Y je funkcionalno zavisан od atributa X (ili kažemo da atribut X funkcionalno određuje atribut Y),*

$$R.X \rightarrow\! R.Y$$

*ako i samo ako svakoj vrijednosti X odgovara jedna i samo jedna vrijednost Y.*

Funkcionalna zavisnost atributa Y o atributu X označava se:

$$X \rightarrow\! Y$$

Ako atribut Y nije funkcionalno zavisan o atributu X, to se označava sa:

$X \dashv\rightarrow Y$

Npr. u relaciji STUDENT1 postoje slijedeće funkcionalne zavisnosti:

BR\_INDEX  $\dashv\rightarrow$  IME

BR\_INDEX  $\dashv\rightarrow$  SEMESTAR

BR\_INDEX  $\dashv\rightarrow$  STUDIJ

BR\_INDEX  $\dashv\rightarrow$  VOD\_STUDIJA

Definicija funkcionalne zavisnosti se može izraziti i drugačije, kao::

*Atribut Y relacije R je funkcionalno zavisan od atributa X relacije R ako i samo ako vrijedi da dvije n-torce relacije R koje imaju istu x-vrednost moraju imati istu i y-vrijednost.*

**Potpuna funkcionalna zavisnost** definira se na slijedeći način:

*Atribut Y relacije R je potpuno funkcionalno zavisan od atributa X relacije R ako je funkcionalno zavisan od atributa X, a nije funkcionalno zavisan ni od jednog pravog podskupa atributa X.*

Npr. relaciju PRIJAVA vrijedi:

BR\_INDEX, ŠIF\_KOLEGIJA  $\dashv\rightarrow$  OCJENA

BR\_INDEX  $\dashv\rightarrow$  OCJENA

ŠIF\_KOLEGIJA  $\dashv\rightarrow$  OCJENA

odnosno atribut OCJENA je *potpuno funkcionalno zavisan* samo od složenog atributa BR\_INDEX, ŠIF\_KOLEGIJA

Vrijedi također:

BR\_INDEX, ŠIF\_KOLEGIJA  $\dashv\rightarrow$  NAZ\_KOLEGIJA

BR\_INDEX  $\dashv\rightarrow$  NAZ\_KOLEGIJA

ŠIF\_KOLEGIJA  $\dashv\rightarrow$  NAZ\_KOLEGIJA

atribut NAZ\_KOLEGIJA je *nepotpuno funkcionalno zavisan* od složenog atributa BR\_INDEX, ŠIF\_KOLEGIJA, jer je funkcionalno zavisan i od njega i od jednog njegovog dijela (ŠIF\_KOLEGIJA).

**Tranzitivna funkcionalna zavisnost** definira se na slijedeći način::

Neka je zadana relacija R sa atributima A, B i C (atributi A, B i C mogu

biti i složeni atributi). **Ako u relaciji R važi:**

**A**  $\rightarrowtail$  **B**

**B**  $\rightarrowtail$  **C**

**A**  $\rightarrowtail$  **C**

**B**  $\not\rightarrowtail$  **A**

**C**  $\not\rightarrowtail$  **A**

*atribut C je tranzitivno funkcionalno zavisan od atributa A.*

*Jednostavnije rečeno, atribut C je tranzitivno funkcionalno zavisan od atributa A ako je funkcionalno zavisan od A i ako je funkcionalno zavisan od nekog atributa B koji je i sam funkcionalno zavisan od A.*

Npr. za relaciju STUDENT1 vrijedi:

**BR\_INDEX**  $\rightarrowtail$  **STUDIJ**

**BR\_INDEX**  $\rightarrowtail$  **VOD\_STUDIJA**

**STUDIJ**  $\rightarrowtail$  **VOD\_STUDIJA**

**STUDIJ**  $\not\rightarrowtail$  **BR\_INDEX**

**VOD\_STUDIJA**  $\not\rightarrowtail$  **BR\_INDEX**

odnosno atribut **VOD\_STUDIJA** je *tranzitivno funkcionalno zavisan* o atributu **BR\_INDEX**, jer je funkcionalno zavisan o atributu **BR\_INDEX** i atributu **STUDIJ** koji je sam funkcionalno zavisan o atributu **BR\_INDEX**.

U definiciji tranzitivne funkcionalne zavisnosti ništa nije rečeno o zavisnosti atributa B od atributa C. To znači da ona može biti bilo funkcionalna, bilo više značna (o više značnoj ovisnosti više će riječi biti kasnije). Ako je ova zavisnost više značna, takva tranzitivna zavisnost C od A se naziva stroga tranzitivna zavisnost. Stroga tranzitivna zavisnost dovodi do veće redundancije podataka.

### 4.3. Druga normalna forma (2NF)

Pogledajmo relaciju **PRIJAVA**.

**PRIJAVA**

BR_INDEX	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	OCJENA
21	123	Baze podataka	2
21	124	Matematika	4
21	267	Fizika	5

77	678	Polja i valovi	4
77	589	Teorija sustava	1
77	245	Modeliranje	4
77	567	Elektronički sklopovi	2
36	678	Polja i valovi	2
36	245	Modeliranje	3
...			

Očigledna je redundancija (ponavljanje) podataka u ovoj relaciji:

[NAZ\\_KOLEGIJA](#), za isti kolegij, se pojavljuje uz svakog studenta koji je taj kolegij položio. U održavanju baze podataka postoje opet iste anomalije:

- Ne mogu se dodati podaci o novom kolegiju ako ga neki student nije položio.
- Ako se iz baze podataka izbaci n-torka studenta koji je jedini položio neki kolegij, gube se sve informacije i o tom kolegiju.
- Ako se promijeni naziv kolegija, to se mora učiniti na onoliko mjesta koliko je studenata položilo taj kolegij.

Uzrok redundanciji i anomalijama u održavanju je nepotpuna funkcionalna zavisnost atributa [NAZ\\_KOLEGIJA](#) od složenog atributa [BR\\_INDEX](#), [ŠIF\\_KOLEGIJA](#), za koju smo ranije pokazali da je redundantna funkcionalna zavisnost.

Definicija *Druge normalne forme* zabranjuje postojanje ovakve zavisnosti:

Relacija R je u Drugoj normalnoj formi (2NF) ako i samo ako je u 1NF i svi njeni neključni atributi potpuno i funkcionalno zavise od primarnog ključa.

Neklučni atributi (atributi koji nisu kandidati za ključ, niti dio kandidata za ključ) relacije [PRIJAVA](#) su [NAZ\\_KOLEGIJA](#) i [OCJENA](#), a primarni ključ je složeni atribut [BR\\_INDEX](#), [ŠIF\\_KOLEGIJA](#). Kako je ranije pokazano atribut [OCJENA](#) je potpuno funkcionalno zavisn od primarnog ključa, a atribut [NAZ\\_KOLEGIJA](#) nije. Zbog toga relacija [PRIJAVA](#) nije u 2NF.

Svođenje na 2NF vrši se rastavljanjem promatrane relacije na dvije (ili više) relacija: u prvoj se ostavlja primarni ključ i svi atributi koji su potpuno funkcionalno zavisni od njega, a u drugoj (i ostalim) relacijama se realiziraju one funkcionalne zavisnosti koje su prouzrokovale nepotpune funkcionalne zavisnosti.

Npr. relacija [PRIJAVA](#) rastavlja se na dvije relacije i to:

## **PRIJAVA1(BR\_INDEX, ŠIF\_KOLEGIJA, OCJENA)**

## **KOLEGIJ(ŠIF\_KOLEGIJA, NAZ\_KOLEGIJA)**

### **PRIJAVA1**

BR_INDEX	ŠIF_KOLEGIJA	OCJENA
21	123	2
21	124	4
21	267	5
77	678	4
77	589	1
77	245	4
77	567	2
36	678	2
36	245	3
...		

### **KOLEGIJ**

ŠIF_KOLEGIJA	NAZ_KOLEGIJA
123	Baze podataka
124	Matematika
267	Fizika
678	Polja i valovi
589	Teorija sustava
245	Modeliranje
567	Elektronički sklopovi
...	

Definicija Druge normalne forme (2NF) može se neformalno formulirati kao:

Relacija R je u 2NF ako svi njeni atributi daju jednoznačne činjenice samo o cijelom ključu.

Atribut **NAZ\_KOLEGIJA** daje jednoznačnu činjenicu o dijelu ključa **ŠIF\_KOLEGIJA**, pa zbog toga, i po ovoj neformalnoj definiciji relacija **PRIJAVA** nije u 2NF.

Iz definicije 2NF i definicije potpune funkcionalne zavisnosti očigledno je da je svaka relacija sa jednostavnim (prostim) primarnim ključem u 2NF, jer prosti ključ nema semantički moguć pravi podskup.

## 4.4. Treća normalna forma (3NF)

Pogledajmo relaciju [STUDENT1](#).

### STUDENT1

BR_INDEX	IME	SEMESTAR	STUDIJ	VOD_STUDIJA
21	Mate	4	Računarstvo	Frane
77	Ana	7	Elektronika	Jure
36	Pero	6	Elektronika	Jure
...				

Očigledno je da u ovoj relaciji postoji redundancija (ponavljanje) podataka: ista vrijednost atributa [VOD\\_STUDIJA](#) se pojavljuje uz svakog studenta upisanog na odgovarajući [STUDIJ](#), i da postoje anomalije u održavanju baze podataka:

- Ne mogu se dodati informacije o novom studiju dok ga neki student ne upiše.
- Ako se iz relacije [STUDENT1](#) izbaci student koji je jedini bio upisan na neki studij, gube se informacije i o tom studiju.
- Ako se promijeni voditelj studija atribut [VOD\\_STUDIJA](#) treba promijeniti na onoliko mesta koliko je studenata upisano na taj studij.

Razlog za redundanciju i anomalije u održavanju baze podataka je postojanje tranzitivne funkcionalne zavisnosti (za koju smo pokazali da je redundantna) atributa [VOD\\_STUDIJA](#) od atributa [BR\\_INDEX](#).

Definicija [\*Treće normalne forme\*](#) zabranjuje postojanje ovakvih zavisnosti u relacijama.

Relacija R je u Trećoj normalnoj formi (3NF) ako i samo ako je u 2NF i ako u njoj ne postoji ni jedan atribut koji je tranzitivno funkcionalno zavisno o primarnom ključu.

Zbog postojanja tranzitivne zavisnosti atributa [VOD\\_STUDIJA](#) o primarnom ključu [BR\\_INDEX](#) relacija [STUDENT1](#) nije u 3NF. Svođenje na 3NF vrši se rastavljanjem relacije u dvije (ili više) relacije: u prvoj se ostavlja primarni ključ i svi neključni atributi koji nisu tranzitivno zavisni o primarnom ključu, a u drugoj (i ostalim) relacijama se realiziraju funkcionalne zavisnosti koje su dovele do tranzitivnih zavisnosti.

Npr. relacija [STUDENT1](#) rastavlja se na dvije relacije i to:

**STUDENT2(BR\_INDEX, IME, SEMESTAR, STUDIJ)**

**STUDIJ(STUDIJ, VOD\_STUDIJA)**

## STUDENT2

BR_INDEX	IME	SEMESTAR	STUDIJ
21	Mate	4	Računarstvo
77	Ana	7	Elektronika
36	Pero	6	Elektronika
...			

## STUDIJ

STUDIJ	VOD_STUDIJA
Računarstvo	Frane
Elektronika	Jure
...	

Neformalno, definicija Treće normalne forme (3NF) koja uključuje i 2NF, može se formulirati kao:

Relacija R je u 3NF ako svi njeni atributi daju jednoznačne činjenice o cijelom ključu i samo o cijelom ključu.

U relaciji **STUDENT1** atribut **VOD\_STUDIJA** je činjenica i o atributu **STUDIJ**, a ne samo o ključu **BR\_INDEX**.

Tranzitivna zavisnost u nekoj relaciji postoji uvijek kada, osim funkcionalne zavisnosti atributa od ključa (jednoznačne činjenice o ključu), u njoj postoji i neka druga funkcionalna zavisnost između atributa (jednoznačne činjenice o nekom atributu koji nije ključ).

Neformalne definicije 2NF i 3NF otkrivaju smisao postupka normalizacije. Naime, baza podataka je, kao što je rečeno, statički model realnog sustava. Samo ako se u njoj jasno prepoznaju tipovi entiteta, veze i atributi entiteta i veza u realnom sustavu, ona će biti pogodna za sve obrade podataka koje odgovaraju svim mogućim zahtjevima korisnika, odnosno neće izazivati anomalije u održavanju, niti će dovoditi do problema u izradi programa za izvještavanje.

Naš primjer odnosi se na realni sustav **FAKULTET**. U kontekstu iznesenih primjera možemo reći da se sustav **FAKULTET** sastoji od tipova entiteta: **STUDENT**, **KOLEGIJ** i **STUDIJ** čija su svojstva opisana preko njima pripadajućih atributa. (Npr. **BR\_INDEX**, **IME** i **SEMESTAR** za tip entiteta **STUDENT** ili **ŠIF\_KOLEGIJA** i **NAZ\_KOLEGIJA** za tip entiteta **KOLEGIJ** itd.). Tipovi entiteta **STUDENT** i **KOLEGIJ** su povezani vezom **PRIJAVA** koja ima atribut **OCJENA**, a tipovi entiteta **STUDENT** i

**STUDIJ** su povezani vezom **UPISAN** koja u modelu nije eksplisitno iskazana, već se ostvaruje preko atributa **STUDIJ** u relaciji **STUDENT2**.

Ovako definiran sustav jasno se prepozna u relacijskom modelu baze podataka sa relacijama u 3NF (**STUDENT2**, **KOLEGIJ**, **STUDIJ** i **PRIJAVA1**). Polazna relacija **STUDENT**, relacije **STUDENT1** i **PRIJAVA** predstavljale su neku "mješavinu" tipova entiteta i veza realnog sustava, a ne posebne, jasno definirane tipove entiteta i veza sa njima svojstvenim atributima. *Postupak normalizacije sigurno svodi model baze podataka na skup relacija od kojih svaka predstavlja neki osnovni tip entiteta ili veza u realnom sustavu.* Atributi takvih relacija su svojstveni atributi odgovarajućih tipova entiteta ili veza.

Primijetimo da bi istu konačnu shemu relacijskog modela odmah dobili da smo u fazi modeliranja entiteta postupili ispravno, te studente, kolegije i studije odmah prikazali kao posebne tipove entiteta.

#### 4.5. Boyce-Codd-ova normalna forma (BCNF)

Navedene definicije 2NF i 3NF je originalno dao E.F.Codd, tvorac relacijskog modela. Pokazalo se da ove definicije nisu dovoljno precizne (stroge), posebno u slučajevima kada relacija ima tzv "preklapajuće" kandidate za ključ (dva ili više složenih kandidata za ključ koji imaju barem jedan zajednički atribut).

Pogledajmo ponovno relaciju **PRIJAVA**.

**PRIJAVA(BR\_INDEX, ŠIF\_KOLEGIJA, NAZ\_KOLEGIJA, OCJENA)**

**PRIJAVA**

BR_INDEX	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	OCJENA
21	123	Baze podataka	2
21	124	Matematika	4
21	267	Fizika	5
77	678	Polja i valovi	4
77	589	Teorija sustava	1
77	245	Modeliranje	4
77	567	Elektronički sklopovi	2
36	678	Polja i valovi	2
36	245	Modeliranje	3
...			

U dosadašnjoj analizi ove relacije pretpostavljali smo da vrijede sljedeće

funkcionalne zavisnosti:

BR\_INDEX, ŠIF\_KOLEGIJA ---> NAZ\_KOLEGIJA

BR\_INDEX, ŠIF\_KOLEGIJA ---> OCJENA

ŠIF\_KOLEGIJA ---> NAZ\_KOLEGIJA

Jedini kandidat za ključ i primarni ključ je složeni atribut BR\_INDEX, ŠIF\_KOLEGIJA. Neključni atributi su NAZ\_KOLEGIJA i OCJENA. Pošto neključni atribut NAZ\_KOLEGIJA nepotpuno funkcionalno zavisi od primarnog ključa, relacija PRIJAVA nije u 2NF. Međutim, prepostavimo da sada postoji još i sljedeća funkcionalna zavisnost (da je NAZ\_KOLEGIJA jednoznačan):

NAZ\_KOLEGIJA ---> ŠIF\_KOLEGIJA

U tom slučaju u relaciji PRIJAVA postoje dva složena i "preklapajuća" kandidata za ključ, BR\_INDEX, ŠIF\_KOLEGIJA i BR\_INDEX, NAZ\_KOLEGIJA. Jedini neključni atribut je sada OCJENA, pa pošto on potpuno funkcionalno zavisi od primarnog ključa (bilo koji od kandidata da je izabran), relacija PRIJAVA je u 2NF. Međutim, sve anomalije u ažuriranju i dalje ostaju! To znači da definicija 2NF (pa samim tim i 3NF) nije dovoljno precizna.

Boyce-Codd-ova definicija uklanja te nepreciznosti. Za iskaz te definicije uvodi se i pojam *determinante relacije*.

*Determinanta relacije R je bilo koji atribut, jednostavan ili složen, o kojem neki drugi atribut u relaciji potpuno funkcionalno zavisi.*

Relacija R je u Boyce-Codd-ovoj normalnoj formi (BCNF) ako i samo ako su sve determinante u relaciji i kandidati za ključ.

Označimo sve determinante (D) i sve kandidate za ključ (KK) relacije PRIJAVA (uz prepostavku da važi i dodatno uvedena funkcionalna zavisnost NAZ\_KOLEGIJA ---> ŠIF\_KOLEGIJA):

BR\_INDEX, ŠIF\_KOLEGIJA ---> NAZ\_KOLEGIJA, (D) (KK)  
OCJENA

BR\_INDEX, NAZ\_KOLEGIJA ---> ŠIF\_KOLEGIJA, (D) (KK)  
OCJENA

ŠIF\_KOLEGIJA ---> NAZ\_KOLEGIJA (D)  
NAZ\_KOLEGIJA ---> ŠIF\_KOLEGIJA (D)

Sve determinante nisu kandidati za ključ pa relacija PRIJAVA nije u BCNF. Svođenje na BCNF vrši se rastavljanjem promatrane relacije na

dvije (ili više) relacija, pri čemu se iz polazne relacije izvlače relacije sa determinantama koje nisu kandidati za ključ. U našem primjeru relacija **PRIJAVA** rastavlja se na dvije relacije:

**PRIJAVA1(BR\_INDEX, ŠIF\_KOLEGIJA, OCJENA)**

**KOLEGIJ(ŠIF\_KOLEGIJA, NAZ\_KOLEGIJA)**

Složeni, "preklapajući" kandidati za ključ su signal da relacija možda nije u BCNF, iako se može pokazati da neka relacija i sa složenim, "preklapajućim" kandidatima za ključ može biti u BCNF.

Definicija BCNF je striktno stroža od definicije 2NF i 3NF. To znači da je svaka relacija koja je u BCNF sigurno i u 2NF i 3NF. Obrnuto ne važi.

## 4.6. Višezačne zavisnosti i Četvrta normalna forma (4NF)

Do sada smo kao jedinu moguću zavisnost između atributa relacije naveli razne vrste funkcionalnih zavisnosti. Međutim, postoje i mnoge druge zavisnosti između atributa u relaciji koje također mogu dovesti do anomalija u održavanju baze podataka. Najčešće se javljaju tzv. višezačne zavisnosti.

Pogledajmo nenormaliziranu relaciju PROGRAM\_KOLEGIJA.

**PROGRAM\_KOLEGIJA**

KOLEGIJ	NASTAVNIK	KNJIGA
Baze podataka	Ljubo	O'Neil
	Mirko	Date
	Jure	
Metode optimizacije	Vlado	Rao
		Arora
...		

Smisao je sljedeći: Jedan kolegij predaje više nastavnika. Za jedan kolegij se koristi više knjiga. Ne postoji nikakva veza između nastavnika i knjiga. (Ne zna se koji nastavnik koristi koju knjige, jednu ili više).

Da bi se ova relacija dovela u 1NF i sačuvala semantika problema neophodno je da se u normaliziranoj relaciji nađu, za dani kolegij i danog nastavnika, n-torce sa svim mogućim knjigama za taj kolegij.

**PROGRAM\_KOLEGIJA(KOLEGIJ, NASTAVNIK, KNJIGA)**

**PROGRAM\_KOLEGIJA**

KOLEGIJ	NASTAVNIK	KNJIGA

Baze podataka	Ljubo	O'Neil
Baze podataka	Ljubo	Date
Baze podataka	Mirko	O'Neil
Baze podataka	Mirko	Date
Baze podataka	Jure	O'Neil
Baze podataka	Jure	Date
Metode optimizacije	Vlado	Rao
Metode optimizacije	Vlado	Arora
...		

Relacija PROGRAM\_KOLEGIJA je u BCNF (sve determinante su kandidati za ključ). Međutim, redundancija (ponavljanje) podataka je očigledna, pa samim tim postoje i anomalije u ažuriranju.

Npr., ubacivanje informacije da se za kolegij Baze podataka koristi i knjiga Harrington, zahtjevalo bi ubacivanje tri nove n-torke, po jednu za svakog nastavnika koji predaje taj kolegij.

Očigledno je da relaciju PROGRAM treba rastaviti na dvije relacije

#### **PREDAJE(KOLEGIJ, NASTAVNIK)**

#### **LITERATURA(KOLEGIJ, KNJIGA)**

#### **PREDAJE**

KOLEGIJ	NASTAVNIK
Baze podataka	Ljubo
Baze podataka	Mirko
Baze podataka	Jure
Metode optimizacije	Vlado
...	

#### **LITERATURA**

KOLEGIJ	KNJIGA
Baze podataka	O'Neil
Baze podataka	Date
Metode optimizacije	Rao
Metode optimizacije	Arora
...	

Rastavljanje je izvršeno bez gubitka podataka. No ono nije u skladu ni s jednim dosada izloženim pravilom, jer je polazna relacija

## PROGRAM\_KOLEGIJA u BCNF.

Veze koje postoje između atributa ove relacije nazivaju se **višeznačnim vezama**.

Definicija **višeznačne zavisnosti** glasi:

*Zadana je relacija  $R(A, B, C)$ .*

**Višeznačna zavisnost**  $A \rightarrow\!\!> B$  vrijedi ako i samo ako: skup  $B$ -vrijednosti koje se u  $R$  pojavljuju uz zadani par ( $A$ -vrijednost,  $C$ -vrijednost) ovisi samo o  $A$ -vrijednosti, a ne i o  $C$ -vrijednosti.

*Atributi  $A$ ,  $B$  i  $C$  mogu biti jednostavni ili složeni atributi.*

Očigledno je se višeznačna zavisnost može definirati u relaciji koja ima najmanje tri atributa.

U našoj relaciji **PROGRAM\_KOLEGIJA** postoje dvije višeznačne zavisnosti:

**KOLEGIJ**  $\rightarrow\!\!> NASTAVNIK$

**KOLEGIJ**  $\rightarrow\!\!> KNJIGA$

*Svaka funkcionalna zavisnost je istovremeno i višeznačna zavisnost, odnosno, funkcionalne zavisnosti su specijalan slučaj višeznačne zavisnosti u kojem skup vrijednosti  $B$  za dati skup vrijednosti  $A$  ima točno jednu vrijednost.*

Očigledno je da "prave" višeznačne zavisnosti (višeznačne zavisnosti koje nisu funkcionalne, kao što je to slučaj u našem primjeru) dovode do anomalija u održavanju baze podataka. Definicija **Četvrte normalne forme** zabranjuje postojanje ovakvih zavisnosti u relaciji.

Relacija  $R$  je u Četvrtoj normalnoj formi (4NF) ako je u BCNF i ako su sve višeznačne zavisnosti funkcionalne zavisnosti o primarnom ključu.

Za praktičnu primjenu može se dati i sljedeća, neformalna i nedovoljno precizna definicija 4NF:

Relacija  $R$  je u 4NF ako u njoj ne postoje dvije (ili više) nezavisnih višeznačnih činjenica.

Relacija **PROGRAM\_KOLEGIJA** nije u 4NF, jer u njoj postoje dvije nezavisne višeznačne činjenice (jedan **KOLEGIJ** predaje više **NASTAVNIKA**, za jedan **KOLEGIJ** se koristi više **KNJIGA**, a ne postoji nikakva veza **NASTAVNIKA** i **KNJIGA**). Relacije **PREDAJE** i **LITERATURA** su u 4NF jer u njima ne postoje višeznačne zavisnosti samim tim što se radi o binarnim relacijama, odnosno obje sadrže samo

po jednu višeznačnu činjenicu.

Kako su funkcionalne zavisnosti samo specijalan slučaj višeznačnih zavisnosti, svaka relacija koja je u 4NF je istovremeno i u BCNF. Obrnuto ne važi.

## 4.7. Zavisnosti spajanja i Peta normalna forma (5NF)

Kazali smo da su višeznačne zavisnosti općenitiji oblik funkcionalnih zavisnosti koje omogućuju predstavljanje semantički složenijih koncepata u modelu. Postoji još općenitiji oblik zavisnosti, tzv. "*"zavisnost spajanja"* (*join dependency*) koja u sebi sadrži višeznačne, pa samim tim i funkcionalne zavisnosti. Zavisnost spajanja pokazati ćemo na istom primjeru odnosa **KOLEGIJ**, **NASTAVNIK**, **KNJIGA**, ali, sada, sa nešto izmijenjenom smislu (semantikom).

Ranije smo prepostavljali da u relaciji **PROGRAM\_KOLEGIJA** ne postoji nikakva veza između atributa **NASTAVNIK** i **KNJIGA**. Prepostavimo sada da ova veza postoji, da jedan nastavnik može da koristi nula, jednu ili više knjiga i da se zna koji nastavnik koristi koje knjige (Ljubo koristi samo knjigu O'Neil, Mirko koristi samo knjigu Date a Jure i Vlado koriste obje knjige). Relacija **PROGRAM\_KOLEGIJA** u tom slučaju bi bila:

**PROGRAM\_KOLEGIJA(KOLEGIJ, NASTAVNIK, KNJIGA)**

**PROGRAM\_KOLEGIJA**

KOLEGIJ	NASTAVNIK	KNJIGA
Baze podataka	Ljubo	O'Neil
Baze podataka	Mirko	Date
Baze podataka	Jure	O'Neil
Baze podataka	Jure	Date
Metode optimizacije	Vlado	Rao
Metode optimizacije	Vlado	Arora
...		

Relacija **PROGRAM\_KOLEGIJA** je u 4NF, jer u njoj ne postoji višeznačne zavisnosti (Jedan **KOLEGIJ** predaje više **NASTAVNIKA**, za jedan **KOLEGIJ** se koristi više **KNJIGA**, ali sada **KNJIGE** zavise od **NASTAVNIKA**). Međutim, redundancija podataka i anomalije u ažuriranju i dalje postoje. Da bi se izbjegla redundancija podataka i istovremeno sačuvala nova semantika relacije **PROGRAM\_KOLEGIJA** očigledno je da je treba rastaviti (bez gubitka podataka), u sljedeće tri relacije:

**PREDAJE(KOLEGIJ, NASTAVNIK)**

**LITERATURA(KOLEGIJ, KNJIGA)**

**NAST\_KNJIGA(NASTAVNIK, KNJIGA)**

### **PREDAJE**

KOLEGIJ	NASTAVNIK
Baze podataka	Ljubo
Baze podataka	Mirko
Baze podataka	Jure
Metode optimizacije	Vlado
...	

### **LITERATURA**

KOLEGIJ	KNJIGA
Baze podataka	O'Neil
Baze podataka	Date
Metode optimizacije	Rao
Metode optimizacije	Arora
...	

### **NAST\_KNJIGA**

NASTAVNIK	KNJIGA
Ljubo	O'Neil
Mirko	Date
Jure	O'Neil
Jure	Date
Vlado	Rao
Vlado	Arora
...	

Specifična vrsta zavisnosti koja postoji u novoj verziji relacije PROGRAM se naziva **zavisnost spajanja**. Kažemo da su u relaciji PROGRAM atributi KOLEGIJ, NASTAVNIK, KNJIGA vezani preko zavisnosti spajanja.

*U relaciji  $R(X, Y, Z)$  postoji zavisnost spajanja ako i samo ako relacija  $R$  nastaje prirodnim spajanjem relacije sa atributima  $X, Y$ , relacije sa atributima  $X, Z$  i relacije sa atributima  $Y, Z$  gdje su  $X, Y$  i  $Z$  podskupovi atributa relacije  $R$ .*

Definicija **Pete normalne forme** može se formulirati kao::

Relacija R je u Petoj normalnoj formi ako i samo ako se svaka zavisnost spajanja može

pripisati kandidatu za ključ.

Relacija **PROGRAM\_KOLEGIJA** se može rekonstruirati iz relacija njenih projekcija spajanjem relacija **NAST\_KNJIGA** i **LITERATURA** po atributu **KNJIGA**, a zatim prirodnim spajanjem tako dobivenog rezultata sa relacijom **PREDAJE** po atributu **NASTAVNIK**. Kako atributi spajanja **KNJIGA** i **NASTAVNIK** nisu i kandidati za ključ relacije **PROGRAM\_KOLEGIJA** ona nije u 5NF.

Praktična, neformalna definicija 5NF može se formulirati kao:

Relacija je u 5NF onda kad se njen informacijski sadržaj ne može rekonstruirati iz relacija nižeg stupnja, s tim što se slučaj relacija nižeg stupnja sa istim ključem isključuje.

Kako je višeznačna zavisnost specijalan slučaj zavisnosti spajanja, ako je relacija u 5NF ona je sigurno i u 4NF, pa samim tim i u svim ostalim.

Obrnuto ne važi.

5NF se često naziva i "projekcija-spajanje normalna forma".

#### 4.8. Normalna forma ključeva i domena

Normalne forme se postepeno uvode teoriju relacijskih baza podataka: počevši od prve tri koje je definirao Codd, 1972. godine, preko Boyce-Codd-ove, 4NF u kojoj su funkcionalne zavisnosti uopćene u višeznačne zavisnosti do 5NF u kojoj su višeznačne zavisnosti uopćene u zavisnosti spajanja. Pitanje je: postoje li i daljnja uopćenje zavisnosti atributa u relacijama i da li se može definirati neka nova (šesta, sedma, ...) normalna forma?

R.Fagin je 1981. dao najopćenitiju definiciju **normalne forme ključeva i domena (DK/NF)** i pokazao je da relacija koja je u DK/NF ne uzrokuje anomalije u održavanju, i obrnuto, da se relacija, koja ne uzrokuje anomalije u ažuriranju, nalazi u DK/NF. Time je isključena potrebu za definiranjem novih normalnih formi. Koje se relacije mogu svesti na DK/NF i koji je opći postupak tog svođenja, još uvijek je otvoreno pitanje.

Definiciju **normalne forme ključeva i domena (DK/NF)** može se izraziti kao

Relacija je u DK/NF ako je svako ograničenje na vrijednosti njenih atributa posljedica definicije ključeva i domena

Ako je relacija u DK/NF ona je sigurno u svim ostalim. Međutim, direktna primjena ove definicije praktično nije moguća, jer otkrivanje "ograničenja"

u relacijama zahtjeva, u osnovi, otkrivanje funkcionalnih, višeznačnih i zavisnosti spajanja, a to praktično znači primjenu svih navedenih definicija.

## 4.9. Razlozi zbog kojih se može odustati od normalizacije

Za većinu praktičnih primjera dovoljno je relacije normalizirati do 3NF. Koji put je potrebno neku relaciju i dalje normalizirati do BCNF ili 4NF.

Peta normalna forma i normalna forma ključeva i domena nisu od praktičnog značaja.

Postoje razlozi zbog kojih iznimno možemo odustati od pune normalizacije. Navesti ćemo dva takva razloga:

### Složeni atribut

Dešava se da nekoliko atributa u relaciji čine cjelinu koja se u aplikacijama nikad ne rastavlja na sastavne dijelove. Na primjer, promatrajmo relaciju

**KUPAC** ( PREZIME IME, POŠTANSKI BROJ, GRAD, ULICA ) .

Strogo govoreći, atribut GRAD je funkcionalno ovisan o atributu POŠTANSKI BROJ, pa relacija nije u 3NF. No mi znamo da POŠTANSKI BROJ, GRAD i ULICA čine cjelinu koja se zove ADRESA. Budući da se podaci iz adrese koriste i ažuriraju "u paketu", ne može doći do prethodno spominjanih anomalija, te stoga nije preporučljivo razbijati ovu relaciju na dvije.

### Efikasno čitanje podataka

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

Projektant baze podataka treba procijeniti kada treba provesti normalizaciju do kraja a kada ne. Za tu procjenu je važno razumijevanje značenja podataka i načina kako će se oni koristiti.

# 5. OPERACIJE RELACIJSKOG MODELA

Postoje dva načina iskazivanja operacija relacijskog modela:

- *Relacijska algebra*, u kojoj se definira skup operacija pomoću kojih je moguće, na proceduralan način, dobiti željenu relaciju (tablicu) iz skupa danih relacija (tablica).
- *Relacijski račun*, koji je neproceduralni način iskazivanja operacija, gdje se, pomoću konstrukcija predikatnog računa prvog reda u kojem su varijable ili n-torce relacija (relacijski račun n-torki) ili domene relacija (relacijski račun domena), definiraju osobine relacije koja se želi dobiti.

Relacijska algebra i relacijski račun (i račun n-torki i račun domena) su fundamentalno ekvivalentni jedno drugom. E.F Codd je pokazao da se svaki izraz relacijskog računa može svesti na semantički ekvivalentan izraz u relacijskoj algebri, a Ullman, da se svaki izraz u relacijskoj algebri može svesti na izraz relacijskih računa, pa se, na osnovu toga, može zaključiti da su ova dva formalizma logički ekvivalentna.

## 5.1. Relacijska algebra

Relacijsku algebru uveo je E.F. Codd u svojim radovima iz 70-tih godina XX. stoljeća. Riječ je o teorijskoj (matematičkoj) notaciji, a ne o praktičnom jeziku kojeg bi ljudi zaista neposredno koristili. Zato niti ne postoji standardna sintaksa.

*Operandi relacijske algebre su relacije a rezultat ponovno relacija.*

Relacijska algebra se svodi na "izračunavanje" relacijskih algebarskih izraza, građenih od relacija kombiniranih sa unarnim ili binarnim operatorima svrstanim u dvije grupe:

- Operacije teorije skupova (Set-theory operations)
- Prirodne relacijske operacije (Native-relation operations)

### 5.1.1 Operacije teorije skupova (Set-theory operations)

Podsjetimo se, relacija se definira kao podskup Kartezijevog produkta i može se tretirati kao skup n-torki. Zbog toga su osnovne operacije algebre skupova i operacije relacijske algebre:

- Unija ( $T = R \cup S$ )
- Razlika ( $T = R - S$ )
- Presjek ( $T = R \cap S$ )
- Produkt ( $T = R \times S$ )

Operacije unije, presjeka i razlike ne mogu se primjeniti, (ili preciznije,

nemaju semantički značaj) na bilo koje dvije relacije.

Npr, unija relacija **STUDENT** i **KOLEGIJ** formalno treba rezultirati relacijom čije su n-torce elementi bilo relacije **STUDENT** bilo relacije **KOLEGIJ**, a tako dobiven rezultat nije relacija u smislu u kojem su definirane polazne relacije.

Za izvođenje operacija unije, presjeka i razlike definira se slijedeći uvjet kompatibilnosti relacija R i S:

Relacije R i S moraju imati isti broj atributa (isti stupanj), a odgovarajući atributi moraju biti definirani nad istim domenama.

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

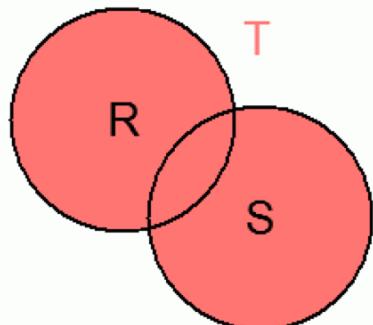
### 5.1.1.1. Unija (union)

Zadane su relacije **R** i **S** koje zadovoljavaju navedeni uvjet kompatibilnosti. Rezultat operacije unije:

$$T = R \dot{\cup} S \quad \text{ili} \quad T = R \text{ union } S$$

je relacija **T** koja sadrži sve n-torce koje se pojavljuju bilo u **R** bilo u **S**.

Grafički se unija može prikazati kao



Npr. unija relacija **T** polaznih relacija **R** i **S** (prikazanih njihovim ekstenzijama) je

R		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

S		
A	B	C
a3	b3	c3
a4	b4	c4
a5	b5	c5

T := R ∪ S		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Uočimo da se u eliminiraju zapis (n-torce) koji se ponavljaju da bi se

očuvalo pravilo jedinstvenosti n-torki po primarnom ključu.

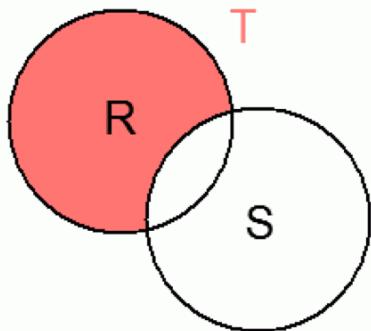
### 5.1.1.2. Razlika (minus)

Zadane su relacije  $R$  i  $S$  koje zadovoljavaju navedeni uvjet kompatibilnosti. Rezultat operacije razlike

$$T = R - S \quad \text{ili} \quad T = R \text{ minus } S$$

je relacija  $T$  u koje se nalaze sve n-torce relacije  $R$  koje *nisu istovremeno* i n-torce relacije  $S$ .

Grafički se razlika može prikazati kao



Npr. razlika relacija  $T$  polaznih relacija  $R$  i  $S$  (prikazanih njihovim ekstenzijama) je

R			
	A	B	C
a1	b1	c1	
a2	b2	c2	
a3	b3	c3	

S			
	A	B	C
a3	b3	c3	
a4	b4	c4	
a5	b5	c5	

T := R - S			
	A	B	C
a1	b1	c1	
a2	b2	c2	

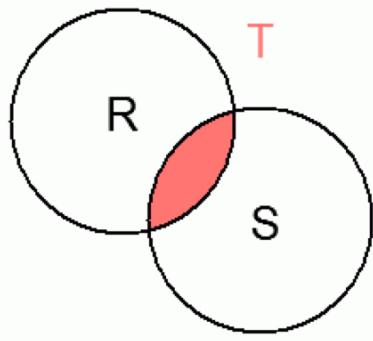
### 5.1.1.3. Presjek (intersect)

Zadane su relacije  $R$  i  $S$  koje zadovoljavaju navedeni uvjet kompatibilnosti. Rezultat operacije presjeka

$$T = R \cap S \quad \text{ili} \quad T = R \text{ intersect } S$$

je relacija  $T$  koja sadrži sve n-torce koje se pojavljuju u obje relacije  $R$  i  $S$ .

Grafički se presjek može prikazati kao



Npr. presjek relacija **T** polaznih relacija **R** i **S** (prikazanih njihovim ekstenzijama) je

<b>R</b>	<b>S</b>	<b>T := R ∩ S</b>
A	A	A
a1 b1 c1	a3 b3 c3	a3
a2 b2 c2	a4 b4 c4	b3
a3 b3 c3	a5 b5 c5	c3

Primijetimo da uvijek vrijedi (Provjerite!):

$$R \cap S = R - (R - S) = S - (S - R)$$

$$R - S = R - (R \cap S)$$

Kažemo da su **presjek i razlika su međuzavisne operacije**.

Operacije unije i razlike su operacije pogodne za ažuriranje baze podataka. Operacijom unije mogu se u neku relaciju dodati nove n-torke, a operacijom razlike mogu se iz neke relacije izbaciti (brisati) neželjene n-torke.

Izmjena vrijednosti pojedinih atributa u nekoj relaciji vrši se uzastopnom primjenom operacije razlike, pomoću koje se izbaci cijela n-torka u kojoj se nalazi promatrani atribut, a zatim se operacijom unije "vraća" izbačena n-torka sa promijenjenom vrijednošću atributa.

#### 5.1.1.4. (Kartezijev) Produkt

Kartezijev produkt se može primijeniti na bilo koje dvije relacije. Rezultat ove operacije

$$T = R \times S \quad \text{ili} \quad T = R \text{ times } S$$

je relacija T čije su n-torke svi "parovi" koje čine svaka n-torka relacije R sa svim n-torkama relacije S.

Općenito za Kartezijev produkt relacija vrijedi:

Ako relacija R ima **m** redaka(n-torki) i **n** stupaca(atributa) i relacija S ima **k** redaka(n-torki) i **j** stupaca(atributa) tada relacija T, koja je Kartezijev produkt relacija R i S ima  **$m \cdot k$**  redaka(n-torki) i  **$n+j$**  stupaca(atributa).

Atributi u relaciji  $T := R \times S$  imaju isto ime kao odgovarajući atributi u **R** odnosno **S**, s time da se po potrebi to ime proširuje imenom polazne relacije.

Npr. (Kartezijev) produkt relacija **T** polaznih relacija **R** i **S** (prikazanih njihovim ekstenzijama) je

<b>R</b>	<b>S</b>	<b><math>T := R \times S</math></b>
A a1 a2 a3	B b1 b2 b3	C
D d1 d2	E e1 e2	
a1 a2 a3	b1 b2 b3	D d1 d2 d1 d2 d1 d2 d2
b1 b2 b3	c1 c2 c3	E e1 e2 e1 e2 e1 e2
c1 c2 c3		

*Za operatore unije, presjeka i produkta vrijede pravila asocijativnosti i komutativnosti.*

$$(R \cup S) \cup T = R \cup (S \cup T) = R \cup S \cup T$$

$$(R \cap S) \cap T = R \cap (S \cap T) = R \cap S \cap T$$

$$(R \times S) \times T = R \times (S \times T) = R \times S \times T$$

## 5.1.2 Prirodne relacijske operacije (Native-relation operations)

Pored skupovnih operacija u relacijskom modelu se definiraju i sljedeće operacije specifične za relacijske:

- Projekcija ( $T = P [a] R$ )
- Selekcija ( $T = R \text{ where } a=x$ )
- Spajanje ( $T = R >< S$ )
- Dijeljenje ( $T = R, S$ )

### 5.1.2.1. Projekcija

*Operacija projekcije je unarna operacija* koja iz neke relacije selektira (izabire) skup navedenih atributa, odnosno "vadi" vertikalni (stupčasti) podskup iz odgovarajuće tablice.

Neka je  $R(A_1, A_2, \dots, A_n)$  relacija, a  $X$  podskup njenih atributa. Označimo

sa Y komplement  $\{A_1, A_2, \dots, A_n\}$  - X. Rezultat operacije projekcije relacije R po atributima X je

$P[X] \cap R = \{x \in \Omega \text{ tako da postoji } y \text{ da je } \langle x, y \rangle \in R\}$ .

Jednostavnije kazano, projekcija tablice R nad atributima X,Y,Z (označava se sa P[X,Y,Z] R)

T := P[X,Y,Z] R

je tablica T koja ima samo atribute X, Y, Z i koja sadržava sve n-torke iz relacije R.

Operacijom projekcije nad zadanim tablicom R dobiva se rezultat koji sadrži isti broj redova kao i tablica R, ali samo one atribute (stupce) po kojima se vrši projekcija.

Grafički se projekcija može prikazati kao

A bar chart with three bars labeled X, Y, and Z. The bars are pink and have black outlines. They are positioned on a grid background. Bar X is the leftmost bar, Bar Y is the middle bar, and Bar Z is the rightmost bar. The height of each bar corresponds to its value.

Category	Value
X	10
Y	10
Z	10

Npr. projekcija  $T$  relacije  $R$  po atributima  $B$  i  $C$  (relacija  $R$  je prikazana svojom ekstenzijom) je

R			T := P[A,B] R		
A	B	C	B	C	
a1	b1	c1	b1	c1	
a2	b2	c2	b2	c2	
a3	b1	c1	b2	c3	
a3	b2	c3	b3	c3	
a3	b3	c3			

(Kada se iz neke relacije selektira (izabere) podskup atributa preko operacije projekcije u rezultatu se mogu pojaviti duplikati n-torki. Operacija projekcije podrazumijeva da se ovi duplikati eliminiraju, kako je to u navedenom primjeru i urađeno).

#### **5.1.2.2. Selekcija (Izdvajanje, Restrikcija)**

*Selekcija je također unarna operacija* koja iz zadane relacije selektira

(izabere) n-torce koje zadovoljavaju zadati uvjet ("vadi" horizontalni podskup tablice).

Formalno se definira na slijeci način:

Neka je zadana relacija  $R(A_1, A_2, \dots, A_n)$  i predikat  $P$  definiran nad njenim atributima. Rezultat operacije selekcije

$$S[P] R = \{ x \frac{1}{2} x \hat{\top} R \mid P(x) \}$$

Operacijom izdvajanja (selekcije) nad zadanom tablicom R izdvaja se skup redaka koji zadovoljavaju uvjet po kojem se selekcija vrši. Tablica koja se dobiva kao rezultat operacije selekcije sadrži sve atribute (stupce) kao i izvorna tablica, ali samo one retke koji zadovoljavaju traženi uvjet. Dobivena tablica predstavlja horizontalni podskup izvorne tablice.

Grafički se projekcija može prikazati kao

Npr. selekcija  $T$  relacije  $R$  gdje je  $C=c_3$  (relacija  $R$  je prikazana svojom ekstenzijom) je

R			T := S[C=c3] R		
A	B	C	A	B	C
a1	b1	c1	a3	b2	c3
a2	b2	c2	a3	b3	c3
a3	b1	c1			
a3	b2	c3			
a3	b3	c3			

Selekciju na relaciji R u skladu s Booleovim uvjetom B označavamo s S[B] R. Uvjet B je logički izraz (jednostavan ili složen) koji se sastoji od:

- operanada koji su ili konstante ili atributi,
  - operatora za usporedivanje  $=$ ,  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $\neq$ ,
  - logičkih operatora and, or, not.

### **5.1.2.3. Spajanje (Join)**

*Spajanje je binarna operacija koja spaja dvije relacije na taj način da se u*

*rezultatu pojavljuju oni parovi n-torki jedne i druge relacije koji zadovoljavaju uvjet zadan nad njihovim atributima.* Formalno se definira na slijedeći način:

Neka su zadane relacije  $R1(A1, A2, \dots, An)$  i  $R2(B1, B2, \dots, Bm)$  i predikat  $Q$  definiran nad njihovim atributima. Označimo sa  $X$  i  $Y$  skupove atributa relacija  $R1$  i  $R2$ , respektivno.

Rezultat operacije spajanja ovih relacija (tzv. *theta spajanje*) je

$$R1 [xQ] R2 = \{ \langle x,y \rangle \mid x \in R1 \text{ AND } y \in R2 \text{ AND } Q(x,y) \}.$$

Oznaka  $xQ$  za operaciju spajanja ukazuje na činjenicu, očiglednu iz definicije theta spajanja, da ova operacija nije primitivna operacija relacijske algebre, već se može izvesti uzastopnom primjenom operacija Kartezijevog produkta ( $x$ ) i selekcije po predikatu  $Q$  iz tako dobivene relacije.

Ako je predikat  $Q$  definiran sa  $Ak = Bj$ , s tim da su i atributi  $Ak$  i  $Bj$  definirani nad istim domenama, tada se takvo spajanje naziva ekvispajanje.

Grafički se ekvispajanje može prikazati kao

The figure consists of three separate horizontal bar charts, each with five bars. The first bar in each chart is light orange, followed by a pink bar, then a long light red bar, a light blue bar, and a teal bar. The bars are set against a background of a grid of light gray lines.

Npr. **ekvispajanje** T polaznih relacija R i S po atributu C (relacije R i S su prikazane njihovim ekstenzijama) je

R			S			T := R [ R.C = S.C] S					
A	B	C	D	E	C	A	B	C	D	E	C
a1	b1	c1	d3	e3	c3	a2	b2	c2	d4	e4	c2
a2	b2	c2	d4	e4	c2	a3	b3	c3	d3	e3	c3
a3	b3	c3	d5	e5	c5						

Očigledno je da se u rezultatu ekvispajanja uvijek pojavljuju dva ista stupca. Ako se jedan od ova dva stupca izbaci, takvo spajanje se naziva prirodno spajanje. Uobičajeno je da se označava sa prirodno spajanje označava sa \*

Npr. *prirodno spajanje* T polaznih relacija R i S po atributu C (relacije R i S su uvek različite)

[S](#) su prikazane njihovim ekstenzijama) je

R		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

S		
D	E	C
d3	e3	c3
d4	e4	c2
d5	e5	c5

T := R [ R.C * S.C] S		
A	B	C
a2	b2	c2
a3	b3	c3
D	E	
a2	b2	d4
a3	b3	d3
		e4
		e3

#### 5.1.2.4. Dijeljenje T:=R , S

Relacija T koja se dobije dijeljenjem relacija R i S je najveća tablica za koju vrijedi da se sve n-torce Kartezijskog produkta T x S nalaze u tablici R.

Formalno se definira na slijedeći način:

Neka su R1(X,Y) i R2(Z) relacije gdje su X, Y i Z skupovi atributa takvi da su Y i Z jednakobrojni, a odgovarajuće domene su im jednake. Rezultat operacije dijeljenja

$$R1 [Y , Z] R2 = R(X)$$

gde n-torka [x](#) uzima vrijednosti iz [R1.X](#), a par [<x,y>](#) postoji u [R1](#) za sve vrijednosti [y](#) koje se pojavljuju u [R2\(Z\)](#).

Npr. [rezultat dijeljenja](#) T polaznih relacija [R](#) i [S](#) (relacije [R](#) i [S](#) su prikazane njihovim ekstenzijama) je

R	S	T = R , S	T x S
A B D	A B	D	( A B D )
s 1 4	s 1	4	
s 1 5		5	
c 2 6			

Dijeljenje je operacija pogodna za upite u kojima se javlja riječ "svi".

Operacija dijeljenja nije primitivna operacija relacijske algebre već se može izvesti na slijedeći način:

$$R(X, Y) [Y , Z] S(Z) = P[X] R - P[X] ((P[X] R x S) - R)$$

Objašnjenje:

Skup atributa X čini atribut R.D, skup atributa Y atributi R.A i R.B a skup atributa Z atributi S.A i S.B.

[P\[X\]](#) R daje sve n-torce koje mogu da učestvuju u rezultatu

R		
---	--	--

S		
---	--	--

P[D] R		
--------	--	--

A	B	D
s	1	4
s	1	5
c	2	6

A	B
s	1

D
4
5
6

$P[D] R \times S$  daje relaciju u kojoj se za svaku vrijednost  $z$  iz  $S$  pojavljuju parovi  $\langle x, z \rangle$  sa svim vrijednostima  $x$ .

R		
A	B	D
s	1	4
s	1	5
c	2	6

A	B
s	1

P[D]	R x S
A	B
s	1
s	1
s	1
c	1

(P[D]) R x S) - R		
A	B	D
c	2	6

P[D]((P[D] R x S) - R)
D
6

$(P[X] R x S) - R$  ne sadrži ni u jednom paru  $\langle x, z \rangle$  one vrijednosti  $x$  za koje u relaciji  $R$ , kao vrijednosti  $y$ , postoje sve vrijednosti  $z$ .

P[D]
R
D
4
5
6

P[D]((P[D] R x S) - R)
R
D

R(X, Y) [Y , Z] S(Z)
D
4
5

Kompletan, izraz prema tome, je relacija koja sadrži one n-torce x za koje postoje u paru  $\langle x, y \rangle$ , kao vrijednosti  $y$ , sve vrijednosti  $z$

### 5.1.3. Prioritet operatora relacijske algebre

Relacijska algebra je proceduralni jezik. Pomoću operacija relacijske algebre tvori se procedura koja dovodi do odgovora na postavljeni upit. Premda je proceduralni jezik, relacijska algebra je znatno moćnija od klasičnih programskih jezika koji su također proceduralni jezici. Razlog za to je što su operandi relacijske algebre relacije (cijele tablice), a operandi operacija sa datotekama u klasičnim jezicima je zapis (redak tablice).

Može se pokazati da se bilo koja relacija (bilo koji upit), izvodljiva iz skupa danih relacija, može dobiti procedurom (algoritmom) od tri koraka:

1. Karteziјev produkt svih relacija koje se koriste.
2. Selekcija n-torki za koje se predikat selekcije evaluira u T (true)
3. Projekcija rezultata po atributima koji se prikazuju.

Ova procedura se može iskazati jednim općim izrazom relacijske algebre

$$P[A_1, A_2, \dots, A_n] ( S[P] (R_1 \times R_2 \times \dots \times R_m)).$$

Međutim, ovakvo izvođenje operacija bilo bi veoma "skupo" jer bi rezultat Kartezijevog produkta relacija  $R_1, R_2, \dots, R_m$  bio relacija sa  $k_1 \times k_2 \times \dots \times k_m$  n-torki, gde su sa  $k_i$  označene kardinalnosti (broj stupaca) odgovarajućih relacija. Zbog toga se i definira operacija spajanja koja istovremeno obavlja Kartezijev produkt i selekciju, smanjujući na taj način broj n-torki koji se pretražuje. Pored toga očigledno je da su algoritmi sa više koraka, u kojima bi se prvo vršile sve selekcije koje se mogu izvesti na pojedinačnim relacijama, zatim projekcije, pa tek onda spajanja, znatno efikasniji. O optimizaciji izvršenja upita više će govora biti kasnije. Gornji opći izraz za realizaciju upita u relacijskoj algebri dat je da prikaže sposobnosti ovog jezika i da omogući usporedba relacijske algebre sa relacijskim računom i nekim komercijalnim upitnim jezicima.

Redoslijed izvođenja relacijskih operacija utvrđuje se prioritetom relacijskih operatora. Složeni po prioritetu operatori relacijske algebre su (najprije se izvode operatori na vrhu popisa, a na kraju operatori na dnu popisa):

1. Projekcija
2. Selekcija
3. (Kartezijev) produkt
4. Spajanje (Join), Dijeljenje
5. Razlika
6. Unija, Presjek

#### 5.1.4. Operacije sa nula vrijednostima

Navedene operacije relacijske algebre nisu uzimale u obzir nula vrijednosti koje mogu postojati u relacijama. Podrazumijevalo se da se vrijednosti predikata izračunavaju na osnovu standardnih, dvovrijednosnih tablica istinitosti. Isto tako, bez dvoumljenja je bilo moguće odrediti duplike n-torki, odnosno kardinalnost pojedinih skupova. Pojavljivanje nula vrijednosti u relacijskoj bazi podataka zahtijeva da se proširi skup definiranih operacija koje bi na neki način uključile i nula vrijednosti.

Osnovne postavke za operacije sa nula vrijednostima su slijedeće:

1. Tablice istinitosti trovrijednosne logike:

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

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

NOT	
T	F
?	?
F	T

(? predstavlja **nula vrijednost**, a može se unutar tablica istinitosti čitati i kao "**možda**").

## 2. Izračunavanje aritmetičkih formula:

Neka **op** označava neki od aritmetičkih operatora (+, -, \*, /) i neka su **x** i **y** dvije numeričke vrijednosti.

Vrijednost aritmetičkog izraza "**x op y**" je po definiciji **nula vrijednost** ako bilo **x**, bilo **y**, bilo oba dobiju **nula vrijednost**.

## 3. Skupovi sa nula vrijednostima:

Pogledajmo kolekciju {1, 2, 3, ?}. Da li je ova kolekcija skup?

Ako ? poprimi vrijednost 1, 2 ili 3 onda nije, ili je možemo tretirati kao skup sa kardinalnošću 3.

Ako ? nije ni 1, ni 2, ni 3 onda gornja kolekcija predstavlja skup sa kardinalnošću 4.

To pokazuje da postoje različite nula vrijednosti, nula vrijednost koja nije 1, nula vrijednost koja nije 2, zatim nula vrijednost koja nije ni 1 ni 2 i tako dalje.

Operacije koje bi uzele u obzir različitost nula vrijednosti bile bi veoma složene. Zbog toga se operacije definiraju sa jednom vrstom nula vrijednosti, a ostale nula vrijednosti se tretiraju kao duplikati.

Imajući u vidu ove opće postavke navesti ćemo primjere nekih ranije definiranih operacija na relacijama koje sadrže nula vrijednosti:

### Unija:

R	S	T := R È S
A   B	A   B	A   B
a   b	x   y	a   b
a   ?	?   b	a   ?
?   b	?   ?	?   b
?   ?	?   ?	?   ?
		x   y

### Razlika:

A	B
a	b
a	?
?	b
?	?

A	B
x	y
?	b
?	?

A	B
a	b
?	?
a	?

### Kartezijski produkt:

Kartezijski produkt ostaje neizmijenjen

### Selekcija:

Operacija selekcije ostaje neizmijenjena, selektiraju (izabiru) se one n-torce za koje se odgovarajući predikat evaluira u T(true) na osnovu trovijednosnih tablica istinitosti. Ova operacija se često zove i "**TRUE SELECTION**" (*istinita selekcija*).

A	B
a	b
a	?
?	b
?	?

A	B
a	b
a	?
a	?

### Projekcija:

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

A	B
a	b
a	?
?	b
?	?

A
a
?

### Spajanje:

Operacija spajanja ostaje neizmijenjena, jer operacije Kartezijskog produkta i Selekcije ostaju neizmijenjene. U rezultatu se pojavljuju one n-torce za koje se predikat spajanja evaluira u T(true), na osnovu trovijednosnih tablica istinitosti (**TRUE TETA JOIN - istinito teta spajanje**).

## 5.1.4.1. Dodatne operacije sa nula vrijednostima

Zbog postojanja nula vrijednosti u bazi podataka, neophodno je da se definira logička funkcija "IS NULL" čiji argument može da bude neki skalarni izraz, a koja se evaluira u T(true) ako je vrijednost tog argumenta nula vrijednost, a inače uzima vrijednost F.

### Možda selekcija (MAYBE\_SELECT):

Selektiraju (izabiru) se one n-torce relacije za koje se predikat selekcije, na osnovu trovijednosnih tablica istinitosti, izračunava u nula vrijednost.

R	T := MAYBE_SELECT[A] R
A   B	A   B
a   b	?   b
a   ?	?   ?
?   b	?   ?
?   ?	?   ?

### Možda spajanje (MAYBE\_JOIN):

U rezultatu spajanja se pojavljuju one n-torce za koje se predikat spajanja izračunava u nula vrijednost, na osnovu trovijednosnih tablica istinitosti.

R	S	T := R [MAYBE_JOIN A = D] S
A   B	C   D   E	A   B   C   D   E
a1   b1	c1   ?   e1	a1   b1   c1   ?   e1
a2   b2	c2   d2   e2	a1   b1   c3   ?   e3
?   b3	c3   ?   e3	a2   b2   c1   ?   e1
		a2   b2   c3   ?   e3
		?   b3   c1   ?   e1
		?   b3   c2   d2   e2
		?   b3   c3   ?   e3

### Vanjsko spajanje (OUTER\_JOIN):

Neka su zadane relacije R(A,B) i S(C,D). Prepostavimo da se vrši operacija ekvispajanja ovih relacija,  $R[A = C]S$ . Ako je  $P[A] R^{-1} P[C] S$  (projekcije relacija po atributima spajanja su različite), tada će se u rezultatu spajanja "izgubiti" neke n-torce relacija R i S. Ako se takvo gubljenje informacija ne želi, VANJSKO\_SPAJANJE ih može sačuvati na taj način što se u rezultat dodaju i ove n-torce i to tako što za takve n-torce relacije R atributi C i D uzimaju nula vrijednosti, a za takve n-torce relacije S atributi A i B uzimaju nula vrijednosti.

R	S	T := R [OUTER_JOIN A = C] S
A   B	C   D	A   B   C   D

1	2
2	1
4	?

3	4
2	2

2	1	2	2
1	2	?	?
4	?	?	?
?	?	3	4

### Lijevo vanjska spajanje (LEFT\_OUTER\_JOIN):

Osnova za realizaciju lijeve vanjske veze je prirodno spajanje, kojem se dodaju oni elementi tablice, koja je u relacijskom izrazu sa lijeve strane, koji ne sudjeluju u vezi.

R	A	B	D
s	1	4	
d	1	5	
c	2	5	
s	3	3	

S	D	E	F
	5	1	5
	1	2	4
	3	6	7

$T := R [LEFT\_OUTER\_JOIN A = D] S$

S

A	B	D	E	F
d	1	5	1	5
c	2	5	1	5
s	3	3	6	7
s	1	4	?	?

U stvaranju  $R [LEFT\_OUTER\_JOIN A = D] S$  najprije se realizira prirodno spajanje  $R [A=C] S$  (time se dobiju prva tri reda tablice T). Potom se dodaju svi redovi tablice R, koji ne sudjeluju u prirodnom spoju. U ovom slučaju jedini red u tablici R koji nije obuhvaćen prirodnim spajanjem je prvi red, a budući on ne sudjeluje u prirodnom spoju vrijednosti atributa E i F su null vrijednosti.

### Desna vanjsko spajanje (RIGHT\_OUTER\_JOIN):

Osnova za realizaciju desne vanjske veze je prirodno spajanje, kojem se dodaju oni elementi tablice, koja je u relacijskom izrazu sa desne strane, koji ne sudjeluju u vezi.

R	A	B	D
s	1	4	
d	1	5	
c	2	5	
s	3	3	

S	D	E	F
	5	1	5
	1	2	4
	3	6	7

$T := R [RIGHT\_OUTER\_JOIN A = D] S$

S

A	B	D	E	F
d	1	5	1	5
c	2	5	1	5
s	3	3	6	7
?	?	1	2	4

U prethodnom primjeru objašnjeno je nastajanje tablice  $R >< S$ . U stvaranju  $R [RIGHT_OUTER_JOIN A = D] S$  najprije se realizira prirodno spajanje  $R [A=D] S$  (time se dobiju prva tri reda tablice T). Potom se dodaju svi redovi tablice S, koji ne sudjeluju u prirodnom spoju. U ovom slučaju jedini red u tablici S koji nije obuhvaćen prirodnim spojem je drugi red ( $D=1$ ), a budući on ne sudjeluje u prirodnom spoju vrijednosti atributa

A i B su null vrijednosti.

### Vanjska unija (OUTER\_UNION):

Kao što je rečeno, operacija unije se može izvesti samo nad relacijama koje zadovoljavaju kriterij kompatibilnosti (da su istog stupanja i da su im odgovarajući atributi definirani nad istim domenama). Dodavanjem novih atributa i postavljanjem njihovih vrijednosti na nula vrijednosti mogu se uvijek dvije nekompatibilne relacije učiniti kompatibilnim. Vanjska unija podrazumijeva da su, na ovaj način, dvije nekompatibilne relacije pretvorene u kompatibilne relacije, pa je nakon toga izvršena operacija unije.

R	S	T := R OUTER_UNION S
A a1 a2	B b1 b1	C c1 c2
a1	4	
a1	7	
a2	5	
		D
		a1
		b1
		c1
		?
		a2
		b1
		c2
		?
		a1
		?
		?
		4
		a1
		?
		?
		7
		a2
		?
		?
		5

## 5.2. Relacijski račun

Dok je relacijska algebra proceduralni, relacijski račun je neproceduralni jezik. Umjesto da programer sastavlja proceduru kojom će se dobiti željeni rezultat, relacijskim računom se samo formalno specificira željeni rezultat. Postoje dva oblika relacijskog računa:

- Relacijski račun n-torki i
- Relacijski račun domena.

### 5.2.1. Relacijski račun n-torki

Relacijski račun n-torki je predikatni račun prvog reda u kome su domene varijabli n-torce relacija zadane baze podataka.

Osnovni pojmovi predikatnog računa su:

1. **Afirmativna rečenica**, koja ima smisla i koja je istinita ili neistinita naziva se **sud**.
2. **Afirmativna rečenica** koja ima smisla i koja sadrži jedan ili više **varijabilnih parametara** i koja postaje **sud** uvijek kada parametri iz rečenice dobiju **konkretnu vrijednost** naziva se **predikat**. Broj parametara u predikatu se naziva dužina predikata. (Npr. predikat je  $x^2 + y^2 \leq 1$ ).
3. **Predikatni ili kvantifikatorski račun je matematička teorija čiji su objekti formule koje predstavljaju predikate. Simboli koji**

se koriste da označe neki sud nazivaju se **atomskim formulama ili atomima**. Atomi u relacijskom računu n-torki su:

- $x \hat{=} R$  gdje je  $x$  n-torka-varijabla, a  $R$  relacija, odnosno varijabla  $x$  uzima vrijednosti iz skupa n-torki relacije  $R$ ,
- $x.A Q y.B$  gdje su  $x$  i  $y$  varijable (n-torke),  $A$  i  $B$  su atributi relacija  $R$  i  $S$  iz čijih n-torki, respektivno, varijable  $x$  i  $y$  uzimaju vrijednosti ( $x \hat{=} R$ ,  $y \hat{=} S$ ), a  $Q$  je operacija uspoređivanja definirana nad domenama atributa  $A$  i  $B$  ( $A$  i  $B$  moraju biti definirani nad istom domenom) i
- $x.A Q c$  gdje su  $x$ ,  $A$  i  $Q$  imaju isto značenje kao i u prethodnoj stavci, a  $c$  je konstanta koja ima isti domenu kao i  $A$ .

Formule se formiraju od atoma preko slijedećih pravila:

- Atom je formula;
- Ako je  $P_1$  formula, tada su formule i  $\text{NOT } P_1$  i  $(P_1)$ ;
- Ako su  $P_1$  i  $P_2$  formule tada su formule i  $P_1 \text{ AND } P_2$  i  $P_1 \text{ OR } P_2$ ;
- Ako je  $P_1(s)$  formula koja sadrži neku slobodnu varijablu  $s$  tada su i  $\$ s (P_1(s))$  i  $" s (P_1(s))$  također formule ( $\$$  - "postoji", egzistencijalni kvantifikator,  $"$  - "za svako", univerzalni kvantifikator).

Jedna varijabla u nekoj formuli se može pojaviti više puta. *Varijable mogu biti "slobodne" i "vezane".*

Vezana varijabla u formuli je neka vrsta "prividne" (dummy) varijable koja ima ulogu da poveže varijablu iza kvantifikatora sa varijablama u zoni djelovanja kvantifikatora. Drugim riječima vezana varijabla  $u$  je  $(\$ u)$ ,  $(" u)$  ili  $(\$ u) A$  ili  $(" u) A$ , gdje je  $A$  formula u kojoj se pojavljuje varijabla  $u$ . Sve varijable koje u nekoj formuli nisu vezane, slobodne su u toj formuli. Na primjer, u formuli

$$\$ x (x > 3)$$

$x$  je vezana varijabla

$U$  formuli

$$\$ x (x > 3) \text{ AND } x < 0$$

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

$\$ y (y > 3) \text{ AND } x < 0$

Neka su **R, S, ..., R<sub>n</sub>** relacije u nekoj bazi podataka. Neka su **A, B, ..., C** atributi ovih relacija, respektivno i neka je **f** formula. Općeniti izraz relacijskog računa n-torki je tada:

$t \hat{\in} R, u \hat{\in} S, \dots, v \hat{\in} R_n$

$t.A, u.B, \dots, v.C \text{ GDJE\_JE } f$

(Prikazuju se vrijednosti atributa **A** relacije **R**, atribut **B** relacije **S**, ... i atribut **C** relacije **R<sub>n</sub>**, za one n-torce koje zadovoljavaju uvjet definiran formulom **f**).

Primjeri:

**Upit 1:** Pronađi sve brojeve kolegija.

{ c.ŠIF\_KOLEGIJA | KOLEGIJ(c) } .

**Upit 2:** Pronađi brojeve indeksa svih studenata na prvoj godini studija.

{ s.BR\_INDEX | STUDENT(s) and s.GOD\_STUDIJA = 1 } .

**Upit 3:** Pronađi brojeve i imena studenata koji su polagali ispit iz kolegij 121.

{ s.ŠIF\_KOLEGIJA, s.IME\_STUDENTA | STUDENT(s) and \$ r  
(POLAŽE(r)  
and r.BR\_INDEX = s.BR\_INDEX and r.ŠIF\_KOLEGIJA = 121)} .

## 5.2.2. Relacijski račun domena

U relacijskom računu domena varijable uzimaju vrijednosti iz nekih domena definirane relacijske baze podataka. Ovdje se, pored navedenih, definira još jedna atomska formula, tzv. *"uvjet članstva" (membership condition)*. Uvjet članstva ima oblik:

$R(\text{term}, \text{term}, \dots)$

gdje je **R** ime neke relacije a svaki **term** ima oblik **A: v**, gdje je **A** neki atribut relacije **R** a **v** je ili varijabla ili konstanta. Uvjet članstva se evaluira u T (true) ako postoji n-torka u relaciji **R** koja ima za zadane vrijednosti navedenih atributa. Na primjer, uvjet članstva

**STUDENT (STUDENT\_ID: '23456')**

se evaluira u T (true) samo ako postoji n-torka u relaciji STUDENT sa navedenom vrijednošću zadanog atributa.

Opći izraz relacijskog računa domena je:

$x, y, \dots, z \text{ GDJE\_JE } f$

gde su  $x, \dots, z$  varijable a  $f$  je formula koja uključuje i uvjet članstva.

Primjeri:

**Upit 1:** Pronađi sve brojeve kolegija.

$\{ C \mid KOLEGIJ(\text{ŠIF\_KOLEGIJA} : C) \}$ .

**Upit 2:** Pronađi brojeve svih studenata na prvoj godini studija.

$\{ S \mid STUDENT(BR\_INDEX : S, GOD\_STUDIJA : 1) \}$ .

### 5.3. Primjeri relacijske algebre i relacijskog računa

U ovom poglavlju, u primjerima ćemo koristiti fakultetsku bazu podataka, koja je opisana slijedećim relacijama (intenzijama):

**STUDENT** ( BR\_INDEX, IME\_STUDENTA, GOD\_STUDIJA ) ,

**KOLEGIJ** ( ŠIF\_KOLEGIJA, NAZ\_KOLEGIJA,  
PREZ\_NASTAVNIKA ) ,

**POLAŽE** ( BR\_INDEX, ŠIF\_KOLEGIJA, OCJENA ) ,

**NASTAVNIK** ( PREZ\_NASTAVNIKA, BR\_SOBE ) .

Ekstenzije gornjih relacija su:

**STUDENT**

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA
	20022	Marko	2
	10020	Branko	1
	30354	Neno	3
	20456	Zdravko	2

ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
110	Baze podataka	Perić
220	Programiranje za internet	Jurić
330	Numerička analiza	Franić
440	Programski prevodioci	Antić

**POLAŽE**

BR_INDEX	ŠIF_KOLEGIJA	OCJENA

**NASTAVNIK**

PREZ_NASTAVNIKA	BR_SOBE

20022	110	82
10020	110	75
10020	220	71
30354	440	49
20456	220	39
20022	330	70
10020	330	69
10020	440	78

Perić	581
Jurić	582
Antić	901
Franić	980

**Upit 1:** Pronađi sve studente na 1 godini studija

**RESULT1 := S[GOD\_STUDIJA = 1] STUDENT**

### STUDENT

BR_INDEX	IME_STUDENTA	GOD_STUDIJA
20022	Marko	2
10020	Branko	1
30354	Neno	3
20456	Zdravko	2

### S[GOD\_STUDIJA = 1] STUDENT

BR_INDEX	IME_STUDENTA	GOD_STUDIJA
10020	Branko	1

**Upit 2:** Pronađi sve studente koji su iz kolegija 110 dobili ocjenu veću od 70.

**RESULT2 := S [ (ŠIF\_KOLEGIJA=110) and  
(OCJENA>70)] POLAŽE**

### POLAŽE

BR_INDEX	ŠIF_KOLEGIJA	OCJENA
20022	110	82
10020	110	75
10020	220	71
30354	440	49
20456	220	39
20022	330	70
10020	330	69
10020	440	78

### S [ (ŠIF\_KOLEGIJA=110) and (OCJENA>70)] POLAŽE

BR_INDEX	ŠIF_KOLEGIJA	OCJENA
20022	110	82
10020	110	75

**Upit 3:** Pronađi ime nastavnika koji predaje kolegij 220.

**RESULT3 := P[PREZ\_NASTAVNIKA] ( S  
[ŠIF\_KOLEGIJA = 220] KOLEGIJ )**

### KOLEGIJ

ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA

110	Baze podataka	Perić
220	Programiranje za internet	Jurić
330	Numerička analiza	Franić
440	Programski prevodioci	Antić

RESULT3 := P[PREZ\_NASTAVNIKA] ( S [ŠIF\_KOLEGIJA = 220]  
KOLEGIJ )

PREZ_NASTAVNIKA
Jurić

**Upit 4:** Pronađi imena svih studenata koji su upisali kolegij 330.

**RESULT4 := P[IME\_STUDENTA] (S  
[ŠIF\_KOLEGIJA = 330] (POLAŽE [BR\_INDEX]  
STUDENT))**

#### POLAŽE [BR\_INDEX] STUDENT

BR_INDEX	ŠIF_KOLEGIJA	OCJENA
20022	110	82
10020	110	75
10020	220	71
30354	440	49
20456	220	39
20022	330	70
10020	330	69
10020	440	78

Upit 2: Pronadi broj sobe nastavnika koji predaje kolegij 351.

RESULT2 := ( (KOLEGIJ where ŠIF\_KOLEGIJA = 351) join NASTAVNIK  
)[BR\_SOBE] .

Upit 3: Pronadi imena nastavnika koji predaju kolegije koje je upisao bar jedan student na stupnju

(godini) 2.

RESULT3 := ( ( (STUDENT where GOD\_STUDIJA=2) join POLAŽE ) join KOLEGIJ )[PREZ\_NASTAVNIKA] .

U Tabelarnom prikazu 3.8 nalazi se detaljnije izvrednjavanje Upita 3.

STUDENT where GOD\_STUDIJA = 2

BR\_INDEX IME\_STUDENTA GOD\_STUDIJA

20022 Marko 2

20456 Zdravko 2

(STUDENT where GOD\_STUDIJA=2) join POLAŽE

BR\_INDEX IME\_STUDENTA GOD\_STUDIJA ŠIF\_KOLEGIJA OCJENA

20022 Marko 2 110 82

20022 Marko 2 330 70

20456 Zdravko 2 220 39

( (STUDENT where GOD\_STUDIJA=2) join POLAŽE ) join KOLEGIJ

BR\_INDEX IME\_STUDENTA GOD\_STUDIJA ŠIF\_KOLEGIJA OCJENA

NAZ\_KOLEGIJA PREZ\_NASTAVNIKA

20022 Marko 2 110 82 Baze podataka Perić

20022 Marko 2 330 70 Numerička analiza Franić

20456 Zdravko 2 220 39 Programiranje za internet Jurić

RESULT3

PREZ\_NASTAVNIKA

Perić

Franić

Jurić

Tabelarni prikaz 3.8: Izvrednjavanje Upita 3 s prirodnim spojem.

Prirodni spoj uvijek se može izraziti preko ostalih operatora. Na primjer, za relacije R(A,B,C) i

S(C,D) vrijedi:

R join S = ( (R times S) where RČ = SČ ) [A,B,C,D].

## 26 3. JEZICI ZA RELACIJSKE BAZE PODATAKA

### 3.1.6 Theta-spoj

Theta-spoj relacija R i S, pisano R join(A B) S, je skup onih n-torki Kartezijevog produkta R sa S

za koje je predikat R.A S.B istina. Simbol predstavlja jedan od operatora za usporedbu (=,>,<

, 6=,,). Theta-spoj se uvijek može izraziti pomoću Kartezijevog produkta i selekcije. Prirodni spoj

se može smatrati specijalnim slučajem theta-spoja.

### 3.1.7 Dijeljenje

Neka je R relacija stupnja n, a S relacija stupnja m, i neka se svi atributi od S pojavljuju i u R.

Rezultat dijeljenja R sa S, oznakom R divideby S, je skup svih (n . m)-torki  $hx, yi$  pojavljuju u R za sve m-torce  $hyi$  u S. Ovdje  $x$  i  $y$  predstavljaju grupu od jedne ili više

vrijednosti atributa.

U Tabelarnom prikazu 3.9 nalazi se najprije jedan apstraktni primjer dijeljenja. Dalje slijede primjeri

s našom fakultetskom bazom podataka.

R

BR\_INDEX ŠIF\_KOLEGIJA

s1 c1

s1 c2

s1 c3

s2 c1

s2 c2

s3 c1

s4 c4

S

ŠIF\_KOLEGIJA

c1

R3

ŠIF\_KOLEGIJA

c1

c2

c3

R divideby S

BR\_INDEX

s1

s2

s3

R divideby R3

BR\_INDEX

s1

Tabelarni prikaz 3.9: Apstraktni primjer djeljenja.

Upit 1: Pronadi imena studenata koji su upisali sve kolegije.

RESULT1 := ( (POLAŽE[BR\_INDEX,ŠIF\_KOLEGIJA] divideby KOLEGIJ[ŠIF\_KOLEGIJA]) join STUDENT ) [IME\_STUDENTA].

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

Upit 2: Pronadi brojeve onih studenata koji su upisali barem one kolegije koje je upisao student s

brojem 30354 (i možda još neke kolegije).

RESULT2 := POLAŽE [BR\_INDEX,ŠIF\_KOLEGIJA] divideby (POLAŽE where BR\_INDEX=30354)[ŠIF\_KOLEGIJA] .

Kao odgovor dobivamo brojeve studenata 30354, 10020.

Operator dijeljenja predstavlja način implementiranja univerzalne kvantifikacije 8 u relacijskoj algebri.

Dijeljenje se također može izraziti pomoću prethodno opisanih operatora.

Na primjer, ako imamo

relacije R(A,B,C,D) i S(C,D), tada je:

R divideby S = R[A,B] minus ( (R[A,B] times S) minus R )[A,B] .

### 3.1.8 Vanjski spoj

Vanjski spoj (outer join) je binarni operator vrlo sličan prirodnom spoju.

Primjenjiv je pod istim

uvjetima i daje kao rezultat relaciju s istom gradom (shemom). No sadržaj od R outerjoin S je nešto

bogatiji nego sadržaj R join S. Naime, pored svih n-torki iz R join S, relacija R outerjoin S sadrži i

sve "nesparene" (nespojene) n-torce iz R odnosno S (s time da su te nesparene n-torce na odgovarajući

način proširene null vrijednostima).

## 3.2. RELACIJSKI RAČUN 27

Dakle za isti primjer R i S kao u Tabelarnom prikazu 3.7, R outerjoin S izgleda kao na Tabelarnom

prikazu 3.10.

R

A B C

a1 b1 c1

a2 b1 c1

a3 b2 c2

a4 b2 c3

S

B C D

b1 c1 d1

b1 c1 d2

b2 c3 d3

R outerjoin S

A B C D

a1 b1 c1 d1

a1 b1 c1 d2

a2 b1 c1 d1

a2 b1 c1 d2

a4 b2 c3 d3

a3 b2 c2 .

Tabelarni prikaz 3.10: Primjer vanjskog spoja.

Vanjski spoj obično se koristi za traženje podataka koji ne zadovoljavaju neki uvjet. Na primjer:

Upit 1: Pronadi imena studenata koji nisu upisali ni jedan kolegij.

RESULT1 := ( (STUDENT outerjoin POLAŽE) where (ŠIF\_KOLEGIJA is null) ) [IME\_STUDENTA] .

### 3.2 Relacijski račun

Relacijski račun također je bio predložen od E.F. Codd-a, kao alternativa relacijskoj algebri. Riječ je

o matematičkoj notaciji zasnovanoj na predikatnom računu. Upit se izražava tako da zadamo predikat kojeg n-torce moraju zadovoljavati.

Postoje dvije vrste relacijskog računa: račun orijentiran na n-torce (gdje su osnovni objekti n-torce)

i račun orijentiran na domene (gdje su osnovni objekti vrijednosti iz domena za atribute).

### 3.2.1 Račun orijentiran na n-torce

U izrazima se pojavljuju sljedeći elementi:

Varijable-n-torce poprimaju vrijednosti iz imenovane relacije. Ako je t varijabla koja "prolazi"

relacijom R, tada t.A označava A-komponentu od t, gdje je A atribut od R.

Uvjeti oblika  $x \ y$ , gdje je operator za uspoređivanje ( $=, <, >, 6=, ,$ ). Bar jedno od x i y mora

biti oblika t.A, a drugo može biti konstanta. Uvjeti također mogu biti oblika  $R(t)$ , što znači da

je t n-torka u relaciji R.

Dobro oblikovane formule (WFF) gradene od logičkih veznika (and, or, not) i kvantifikatora 9

(postoji) i 8 (za svako), u skladu sa sljedećim pravilima:

- . Svaki uvjet je WFF.
- . Ako su f1 i f2 WFF, tada su to i f1 and f2, f1 or f2, not f1, i not f2.
- . Ako je f jedna WFF u kojoj se t pojavljuje kao slobodna varijabla, tada su 9 t (f) i

8 t (f) također WFF. (Pojava varijable u WFF je vezana ukoliko je ta varijabla uvedena

kvantifikatorom. Inače je slobodna).

- . Ništa drugo nije WFF.

U ovom poglavlju, u primjerima ćemo koristiti fakultetsku bazu podataka, koja je opisana sljedećim relacijama (intenzijama):

**STUDENT** ( BR\_INDEX, IME\_STUDENTA, GOD\_STUDIJA ) ,

**KOLEGIJ** ( ŠIF\_KOLEGIJA, NAZ\_KOLEGIJA,  
PREZ\_NASTAVNIKA ) ,

**POLAŽE** ( BR\_INDEX, ŠIF\_KOLEGIJA, OCJENA ) ,

**NASTAVNIK** ( PREZ\_NASTAVNIKA, BR\_SOBE ) .

Ekstenzije gornjih relacija su:

**STUDENT**

BR_INDEX	IME_STUDENTA	GOD_STUDIJA
20022	Marko	2
10020	Branko	1
30354	Neno	3
20456	Zdravko	2

**KOLEGIJ**

ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
110	Baze podataka	Perić
220	Programiranje za internet	Jurić
330	Numerička analiza	Franić
440	Programski prevodioci	Antić

**POLAŽE**

BR_INDEX	ŠIF_KOLEGIJA	OCJENA
20022	110	82
10020	110	75
10020	220	71
30354	440	49
20456	220	39
20022	330	70
10020	330	69
10020	440	78

**NASTAVNIK**

PREZ_NASTAVNIKA	BR_SOBE
Perić	581
Jurić	582
Antić	901
Franić	980

Relacija **STUDENT** popisuje studente, **KOLEGIJ** nabraja kolegije, a **NASTAVNIK** sadrži podatke o nastavnicima. Student je jednoznačno određen svojim brojem indeksa (atribut **BR\_INDEX**), kolegij je jednoznačno određen svojom šifrom (atribut **ŠIF\_KOLEGIJA**), a nastavnici se razlikuju po svojim prezimenima (atribut **PREZ\_NASTAVNIKA**). Za svakog studenta pamtimo njegovo ime (atribut **IME\_STUDENTA**) i godinu studija (atribut **GOD\_STUDIJA**). Za kolegij se pamti njegov naziv (atribut **NAZ\_KOLEGIJA**) i prezime nastavnika koji ga predaje (atribut **PREZ\_NASTAVNIKA**). Jedan kolegij predaje samo jedan nastavnik. Za svakog nastavnika poznat nam je broj njegove sobe (atribut **BR\_SOBE**). Relacija **POLAŽE** bilježi koji je student polagao koji kolegij i koju ocjenu je dobio (atribut **OCJENA**).

## 6. SQL (Structured Query Language)

SQL jezik (Structured Query Language – strukturni jezik za pretraživanje) jest srce relacijske tehnologije. Razvoj SQL-a kao jezika za manipulaciju podacima tekao je usporedo s razvojem relacijskog modela podataka, osnove kojeg je postavio 1971. godine E.F.Codd. Kada je Codd postavio svoj koncept relacijskog modela podataka, ustvrdio je da "... usvajanje relacijskog modela podataka... dopušta razvoj univerzalnog jezika podataka baziranog na primjeni relacijske algebре". Iako je odmah uočio zahtjeve koje bi takav jezik trebao ispunjavati, kao i njegove prednosti pri manipulaciji podacima, tada nije pokušao razviti takav jezik u praksi.

S obzirom na brzinu kojom se prihvaćaju nove ideje u informatici, razvoj SQL-a tekao je razmjerno sporo. Tek 1974. godine pojavljuje se članak autora D.D. Chamberlaina i R.F. Boycea u kojem oni opisuju strukturni jezik za pretraživanje podataka, nazvan SEQUEL (Structured English Query Language). Godine 1975. Boyce, Chamberlain i Hammer predstavljaju koncept jezika SQUARE koji je koristio matematičke izraze, za razliku od engleskih termina koje je koristio SEQUEL. Oba jezika su bila relacijski potpuna prema kriteriju koji je postavio E.F.Codd. Uskoro SQUARE mijenja ime u SEQUEL2, i taj jezik je korišten u razvoju prvog prototipa relacijskog sustava za upravljanje bazama podataka, nazvanog "SystemR", razvijenog u IBM-u. Kasnije jezik mijenja ime u "SQL".

SQL je 1986 godine postao američki nacionalni standard (ANSI X3:135 - 1986). Danas se SQL jezik koristi u velikoj većini relacionih baza podataka. Svi ovi sustavi za upravljanje bazama podataka, kako oni najmanji na PC platformama, tako i veliki client-server sustavi, nastoje što je moguće više slijediti originalni standard SQL-a, ali ga i obogaćuju raznim dodatnim opcijama.

SQL je homogeno relacioni jezik, što znači da može da se koristi i interaktivno, sa terminala i u okviru standardnih programskih jezika, za komunikaciju sa bazom podataka. Pored konstrukcija analognih relacionih algebri ili relacionom računu (koje su osnove jezika za rukovanje podacima) SQL sadrži i konstrukcije za opisivanje baze podataka (jezik za opis podataka), operacije koje omogućuju vezivanje SQL-a sa nekim standardnim jezikom (CURSOR operacije), kao i kontrolne konstrukcije za upravljanje konkurentnim obradama, oporavak baze podataka, definiranje zaštite podataka i integriteta baze podataka.

### 6.1. Definiranje baze podataka primjenom SQL (DATA DEFINITION LANGUAGE)

Do sada smo prešli niz etapa na putu početne ideje do implementacije i korištenja podataka iz baze podataka. U realnom svijetu uočili smo problem koji bi se uspješno mogao riješiti primjenom računalne tehnologije. Uočili smo objekte(entitete) i veze među njima, odnosno izgradili smo model podataka. Model podataka zapisan u ER-dijagramu (i popratnoj dokumentaciji) pretvorili smo u relacijski model. Proveli smo normalizaciju relacijskog modela i otklonili sve nepotrebne redundancije podataka. Iza tog našeg rada ostala je opsežna dokumentacija i znanje o problemu koji pokušavamo riješiti.

Spremni smo za slijedeći korak: kreiranje baze podataka na računalu.

Relacije do kojih smo došli na kraju prethodne fazi našeg puta mogu se prikazati kao tablice. Za korisnika ili SQL programera, baza podataka je kolekcija jedne ili više tablica.

Mnogi DBMS sustavi imaju grafički podsistem za dodavanje, mijenjanje i brisanje tablica iz baze podataka, te postavljanje ograničenja na vrijednosti u stupcima i tablici općenito. Ovdje ćemo pokazati kako ove akcije možemo obaviti primjenom SQL-a:

- naredba CREATE DATABASE kreira novu bazu podataka
- naredba DATABASE omogućava pristup postojećoj bazi podataka
- naredba DROP DATABASE briše postojeću bazu podataka
- naredba CREATE TABLE stvara novu tablicu.
- naredba ALTER TABLE mijenja strukturu postojeće tablice.
- naredba DROP TABLE briše (uništava) postojeću tablicu i sve podatke u njoj.
- naredba CREATE TEMPORARY TABLE kreira novu tablicu koju DBMS uništava automatski kada više nije potrebna
- naredba CREATE TABLE AS kreira novu tablicu na temelju neke već postojeće tablice.

Ove naredbe ne vraćaju nikakav rezultat, ali DBMS sustav može prikazati poruku kojom nas obaveještava da li je naredba uspješno izvršena.

Obzirom da ove naredbe mijenjaju objekte baze podataka i same podatke, administrator baze podataka morati će nas ovlastiti za izvođenje ovih naredbi.

### 6.1.1. Kreiranje tablica

Da bi smo kreirali tablicu u bazi podataka moramo odrediti:

- naziv tablice
- nazive stupaca
- tipove podataka u stupcima (character, numeric, datetime, ili neki drugi tip podataka)
- inicijalne (predodređene) vrijednosti za podatke u stupcima (predodređene vrijednosti primjenjuju se ako kod upisa novog retka u bazu podataka neki od podataka nije zadan)
- postaviti ograničenja (ključevi, dozvola NULL vrijednosti, dozvoljene vrijednosti)

Novu tablicu kreiramo koristeći naredbu **CREATE TABLE**. Sintaksa naredbe CREATE TABLE je::

```
CREATE TABLE tablica
(
    stupac1 tip_podataka1 [svojstvo1] [ograničenja_stupca1],
    stupac2 tip_podataka2 [svojstvo2] [ograničenja_stupca2],
    ...
    stupacN tip_podatakaN [svojstvoN] [ograničenja_stupcaN]
    [, ograničenje_tablice1]
    [, ograničenje_tablice2]
    ...
    [, ograničenje_tabliceM]
);
```

gdje je:

- **tablica** naziv nove tablice koju želimo kreirati.
- **stupac1, stupac2,..., stupacN** su nazivi stupaca u tablici.

Tablica mora sadržavati barem jedan stupac.

- *tip\_podataka1, tip\_podataka2,..., tip\_podatakaN* označavaju tipove podataka koji se spremaju u svakom stupcu. Uz tip podataka navodi se dužina, skala i preciznost gdje je to potrebno.

Unutar jednog DBMS sustava ime baze podataka mora biti jedinstveno. Unutar baze podataka ime tablice mora biti jedinstveno, a ime stupca mora biti jedinstveno na nivou tablice. Ime baze podataka, tablice, stupca i ograničenja moraju biti valjani SQL identifikatori, tj moraju počinjati sa slovom ili podcrtom (\_) iza kojih slijedi proizvoljan broj znakova, slova i podcrtica. SQL je "neosjetljiv" na mala i velika slova. Identifikatori mojaBaza i MOJABAZA su za SQL isti identifikatori.

Svaki stupac definira se tako da se navede njegov naziv, tip podataka i da se optionalno navedu ograničenja koja vrijede za podatke u tom stupcu.

#### Napomene!

- Ako pokušamo kreirati tablicu sa postojećim imenom dobiti ćemo poruku o greški. Da bi nas spriječio da uništimo podatke SQL prije kreiranja tablice zahtjeva da naredbom DROP TABLE obrišemo postojeći tablicu sa istim imenom..
- Novo kreirana tablica je prazna (ima nula redaka).
- Standardno se uzima da su dozvoljene NULL vrijednosti u stupcima.

### 6.1.1.1. Tipovi podataka

Prilikom definiranja stupaca u tablici potrebno je definirati tip podataka za svaki stupac, kao ključni segment određivanja domene nad kojom su definirani pojedini podaci. Tipovi podataka (format) mogu se podijeliti u 4 ključne kategorije

- znakovni tipovi podataka
- binarni podaci
- numerički tipovi podataka
- podaci tipa datum-vrijeme

### Znakovni tip podataka

#### CHARACTER

*Char (n)* – string dužine točno n znakova. Svaki znak prikazuje se jednim bajtom. Znakovi mogu biti alfanumerički i specijalni. Dužina tipa *char* je uvijek n bajtova, bez obzira na veličinu podatka koji se unosi, a maksimalan broj znakova je 255. Ako je stvarna dužina unesenog stringa manja od n, sustav dodaje prazne znakove do dužine n.

Npr. Ako je n=6, a uneseni string 'ABC' , on se u memoriji pohranjuje kao 'ABC####', tj. uvijek fiksne dužine n=6.

Tip podatka char pridjeljuje se alfanumeričkim atributima, za koje se pretpostavlja da su približno iste dužine.

#### CHARACTER VARYING

*Varchar(n)* – string promjenjive dužine, maksimalno n karaktera. Pri tome maksimalna dužina zavisi od mogućnosti procesora i operativnog sustava, a najčešće iznosi 255 znakova. Pri unosu podataka, dužina stringa nije fiksna već je određena veličinom

podatka koji se unosi.

Npr. ako je n=25, a uneseni podatak 'Ivo', ovaj podatak pamti se u dužini od 4 bajta kao 'Ivo3', pri čemu zadnji bajt 3 označava stvarnu dužinu stringa.

Tip podatka varchar pridjeljuje se alfanumeričkim atributima promjenjive dužine.

## TEXT

*Text* - string neograničene veličine.

## Binarni podaci

### BIT

*Bit* – podatak koji predstavlja binarnu vrijednost (1 ili 0). Veličina rezerviranog memorijskog prostora je 1 bajt. Ukoliko se bit tipu podataka pokuša pridijeliti cijelobrojna vrijednost, ona se interpretira kao 1.

### BINARY(n)

*Binary(n)* - Sadržava maksimalno 255 bajta binarnih podataka, te se koristi za spremanje binarnih i heksadecimalnih vrijednosti

## Numerički tipovi podataka

### INTEGER

*int* - cijelobrojni format s predznakom ili bez njega. Zauzima 4 bajta i sadržava cijele brojeve od  $-2^{31}$  (-2,147,483,648) do  $(2^{31} - 1)$  (2,147,483,647).

### SMALL INTEGER

*smallint* - cijelobrojni format s predznakom ili bez njega. Zauzima 2 bajta i sadržava cijele brojeve od  $-2^{15}$  (-32,768) do  $(2^{15}-1)$  (32,767).

### TINY INTEGER (BYTE)

*tinyint* - cijelobrojni format koji prikazuje brojeve od 0 do 255. Zauzima 1 bajt.

### NUMERIC

*numeric(p,q)* - decimalni broj, ukupno p znamenki i predznak, q znamenki iza decimalnog zareza.

### DECIMAL

*decimal(p,q)* - decimalni broj, koji se interno bilježi sa m znamenki, pri čemu vrijedi  $0 < p < q < m$ .

Npr. decimal (7,2) dopušta unos broja koji ima točno dvije znamenke iza decimalnog zareza, a ukupan broj znamenaka mu je najmanje 7. Stvarni broj znamenaka koji se rezervira za interni prikaz u bazi zavisi od operacijskog sustava.

### FLOAT

*float(n)* - realni broj u formatu pomičnog zareza (floating point). Zauzima 4 bajta memorije.

### DOUBLE

**double(n)** - realni broj dvostrukog preciznosti u formatu pomicnog zareza (floating point). Zauzima 8 bajta memorije.

## Podaci tipa datum-vrijeme

Datum i vrijeme sastavljeni su od alfanumeričkih podataka koji predstavljaju podatke o datumu i vremenu. Uobičajeni format prikaza je "dan-mjesec-godina sat:minut: sekunda AM"

### DATETIME

**datetime** - je tip podatka koji je predstavljen sa 8 bajtova ili dva 4-bajtna cjela broja: 4 bajta za prikaz datuma i 4 bajta za prikaz vremena (sati). Moguće je prikazati datume od 1. siječnja 1753 do 31. prosinca 9999, sa točnošću od 3.33 milisekunde.

### SMALLDATETIME

**smalldatetime** - skraćeni spremanje podataka o datumu i vremenu koji je predstavljen sa 4 bajta: 2 bajta za prikaz datuma i 2 bajta za prikaz vremena (sati). Moguće je prikazati datume od 1. siječnja 1900 do 6. lipnja 2079, sa točnošću od minute.

### 6.1.1.2. Što su ograničenja (Constraints)

Ograničenja su pravila koja određuju koje su vrijednosti dozvoljene u nekom stupcu. Ograničenja omogućavaju DBMS da automatski očuva integritet podataka u bazi podataka..

Ograničenja se mogu postaviti na dva načina, kao:

- ograničenja stupca, ako se pravila primjenjuju samo na vrijednosti zapisane u tom stupcu
- ograničenja tablice, ako se ograničenje proteže na vrijednosti zapisane u više stupaca.

Tipovi ograničenja su:

Ograničenje (CONSTRAINT)	Opis
NOT NULL	zabranju se pojava NULL vrijednosti u stupcu
PRIMARY KEY	označava da stupac(stupci) sadrži PRIMARNI KLJUČ tablice
FOREIGN KEY	označava da stupac(stupci) sadrži STRANI KLJUČ neke druge tablice
UNIQUE	Zabranjuje ubacivanje istih vrijednosti (duplicata) u stupcu
CHECK	ograničava skup vrijednosti koje se mogu upisati u stupcu primjenom logičkih izraza

Neka ograničenja mogu se zadati ili kao ograničenja stupca ili kao ograničenja tablice. Npr ako primarni ključ čini samo jedan stupac ograničenje se može upisati ili kao ograničenje stupca ili kao ograničenje tablice. Ako pak primarni ključ čini više stupaca tada se ograničenje mora upisati kao ograničenje tablice. Ograničenja tablice zapisuju se nakon definicije svih polja u tablici.

Opcionalno se svakom ograničenju može pridijeliti ime. Imenovanje ograničenja omogućava nam efikasnije održavanje ograničenja.

Primjenom odgovarajućih naredbi uz navođenje imena ograničenja možemo to ograničenje promijeniti ili obrisati. Dobra je praksa da se svako ograničenje imenuje (izuzetak je NOT NULL ograničenje koje se obično ostavlja bez imena). Ako pak ne imenujemo ograničenja, DBMS sustav imenovati će ih sam, automatski. Imena će se sastojati od slučajne kombinacije znakova i brojeva. Imena ograničenja mogu se pojaviti u porukama o greškama i upozorenjima. Tada će za razumijevanje poruke, biti znatno ugodnije pročitati neki suvisli tekst umjesto slučajne kombinacije znakova i brojeva.

Imenovanje ograničenja obavlja se tako da se ispred ograničenja ubaci:

```
CONSTRAINT ime_ograničenja
```

s tim da je *ime\_ograničenja* valjni SQL identifikator. Unutar jedne tablice imena ograničenja moraju biti jedinstvena.

#### 6.1.1.3. Svojstvo IDENTITY[(seed, increment)]

Generira inkrementalnu vrijednost za svaki novi redak podataka u tablici na osnovu *seed* i *increment* parametara. Ako je navedena, *seed* vrijednost označava početak brojača i bit će dodijeljena prvom retku u tablici, a svaki slijedeći redak dobit će jedinstvenu *identity* vrijednost, koja je jednaka posljednjoj *identity* vrijednosti uvećanoj za vrijednost parametra *increment*. Ako vrijednosti *seed* i *increment* nisu navedene smatra se da je njihova vrijednost 1. Pridjeljivanje *IDENTITY* svojstva koloni podrazumijeva njezino svojstvo NOT NULL. Samo jedan stupac u tablici može biti definiran sa svojstvom *identity*. Svojstvo *identity* može se dodijeliti onom stupcu koji je definiran kao numerički tip podataka.

#### 6.1.1.4. Zabranjivanje NULL vrijednosti

Pojava NULL vrijednosti u stupcu znači da podatak trenutno nije raspoloživ(poznat) ili je opcionalan. Vrijednost NULL nije isto što i nula(0), prazno ili prazni string(""). NULL vrijednost može biti upisana u bilo koji stupac koji dozvoljava NULL vrijednosti, bez obzira koji je tip podataka vezan uz taj stupac.

Za ograničenje kojim se zabranjuje pojava NULL vrijednosti u stupcu koriste se ključne riječi *NOT NULL*. Vrijedi:

- NOT NULL ograničenje je uvijek ograničenja stupca
- Standardno se uzima da su dozvoljene NULL vrijednosti u stpcima.
- Kad god je to moguće treba izbjegavati pojavu NULL vrijednosti jer one otežavaju upite, dodavanja i izmjene podataka
- U stpcima koji su dio primarnog ključa ne smije se dozvoliti pojava NULL vrijednosti.
- U stpcima koji su dio stranog ključa (iz neke druge tablice) pojava NULL vrijednosti je dozvoljena ali utječe na provjeru referencijalnog integriteta.
- Kod dodavanja podataka u neku tablicu (upisivanja novog retka) za stupce u kojima nije dozvoljena pojava NULL vrijednosti obavezno se mora navesti vrijednost podatka.

Sintaksa *NOT NULL* ograničenja je:

```
[CONSTRAINT ime_ograničenja]  
NOT NULL
```

Većina DBMS sustava omogućava da se upotrijebi konstrukcija NULL (bez NOT) kojom se dozvoljava pojava NULL vrijednosti unutar nekog stupca.

#### 6.1.1.5. Predodređene vrijednosti

Da bi odredili koje će vrijednosti biti upisane u stupcu kada pri dodavanju novog retka izostavimo podatak za taj stupac koristimo ograničenje DEFAULT. Vrijedi:

- DEFAULT ograničenje je uvijek ograničenja stupca
- Predodređena vrijednost može biti bilo koji izraz čiji je rezultat konstanta.
- Vrijednost upisana iza DEFAULT mora biti istog tipa kao i stupac ili se u taj tip može pretvoriti.
- Ako se dodaje nove redak i podatak za neki stupac nije naveden (a postoji predodređena vrijednost) upotrijebiti će se predodređena vrijednost.
- Za stupce za koje nije upisano ograničenje DEFAULT predodređena je vrijednost NULL

Sintaksa **DEFAULT** ograničenja je:

**DEFAULT** *expr*

gdje je *expr* izraz koji se evoluira u konstantu.

#### 6.1.1.6. Označavanje primarnog ključa tablice

Primarni ključ čine podaci zapisani u jednom ili više stupaca koji jednoznačno određuju svaki redak u tablici. Dva retka u tablici ne smiju imati istu vrijednost primarnog ključa. U stupcima koji čine primarni ključ ne smiju biti dozvoljene NULL vrijednosti. Svaka tablica ima samo samo jedan primarni ključ.

Za ograničenje kojim se za jedan ili više stupaca ustanavljava da tvore primarni ključ koriste se ključne riječi **PRIMARY KEY**. Vrijedi:

- Ako primarni ključ čini jedan stupac ograničenje se navodi kao ograničenje stupca ili ograničenje tablice. Ako primarni ključ čini više stupaca ograničenje se obavezno navodi kao ograničenje tablice.
- Kad se ograničenje navodi kao ograničenje tablice moraju se navesti imena svih stupaca koji čine primarni ključ. Ako se upisuje kao ograničenje stupca podrazumijeva se da je primarni ključ stupac uz koji se ograničenje navodi.
- SQL će dozvoliti kreiranje tablice bez primarnog ključa (iako je to u suprotnosti sa relacijskim modelom). Ipak preporučuje se da se za svaku tablicu odredi primarni ključ.
- U jednoj tablici dozvoljeno je samo jedno ograničenje PRIMARY KEY.
- Vrijednosti u stupcima koji čine primarni ključ uglavnom se ne mijenjaju se nakon što su jednom upisane.
- DBMS će kreirati indeks po primarnom ključu automatski.

Sintaksa ograničenja **PRIMARY KEY** je:

- ako se primarni ključ specificira kao ograničenje stupca uz definiciju stupca dodati

```
[CONSTRAINT ime_ograničenja]
PRIMARY KEY
ili
• ako se primarni ključ specificira kao ograničenje
```

```
[CONSTRAINT ime_ograničenja]
PRIMARY KEY (ime_stupca1 [, ime_stupcaN ])
```

gdje su

*ime\_stupca1, …, ime\_stupcaN* imena za stupce koji tvore primarni ključ

#### 6.1.1.7. Označavanje stranog ključa u tablici

Mehanizmom stranih ključeva vrši se povezivanje tablica. Strani ključ u tekućoj tablici je primarni ključ iz neke druge tablice (ili kandidat za ključ te druge tablice). Preko stranog ključa neki redak tablice povezuje se (referencira) sa podacima u drugim tablicama. Za razliku od stupca (ili stupaca) koji tvore primarni ključ, u stupcu (odnosno stupcima) koji tvore strani ključ dozvoljene su NULL vrijednost. Svaka tablica ne mora sadržavati strani ključ. Vrijednost stranih ključeva u nekoj tablici ne mora biti jedinstvena.

Za ograničenje kojim se za jedan ili više stupaca ustanavljava da tvore strani ključ koriste se ključne riječi FOREIGN KEY ili REFERENCES.

Vrijedi:

- Ako strani ključ čini jedan stupac ograničenje se navodi kao ograničenje stupca ili ograničenje tablice. Ako strani ključ čini više stupaca ograničenje se obavezno navodi kao ograničenje tablice.
- Kad se ograničenje navodi kao ograničenje tablice moraju se navesti imena svih stupaca koji čine strani ključ.
- U jednoj tablici ne mora biti ni jedan strani ključ a može ih biti više.
- Stupci koji čine strani ključ mogu imati isto ime kao u stupci u drugoj tablici sa kojom nas taj strani ključ povezuje.
- Vrijednost stranog ključa mora biti istog tipa kao vrijednost odgovarajućih stupaca u drugoj tablici ili se u taj tip može pretvoriti.
- Stupci koji su označeni kao FOREIGN KEY ne moraju referencirati samo stupce koju su označeni kao PRIMARY KEY u drugoj tablici; mogu referencirati i stupce koji su označeni kao UNIQUE u drugoj tablici.
- Broj stupaca i tip podataka u svakom stupcu koji se navodi u FOREIGN KEY ograničenju mora se poklapati sa onima navedenim u REFERENCES dijelu.

Ako stupci koji su označeni sa FOREIGN KEY referenciraju stupce koji su označeni kao PRIMARY KEY u drugoj tablici, DBMS sustav vrši sljedeće provjere referencijskog integriteta::

- *kod dodavanja retka u tablicu koja sadrži strani ključ*: provjerava se da li strani ključ ima istu vrijednost kao i primarni ključ u drugoj (nadređenoj, roditeljskoj) tablici. Ako u nadređenoj tablici ne postoji primarni ključ sa odgovarajućom vrijednosti ne dozvoljava se ubacivanje novog retka.

- **kod izmjene podataka u retku tablice koja sadrži strani ključ:** provjerava se da li izmijenjena vrijednost stranog ključa ima istu vrijednost kao i primarni ključ u drugoj (nadređenoj, roditeljskoj) tablici. Ako u nadređenoj tablici ne postoji primarni ključ sa odgovarajućom vrijednosti ne dozvoljava se izmjena retka.
- **kod brisanja retka iz tablicu koja sadrži strani ključ:** provjera referencijalnog integriteta nije nužna.
- **kod dodavanja retka u nadređenu tablicu:** provjera referencijalnog integriteta nije nužna.
- **kod izmjene podataka u retku nadređene tablicu:** provjerava da ne postoji ni jedna vrijednost stranog ključa u podređenoj tablici koja odgovara primarnom ključu koji se mijenja. Ako postoji barem jedna vrijednost stranog ključa u podređenoj tablici izmjena se ne dozvoljava.
- **kod brisanja retka u nadređenoj tablici:** provjerava da ne postoji ni jedna vrijednost stranog ključa u podređenoj tablici koja odgovara primarnom ključu koji se briše. Ako postoji barem jedna vrijednost stranog ključa u podređenoj tablici brisanje se dozvoljava.

Ako strani ključ sadrži NULL vrijednosti ne vrši se provjera referencijalnog integriteta.

Sintaksa ograničenja **FOREIGN KEY** je:

- ako se strani ključ specificira kao ograničenje stupca uz definiciju stupca dodati

```
[CONSTRAINT ime_ograničenja]
```

```
    REFERENCES ref_tablica(ref_stupac)
```

ili

- ako se primarni ključ specificira kao ograničenje tablice dodati

```
[CONSTRAINT ime_ograničenja]
```

```
    FOREIGN KEY (stupac_ključa1 [, stupac_ključaN ])
```

```
    REFERENCES ref_tablica(ref_stupac1 [, ref_stupacN ])
```

gdje su

- **stupac\_ključa1, ..., stupac\_ključaN** imena za stupce koji tvore strani ključ,
- **ref\_tablica** ime nadređene tablice sa kojom se povezujemo preko stranog ključa,
- **ref\_stupac1, ..., ref\_stupacN** imena stupaca u nadređenoj tablici preko kojih se uspostavlja veza.

#### **6.1.1.8. Zabrana ponavljanja istih vrijednosti**

Ograničenje UNIQUE zabranjuje da se u jednom stupcu (ili u kombinaciji sastavljenoj više stupaca) ista vrijednost pojavi više puta. Ograničenje UNIQUE slično je kao i ograničenje PRIMARY KEY, s tom razlikom da se u UNIQUE stupcima mogu pojaviti NULL vrijednosti i da grupa označenih kao UNIQUE u jednoj tablici može biti više.

Za ograničenje kojim se za jedan ili više stupaca traži da imaju jedinstvene vrijednosti koristi se ključna riječ **UNIQUE**. Vrijedi:

- Vrijednost upisana u jednom retku za grupa označenu sa UNIQUE ne smije se ponoviti ni u jednom drugom retku.  
Vrijednosti u pojedinim stupcu koji su zajedno označeni kao

grupa UNIQUE mogu se ponavljati.

- Ako je sa UNIQUE označen samo jedan stupac takvo se ograničenje može upisati ili kao ograničenje stupca ili kao ograničenje tablice. Ako je sa UNIQUE označena grupa stupaca tada se takvo ograničenje mora upisati samo kao ograničenje tablice.
- Ako se navodi kao ograničenje tablice moraju se navesti imena svih stupaca na koje se ograničenje odnosi. Ako se navodi kao ograničenje stupca podrazumijeva se da se ograničenje odnosi na stupac u kojem je definirano ograničenje.
- U jednoj tablici ne mora biti ni jedno ograničenje tip UNIQUE a može ih biti više.
- SQL sustav automatski stvara jedinstveni(UNIQUE) indeks za stupac ograničen sa UNIQUE.

Sintaksa **UNIQUE** ograničenja je:

- ako se ograničenje navodi kao ograničenje stupca uz definiciju stupca dodati

```
[CONSTRAINT ime_ograničenja]
```

```
    UNIQUE
```

```
ili
```

- ako se ograničenje navodi kao ograničenje tablice dodati

```
[CONSTRAINT ime_ograničenja]
```

```
    UNIQUE (unique_stupac1 [, unique_stupacN ] )
```

gdje su

*unique\_stupac1, ..., unique\_stupacN* imena za stupce u čija se vrijednost ne smije ponavljati.

#### 6.1.1.9. Dodatna provjera upisanih vrijednosti

Do sada su vrijednosti koje smo upisivali u nekom stupcu bile ograničene tipom podatka i veličinu(širinom, preciznošću) podatka. Ograničenje **CHECK** dodatno ograničava skup dozvoljenih vrijednosti unutar nekog stupca. Obično se ograničava:

- maksimalna i/ili minimalna vrijednost koja se može upisati
- dozvoljava samo upis nekih vrijednosti (npr. dozvoljeno je samo upisati 'biologija', 'kemija', ili 'fizika' u stupac PREDMET).
- opseg vrijednosti (npr. vrijednosti između 0-10% za stupac KAMATA).

Za ograničenje kojim se pri upisu podataka nalaže dodatna provjera vrijednosti koje se upisuju koristi se ključna riječ **CHECK**. Vrijedi:

- Ograničenje CHECK može se navesti kao ograničenje stupca ili ograničenje tablice.
- unutar stupca može se nalaziti više CHECK ograničenja, a ne mora se nalaziti ni jedno.
- uvjet unutar CHECK ograničenja može se odnositi na bilo koji stupac u tablici ali i na stupce unutar drugih tablica.

Sintaksa CHECK ograničenja je:

```
[CONSTRAINT ime_ograničenja]
    CHECK (uvjet)
```

gdje je

**uvjet** logički izraz koji se evaluira pri svakom dodavanju, izmjeni ili brisanju unutar tablice. Ako je rezultat true ili nepoznato( u slučaju postojanja NULL vrijednosti) tražena radnja se izvodi, inače se dojavljuje greška a stanje u tablici ostaje nepromijenjeno.

Definicija baze podataka iz jednog od prethodnih poglavlja primjenom SQL naredbi izgleda ovako:

```
CREATE TABLE mjesto
(
    mjesto_id      int          IDENTITY CONSTRAINT PK_MJESTO PRIMARY KEY,
    ime_mjesta     varchar(30)  NOT NULL,
    post_br        int,
    drzava         varchar(25)  NOT NULL
);

CREATE TABLE student
(
    student_id      int          IDENTITY CONSTRAINT PK_STUDENT PRIMARY KEY,
    ime             varchar(20)  NOT NULL,
    prezime         varchar(30)  NOT NULL,
    jmbg            char(13),
    ime_oca         varchar(20),
    dat_rođenja    datetime    NOT NULL,
    mjesto_rod     int          NOT NULL CONSTRAINT FK_MJESTO_ST FOREIGN KEY REFERENCES m
    dat_upisa       smalldatetime NOT NULL DEFAULT getdate()
);

CREATE TABLE upisni_list
(
    ulist_id        int          IDENTITY CONSTRAINT PK_UPISNI PRIMARY KEY,
    student_id      int          NOT NULL CONSTRAINT FK_ST_UPIS FOREIGN KEY REFERENCES stu
    sem             tinyint      NOT NULL,
    sk_god          varchar(8)   NOT NULL,
    obr_prog_id     int          NOT NULL CONSTRAINT FK_OBR_UPIS FOREIGN KEY REFERENCES ob
    CONSTRAINT CHK_SEM CHECK (sem>0 AND sem<6)
);

CREATE TABLE predmet
(
    predmet_id      int          IDENTITY,
    ime_predmeta   varchar(25)  NOT NULL,
    sati_pred       tinyint      NOT NULL,
    sati_au_vj      tinyint      NOT NULL DEFAULT 0,
    sati_lab_vj     tinyint      NOT NULL DEFAULT 0,
    CONSTRAINT PK_PREDMET PRIMARY KEY(predmet_id)
);

CREATE TABLE upisuje
(

```

```

ulist_id      int          CONSTRAINT FK_UPISNI_UPISUJE FOREIGN KEY REFERENCES upisni
predmet_id    int          CONSTRAINT FK_PREDM_UPISUJE FOREIGN KEY REFERENCES predme
testiran      bit          NOT NULL DEFAULT 0,
ocjena        tinyint,
datum_polag  smalldatetime,
CONSTRAINT PK_UPISUJE PRIMARY KEY (ulist_id,predmet_id)
);

```

### 6.1.2. Kreiranje privremenih tablica

Sve tablice koje smo do sada kreirali su stalne (permanentne) i nazivamo ih bazne tablice. Podaci u njima postoje dok se izričito ne izda nalog da se neka tablica ne obriše (uništi). SQL nam omogućava i kreiranje privremenih tablica za spremanje međurezultata obrada.

Privremena tablica se automatski uništava na kraju sesije(vrijeme unutar kojeg smo spojeni na bazu podataka, vrijeme između prijave i odjave korisnika u kojem DBMS sustav izvodi operacije koje zahtijevamo), obrade ili transakcije. Zajedno sa privremenom tablicom brišu se i svi podaci koji su u njoj pohranjeni.

Za kreiranje privremenih tablica vrijedi:

- Vrijede ista pravila kao i za kreiranje baznih tablica
- Kreirane privremene tablice su nakon kreiranja prazne (nema ni jednog upisanog retka).
- Ako se kreira velika privremena tablica, dobro ju je uništiti čim nam više ne treba, umjesto da čekamo da DBMS to učini na kraju sesije.
- Da bi kreirali privremene tablice moramo imati potrebne ovlasti i radni prostor

Sintaksa **CREATE TEMPORARY** naredbe je:

```

CREATE {LOCAL | GLOBAL} TEMPORARY TABLE tablica
(
  stupac1 tip_podataka1 [svojstvo1] [ograničenja_stupca1],
  stupac2 tip_podataka2 [svojstvo2] [ograničenja_stupca2],
  ...
  stupacN tip_podatakaN [svojstvo1] [ograničenja_stupcaN]
  [, ograničenje_tablice]
);

```

gdje je

**tablica** naziv privremene tablice,

**LOCAL** označava da se radi o lokalnoj privremenoj tablici, a

**GLOBAL** da se radi o globalnoj privremenoj tablici.

### 6.1.3. Kreiranje nove tablice na temelju postojeće tablice

Naredba **CREATE TABLE AS** kreira novu tablicu i popunjava rezultatom obavljenog upita.

Sintaksa naredbe **CREATE TABLE AS** je:

```
CREATE TABLE nova_tablica
```

```
AS podupit;
```

gdje je

*nova\_tablica* ime tablice koja se kreira,

a *podupit* je upit koji određuje strukturu tablice i vraća retke kojim se popunjava nova tablica.

## 6.2. Izmjena postojeće tablice

Naredbu **ALTER TABLE** koristimo da bismo izmijenili definiciju tablice, dodavanjem izmjenom ili brisanjem stupaca i ograničenja.

Unatoč SQL standardu, postoje značajne razlike u implementacijama naredbe ALTER TABLE u raznim DBMS sustavima. Preporučuje se stoga dobro proučiti literaturu vezanu uz DBMS sustav koji se koristi da bi se znalo što se može a što ne.

Primjenom naredbe **ALTER TABLE** obično se može:

- dodati ili obrisati stupac
- izmijeniti tip podataka u nekom stupcu
- dodati, izmijeniti ili odbaciti predodređene vrijednosti stupca i dozvolu pojave NULL vrijednosti
- dodati, izmijeniti ili odbaciti ograničenja stupca ili ograničenja tablice (PRIMARY KEY, FOREIGN KEY, UNIQUE i CHECK)
- preimenovati stupac
- preimenovati tablicu

Sintaksa ALTER TABLE naredbe je:

```
ALTER TABLE tablica  
    akcija_za_izmjenu_tablice;
```

gdje je

*tablica* naziv tablice koja se mijenja,

a *akcija\_za\_izmjenu\_tablice* određuje akciju koju treba obaviti.

Akcija počinje ključnim riječima ADD, ALTER, or DROP.

Npr. moguće akcije su:

```
ADD COLUMN stupac tip_podatka [ograničenja]  
ALTER COLUMN stupac SET DEFAULT expr  
DROP COLUMN stupac [RESTRICT | CASCADE]  
ADD ograničenje_tablice  
DROP CONSTRAINT ime_ograničenja
```

### 6.2.1. Brisanje stupca ili ograničenja (constraints) iz tablice

Sintaksa naredbe za brisanje stupca ili ograničenja iz baze podataka je:

```
ALTER TABLE tablica DROP  
    COLUMN ime_stupaca,  
    CONSTRAINT ime_ograničenja
```

gdje je

*tablica* naziv tablice u kojoj se vrše izmjene,

*ime\_stupca* naziv za stupac koji se briše iz tablice,

*ime\_ograničenja* naziv za ograničenje koje se briše iz definicije tablice.

Npr:

```
ALTER TABLE student DROP COLUMN ime_oca
```

Brisanje postojećih stupaca u tablici primjenom instrukcije ALTER TABLE  
DROP može se primijeniti samo na one stupce koje ne podliježu  
nikakvim ograničenjima. Drugim riječima, nije moguće ukloniti stupce:

- koji čine primarni ključ ili dio primarnog ključa
- koji predstavljaju strani ključ
- koji su vezani ograničenjem tipa CHECK
- jedinstvene stupce (UNIQUE)
- stupce koje imaju predodređenu (DEFAULT) vrijednost.
- stupce po kojima postoji indeks.

Za uklanjanje ovakvih stupaca, potrebno je prethodno izbrisati sva  
ograničenja vezana za te stupce.

### 6.2.2. Promjene svojstava postojećih stupaca u tablici

Sintaksa naredbe za brisanje stupca ili ograničenja iz baze podataka je:

```
ALTER TABLE tablica  
    ALTER COLUMN ime_stupca | svojstva
```

gdje je

*tablica* naziv tablice u kojoj se vrše izmjene,

*ime\_stupca* naziv za stupac koji se mijenja,

Npr.:

```
ALTER TABLE student ALTER COLUMN prezime varchar(20) NOT NULL
```

Na tablici je moguće promijeniti samo svojstva stupaca, a ne i pojedina  
ograničenja. Kod promjene pojedinih ograničenja, potrebno je najprije  
ukloniti postojeće ograničenje, a nakon toga definirati novo. Stoga je  
instrukcijom ALTER TABLE moguće samo mijenjanje svojstava vezanih  
za pojedini stupac (tip podataka i NOT NULL ili NULL svojstvo).

Primjenom ove instrukcije nije moguće promijeniti svojstva stupaca:

- koji čine primarni ključ ili dio primarnog ključa
- koji predstavljaju strani ključ
- koji su vezani ograničenjem tipa CHECK ili UNIQUE. Jedina  
promjena koja se može izvršiti jest ako su te stupce  
definirane kao niz znakova varchar, pri čemu je dozvoljeno  
mijenjanje dužine niza.
- stupce po kojima postoji indeks.

### 6.2.3. Dodavanje novih stupaca

Dodavanje novih stupaca može se slobodno realizirati dok je tablica  
prazna, te ne sadrži podatke. Međutim ako tablica već sadrži neke  
podatke, svaki od već postojećih redaka u tablici proširuje se sa novim  
stupcem, za koji prethodno nije definirana njegova vrijednost. U ovakvim  
slučajevima, moguća su dva rješenja :

1. Stupac koji se dodaje u tablicu treba se definirati sa svojstvom NULL. Na taj način svim već postojećim redovima dodaje se nova stupaca, a vrijednosti u toj koloni za svaki redak u tablici bit će NULL.
2. Ako se stupac ne želi definirati sa svojstvom NULLI, za njega je potrebno navesti DEFAULT vrijednost. U takvim slučajevima svi postojeći retci u novostvorenom stupcu poprimaju vrijednost koja je određena kao DEFAULT.

Npr.

```
ALTER TABLE student ADD ime_majke varchar(20)
ALTER TABLE student ADD ime_majke varchar(20) NOT NULL DEFAULT 'Nije poznato'
```

#### 6.2.4. Dodavanje novih ograničenja

Kod dodavanja novih ograničenja treba voditi računa da svako ograničenje počinje važiti od trenutka kada je definirano. Ukoliko se nekoj tablici, koja već sadrži određene podatke (određeni broj redaka) želi dodati ograničenje tipa CHECK ili FOREIGN KEY, potrebno je definirati kako se novo ograničenje odnosi prema već postojećim podacima.

Neka je tablica upisni\_list definirana kao:

```
CREATE TABLE upisni_list
(
    ulist_id           int      IDENTITY CONSTRAINT PK_UPISNI PRIMARY KEY,
    student_id         int      NOT NULL CONSTRAINT FK_ST_UPIS FOREIGN KEY REFERENCES stu
    sem                tinyint  NOT NULL,
    sk_god             varchar(8) NOT NULL,
    obr_prog_id        int      NOT NULL,
    CONSTRAINT FK_OBR_UPIS FOREIGN KEY REFERENCES obr_prog(obr_prog_id)
);
```

Pretpostavimo da su u tablicu uneseni neki upisni listovi, nakon čega primijenimo ograničenje na vrijednosti semestra izrazom

```
ALTER TABLE upisni_list ADD CONSTRAINT CHK_SEM CHECK (sem>0 AND sem<6)
```

Novopostavljeno ograničenje vrijedit će za sve podatke koji se unesu nakon što je ograničenje definirano, ali je pitanje kako se odnosi prema onim retcima u tablici koji su prethodno uneseni. Djelovanje novog ograničenja na već postojeće podatke određuje se opcijama **WITH CHECK** i **WITH NOCHECK**.

Ako se ograničenje definira sa opcijom **WITH CHECK**, automatski će biti provjereni i svi prethodno uneseni podaci da li zadovoljavaju pravilo. Primjenom opcije **WITH NOCHECK**, novo ograničenje primjenit će se samo na one podatke koji se unesu nakon ograničenja, bez provjere prethodno unesenih podataka.

```
ALTER TABLE upisni_list WITH CHECK ADD CONSTRAINT CHK_SEM CHECK (sem>0 AND sem<6)
```

#### 6.3. Brisanje tablice

Tablicu iz baze podataka uklanjamo naredbom **DROP TABLE**. Vrijedi:

- ukloniti se može bazna ili privremena tablica.
- uklanjanje tablice iz baze podataka uništava se uglavnom

- sve što je vezano za nju: struktura baze podataka, sami podaci, indeksi, ograničenja, dozvole i td.
- uklanjanje tablice nije isto što i brisanje redaka iz tablice.

Sintaksa naredbe DROP TABLE je:

```
DROP TABLE tablica;
```

gdje je **tablica** naziv tablice koja se briše.

## 6.4. Indeksi

Uvođenje indeksa u bazu podataka ima za osnovni cilj ubrzavanje pretraživanja, tj. smanjenje broja pristupa disku. Indeks definiran nad tablicom vezuje se za određeni stupac ili više stupaca u tablici.

Indeks za određeni atribut jest tablica sa dva stupca i jednakim brojem redaka, kao i tablica nad kojom je indeks definiran. Prvi stupac u indeksu je atribut za koji je indeks definiran i naziva se ključ indeksa (index key), a u drugom stupcu je disk pointer (pokazivač na mjesto na disku na kojem se nalazi podatak sa određenom vrijednošću indeksnog ključa). Važna osobina indeksa je da je tablica indeksa uvijek striktno uređena po vrijednostima ključa, bilo u rastućem redoslijedu (ascending) ili padajućem (descending).

Koncept indeksiranja može se usporediti sa katalogizacijom knjiga u biblioteci. Za svaku knjigu postoji odgovarajuća kartica sa osnovnim podacima o knjizi i podatkom gdje je smještena. Pri tome može postojati nekoliko odvojenih kataloga, u jednom su knjige sortirane prema naslovu, u drugom prema imenu autora itd. Kad korisnik zatraži knjigu, npr. od željenog autora, potrebno je pogledati u katalog gdje su knjige sortirane po autoru, te se nađe pripadajuća kartica i pogleda u kojem je dijelu knjižnice ta knjiga.

Prilikom pretraživanja podataka, ukoliko se podacima pristupa po vrijednostima atributa nad kojima postoji indeks, sustav ne pretražuje cijelu polaznu tablicu, već umjesto toga, pretražuje se tablica indeksa, koja je po veličini mnogo manja. Pri tome nije nužno pretražiti cijelu tablicu indeksa, već samo do zadane vrijednosti indeksnog ključa. Kada se u pretraživanju indeksa dođe do zadane vrijednosti indeksnog ključa i lociraju svi podaci, pretraživanje indeksa se završava, budući je indeks uređen po vrijednostima ključa, pa ostatak tablice indeksa sigurno ne sadrži više podatke sa zadanom vrijednošću ključa.

Za neke oblike pretraživanja čak nije ni potrebno pretraživati cijelu tablicu, već je dovoljno pretraživanje samo tablice indeksa, po vrijednostima ključa. Takvi upiti ispituju samo postojanje određenog podatka, te indeksiranje tablice po traženoj vrijednosti atributa, čini dovoljnim pretraživanje indeksa, bez pretraživanja osnovne tablice.

Negativna strana indeksiranja ogleda se u činjenici da se smanjuje brzina ažuriranja i dodavanja novih podataka. Kod ovih operacija, uz promjene u samoj tablici, potrebno je promjeniti i tablicu indeksa.

**Višestruki indeksi** predstavljaju indeksiranje bazne tablice po većem broju atributa. Indeksna tablica zadržava istu strukturu, tj. sastoji se od već opisanih stupaca ključeva indeksa(key) i pokazivača – disk pointera. Ključ indeksa složen je od vrijednosti svih atributa koji su uključeni u indeks.

Ovakvi indeksi pogodni su za pretraživanje po jednoj ili više vrijednosti

atributa. Pretraživanje po jednoj vrijednosti atributa identično je pretraživanju indeksa sa jednim atributom. Pretraživanje po više vrijednosti daje rezultat samo one vrijednosti gdje svi atributi zadovoljavaju tražene uvjete.

#### 6.4.1. CLUSTERED INDEKS

Uobičajeni način popunjavanja tablice prilikom unosa novih redova podrazumijeva slijedno unošenje, tj. jedan red iza drugog, po redu. Uvođenjem indeksa formira se indeksna tablica, koja sadržava podatke o položaju podataka u tablici prema indeksnom ključu, ali redoslijed podataka ostaje nepromijenjen. Pretraživanjem tablice indeksa dobije se podatak o smještaju traženih podataka, ali su oni u osnovnoj tablici raspršeni. Ukoliko se indeks definira svojstvom CLUSTERED, bitno se mijenja raspored podataka u osnovnoj tablici. CLUSTERED indeks podrazumijeva smještaj podataka u osnovnoj tablici u uređenom redoslijedu prema vrijednostima ključa.

U jednoj tablici može postojati samo jedan CLUSTERED indeks.

#### 6.4.2. JEDINSTVENI INDEKS (UNIQUE INDEX)

Definiranjem indeksa sa svojstvom jedinstvenosti (UNIQUE), određuje se da u indeksnoj tablici ne mogu postojati dvije iste vrijednosti indeksnog ključa, tj. ključ indeksa je jedinstven. Budući je ključ indeksa vrijednost atributa iz tablice, time se definira da u tablici ne mogu postojati dva retka sa istom vrijednošću atributa po kojoj se stvara indeks. Jedinstvenost indeksa, a time i atributa u tablici podrazumijeva da vrijednost atributa ne može biti NULL, tj. atribut po kojem se stvara jedinstveni indeks mora imati obaveznu prisutnost. Za višestruke indekse, koji predstavljaju kombinaciju više atributa, jedinstveni indeks podrazumijeva jedinstvenost kombinacije atributa koji su obuhvaćeni indeksom. Npr. ukoliko se definira jedinstveni indeks na tablici STUDENT po atributima ime i prezime, to podrazumijeva da u tablici ne mogu postojati dva studenta koji imaju jednaka oba atributa (ime i prezime).

#### 6.4.3. Kreiranje i brisanje indeksa

Za formiranje indeksa koristi se instrukcija **CREATE INDEX** oblika:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX ime_indeksa ON ime_tablice(stupac1, ..., stupacN)
```

gdje

**UNIQUE** – definira jedinstveni indeks

**CLUSTERED | NONCLUSTERED** – određuje clustered indeks. Ukoliko se ovaj podatak ne navede indeks se smatra NONCLUSTERED

Npr:

```
CREATE UNIQUE INDEX IX_JMBG ON student(JMBG)
```

Brisanje indeksa vrši se instrukcijom oblika:

```
DROP INDEX ime_tablice.ime_indeksa
```

## 7. PRETRAŽIVANJE BAZE PODATAKA PRIMJENOM SQL NAREDBI

Tipični SQL upitni blok ima slijedeći oblik:

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

gdje je

- *A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub> su stupci (atributi) relacija koji se žele u rezultatu.* Lista stupaca može se zamjeniti \*, što znači da se u rezultatu pojavljuju svi stupci po redoslijedu navedenom kod formiranja tablice (CREATE TABLE). Ako ih ima više, imena stupaca se međusobno odvajaju zarezom.  
Očigledno je da SELECT odgovara operaciji projekcije u relacijskoj algebri.
- *R<sub>1</sub>, R<sub>2</sub>, ..., R<sub>m</sub> su tablice (relacije) koje se koriste u pretraživanju,* pa je iskaz FROM ekvivalentan operaciji Kartezijevog produkta u relacijskoj algebri. Tablice moraju prethodno biti definirane u bazi podataka. Uz imena tablice mogu se koristiti i logičke veze koje opisuju operacije prirodnog spajanja među tablicama (inner join, outer join itd). Realizacija logičkih veza podrazumijeva navođenje imena tablica i načina njihovog povezivanja.
- *P je predikat koga zadovoljavaju selektirane n-torce u rezultatu,* pa je iskaz WHERE ekvivalentan operaciji selekcije u relacijskoj algebri.

Prilikom definiranja select izraza koji obuhvaća stupce (attribute) iz više tablica (relacija), treba voditi računa o jedinstvenosti naziva izabranih atributa. To znači da, ukoliko se navode stupci iz više tablica koji imaju isti naziv, uz naziv stupca treba navesti iz koje se tablice selektira traženi stupac.

Tipični upitni blok SQL jezika je ekvivalentan ranije danom općem izrazu za realizaciju upita u relacijskoj algebri:

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

Isto je tako očigledno da se može pokazati i ekvivalencija općeg iskaza upita u relacijskom računu n-torki sa tipičnim SQL upitnim blokom,

```
t Î R1, u Î R2, ..., v Î Rn  
t. A1, u. A2, ..., v. An GDJE_JE P
```

čime se posredno pokazuje i ekvivalentnost relacionog računa i relacione algebre. Ponovo napominjemo da ekvivalentnost SQL upitnog bloka sa općim izrazom za realizaciju upita u relacijskoj algebri ne znači da se SQL upit implementira na taj način.

Tipičnom SQL bloku mogu se dodati još mnogi detalji. Umjesto definiranja opće sintakse ovoga bloka, prikazati ćemo nekoliko karakterističnih primjera

Demonstracija SQL upita u ovom poglavlju vrši se na bazi podataka **FakultetskaBaza** unutar MS Access DBMS-a.

## 7.1. Upit nad jednom relacijom



**UPIT:** Prikaži brojeve indeksa svih studenata koji su polagali ispite.

`SELECT BR_INDEX FROM POLAŽE;`

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

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

BR_INDEX
10020
10020
10020
10020
20022
20022
20456
30354

Rezultat nije prava projekcija. Ako se želi prava projekcija upit izgleda:

`SELECT DISTINCT BR_INDEX FROM POLAŽE;`

Rezultat ovog upita je tablica:

BR_INDEX
10020
20022
20456
30354

--

Prava projekcija, odnosno eliminacija duplikata je skupa operacija pa se preporučuje samo onda kada je to stvarno neophodno.

## 7.2. Uvjetni izraz (search condition)

Općeniti oblik uvjetnog izraza je:

```
SELECT lista_stupaca  
FROM ime_tablice  
WHERE uvjetni izraz
```

### 7.2.1. Predikati usporedbe (Comparision predicates)

Predikati usporedbe primjenjivi su na sve tipove podataka. Opći oblik je

**naziv\_stupca OperatorUsporedbe vrijednost**

Na raspolaganju su nam slijedeći operatori:

Operator	Značenje
=	Jednako
>	Veće od
<	Manje od
>=	Veće ili jednako
<=	Manje ili jednako
<>	Različito od
!>	Nije veće
!<	Nije manje

Npr.



**UPIT:** Prikaži sve studente koji se zovu Marko.

```
SELECT * FROM STUDENT
WHERE IME_STUDENTA="Marko";
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	20022	Marko		2	2003	R

### 7.2.2. Predikati područja (range predicates)

Predikati područja primjenjivi su na sve tipove podataka. Opći oblik je::

**naziv\_stupca BETWEEN x AND y**

Selektiramo retke u kojima atribut **naziv\_stupca** ima vrijednost između **x** i **y**

Npr.



**UPIT:** Prikaži sve studente sa brojevima indeksa između 20000 i 29999.

```
SELECT * FROM STUDENT  
WHERE BR_INDEX BETWEEN 20000 AND  
29999;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	20021	Jure		2	2003	R
	20022	Marko		2	2003	R
	20456	Zdravko		2	2004	R

Drugi oblik predikata područja je:

**naziv\_stupca NOT BETWEEN x and y**

Selektiramo retke u kojima atribut **naziv\_stupca** ima vrijednost izvan opsega **x** i **y**.

Npr.



**UPIT:** Prikaži sve studente koji nisu upisali studij u periodu do 2003-2004 godine.

```
SELECT * FROM STUDENT  
WHERE GOD_UPISA NOT BETWEEN 2003  
AND 2004;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	10020	Branko		1	2005	I

	30354	Neno	3	2002	I	
--	-------	------	---	------	---	--

### 7.2.3. Predikati liste (list predicates)

Predikati liste primjenjivi su na sve tipove podataka. Opći oblik je:

**naziv\_stupca IN (lista vrijednosti)**

Selektiramo retke u kojima atribut **naziv\_stupca** ima vrijednost koja je jednaka nekoj od vrijednosti navedenoj u listi.

Npr.



**UPIT:** Prikaži sve studente koji su rođeni u mjestima Split ili Hvar.

```
SELECT * FROM STUDENT
WHERE MJ_ROĐENJA IN ("Split", "Hvar");
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	20022	Marko		2	2003	R
	10020	Branko		1	2005	I
	20456	Zdravko		2	2004	R

Uočite da bismo isti rezultat dobili da smo postavili upit:

```
SELECT * FROM STUDENT
WHERE MJ_ROĐENJA = "Split" OR MJ_ROĐENJA =
"Hvar";
```

Drugi oblik predikata liste je:

**naziv\_stupca NOT IN (lista vrijednosti)**

Selektiramo retke u kojima atribut **naziv\_stupca** ima vrijednost koja nije jednaka ni jednoj od vrijednosti navedenoj u listi.

Npr.



**UPIT:** Prikaži sve studente koji nisu rođeni Hvaru i Makarskoj.

```
SELECT * FROM STUDENT
WHERE MJ_ROĐENJA NOT IN ("Hvar", "Makarska");
```

Rezultat ovog upita je tablica (obratite pažnju na redoslijed redaka):

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	20022	Marko		2	2003	R
	10020	Branko		1	2005	I

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	10020	Branko		1	2005	I
	20021	Jure		2	2003	R
	20022	Marko		2	2003	R
	20456	Zdravko		2	2004	R
	30354	Neno		3	2002	I

Uočite da bismo isti rezultat dobili da smo postavili upit:

```
SELECT * FROM STUDENT
WHERE MJ_ROĐENJA <> "Hvar" AND MJ_ROĐENJA
<> "Makarska";
```

#### 7.2.4. Predikati uzorka (pattern predicates)

Predikati uzorka primjenjivi su na tekstualne i datumske tipove podataka.

Opći oblik predikata uzorka je:

**naziv\_stupca LIKE uzorak**

Selektiramo retke u kojima se vrijednost atributa **naziv\_stupca** ima vrijednost koja nije jednaka ni jednoj od vrijednosti navedenoj u listi. Selektiraju se - vrijednost atributa poklapa se sa navedenim uzorkom

Vrste uzoraka su:

Uzorak	Opis
* ( % )	Niz karaktera bilo koje duljine.
? ( _ )	Bilo koji pojedinačni karakter
#	Bilo koja znamenka
[ ]	Bilo koji karakter u navedenim granicama
[! ]	Bilo koji karakter izvan navedenih granica

Npr.



**UPIT:** Prikaži sve nazive kolegija čiji naziv počinje slovom P

```
SELECT NAZ_KOLEGIJA FROM KOLEGIJ  
WHERE NAZ_KOLEGIJA LIKE "P*"
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: Query	
	NAZ_KOLEGIJA
	Programiranje za internet
	Programski prevoditelji

Tablica: Query						
BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA	
10020	Branko		1	2005	I	Split
20021	Jure		2	2003	R	Makarska
20022	Marko		2	2003	R	Split
20456	Zdravko		2	2004	R	Hvar
30354	Neno		3	2002	I	

Kolegij : Table			
	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
► +	110	Baze podataka	Perić
► +	220	Programiranje za internet	Jurić
► +	330	Numerička analiza	Franić
► +	440	Programski prevoditelji	Antić
*			

Query1 : Sele... [-] [X]			
	NAZ_KOLEGIJA		
►	Programiranje za internet		
	Programski prevoditelji		
*			



**UPIT:** Prikaži sve nastavnike čije ime završava sa 'ić'.

```
SELECT PREZ_NASTAVNIKA FROM KOLEGIJ  
WHERE PREZ_NASTAVNIKA LIKE "*ić"
```

PREZ_NASTAVNIKA
► Perić
Jurić
Franić
Antić
*

Record: 1



**UPIT:** Prikaži sve nastavnike kojima se u prezimenu nalazi "rić", a da se ispred nalaze bilo koja dva slova.

```
SELECT PREZ_NASTAVNIKA FROM KOLEGIJ
WHERE PREZ_NASTAVNIKA LIKE "??"rić"
```

PREZ_NASTAVNIKA
► Perić
Jurić
*

Record: 1



**UPIT:** Prikaži sve nastavnike kojima prezime počinje sa slovom između A i J, a prezime završava na "ić"

```
SELECT PREZ_NASTAVNIKA FROM KOLEGIJ
WHERE PREZ_NASTAVNIKA LIKE "[A-J]*ić"
```

PREZ_NASTAVNIKA
► Jurić
Franić
Antić
*

Record: 1



**UPIT:** Prikaži sve nastavnike kojima prezime počinje sa slovom između A ili J.

```
SELECT PREZ_NASTAVNIKA FROM KOLEGIJ
WHERE PREZ_NASTAVNIKA LIKE "[AJ]**"
```

PREZ_NASTAVNIKA
► Jurić
Antić
*

Record: 1

## 7.2.5. Predikati nepoznate vrijednosti

Predikati IS NULL, IS NOT NULL primjenjivi su na sve tipove podataka.

Opći im je oblik:

**naziv\_stupca IS NULL**

ili

**naziv\_stupca IS NOT NULL**

Npr.



**UPIT:** Prikaži sve studente kojima je mjesto rođenja NULL.

```
SELECT BR_INDEX, IME_STUDENTA FROM  
STUDENT  
WHERE MJ_ROĐENJA IS NULL;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

BR_INDEX	IME_STUDENTA
30354	Neno
*	

Record: [Navigation Buttons] 1 [Next] [Last]

## 7.2.5. Logički operatori - AND, OR, NOT

Logički operatori AND i OR povezuju više uvjeta u uvjetnom izrazu.

Logički izraz AND povezuje dva ili više uvjeta i vraća rezultat samo ako su svi uvjeti zadovoljeni (istiniti).

Npr.



**UPIT:** Prikaži sve studente kojima su 2003 god. i imaju broj indeksa između 20000 i 29999 danas.

```
SELECT BR_INDEX, IME_STUDENTA FROM  
STUDENT  
WHERE GOD_UPISA = 2003 AND BR_INDEX  
BETWEEN 20000 AND 29999;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

BR_INDEX	IME_STUDENTA
20022	Marko
20021	Jure
*	

Record: [Navigation Buttons] 1 [Next] [Last]

Logički izraz OR povezuje dva ili više uvjeta i vraća rezultat ako je bilo koji uvjet zadovoljeni (istinit).

Npr.



**UPIT:** Prikaži sve studente koji su rođeni u Split ili im mjesto rođenja nije poznato (NULL).

```
SELECT BR_INDEX, IME_STUDENTA FROM
STUDENT
WHERE MJ_ROĐENJA IS NULL OR MJ_ROĐENJA =
"Split";
```

Query1 : Select ...	
	BR_INDEX   IME_STUDENTA
▶	20022 Marko
	10020 Branko
	30354 Neno
*	

Logički izraz NOT negira uvjet koji mu slijedi

Npr.



**UPIT:** Prikaži sve studente koji nisu rođeni u Splitu.

```
SELECT BR_INDEX, IME_STUDENTA FROM
STUDENT
WHERE NOT MJ_ROĐENJA = "Split";
```

Query1 : Select ...	
	BR_INDEX   IME_STUDENTA
▶	20456 Zdravko
	20021 Jure
*	

Kada se u uvjetnom dijelu izraza pojavljuje više logičkih operatora, logički izraz formira se prema prioritetu djelovanja operatora

Logički operator	Prioritet
NOT	najviši prioritet
AND	srednji prioritet
OR	najniži prioritet

Npr.



**UPIT:** Prikaži sve studente koji su upisani 2003 ili su upisani 2002 god i na 3 su godini studija.

```
SELECT BR_INDEX, IME_STUDENTA FROM
STUDENT
WHERE GOD_UPISA = 2003 OR GOD_UPISA = 2002
AND GOD_STUDIJA = 3;
```

	BR_INDEX	IME_STUDENTA
▶	20022	Marko
	30354	Neno
	20021	Jure
*		

Record: [◀] [◀] [▶] [▶] 1 [◀] [▶]

Sličan upit sa zagradama potpuno mijenja smisao:



**UPIT:** Prikaži sve studente koji su upisani 2003 ili 2002 god i na 3 su godini studija.

```
SELECT BR_INDEX, IME_STUDENTA FROM
STUDENT
WHERE (GOD_UPISA = 2003 OR GOD_UPISA =
2002) AND GOD_STUDIJA = 3;
```

	BR_INDEX	IME_STUDENTA
▶	30354	Neno
*		

Record: [◀] [◀] [▶] [▶] 1 [◀] [▶]

### 7.3 Upiti nad više relacija

U jednom upit može se povezati više relacija.



**UPIT:** Prikaži listu studenata, ispita koje se polagali i ocjena koje su dobili. Umjesto brojeva indeksa i šifri kolegija neka se u listi pojave imena studenata i nazivi kolegija.

```
SELECT IME_STUDENTA, NAZ_KOLEGIJA, OCJENA
FROM POLAŽE, STUDENT, KOLEGIJ
WHERE POLAŽE.BR_INDEX =
STUDENT.BR_INDEX AND
POLAŽE.ŠIF_KOLEGIJA = KOLEGIJ.ŠIF_KOLEGIJA;
```

	BR_INDEX	IME_STUDENTA
▶	30354	Neno
*		

Record: [◀] [◀] [▶] [▶] 1 [◀] [▶]

Općenitije napisan izraz za selekciju podataka bio bi:

<b>SELECT [ ALL   DISTINCT ] imena_stupaca</b> <b>FROM imena_tablica   logičke veze</b> <b>WHERE uvjetni_izraz</b>
--

Unutar FROM sekcije izraza za selekciju podataka mogu se osim imena tablica pojaviti i logičke veze.

Logičke veze opisuju operacije prirodnog spajanja među tablicama (inner join, left inner join, right inner join, outer join itd). Realizacija logičkih veza podrazumijeva navođenje imena tablica i načina njihovog povezivanja (join) u obliku:

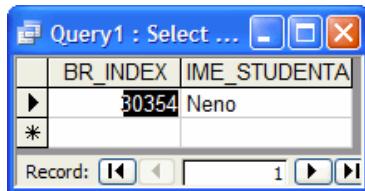
<i>tablica1 Operator_veze tablica2 ON tablica1.stupac1=tablica2.stupac1</i>
---

Npr.



**UPIT:** Prikaži sve imena studente i brojeve indeksa svih studenata koji su dosada polagali barem jedan ispit.

```
SELECT DISTINCT POLAŽE.BR_INDEX,  
IME_STUDENTA  
FROM POLAŽE INNER JOIN STUDENT ON  
POLAŽE.BR_INDEX=STUDENT.BR_INDEX;
```



	BR_INDEX	IME_STUDENTA
▶	30354	Neno
*		

Uočite da bi isti rezultat dobili i upitom:

```
SELECT DISTINCT POLAŽE.BR_INDEX, IME_STUDENTA  
FROM POLAŽE, STUDENT  
WHERE POLAŽE.BR_INDEX=STUDENT.BR_INDEX;
```

Prilikom definiranja select izraza koji obuhvaća atribute (stupce) iz više tablica, treba voditi računa o jedinstvenosti naziva izabranih atributa. To znači da, ukoliko se navode stupci iz više tablica koji imaju isti naziv, uz naziv stupca obavezno treba definirati iz koje tablice se selektira traženi stupac.

Upit SQL naredbom:

```
SELECT DISTINCT BR_INDEX, IME_STUDENTA  
FROM POLAŽE, STUDENT  
WHERE POLAŽE.BR_INDEX=STUDENT.BR_INDEX;
```

**nije ispravan** jer nije određeno iz koje se tablice preuzima stupac BR\_INDEX, budući da on postoji i u tablici POLAŽE i u tablici STUDENT

## 7.4. Oblikovanje izlaznih rezultata

### 7.4.1. Preimenovanje stupaca u rezultatu

Kod postavljanja upita (select) kojim se dohvaćaju podaci, izlazna lista (rezultat) formira se na način da naziv kolone koja se dohvaća postaje zaglavlje te kolone.

Npr.



**UPIT:** Prikaži sve studente koji su upisani 2003 god..

```
SELECT BR_INDEX, IME_STUDENTA , GOD_UPISA  
FROM STUDENT  
WHERE GOD_UPISA = 2003;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Query1 : Select Query		
	BR_INDEX	IME_STUDENTA
▶	20022	Marko
	20021	Jure
*		0

Za postizanje izlazne liste rezultata, koja je korisniku razumljivija, može se u sklopu select izraza primijeniti preimenovanje izlaznih rezultata u obliku:

**naziv\_stupca AS zaglavlje\_stupca**

Ponovimo prethodni primjer uz preimenovanje stupaca:

```
SELECT BR_INDEX AS "INDEKS" , IME_STUDENTA AS  
"IME", GOD_UPISA AS "GODINA UPISA"  
FROM STUDENT  
WHERE GOD_UPISA = 2003;
```

Rezultirajuća relacija je:

Query1 : Select Query			
	BROJ_INDEKSA	IME_STUDENTA	GODINA_UPISA
▶	20022	Marko	2003
	20021	Jure	2003
*			0

#### 7.4.2. Zamjena podataka u rezultatu upita

Proširimo tablicu STUDENT uvođenjem novog stupca TIP\_STUDIJA.

Podaci u tom stupcu imati će samo dvije vrijednosti: "R" ili "I", kao oznaka za redovne ili izvanredne studente.

Npr.



**UPIT:** Prikaži sva imena studenata i tip studija.

```
SELECT IME_STUDENTA, TIP_STUDIJA FROM  
STUDENT
```

Student : Table						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_RODENJA	TIP_STUDIJA
	10020	Branko		1	2005	Split
	20021	Jure		2	2003	Makarska
	20022	Marko		2	2003	Split
	20456	Zdravko		2	2004	Hvar
*	30354	Neno		3	2002	I
				1	0	

Record: [◀◀] [◀] [▶] [▶▶] [◀◀◀◀] [▶▶▶▶] of 5

Query1 : Select Qu...	
	IME_STUDENTA   TIP_STUDIJA
▶	Branko   I
	Jure   R
	Marko   R
	Zdravko   R
*	Neno   I

Record: [◀◀] [◀] [▶] [▶▶] [◀◀◀◀] [▶▶▶▶]

Prethodni upit daje rezultat u kojem se u stupcu TIP\_STUDIJA nalaze znakovi 'R' ili 'I' kao oznaka tipa studiranja. Kako bi se ove kratice učinile razumljivijim mogu se u izlaznom rezultatu dodijeliti puni nazivi korištenjem SWITCH funkcije

```
SELECT IME_STUDENTA,  
SWITCH(TIP_STUDIJA="R", "Redovni", TIP_STUDIJA="I", "Izvanredni")  
AS "TIP STUDIJA"  
FROM STUDENT;
```

Query1 : Select Qu...	
	IME_STUDENTA   "TIP STUDIJA"
	Branko   Izvanredni
	Jure   Redovni
	Marko   Redovni
	Zdravko   Redovni
*	Neno   Izvanredni

Record: [◀◀] [◀] [▶] [▶▶] [◀◀◀◀] [▶▶▶▶]

### 7.4.3. Povezivanje podataka u rezultatu upita

U tablici koja je rezultat upita stupci polaznih relacija mogu se spajati.



**UPIT:** Prikaži sva imena studenata, godinu studija i tip studija.

```
SELECT IME_STUDENTA, GOD_STUDIJA,  
TIP_STUDIJA FROM STUDENT
```

Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_ROĐENJA	TIP_STUDIJA
	10020	Branko		1	2005	Split
	20021	Jure		2	2003	Makarska
	20022	Marko		2	2003	Split
	20456	Zdravko		2	2004	Hvar
.	30354	Neno		3	2002	I
*				1	0	

Record: [◀◀] [◀] [▶] [▶▶] [◀▶] [▶◀] \* of 5

Query1 : Select Query

	IME_STUDENTA	GOD_STUDIJA	TIP_STUDIJA
▶	Branko	1	I
	Jure	2	R
	Marko	2	R
	Zdravko	2	R
	Neno	3	I
*		1	

Record: [◀] [◀] [◀▶] [▶] [▶▶] [◀▶] \* of 5

Povezivanjem stupaca IME\_STUDENTA, TIP\_STUDIJA i GOD\_STUDIJA može se dobiti čitljiviji prikaz studenta:

```
SELECT IME_STUDENTA + " +
SWITCH(TIP_STUDIJA="R","redovni",
      TIP_STUDIJA="I","izvanredni") +
" student "+STR(GOD_STUDIJA)+" godine" AS
"STUDENT"
FROM STUDENT
```

Query1 : Select Query

	"STUDENT"
▶	Branko izvanredni student 1 godine
	Jure redovni student 2 godine
	Marko redovni student 2 godine
	Zdravko redovni student 2 godine
	Neno izvanredni student 3 godine
*	

Record: [◀] [◀] [◀▶] [▶] [▶▶] [◀▶] \* of 5

#### 7.4.4. Sortiranje izlaznih rezultata

Za sortiranje izlaznih rezultata koristi se izraz **ORDER BY**, kao dio select instrukcije.

```
SELECT [ALL | DISTINCT] nazivi_stupaca
FROM nazivi_tablica | logičke veze
WHERE uvjetni izraz
ORDER BY stupci_za_sortiranje
```

Sintaksa za sortiranje je:

**ORDER BY** naziv\_stupca tip\_sortiranja, naziv\_stupca

tip\_sortiranja,...

Sortiranje može biti rastuće (ascending ASC) ili padajuće (descending DESC). Ako nije naveden tip sortiranja podrazumijeva se rastuće (ASC) sortiranje.

Npr.



**UPIT:** Prikaži sve studente sortirane uzlazno po imenima.

```
SELECT * FROM STUDENT ORDER BY  
IME_STUDENTA;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_ROĐENJA
▶	10020	Branko	1	2005	Split
	20021	Jure	2	2003	Makarska
	20022	Marko	2	2003	Split
	30354	Neno	3	2002	
	20456	Zdravko	2	2004	Hvar
*			1	0	

Umjesto naziva stupaca u izrazu ORDER BY može se koristiti redni broj stupca u select listi.



**UPIT:** Prikaži sve studente sortirane silazno po imenima.

```
SELECT * FROM STUDENT ORDER BY 2 DESC;
```

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_ROĐENJA
▶	20456	Zdravko	2	2004	Hvar
	30354	Neno	3	2002	
	20022	Marko	2	2003	Split
	20021	Jure	2	2003	Makarska
	10020	Branko	1	2005	Split
*			1	0	

#### 7.4.5. Izračunate kolone (Computed values)

SQL dozvoljava formiranje novih stupaca u select izrazu, čije se vrijednosti izračunavaju na temelju podataka koji se nalaze u numeričkim stupacama postojećih tablica.

Npr.



**UPIT:** Prikaži za svakog studenta koliko će još minimalno godina biti na fakultetu ako studij traje 4 godine.

```
SELECT BR_INDEX, IME_STUDENTA, 5-
```

GOD\_STUDIJA AS "JOŠ STUDIRA" FROM STUDENT  
ORDER BY 3 ASC;

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Query2 : Select Query			
	BR_INDEX	IME_STUDENTA	"JOŠ STUDIRA"
▶	30354	Neno	2
	20021	Jure	3
	20456	Zdravko	3
	20022	Marko	3
	10020	Branko	4
*			

Pri izračunavanju vrijednosti u pojedinim stupcima mogu se koristiti vrijednosti postojećih stupaca za koje su definirani numerički tipovi podataka i neki od numeričkih operatora +,-,\*,/, % (Modulo operator – ostatak pri cjelobrojnom dijeljenju).

#### 7.4.6. Ograničavanje ispisa rezultata

Upit tipa:

```
SELECT [ALL | DISTINCT] imena_stupaca  
FROM imena_tablica | logičke veze  
WHERE uvjetni_izraz
```

prikazuje sve rezultate iz izabranih tablica koji zadovoljavaju uvjetni izraz. U slučaju potrebe za prikazom samo dijela izlaznih rezultata ispis se može ograničiti primjenom sintakse

```
SELECT [ALL | DISTINCT] [TOP n [PERCENT]] imena_stupaca  
FROM imena_tablica | logičke veze  
WHERE uvjetni_izraz  
ORDER BY kolone_sortiranja
```

pri čemu je select izraz proširen modifikatorom TOP n ili TOP n PERCENT. Ovaj modifikator označava da će u ispisu rezultata biti prikazano samo prvih n redova (TOP n), odnosno samo n% (TOP n PERCENT) od ukupnog broja redova koji zadovoljavaju postavljeni uvjet.

Npr.



**UPIT:** Prikaži podatke o studentima ali se u ispisu ograniči na samo prva dva studenta.

**SELECT TOP 2 \* FROM STUDENT;**

Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_ROĐENJA	TIP_STUDIJA
	10020	Branko		1	2005	Split
	20021	Jure		2	2003	Makarska
	20022	Marko		2	2003	Split
	20456	Zdravko		2	2004	Hvar
*	30354	Neno		3	2002	I
				1	0	

Record: [◀◀] [◀] [▶] [▶▶] [\*] of 5

Query1 : Select Query

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_ROĐENJA	TIP_STUDIJA
▶	20022	Marko		2	2003	Split
*	10020	Branko		1	2005	Split
				1	0	

Record: [◀◀] [◀] [▶] [▶▶] [\*] of 2



**UPIT:** Prikaži podatke o studentima ali se u ispisu ograniči na 75 posto podataka.

**SELECT TOP 75 PERCENT \* FROM STUDENT;**

Query1 : Select Query

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	MJ_ROĐENJA	TIP_STUDIJA
▶	20022	Marko		2	2003	Split
*	10020	Branko		1	2005	Split
	30354	Neno		3	2002	I
	20456	Zdravko		2	2004	Hvar
*				1	0	

Record: [◀◀] [◀] [▶] [▶▶] [\*] of 4

### 7.3.5. Upit koji se kreira nova tablica (MAKE TABLE QUERY) - SELECT INTO

Opći oblik ove SQL naredbe je:

```
SELECT [ALL | DISTINCT] nazivi_stupaca
INTO nova_tablica
FROM nazivi_tablica
WHERE uvjetni izraz
```

Izraz **select into** stvara novu tablicu na osnovu stupaca navedenih u *sekciji nazivi\_stupaca* i redaka koji se biraju uvjetnim izrazom iz postojećih tablica u bazi. Podatak *nova\_tablica* predstavlja ime nove tablice, koja će se formirati iz rezultata upita. Ta tablica ne smije prethodno biti definirana u bazi.

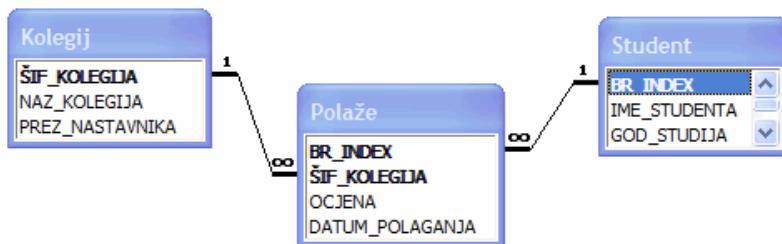
**SELECT INTO** je operacija koja se odvija u dva koraka. U prvom koraku formira se nova tablica, a drugim korakom u novu tablicu unose se retci koji zadovoljavaju uvjetni izraz. Ako se drugi korak ne izvrši iz bilo kojeg razloga, nova tablica ostaje ali u njoj nema nikakvih podataka. Ovim izrazom omogućeno je automatsko formiranje nove tablice bez njezinog prethodnog definiranja naredbom **CREATE TABLE**. Definicija podataka u

novoj tablici formira se automatski i ona je identična definiciji pojedinih atributa(stupaca) u postojećim tablicama, koje su obuhvaćene SELECT izrazom.

Npr.



**UPIT:** Na temelju strukture naše fakultetske baze podataka potrebno je formirati novu tabelu koja će sadržavati podatke o studentima koji su na drugoj godini i polagali su ispit iz predmeta Baze podataka.



```
SELECT IME_STUDENTA INTO STUDENT2
FROM POLAŽE, STUDENT, KOLEGIJ WHERE
POLAŽE.BR_INDEX = STUDENT.BR_INDEX AND
POLAŽE.ŠIF_KOLEGIJA = KOLEGIJ.ŠIF_KOLEGIJA AND
NAZ_KOLEGIJA="Baze podataka" AND GOD_STUDIJA=1;
```

Isti rezultat dobili bismo da smo upit postavili i slijedećom SQL naredbom:

```
SELECT IME_STUDENTA INTO STUDENT2
FROM (POLAŽE INNER JOIN STUDENT ON
POLAŽE.BR_INDEX = STUDENT.BR_INDEX) INNER
JOIN KOLEGIJ ON POLAŽE.ŠIF_KOLEGIJA =
KOLEGIJ.ŠIF_KOLEGIJA WHERE NAZ_KOLEGIJA="Baze
podataka" AND GOD_STUDIJA=1;
```

Kolegij : Table			
	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶ +	110	Baze podataka	Perić
▶ +	220	Programiranje za internet	Jurić
▶ +	330	Numerička analiza	Franić
▶ +	440	Programski prevoditelji	Antić
*			

**Polaže : Table**

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 8

**Student : Table**

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko		1	2005 I	Split
+ +	20021	Jure		2	2003 R	Makarska
+ +	20022	Marko		2	2003 R	Split
+ +	20456	Zdravko		2	2004 R	Hvar
*	30354	Neno		3	2002 I	
				1	0	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 5

Rezultirajuća relacija je:

**ST... : Table**

	IME_STUDENT
▶	Branko
*	

## 7.5. Append upiti (proširivanje tablice podacima iz drugih tablica)

Upitima tipa Append (dodavanje) u postojeću tablicu u bazi podataka dodaju se podaci iz drugih tablica prema zadanim uvjetima. Standardna SQL sintaksa ovih izraza je oblika

```
INSERT INTO ime_tablice
  SELECT nazivi_stupaca
  FROM tablice | logičke veze
  WHERE uvjetni_izraz
```

gdje *ime\_tablice* predstavlja naziv neke od postojećih tablica u bazi podataka.

Npr.



**UPIT:** Tablicu Student2 koju smo kreirali u prethodnom primjeru želimo nadopuniti podacima o svim ostalim studentima koji su polagali predmet "Baze podataka".

```
INSERT INTO STUDENT2 SELECT IME_STUDENTA
  FROM POLAŽE, STUDENT, KOLEGIJ WHERE
    POLAŽE.BR_INDEX = STUDENT.BR_INDEX AND
```

```
POLAŽE.ŠIF_KOLEGIJA = KOLEGIJ.ŠIF_KOLEGIJA  
AND NAZ_KOLEGIJA="Baze podataka" AND  
GOD_STUDIJA<>1;
```

Isti rezultat dobili bismo da smo dodavanje izveli slijedećom SQL naredbom:

```
INSERT INTO STUDENT2 SELECT IME_STUDENTA  
FROM (POLAŽE INNER JOIN STUDENT ON  
POLAŽE.BR_INDEX = STUDENT.BR_INDEX) INNER  
JOIN KOLEGIJ ON POLAŽE.ŠIF_KOLEGIJA =  
KOLEGIJ.ŠIF_KOLEGIJA WHERE  
NAZ_KOLEGIJA="Baze podataka" AND  
GOD_STUDIJA<>1;
```

IME_STUDENT
Branko
*

Rezultirajuća relacija je:

IME_STUDENT
Branko
Marko
*

## 7.6. Ažuriranje podataka u tablici

Ažuriranje podataka u tablici provodi se naredbom UPDATE prema sintaksi

```
UPDATE naziv_tablice SET  
naziv_stupca1 = nova vrijednost1,  
naziv_stupca2 = nova_vrijednost2  
WHERE uvjetni izraz
```

Npr.



**UPIT:** U tablici STUDENT studentu (ili bolje rečeno retku) kojem je broj indeks 20022 treba promijeniti ime u "Darko".

```
UPDATE STUDENT SET IME_STUDENTA="Darko"  
WHERE BR_INDEX=20022;
```

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko		1	2005 I	Split
▶ +	20021	Jure		2	2003 R	Makarska
▶ +	20022	Marko		2	2003 R	Split
▶ +	20456	Zdravko		2	2004 R	Hvar
▶ +	30354	Neno		3	2002 I	
*				1	0	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 5

Rezultirajuća relacija je:

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko		1	2005 I	Split
▶ +	20021	Jure		2	2003 R	Makarska
▶ +	20022	Darko		2	2003 R	Split
▶ +	20456	Zdravko		2	2004 R	Hvar
▶ +	30354	Neno		3	2002 I	
*				1	0	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 5

## 7.7. Brisanje podataka iz tablice

Brisanje podataka iz tablice provodi se naredbom DELETE prema sintaksi

```
DELETE [naziv_tablice.*]
FROM naziv_tablice
WHERE uvjetni izraz
```

Npr.



**UPIT:** U tablici STUDENT obriši studenta sa brojem indeksa 99999.

[DELETE STUDENT WHERE BR\\_INDEX=99999;](#)

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko		1	2005 I	Split
▶ +	20021	Jure		2	2003 R	Makarska
▶ +	20022	Darko		2	2003 R	Split
▶ +	20456	Zdravko		2	2004 R	Hvar
▶ +	30354	Neno		3	2002 I	
..� +	99999	Ivica		1	2000 I	
*				1	0	

Record: [◀] [◀] [6] [▶] [▶] [▶\*] of 6

Rezultirajuća relacija je:

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko	1	2005	I	Split
▶ +	20021	Jure	2	2003	R	Makarska
▶ +	20022	Darko	2	2003	R	Split
▶ +	20456	Zdravko	2	2004	R	Hvar
*	30354	Neno	3	2002	I	
			1	0		

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶] \* of 5

## 7.8. Alias-i

Alias (drugo ime) koristi se u upitima SQL jezika za promjenu naziva objekta (tablice) kojoj se pristupa. Svakoj tablici može se dodijeliti alias, bilo zbog lakše upotrebe ili razlikovanja kod tzv. self-join query postupaka. Sintaksa je

**naziv\_tablice AS alias**

Npr.



**UPIT:** Prikazati sve parove studenata koji su na drugoj godini studija i broj indeksa prvog studenta u paru je veći od broja indeksa drugog studenta u paru.

```
SELECT STUDENT_1.BR_INDEX,
STUDENT_1.IME_STUDENTA, STUDENT_2.BR_INDEX,
STUDENT_2.IME_STUDENTA FROM STUDENT AS
STUDENT_1, STUDENT AS STUDENT_2 WHERE
STUDENT_1.GOD_STUDIJA=STUDENT_2.GOD_STUDIJA
and STUDENT_1.BR_INDEX>STUDENT_2.BR_INDEX
```

ili

```
SELECT STUDENT_1.BR_INDEX,
STUDENT_1.IME_STUDENTA, STUDENT_2.BR_INDEX,
STUDENT_2.IME_STUDENTA FROM STUDENT AS
STUDENT_1 INNER JOIN STUDENT AS STUDENT_2 ON
STUDENT_1.GOD_STUDIJA=STUDENT_2.GOD_STUDIJA
WHERE STUDENT_1.BR_INDEX >
STUDENT_2.BR_INDEX
```

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko	1	2005	I	Split
▶ +	20021	Jure	2	2003	R	Makarska
▶ +	20022	Darko	2	2003	R	Split
▶ +	20456	Zdravko	2	2004	R	Hvar
*	30354	Neno	3	2002	I	
			1	0		

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶] \* of 5

Rezultirajuća relacija je:

	STUDENT_1	STUDENT_1.IME	STUDENT_2	STUDENT_2.IME
▶	20456	Zdravko	20022	Darko
	20456	Zdravko	20021	Jure
	20022	Darko	20021	Jure

Record: [◀] [◀] [1] [▶] [▶] [▶] \* of 3

## 7.9. Agregatne funkcije

Agregatne funkcije generiraju zbirne (sumarne) vrijednosti nad pojedinim stupcima tablice. Agregatne funkcije koriste se u SELECT izrazu, te mogu biti primijenjene nad svim recima tablice ili samo nad onim redovima koji su definirani WHERE izrazom.

Funkcije za dobivanje sumarnih informacija su:

**AVG(atribut)** - izračunava srednju vrijednost danog atributa(stupca),

**SUM(atribut)** - izračunava sumu svih vrijednosti atributa(stupca),

**MIN(atribut)** - nalazi minimalnu vrijednost atributa(stupca),

**MAX(atribut)** - nalazi najveću vrijednost atributa(stupca),

**COUNT(\*)** - nalazi broj n-torki u relaciji (grupi),

**COUNT(atribut)** - nalazi broj n-torki sa ne nula vrijednostima atributa i

Agregatne funkcije SUM i AVG mogu se koristiti samo za numeričke tipove podataka.

Općeniti oblik SQL naredbe sa korištenjem agregatnih funkcija je:

```
SELECT aggregate(naziv_stupca), aggregate(naziv_stupca)
FROM tablice
WHERE uvjetni izraz
```

Npr.



**UPIT:** Izračunaj prosječnu ocjenu svih položenih ispita.

**SELECT AVG(OCJENA) FROM POLAŽE;**

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] \* of 8

Rezultirajuća relacija je:

Q...	
Expr1000	
▶	3,375
Record: [◀] [◀] [▶]	



**UPIT:** Izračunaj prosječnu ocjenu svih položenih ispita u 2005 godini.

```
SELECT AVG(OCJENA) FROM POLAŽE  
WHERE YEAR(DATUM_POLAGANJA)=2005;
```

Polaže : Table				
	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Rezultirajuća relacija je:

Q...	
Expr1000	
▶	4
Record: [◀] [◀] [▶] [▶*] of 8	



**UPIT:** Pronađi koja je bila minimalna ocjena iz kolegija "Baze podataka"

```
SELECT MIN(OCJENA) FROM POLAŽE,KOLEGIJ  
WHERE  
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA  
AND  
NAZ_KOLEGIJA = "Baze podataka";
```

Polaže : Table				
	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

**Kolegij : Table**

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶ +	110	Baze podataka	Perić
▶ +	220	Programiranje za internet	Jurić
▶ +	330	Numerička analiza	Franić
▶ +	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 4

Rezultirajuća relacija je:

**Expr1000**

Expr1000	2
----------	---

Record: [◀] [◀] [1]



**UPIT:** Pronađi koliko je studenata polagalo ispit iz predmeta "Baze podataka".

```
SELECT COUNT(*) FROM POLAŽE,KOLEGIJ
WHERE
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
AND
NAZ_KOLEGIJA = "Baze podataka";
```

**Polaže : Table**

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*			1	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 8

**Kolegij : Table**

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶ +	110	Baze podataka	Perić
▶ +	220	Programiranje za internet	Jurić
▶ +	330	Numerička analiza	Franić
▶ +	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 4

Rezultirajuća relacija je:

**Expr1000**

Expr1000	2
----------	---

Record: [◀] [◀] [1]

## 7.10. Grupni upiti (GROUP BY Query)

Opći oblik SQL naredbe za postavljanje grupnih upita je:

---

```

SELECT [ALL | DISTINCT] nazivi stupaca, agregatne kolone
FROM nazivi_tablica | join veze
[WHERE uvjetni izraz]
[GROUP BY nazivi stupaca]
[HAVING uvjetni izraz]
[ORDER BY sort_izraz]

```

### 7.10.1. GROUP BY

Određuje grupe po kojima se dijele izlazni rezultati, dobiveni pomoću select izraza, te primjenu agregatnih funkcija nad pojedinim grupama. GROUP BY izraz može se primijeniti na sve kolone, osim onih tipa *text*.

Primjenom GROUP BY izraza, rezultati dobiveni dijelom izraza

```

SELECT nazivi_stupca
FROM tablice | join veze
WHERE uvjetni izraz

```

grupiraju se na način da oni redovi izlaznog rezultata koji u svim stupacima navedenim u GROUP BY izrazu imaju iste vrijednosti spadaju u istu grupu, nad kojom se može zadati određena agregatna funkcija.



**UPIT:** Pronađi koliko je kojih ocjena dobiveno na ispitima iz svih predmeta.

```

SELECT OCJENA, COUNT(*) FROM POLAŽE
GROUP BY OCJENA

```

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] \* of 8

Rezultirajuća relacija je:

	OCJENA	Expr1001
▶	1	1
	2	1
	3	2
	4	2
	5	2

Record: [◀] [◀] [1] [▶] [▶]



**UPIT:** Pronađi prosjek ocjena po predmetima.

```

SELECT NAZ_KOLEGIJA, AVG(OCJENA) AS
[SREDNJA OCJENA]
FROM POLAŽE INNER JOIN KOLEGIJ ON
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
GROUP BY KOLEGIJ.NAZ_KOLEGIJA

```

The screenshot shows two Microsoft Access tables side-by-side.

**Polaže : Table**

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
				1

Record: [◀] [◀] [▶] [▶] [▶] [▶] \* of 8

**Kolegij : Table**

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶ +	110	Baze podataka	Perić
▶ +	220	Programiranje za internet	Jurić
▶ +	330	Numerička analiza	Franić
▶ +	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [▶] [▶] [▶] [▶] \* of 4

Rezultirajuća relacija je:

The screenshot shows a Microsoft Access query result window titled "Query1 : Select Query".

	NAZ_KOLEGIJA	SREDNJA OCJ
▶	Baze podataka	3,5
	Numerička analiza	3,5
	Programiranje za internet	2
	Programski prevoditelji	4,5

Record: [◀] [◀] [▶] [▶] [▶] [▶] \* of 4

Djelovanje GROUP BY operatora podrazumijeva grupiranje izlaznih rezultata na način da se svi redovi koji u stupacama navedenim u GROUP BY izrazu imaju iste vrijednosti, povezuju u zajedničku grupu. U gornjem primjeru grupa je određena stupacama NAZ\_KOLEGIJA, što znači da se svi redovi koji u ovim stupacama imaju istu vrijednost, povezuju u zajedničku grupu, nad kojom se provodi određena agregatna funkcija.

## 7.10.2. HAVING uvjetni izraz

Predstavlja ograničenje izlaznih rezultata po zadanim uvjetima nakon izvršenja GROUP BY operacije. Osnovna razlika između WHERE i HAVING izraza jest u činjenici da WHERE izraz ograničava izlazne rezultate prije njihovog grupiranja tj. uvjetuje broj redova koji ulaze u grupiranje, dok HAVING izraz postavlja uvjet na izlazne rezultate nakon što je grupiranje završeno.



**UPIT:** Prikaži predmete kod kojih je prosjek ocjena <= 3.

```
SELECT NAZ_KOLEGIJA, AVG(OCJENA) AS  
[SREDNJA OCJENA]  
FROM POLAŽE INNER JOIN KOLEGIJ ON  
POLAŽE.ŠIF_KOLEGIJA = KOLEGIJ.ŠIF_KOLEGIJA  
GROUP BY KOLEGIJ.NAZ_KOLEGIJA HAVING  
AVG(OCJENA) <= 3
```

Polaže : Table				
	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*				1

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 8

Kolegij : Table			
	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶	110	Baze podataka	Perić
+	220	Programiranje za internet	Jurić
+	330	Numerička analiza	Franić
+	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 4

Rezultirajuća relacija je:

Query1 : Select Query		
	NAZ_KOLEGIJA	SREDNJA OCJ
▶	Programiranje za internet	2
*		

 **UPIT:** Prikaži koliko je koji student položio ispita.

```
SELECT STUDENT.BR_INDEX,  
IME_STUDENTA,COUNT("OCJENA") AS  
[POLOZIO_PREDMETA]  
FROM POLAŽE INNER JOIN STUDENT ON  
POLAŽE.BR_INDEX=STUDENT.BR_INDEX  
GROUP BY STUDENT.BR_INDEX, IME_STUDENTA
```

Polaže : Table

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 8

Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko		1	2005 I	Split
+ +	20021	Jure		2	2003 R	Makarska
+ +	20022	Marko		2	2003 R	Split
+ +	20456	Zdravko		2	2004 R	Hvar
* +	30354	Neno		3	2002 I	
				1	0	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 5

Rezultirajuća relacija je:

Query1 : Select Query

	BR_INDEX	IME_STUDENTA	POLOZIO_PRE
▶	10020	Branko	4
	20022	Darko	2
	20456	Zdravko	1
*	30354	Neno	1

 UPIT: Prikaži koliko je koji student položio ispita ocjenom većom ili jednakom 4.

```
SELECT STUDENT.BR_INDEX,
       IME_STUDENTA,COUNT("OCJENA") AS
       [POLOZIO_PREDMETA]
  FROM POLAŽE INNER JOIN STUDENT ON
  POLAŽE.BR_INDEX=STUDENT.BR_INDEX
 WHERE OCJENA >= 4
 GROUP BY STUDENT.BR_INDEX, IME_STUDENTA
```

Polaže : Table

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 8

Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko		1	2005	I
▶ +	20021	Jure		2	2003	R
▶ +	20022	Marko		2	2003	R
▶ +	20456	Zdravko		2	2004	R
▶ +	30354	Neno		3	2002	I
*				1	0	

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶\*] of 5

Rezultirajuća relacija je:

Query1 : Select Query

	BR_INDEX	IME_STUDENTA	POLOZIO_PRE
▶	10020	Branko	3
	30354	Neno	1

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶\*] of 2



**UPIT:** Prikaži koliko je studenata položilo 2 ili više ispita

```
SELECT STUDENT.BR_INDEX,
       IME_STUDENTA,COUNT("OCJENA") AS
       [POLOZIO_PREDMETA]
  FROM POLAŽE INNER JOIN STUDENT ON
    POLAŽE.BR_INDEX=STUDENT.BR_INDEX
 GROUP BY STUDENT.BR_INDEX, IME_STUDENTA
 HAVING COUNT("OCJENA") >=2
```

Polaže : Table

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*			1	

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶\*] of 8

Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_RODENJA
▶ +	10020	Branko		1	2005	I
▶ +	20021	Jure		2	2003	R
▶ +	20022	Marko		2	2003	R
▶ +	20456	Zdravko		2	2004	R
▶ +	30354	Neno		3	2002	I
*				1	0	

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶\*] of 5

Rezultirajuća relacija je:

Query1 : Select Query

	BR_INDEX	IME_STUDENTA	POLOZIO_PRE
▶	10020	Branko	4
	20022	Darko	2

Record: [◀] [◀] [◀] [1] [▶] [▶] [▶\*] of 2



**UPIT:** Prikazati statistiku ocjena studenata iz pojedinih predmeta u obliku

ime_predmeta	ocjena	broj studenata koji su dobili ocjenu
--------------	--------	--------------------------------------

```
SELECT NAZ_KOLEGIJA, OCJENA, COUNT("OCJENA") AS [BROJ OCJENA]
FROM POLAŽE INNER JOIN KOLEGIJ ON
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
GROUP BY KOLEGIJ.NAZ_KOLEGIJA, OCJENA
```

Polaže : Table				
	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
				1

Record: [◀] [◀] [1] [▶] [▶] [▶] [▶] [\*] of 8

Kolegij : Table			
	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶	110	Baze podataka	Perić
+	220	Programiranje za internet	Jurić
+	330	Numerička analiza	Franić
+	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [1] [▶] [▶] [▶] [▶] [\*] of 4

Rezultirajuća relacija je:

Query1 : Select Query			
	NAZ_KOLEGIJA	OCJENA	BROJ OCJENA
▶	Baze podataka	2	1
	Baze podataka	5	1
	Numerička analiza	3	1
	Numerička analiza	4	1
	Programiranje za internet	1	1
	Programiranje za internet	3	1
	Programski prevoditelji	4	1
	Programski prevoditelji	5	1

Record: [◀] [◀] [1] [▶] [▶] [▶] [▶] [\*] of 8



**UPIT:** Prikaži prosječnu ocjenu po predmetima i mjestu rođenja.

```
SELECT MJ_ROĐENJA, NAZ_KOLEGIJA,
AVG(OCJENA) AS [SREDNJA OCJENA]
FROM (STUDENT INNER JOIN POLAŽE ON
STUDENT.BR_INDEX=POLAŽE.BR_INDEX) INNER
JOIN KOLEGIJ ON
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
GROUP BY MJ_ROĐENJA, KOLEGIJ.NAZ_KOLEGIJA
```

Isti upit uz uvjet da je mjesto rođenja poznato glasi:

```
SELECT MJ_ROĐENJA, NAZ_KOLEGIJA,
AVG(OCJENA) AS [SREDNJA OCJENA]
FROM (STUDENT INNER JOIN POLAŽE ON
STUDENT.BR_INDEX=POLAŽE.BR_INDEX) INNER
JOIN KOLEGIJ ON
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
WHERE MJ_ROĐENJA IS NOT NULL
GROUP BY MJ_ROĐENJA,KOLEGIJ.NAZ_KOLEGIJA
```

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶ +	110	Baze podataka	Perić
▶ +	220	Programiranje za internet	Jurić
▶ +	330	Numerička analiza	Franić
▶ +	440	Programski prevoditelji	Antić
*			

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*			1	

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko	1	2005	I	Split
▶ +	20021	Jure	2	2003	R	Makarska
▶ +	20022	Marko	2	2003	R	Split
▶ +	20456	Zdravko	2	2004	R	Hvar
▶ +	30354	Neno	3	2002	I	
*			1	0		

Rezultirajuća relacija je:

	MJ_ROĐENJA	NAZ_KOLEGIJA	SREDNJA OCJ
▶		Programski prevoditelji	4
	Hvar	Programiranje za internet	1
	Split	Baze podataka	3,5
	Split	Numerička analiza	3,5
	Split	Programiranje za internet	3
	Split	Programski prevoditelji	5
Record:	[◀] [◀] [▶] [▶] [▶*]	of 6	

## 7.11. Ugnježdeni upiti - podupiti

Podupit (Subquery) je SELECT izraz uključen u glavni SELECT,

INSERT, UPDATE ili DELETE upit. Upit koji uključuje podupit (subquery), svoj uvjetni dio izraza temelji na rezultatima koji su dobiveni izvršenjem podupita. Pri tome podupit i glavni upit mogu djelovati ne istu tablicu ili na različite tablice. Mnogi SQL izrazi koji uključuju podupite mogu se realizirati i stvaranjem upita sa povezivanjem (join).

Podupit se uključuje u glavni upit u uvjetnom dijelu izraza tj. u dijelu WHERE ili HAVING.

```
SELECT [ALL | DISTINCT] nazivi stupaca, agregatne kolone
FROM nazivi_tablica| join veze
[WHERE naziv_stupca predikat (subquery)]
[GROUP BY nazivi stupaca]
[HAVING naziv_stupca predikat (subquery)]
```

pri čemu podupit ima sintaksu slijedećeg oblika

```
SELECT [ALL | DISTINCT] nazivi stupaca, agregatne kolone
FROM nazivi_tablica| join veze
[WHERE uvjetni izraz]
[GROUP BY nazivi stupaca]
[HAVING uvjetni izraz]
```

Prilikom izvršenja SQL izraza koji uključuje podupit, najprije se izvršava podupit, a rezultati koje podupit daje postaju dio uvjetnog izraza glavnog upita. Primjena podupita podrazumijeva postojanje određenih ograničenja i pravila:

U select listi (lista stupaca) koje se dobijaju kao rezultat podupita može biti sadržan naziv samo jedne kolone, osim u slučajevima primjene podupita postojanja (Exists).

- Podupiti sa operatorom usporedbe moraju uvijek davati samo jednu vrijednost, te ne mogu uključivati grupiranje (GROUP BY i HAVING strukturu).
- Modifikator DISTINCT ne može se primijeniti sa podupitim kojima uključuju GROUP BY izraz.
- Podupit, tj. select izraz kojim se podupit definira uvijek mora biti naveden u zagradama.
- U sklopu podupita nije moguće koristiti ORDER BY izraze.

Postoje tri osnovna oblika podupita (subqueries):

1. podupiti liste na koje se primjenjuje IN predikat
2. podupiti sa predikatom usporedbe
3. podupiti kojima se ispituje postojanje određenih podataka

### 7.11.1. Podupiti liste

Podupit vraća niz rezultata (listu), koju glavni upit koristi u svom uvjetnom izrazu preko predikata liste (IN ili NOT IN).

Npr.



**UPIT:** Prikaži studente koji su polagali bilo koji ispit u 2004 godini. Neka prikazani podaci budu uzlazno posloženi po imenu studenta.

```
SELECT DISTINCT BR_INDEX, IME_STUDENTA
FROM STUDENT
WHERE BR_INDEX IN (SELECT BR_INDEX FROM
POLAŽE
WHERE YEAR(DATUM_POLAGANJA) =2004)
ORDER BY IME_STUDENTA
```

Polaže : Table				
	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] \* of 8

Student : Table						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko	1	2005	I	Split
+ +	20021	Jure	2	2003	R	Makarska
+ +	20022	Marko	2	2003	R	Split
+ +	20456	Zdravko	2	2004	R	Hvar
+ +	30354	Neno	3	2002	I	
*			1	0		

Record: [◀] [◀] [1] [▶] [▶] [▶] \* of 5

Rezultirajuća relacija je:

Query1 : Select Q...		
	BR_INDEX	IME_STUDENTA
▶	10020	Branko
	20022	Darko
	20456	Zdravko

Record: [◀] [◀] [1] [▶] [▶]

Izraz se izvršava u dva koraka. Prvo podupit iz tablice POLAŽE vraća BR\_INDEX za sve studente koji imaju su polagali ispite u 2004 god.)

Rezultat je lista sa podacima BR\_INDEX. Potom glavni upit iz tablice STUDENT vraća podatke o svim studentima čiji se BR\_INDEX nalazi u listi koristeći predikat liste (IN).

Isti problem može se realizirati i primjenom veze(join) između tablica STUDENT i POLAŽE.

```
SELECT DISTINCT STUDENT.BR_INDEX, IME_STUDENTA  
FROM STUDENT INNER JOIN POLAŽE ON  
STUDENT.BR_INDEX=POLAŽE.BR_INDEX  
WHERE YEAR(DATUM_POLAGANJA)=2004  
ORDER BY IME_STUDENTA
```



**UPIT:** Prikazati studente koji nisu polagali ni jedan ispit.

```
SELECT DISTINCT BR_INDEX, IME_STUDENTA  
FROM STUDENT  
WHERE BR_INDEX NOT IN  
(SELECT BR_INDEX FROM POLAŽE)  
ORDER BY IME_STUDENTA
```

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*				1

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 8

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶	10020	Branko	1	2005	I	Split
+	20021	Jure	2	2003	R	Makarska
+	20022	Marko	2	2003	R	Split
+	20456	Zdravko	2	2004	R	Hvar
+	30354	Neno	3	2002	I	
*			1	0		

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 5

Rezultirajuća relacija je:

Query1 : Select Q...	
	BR_INDEX
▶	20021 Jure

Record: [◀] [◀] [1] [▶] [▶]

## 7.11.2. Podupiti sa predikatom usporedbe

Rezultat podupita može se u glavni upit uključiti preko jednog od predikata usporedbe (=,<,>,<,....)

Takva primjena podupita podrazumijeva da podupit mora vratiti samo jednu vrijednost, a ne kao u prethodnom primjeru listu vrijednosti. Ukoliko podupit vraća više od jedne vrijednosti javlja se greška u izvršavanju

glavnog upita.

Npr.



**UPIT:** Prikazati sve studente koji su rođeni u istom mjestu kao i student sa brojem indeksa 20022.

```
SELECT BR_INDEX, IME_STUDENTA FROM
STUDENT WHERE MJ_ROĐENJA =
(SELECT MJ_ROĐENJA FROM STUDENT AS
STUDENT_1 WHERE BR_INDEX=20022)
ORDER BY IME_STUDENTA
```

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶	+ 10020	Branko	1	2005	I	Split
▶	+ 20021	Jure	2	2003	R	Makarska
▶	+ 20022	Marko	2	2003	R	Split
▶	+ 20456	Zdravko	2	2004	R	Hvar
*	+ 30354	Neno	3	2002	I	
			1	0		

Rezultirajuća relacija je:

Query1 : Select ...		
	BR_INDEX	IME_STUDENTA
▶	+ 10020	Branko
▶	+ 20022	Darko
*		



**UPIT:** Izbrisati podatke o položenim ispitima u 2004 godini za studenta koji se zove "Darko"

```
DELETE FROM POLAŽE WHERE
YEAR(DATUM_POLAGANJA)=2004 AND BR_INDEX
= (SELECT BR_INDEX FROM STUDENT WHERE
IME_STUDENTA="Darko")
```



**UPIT:** Za studenta koji se zove "Branko" prikazati sve položene ispite osim posljednjeg.

```
SELECT NAZ_KOLEGIJA, OCJENA
FROM STUDENT, POLAŽE, KOLEGIJ
WHERE
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
AND POLAŽE.BR_INDEX=STUDENT.BR_INDEX AND
IME_STUDENTA="Branko" AND
DATUM_POLAGANJA <
(SELECT MAX(DATUM_POLAGANJA) FROM
POLAŽE WHERE BR_INDEX=( SELECT BR_INDEX
FROM STUDENT WHERE
```

IME\_STUDENTA="Branko"))

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶ +	110	Baze podataka	Perić
▶ +	220	Programiranje za internet	Jurić
▶ +	330	Numerička analiza	Franić
▶ +	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 4

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*			1	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 8

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko		1	2005 I	Split
▶ +	20021	Jure		2	2003 R	Makarska
▶ +	20022	Marko		2	2003 R	Split
▶ +	20456	Zdravko		2	2004 R	Hvar
▶ +	30354	Neno		3	2002 I	
*				1	0	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 5

Rezultirajuća relacija je:

Query1 : Select Query		
	NAZ_KOLEGIJA	OCJENA
▶	Baze podataka	5
	Programiranje za internet	3
	Numerička analiza	4

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 3



**UPIT:** Studentu koji se zove "Darko" i trenutno je na 2 godini studija promjeni ocjenu iz kolegija "Baze podataka" u 4.

```
UPDATE POLAŽE SET OCJENA=4 WHERE
    BR_INDEX=(SELECT BR_INDEX FROM STUDENT
    WHERE IME_STUDENTA="Darko" AND
    GOD_STUDIJA=2) AND ŠIF_KOLEGIJA=(SELECT
    ŠIF_KOLEGIJA FROM KOLEGIJ WHERE
    NAZ_KOLEGIJA="Baze podataka")
```

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [▶] \* of 8

Rezultirajuća relacija je:

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	4	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [▶] \* of 8

Poseban slučaj podupita sa predikatom usporedbe predstavlja primjena modifikatora ANY - bilo koji i ALL - svi. U slučaju primjene ovih modifikatora podupiti sa predikatom usporedbe mogu vraćati listu vrijednosti, a ne samo jednu vrijednost. Ovi modifikatori primjenjuju se u obliku

```
SELECT [ALL | DISTINCT] nazivi stupaca, agregatne kolone
FROM nazivi_tablica| join veze
WHERE naziv_stupca predikat_usporedbe ANY| ALL
(subquery)]
```

Na primjer, operator > (veći od) primjenjen sa modifikatorom ALL , >ALL znači veći od svih vrijednosti koje se nalaze u listi, tj. veći od maksimalne vrijednosti iz liste. >ALL (1,2,3) znači veći od 3. Sa druge strane modifikator ANY znači bilo koji (odnosno barem jedan). > ANY znači veći od bilo kojeg, tj. veći od najmanje vrijednosti u listi. > ANY (1,2,3) znači veći od 1.

Modifikator	Način djelovanja
> ANY (lista vrijednosti)	>(min(lista_vrijednosti))
> ALL (lista vrijednosti)	>(max(lista_vrijednosti))
< ANY (lista vrijednosti)	<(max(lista_vrijednosti))
< ALL (lista vrijednosti)	<(min(lista_vrijednosti))
= ANY (lista vrijednosti)	IN(lista_vrijednosti)
= ALL (lista vrijednosti)	Nije dozvoljeno (greška)

<> ALL (lista vrijednosti)

NOT IN (lista vrijednosti)



**UPIT:** Za studenta koji se zove "Branko" prikazati sve položene ispite osim posljednjeg.

```
SELECT NAZ_KOLEGIJA, OCJENA,
DATUM_POLAGANJA FROM POLAŽE,
STUDENT,KOLEGIJ WHERE
POLAŽE.BR_INDEX=STUDENT.BR_INDEX AND
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA
AND POLAŽE.BR_INDEX = (SELECT BR_INDEX
FROM STUDENT WHERE
IME_STUDENTA="Branko") AND
DATUM_POLAGANJA < ANY (SELECT
DATUM_POLAGANJA FROM POLAŽE WHERE
BR_INDEX=( SELECT BR_INDEX FROM STUDENT
WHERE IME_STUDENTA="Branko"))
```

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶	110	Baze podataka	Perić
▶	220	Programiranje za internet	Jurić
▶	330	Numerička analiza	Franić
▶	440	Programski prevoditelji	Antić
*			

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 4

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
▶	10020	220	3	7.6.2004
▶	10020	330	4	15.6.2005
▶	10020	440	5	9.9.2005
▶	20022	110	2	1.6.2004
▶	20022	330	3	15.6.2005
▶	20456	220	1	1.6.2004
▶	30354	440	4	15.9.2005
*			1	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 8

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶	10020	Branko	1	2005	I	Split
▶	20021	Jure	2	2003	R	Makarska
▶	20022	Marko	2	2003	R	Split
▶	20456	Zdravko	2	2004	R	Hvar
▶	30354	Neno	3	2002	I	
*			1	0		

Rezultirajuća relacija je:

 Query1 : Select Query

NAZ_KOLEGIJA	OCJENA	DATUM_POLAGANJA
Baze podataka	5	3.2.2004
Programiranje za internet	3	7.6.2004
Numerička analiza	4	15.6.2005

Record:   | 1   \* of 3



**UPIT:** Prikazati sve studente koji su polagali ispite prije između 01.01.2005 i 30.06.2005 godine.

```
SELECT DISTINCT STUDENT.BR_INDEX,
IME_STUDENTA
FROM STUDENT
WHERE STUDENT.BR_INDEX=ANY (SELECT
BR_INDEX FROM POLAŽE WHERE
DATUM_POLAGANJA BETWEEN #1/1/2005# AND
#30/6/2005#)
ORDER BY IME_STUDENTA
```

 Polaže : Table

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record:   | 1   \* of 8

 Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko	1	2005	I	Split
+ +	20021	Jure	2	2003	R	Makarska
+ +	20022	Marko	2	2003	R	Split
+ +	20456	Zdravko	2	2004	R	Hvar
+ +	30354	Neno	3	2002	I	
*			1	0		

Record:   | 1   \* of 5

Rezultirajuća relacija je:

 Query1 : Select ...

BR_INDEX	IME_STUDENTA
10020	Branko
20022	Darko

Record:   | 1  

## Korelirani pod-upiti (Cross-correlated queries)

Prethodni primjeri pod-upita bili su izraženi na način da se podupit izvršio samo jednom i rezultati koje vraća podupit su se proslijedivali u uvjetni dio glavnog upita. U upitim kojima uključuju korelirane podupite, izračunavanje podupita zavisi od glavnog upita, što znači da se podupit izvodi za svaki

podatak glavnog upita.



**UPIT:** Prikaži sve studente koji su polagali ispite iz kolegija "Baze podataka"

```
SELECT * FROM POLAŽE WHERE ŠIF_KOLEGIJA=
(SELECT ŠIF_KOLEGIJA FROM KOLEGIJ
WHERE NAZ_KOLEGIJA="Baze podataka")
```

	ŠIF_KOLEGIJA	NAZ_KOLEGIJA	PREZ_NASTAVNIKA
▶	+ 110	Baze podataka	Perić
▶	+ 220	Programiranje za internet	Jurić
▶	+ 330	Numerička analiza	Franić
▶	+ 440	Programski prevoditelji	Antić
*			

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
	30354	440	4	15.9.2005
*			1	

Rezultirajuća relacija je:

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	20022	110	4	1.6.2004
	10020	110	5	3.2.2004
*			1	

Ovo je primjer pod-upita koji je nezavisan u glavnom upitu, te se izvodi samo jednom.

Isti rezultat može se dobiti pomoću koreliranog podupita

```
SELECT * FROM POLAŽE WHERE "Baze
podataka"=(SELECT NAZ_KOLEGIJA FROM KOLEGIJ
WHERE POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA)
```

Izvršavanje upita počinje od tablice POLAŽE. Uzima se prvi red u tablici POLAŽE, te se za taj ispit vadi vrijednost ŠIF\_KOLEGIJA i proslijedi u podupit. Podupit traži redak u tablici KOLEGIJ gdje je vrijednost ŠIF\_KOLEGIJA jednaka onoj koja odgovara prvom ispitu. Kad nađe na takav podatak u tablici KOLEGIJ, provjerava NAZ\_KOLEGIJA i

uspoređuje sa zadanim vrijednošću ('Baze podataka'), i tako za sve redove u tablici POLAŽE.



**UPIT:** Izraditi korelirani podupit kojim se ispisuju podaci o studentima koji su polagali ispite u 2004 godini.

```
SELECT IME_STUDENTA FROM STUDENT where  
BR_INDEX IN (SELECT BR_INDEX FROM POLAŽE  
WHERE STUDENT.BR_INDEX=POLAŽE.BR_INDEX  
AND YEAR(DATUM_POLAGANJA)=2004)
```

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 8

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶	+ 10020	Branko	1	2005	I	Split
	+ 20021	Jure	2	2003	R	Makarska
	+ 20022	Marko	2	2003	R	Split
	+ 20456	Zdravko	2	2004	R	Hvar
*	30354	Neno	3	2002	I	
			1	0		

Record: [◀] [◀] [1] [▶] [▶] [▶\*] of 5

Rezultirajuća relacija je:

	IME_STUDENTA
▶	Darko
	Branko
	Zdravko
*	

### 7.11.3. Podupiti sa predikatom postojanja (EXISTS, NOT EXISTS)

Operator EXISTS ispituje postojanje podataka uz navedene uvjete i najčešći je način korištenja koreliranih podupita. U ovakvoj strukturi ugnježđenog upita, uvjetni izraz glavnog upita ispituje postojanje podataka koje vraća podupit. Uvjetni izraz glavnog upita provjerava postojanje podataka u podupitu. Podupit sa predikatom postojanja ne vraća nikakve podatke, već daje samo rezultat TRUE ili FALSE.



**UPIT:** Prikazati sve studente nisu polagali ni jedan ispit 2004 godine.

```

SELECT BR_INDEX, IME_STUDENTA FROM
STUDENT WHERE NOT EXISTS (SELECT * FROM
POLAŽE WHERE
STUDENT.BR_INDEX=POLAŽE.BR_INDEX AND
YEAR(DATUM_POLAGANJA) = 2004)

```

**Polaže : Table**

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 8

**Student : Table**

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶	+ 10020	Branko		1 2005	I	Split
	+ 20021	Jure		2 2003	R	Makarska
	+ 20022	Marko		2 2003	R	Split
	+ 20456	Zdravko		2 2004	R	Hvar
*	30354	Neno		3 2002	I	
				1 0		

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 5

Rezultirajuća relacija je:

**Query1 : Select ...**

	BR_INDEX	IME_STUDENTA
▶	30354	Neno
	20021	Jure
*		

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*]



**UPIT:** Prikazati sva mjesta (poznata) u kojima su rođeni studenti koji su polagali barem jedan ispit.

```

SELECT DISTINCT MJ_ROĐENJA FROM STUDENT
WHERE MJ_ROĐENJA IS NOT NULL AND EXISTS
(SELECT * FROM POLAŽE WHERE
STUDENT.BR_INDEX=POLAŽE.BR_INDEX)

```

**Polaže : Table**

	BR_INDEX	ŠIF_KOLEGIJA	OCJENA	DATUM_POLAGANJA
▶	10020	110	5	3.2.2004
	10020	220	3	7.6.2004
	10020	330	4	15.6.2005
	10020	440	5	9.9.2005
	20022	110	2	1.6.2004
	20022	330	3	15.6.2005
	20456	220	1	1.6.2004
*	30354	440	4	15.9.2005
			1	

Record: [◀] [◀] [1] [▶] [▶] [▶] [\*] of 8

Student : Table

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
▶ +	10020	Branko	1	2005	I	Split
▶ +	20021	Jure	2	2003	R	Makarska
▶ +	20022	Marko	2	2003	R	Split
▶ +	20456	Zdravko	2	2004	R	Hvar
*	30354	Neno	3	2002	I	
			1	0		

Record: [◀] [◀] [▶] [▶] [▶] [▶] of 5

Rezultirajuća relacija je:

Q... [ ] [ ] [ ]

MJ_ROĐENJA
Hvar
Split

Record: [◀] [◀] [ ]

## 7.12. Unija

Operator unije (UNION) definira relacijsku operaciju unije, tj. povezuje rezultate više upita u jedinstveni skup rezultata.

Sintaksa:

```
upit1 UNION [ALL]
upit2 ...
ORDER BY sort
```

Pri formiranju unije mora vrijediti:

- upiti čiji se rezultati uniraju moraju imati jednak broj izlaznih stupaca
- odgovarajući izlazni stupci u upitim kojih se povezuju unijom moraju biti istog tipa podataka

Npr.



**UPIT:** Prikaži sve studente koji su upisani 2003 i 2004 god.

```
SELECT BR_INDEX, IME_STUDENTA , GOD_UPISA
FROM STUDENT
WHERE GOD_UPISA = 2003 OR GOD_UPISA =
2004;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: Query

	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	20021	Jure	2	2003	R	Makarska

	20022	Marko	2	2003	R	Split	
	20456	Zdravko	2	2004	R	Hvar	

Do istog smo rezultata mogli doći primjenom unije:

```
SELECT * FROM STUDENT
WHERE GOD_UPISA = 2003
UNION
SELECT * FROM STUDENT
WHERE GOD_UPISA = 2004;
```

[PRIKAŽI POLAZNE TABLICE I REZULTATE POJEDINIH UPITA]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	20021	Jure	2	2003	R	Makarska
	20022	Marko	2	2003	R	Split
	20456	Zdravko	2	2004	R	Hvar

UNION operator uklanja duplicitirane redove rezultata obaju upita.



**UPIT:** Prikaži sve studente koji su upisani 2003 god. i studente koji su rođeni u Splitu.

```
SELECT * FROM STUDENT
WHERE GOD_UPISA = 2003
UNION
SELECT * FROM STUDENT
WHERE MJ_ROĐENJA = "Split"
ORDER BY BR_INDEX;
```

[PRIKAŽI POLAZNE TABLICE I REZULTATE POJEDINIH UPITA]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	10020	Branko	1	2005	I	Split
	20021	Jure	2	2003	R	Makarska
	20022	Marko	2	2003	R	Split

Ukoliko se želi prikaz svih redova, bez eliminacije duplikata, potrebno je navesti opciju ALL.

```

SELECT * FROM STUDENT
WHERE GOD_UPISA = 2003
UNION ALL
SELECT * FROM STUDENT
WHERE MJ_ROĐENJA = "Split"
ORDER BY BR_INDEX;

```

[PRIKAŽI POLAZNE TABLICE I REZULTATE POJEDINIH UPITA]

Rezultat ovog upita je tablica:

Tablica: Query						
	BR_INDEX	IME_STUDENTA	GOD_STUDIJA	GOD_UPISA	TIP_STUDIJA	MJ_ROĐENJA
	10020	Branko		1	2005	I
	20021	Jure		2	2003	R
	20022	Marko		2	2003	R
	20022	Marko		2	2003	R

Kod primjene operatora unije, pojedinačni upiti ne mogu imati svoje sortiranje (ORDER BY dio), već je primjena ORDER BY operatora moguća nakon navođenja posljednjeg upita koji sudjeluje u uniji, i sortiranje se primjenjuje na završni skup rezultata (uniju). Rezultat uniranja više upita je skup rezultata čije kolone nose naziv isti kao i kolone rezultata prvog upita koji se navodi u uniji.

Broj UNION operatora u okviru SQL izraza nije ograničen, tj. moguće je uniranje po volji velikog broja upita i njihovih rezultata.

```

SELECT * FROM TableA
UNION
SELECT * FROM TableB
UNION
SELECT * FROM TableC
...

```

## 7.13 POGLED (VIEW)

Pogledi su virtualne tablice, koje se formiraju kao posebna organizacijska jedinica u bazi podataka.

Osnovne tablice postoje kao stvarne, fizičke tablice u bazi podataka. One imaju svoju definiciju koja je pohranjena u sistemskom katalogu baze podataka, svoje podatke koji su pohranjeni na disku i svoje indekse. Pogled (view), kao virtualna tablica, nema svoje indekse, niti datoteke s podacima, ali mu korisnik pristupa kao i svakoj drugoj tablici.

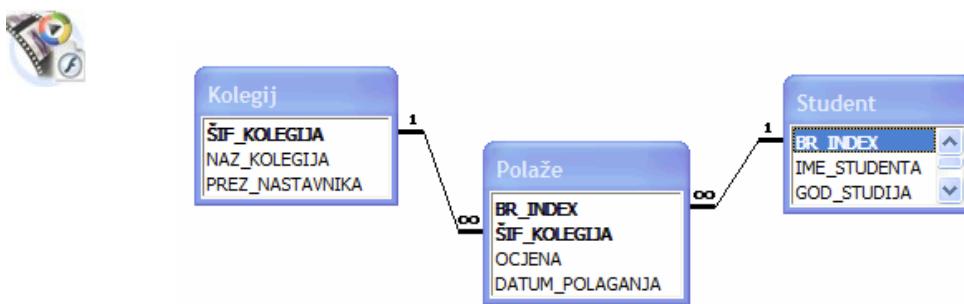
## 7.13.1. Kreiranje pogleda

Pogled se definira naredbom

```
CREATE VIEW naziv_pogleda AS  
SELECT ...select query  
[ WITH CHECK ]
```

U sklopu definicije pogleda moguće je navesti bilo koji select query, obični ili grupni uz postojeća pravila u okviru SQL-a. Atributi pogleda imaju iste nazive kao i atributi tablica nad kojima je pogled definiran.

Npr.



Kreirajte pogled formiran nad tablicama STUDENT, POLAŽE i KOLEGIJ, sa izlaznim atributima BR\_INDEX, IME\_STUDENTA i NAZ\_KOLEGIJA.

```
CREATE VIEW POPISPREDMETA AS  
SELECT STUDENT.BR_INDEX,  
STUDENT.IME_STUDENTA, KOLEGIJ.NAZ_KOLEGIJA  
FROM (STUDENT INNER JOIN POLAŽE ON  
STUDENT.BR_INDEX=POLAŽE.BR_INDEX)  
INNER JOIN KOLEGIJ ON  
POLAŽE.ŠIF_KOLEGIJA=KOLEGIJ.ŠIF_KOLEGIJA;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: POPISPREDMETA			
	BR_INDEX	IME_STUDENTA	NAZ_KOLEGIJA
	20022	Marko	Baze podataka
	10020	Branko	Baze podataka
	10020	Branko	Programiranje za internet
	20022	Marko	Numerička analiza
	10020	Branko	Numerička analiza
	10020	Branko	Programski prevoditelji
	20456	Zdravko	Programiranje za internet

Ovakvom pogledu moguće je pristupati kao i svakoj drugoj tablici:



**UPIT:** Prikaži sve studente koji su polagali ispit iz kolegija "Baze podataka". Studente poredajte po abecedi.

```
SELECT * FROM POPISPREDMETA
WHERE NAZ_KOLEGIJA="Baze podataka"
ORDER BY IME_STUDENTA;
```

[PRIKAŽI/SAKRIJ POLAZNE TABLICE]

Rezultat ovog upita je tablica:

Tablica: POPISPREDMETA		
BR_INDEX	IME_STUDENTA	NAZ_KOLEGIJA
20022	Marko	Baze podataka
10020	Branko	Baze podataka

### 7.13.2. Preimenovanje atributa pogleda

Prilikom uobičajenog definiranja pogleda, njegovi atributi imaju iste nazive kao i atributi tablica nad kojima je pogled definiran. Ako postoji potreba za promjenom naziva pojedinih atributa, to se može ostvariti na tri načina:

1. Uobičajeno pravilo promjene naziva atributa u SELECT upitima tipa *novi\_naziv=atribut*

Primjer:

```
CREATE VIEW V_STUDENTI AS SELECT IME, PREZIME,
SK_GOD, UPISANI_SEMESTAR=SEM FROM STUDENT INNER JOIN
UPISNI_LIST
ON STUDENT.STUDENT_ID =UPISNI_LIST.STUDENT_ID
```

2. Korištenjem izraza AS u obliku: *atribut AS novi\_naziv*

Primjer: *CREATE VIEW V\_STUDENTI AS SELECT IME, PREZIME, SK\_GOD, SEM AS UPISANI\_SEM FROM STUDENT INNER JOIN UPISNI\_LIST ON STUDENT.STUDENT\_ID =UPISNI\_LIST.STUDENT\_ID*

3. Navođenjem novih naziva atributa u definiciji pogleda u obliku

```
CREATE VIEW naziv_pogleda (nazivi_atributa) AS
```

Primjer: *CREATE VIEW V\_STUDENTI (IME, PREZIME, SK\_GOD, UPISANI\_SEM)*

```
AS SELECT IME, PREZIME, SK_GOD, SEM FROM STUDENT INNER
JOIN UPISNI_LIST ON STUDENT.STUDENT_ID
=UPISNI_LIST.STUDENT_ID
```

### 7.13.3. Ažuriranje, unos i brisanje podataka u pogledu

Za razliku od stvarnih tablica, mogućnosti promjene podataka u pogledu podliježu znatnim ograničenjima.

Mijenjanje, unos i brisanje podataka u pogledu nije dozvoljeno

- ako je pogled formiran dohvaćanjem podataka iz više od jedne tablice
- ako je pogled formiran grupnim upitom (group by)
- ako je bilo koji atribut pogleda stvoren pomoću agregatne funkcije
- ako se pri formiraju pogleda koristi opcija DISTINCT
- ako je pogled formiran kao UNION query
- ako je pogled formiran pomoću složenog upita sa podupitom

Tako definirana ograničenja čine mogućnosti promjene podataka u pogledu vrlo ograničenim.

Dodatna ograničenja mogu se aktivirati korištenjem izraza WITH CHECK u definiciji pogleda. Pomoću ove opcije provjerava se da li se unosom ili promjenom podataka narušava struktura pogleda, tj. da li promjena ili unos podataka zadovoljava uvjet naveden u upitu. Ako se opcija WITH CHECK ne navede u pogled je moguće unijeti podatke koji narušavaju strukturu pogleda

Primjer:

```
CREATE VIEW V1_STUDENT AS
SELECT IME, PREZIME, ROD_DAT FROM STUDENT
WHERE ROD_DAT>'01.01.1980'
```

je pogled koji prikazuje sve studente koji su rođeni nakon 1. siječnja 1980. Ovakav pogled može se ažurirati i brisati, budući zadovoljava sva gore navedena pravila.

```
INSERT V1_STUDENT VALUES ( 'Z','D','01.05.1979')
```

što je isto kao

```
INSERT STUDENT(IME,PREZIME,ROD_DAT) VALUES (  
'Z','D','01.05.1979')
```

Primjer:

```
CREATE VIEW V1_STUDENT AS  
SELECT IME, PREZIME, ROD_DAT FROM STUDENT  
WHERE ROD_DAT>'01.01.1980' WITH CHECK
```

Uvođenjem opcije provjere (WITH CHECK) instrukcija

```
INSERT V1_STUDENT VALUES ('Z','D','01.05.1979')
```

neće dati rezultat jer novi podatak koji se unosi ne zadovoljava uvjet  
ROD\_DAT>'01.01.1980'

Ako se u tablici STUDENT nalazi podatak sa vrijednostima atributa

```
IME='P', PREZIME='R' , ROD_DAT='01.12.1980'
```

instrukcija *UPDATE V1\_STUDENT SET ROD\_DAT='01.11.1980' where  
IME='P' AND PREZIME='R'*

može se izvesti jer je zadovoljeno ograničenje ROD\_DAT>'01.01.1980'

#### 7.13.4. Uklanjanje pogleda

Pogled se uklanja iz baze podataka naredbom:

```
DROP VIEW naziv_pogleda
```

Primjer:

```
DROP VIEW V1_STUDENT
```

Očigledno je da se, najopćije gledano, nad pogledima ne mogu vršiti operacije ažuriranja baze podataka, jer pogled fizički ne postoji, a ažuriranje se može preneti na relacije od kojih je pogled sačinjen samo u izuzetnim slučajevima (kada je pogled izведен iz jedne relacije, ili kada se ažuriranje vrši samo nad jednom od relacija koje čine pogled i kada je prenos ažuriranja iz pogleda na baznu relaciju nedvosmislen). Neki komercijalni sistemi ne dozvoljavaju ažuriranje pogleda uopće, a neki dozvoljavaju u tačno navedenim posebnim slučajevima.

Upotreba i značaj pogleda u razvoju aplikacija nad relacijskom bazom podataka je višestruka:

- Ostvarivanje logičke nezavisnosti programa i podataka. Da bi se ostvarila nezavisnost programa od logičke strukture baze podataka, programe treba razvijati nad pogledima, a ne nad baznim relacijama. Ovo je posebno značajno u prototipskom razvoju, kada treba formirati poglede čak i potpuno ekvivalentne baznim relacijama. Na taj način aplikacioni programi se neće menjati, čak i ako se jedna bazna relacija "vertikalno" dekomponuje na dve, ili se izvrši spajanje relacija, ili izmena naziva atributa i slično. Jedino što je, tada, neophodno uraditi je redefinisanje odgovarajućeg pogleda.
- Pogledi omogućuju da se isti podaci, različitim korisnicima, prikažu na različite načine, upravo onako kako oni žele da ih vide. Time se značajno može pojednostaviti korišćenje podataka iz baze podataka.
- Pogledi omogućuju i jedan stepen zaštite podataka, jer se neki podaci u baznim tabelama, dajući korisniku na raspolaganje samo poglede, mogu sakriti.

#### 9.17. OSNOVE VIŠEKORISNIČKOG RADA

Zbog većeg broja korisnika koji pristupaju bazama podataka, administrator baze ima zadatku ograničiti prava pristupa i korištenja podataka. Osnovni razlozi ograničenja prava pristupa podacima u bazi su:

- mogućnosti neovlaštenog pristupa podacima
- nestručno rukovanje podacima
- moguće zloupotrebe

DBA- Database administrator ( Administrator baze podataka) , kao ključna osoba u održavanju baze podataka je korisnik najvišeg prioriteta, koji ima sva prava pristupa. Ovisno o zadacima koje obavljaju pojedini korisnici DBA formira nove korisnike u bazi

Upit se vrši primjenom slijedeće SQL naredbe:

```
SELECT BR_INDEX, IME_STUDENTA  
      FROM STUDENT  
     WHERE GOD_STUDIJA = 2 AND GOD_UPISA >= 2002 ;
```

MLB	IME
1010943..	Pera
0203955..	Mira

Ako se žele prikazati svi atributi neke relacije, umjesto njihovog navođenja može se napisati SELECT \*.

(3) Redoslijed n-torki u rezultatu nije poznat. Ako se želi određeni redoslijed u rezultatu upotrebljava se klauzula

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

Prikaži šifre i imena radnika iz Beograda starih između 22 i 30 godina, po opadajućem redoslijedu starosti.

```
SELECT RADNIK. RADN#, RADNIK. IME
```

```
      FROM RADNIK
```

```
      WHERE MJESTO_ROĐ = 'Beograd' AND
```

```
STAROST BETWEEN 22 AND 30
```

```
ORDERED BY STAROST DESC;
```

(4) Klauzula LIKE omogućuje pretraživanje vrijednosti atributa kao niza znakova sa samo djelomično zadatom vrijednošću. Pri tome se koriste oznake:

% predstavlja string od 0 ili više karaktera i

\_ predstavlja poziciju karaktera.

Prikaži imena radnika koja počinju sa M.

SELECT IME

FROM RADNIK

WHERE IME LIKE '\_M%';

Prikaži imena radnika koja imaju treći karakter R.

SELECT IME

FROM RADNIK

WHERE IME LIKE '\_R%';

(5) Testiranje nula vrijednosti se vrši sa klauzulama IS NULL i IS NOT NULL.

Prikaži šifre radnika čija su imena nepoznata.

SELECT RADN#

FROM RADNIK

WHERE IME IS NULL;

Izrazi i ugrađene funkcije

Pored prikazivanja vrijednosti atributa koje su zapamćene u bazi podataka, SQL omogućuje sračunavanja preko aritmetičkih izraza i u njega su ugrađene funkcije za dobijanje sumarnih informacija, neke aritmetičke funkcije, funkcije nad nizovima karaktera i NULL funkcija (koja privremeno menja nula vrednost sa nekom vrednošću koju sami odredimo).

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

SELECT IME , 'Trenutna starost meseci', STAROST \* 12 + 6

FROM RADNIK

WHERE MJESTO\_ROĐ = 'Zagreb';

Rezultat ima oblik

IME

Sima Trenutna starost meseci 294

Aca Trenutna starost meseci 306

Funkcije za dobijanje sumarnih informacija su:

AVG(atribut) - izračunava srednju vrednost datog atributa,

SUM(atribut) - izračunava sumu svih vrijednosti atributa,

MIN(atribut) - nalazi minimalnu vrednost atributa,

MAX(atribut) - nalazi najveću vrednost atributa,

COUNT(\*) - nalazi broj n-torki u relaciji (grupi),

COUNT(atribut) - nalazi broj n-torki sa ne nula vrijednostima atributa i

COUNT(DISTINCT atribut) - nalazi broj n-torki sa različitim vrijednostima atributa.

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

```
SELECT SUM(DANARADA), AVG(DANARADA)
```

```
FROM RASPORED
```

```
WHERE PROJ# = 'pr1';
```

Rezultat ima oblik:

SUM(DANARADA)	AVG(DANARADA)
---------------	---------------

183	20.33
-----	-------

Aritmetičke funkcije (stepenovanje, kvadratni koren, absolutna vrednost i druge) i funkcije nad nizovima karaktera (konkatenacija, određivanje dužine niza i druge) neće se ovde detaljnije prikazivati.

GROUP BY klauzula logički deli relaciju na grupe n-torki tako da unutar jedne grupe sve n-torce imaju istu vrednost zadatog atributa. Time se omogućuje da se funkcije za dobijanje sumarnih informacija primene na svaku ovakvu grupu posebno, umjesto na celu relaciju.

(8) Prikaži srednju starost radnika po organizacionim jedinicama.

```
SELECT ORGJED#, AVG(STAROST)
```

```
FROM RADNIK
```

```
GROUP BY ORGJED#;
```

Rezultat je oblika

ORGJED#	AVG(STAROST)
---------	--------------

oj1	38.8
-----	------

oj2	31
-----	----

oj3	25.5
-----	------

Klauzula HAVING koristi se zajedno sa GROUP BY i za grupu n - torki ima isti efekat kao WHERE klauzula za pojedinačne n-torce.

(9) Prikazati zadatke koje obavlja više od dva radnika na jednom projektu.

```
SELECT PROJ#, BRZAD, COUNT (*)
```

```
FROM RASPORED
```

```
GROUP BY PROJ#, BRZAD
```

```
HAVING COUNT (*) > 2;
```

Izgled rezultata je:

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

Upiti nad više relacija

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

Ulaganje upita može se primeniti kada se u pretraživanju koristi više relacija, ali se prikazuju samo atributi jedne relacije. Uloženi upit se naziva podupit (subquery).

(10) Prikaži šifre i imena radnika koji rade u organizacionoj jedinici Razvoj.

```
SELECT RADN#, IME
```

```
FROM RADNIK
```

```
WHERE ORGJED# =
```

```
(SELECT ORGJED#
```

```
FROM ORGANIZACIJA
```

```
WHERE NAZIVORG = 'Razvoj');
```

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

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

(11) Prikaži nazive projekata u kojima rade radnici rođeni u Beogradu.

```
SELECT NAZIVPR
```

```
FROM PROJEKAT
```

```
WHERE PROJ# IN
```

```
(SELECT PROJ#
FROM RASPORED
WHERE RADN# IN
(SELECT RADN#
FROM RADNIK
WHERE MJESTO_ROĐ = 'Beograd'));
```

(12) Prikaži nazine projekata i imena radnika koji rade na njima, a rođeni su u Beogradu. Mada u osnovi potpuno ekvivalentan sa prethodnim, ovaj upit, pošto zahteva prikazivanje atributa iz dve relacije ne može se realizovati ulaganjem upita, već samo pomoću upita spajanja:

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

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

(13) Spajanje relacije sa samom sobom. Prikaži parove radnika iste starosti.

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

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

(2.2) Operacije za održavanje baze podataka

Operacije za održavanje su UPDATE, DELETE i INSERT. Opći oblik naredbe UPDATE je:

UPDATE relacija

SET atribut = izraz

[ , atribut = izraz]

[WHERE predikat];

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

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

UPDATE RADNIK

SET ORGJED# = oj3

WHERE STAROST > 25;

Opći oblik naredbe DELETE je:

DELETE

FROM relacija

[WHERE predikat];

Izbacuju se sve n-torce koji zadovoljavaju dati predikat.

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

DELETE

FROM RASPORED

WHERE DANARADA < 15;

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

DELETE

FROM RADNIK

WHERE ORGJED# IN

(SELECT ORGJED#

FROM ORGANIZACIJA

WHERE MJESTO\_ORG = 'Zagreb');

Opći oblik naredbe INSERT je:

INSERT

INTO relacija [(atribut [ ,atribut] ...)]

VALUES (konstanta [ ,konstanta] ...);

ili

INSERT

INTO relacija [(atribut [ ,atribut] ...)]

SELECT ... FROM ... WHERE ... ;

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

INSERT

INTO ORGANIZACIJA (ORGJED#, NAZIVORG, MJESTO\_ORG)

VALUES ('oj4', 'Nabavka', 'Beograd');

(2.3) Formiranje pogleda (Vier)

Pogled (vier) u SQL-u se definiše kao izvedena relacija, izvedena iz skupa baznih tabela i drugih definisanih pogleda. Pogled je virtuelna relacija, ona fizički ne postoji, ali je korisnik, po pravilu (izuzeci se odnose uglavnom na operacije održavanja baze podataka) može koristiti kao bilo koju drugu relaciju u sistemu.

Opšta sintaksa za kreiranje pogleda je:

CREATE VIEW ime-pogleda

[(ime-atributa [ ,ime-atributa] ...)]

AS SELECT ... WHERE ... FROM ... ;

Pogled se uništava sa iskazom:

DROP VIEW ime-pogleda;

Primeri:

(18) CREATE VIEW STARI\_RAD

AS SELECT \*

FROM RADNIK

WHERE STAROST > 50;

{Ako se ne navedu imena atributa pogleda on nasleđuje ime atributa iz relacija od kojih se formira, na očigledan način}.

(19) Podskup radnika, okarakterisan sa njihovim šiframa i imenima koji su angažovani na projektima ukupno više od 50 dana, možemo nazvati DOBRI\_RADN i kreirati odgovarajuću tabelu - pogled:

CREATE VIEW DOBRI\_RADN (ŠIFDR, IMEDR)

AS SELECT RADN#, IME

FROM RADNIK

WHERE RADN# IN

(SELECT RADN#

```
FROM RASPORED  
GROUP BY RADN#  
HAVING SUM(DANARADA) > 50);
```

Naredbom CREATE VIEW ne izvršava se upit preko koga je pogled definisan, već se odgovarajući iskaz smešta u katalog baze podataka. Tek kada se nad pogledom izvršava neka operacija, tada se ta operacija izvodi zajedno sa operacijama preko kojih se pogled kreira, a koje su zapamćene u katalogu.

Upit sa kojim se pogled kreira može se formirati na isti način (sa istim klauzulama) kao i bilo koji upit. Izuzetak, očigledno čini klauzula ORDERED BY, jer pogled definiše relaciju, a u relaciji redoslijed n-torki nije bitan. Neki komercijalni sistemi obično dodaju još neka ograničenja.

Sa tačke gledišta pretraživanja (izveštavanja), pogled se tretira kao bilo koja druga relacija i nad njim se mogu izvršavati bilo kakvi upiti. Na primer,

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

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

Očigledno je da se, najopćije gledano, nad pogledima ne mogu vršiti operacije ažuriranja baze podataka, jer pogled fizički ne postoji, a ažuriranje se može preneti na relacije od kojih je pogled sačinjen samo u izuzetnim slučajevima (kada je pogled izведен iz jedne relacije, ili kada se ažuriranje vrši samo nad jednom od relacija koje čine pogled i kada je prenos ažuriranja iz pogleda na baznu relaciju nedvosmislen). Neki komercijalni sistemi ne dozvoljavaju ažuriranje pogleda uopće, a neki dozvoljavaju u tačno navedenim posebnim slučajevima.

Upotreba i značaj pogleda u razvoju aplikacija nad relacijskom bazom podataka je višestruka:

(a) Ostvarivanje logičke nezavisnosti programa i podataka. Da bi se ostvarila nezavisnost programa od logičke strukture baze podataka, programe treba razvijati nad pogledima, a ne nad baznim relacijama. Ovo je posebno značajno u prototipskom razvoju, kada treba formirati poglede čak i potpuno ekvivalentne baznim relacijama. Na taj način aplikacioni

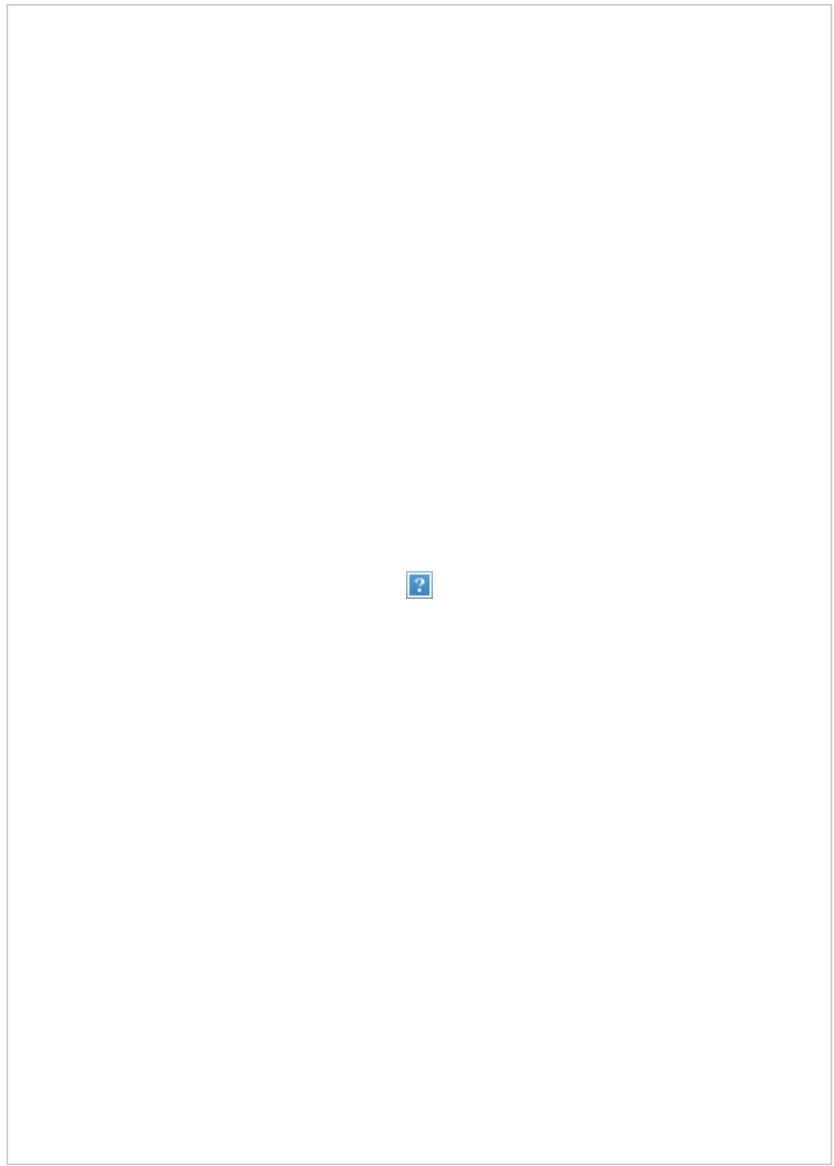
programi se neće menjati, čak i ako se jedna bazna relacija "vertikalno" dekomponuje na dve, ili se izvrši spajanje relacija, ili izmena naziva atributa i slično. Jedino što je, tada, neophodno uraditi je redefinisanje odgovarajućeg pogleda.

(b) Pogledi omogućuju da se isti podaci, različitim korisnicima, prikažu na različite načine, upravo onako kako oni žele da ih vide. Time se značajno može pojednostaviti korišćenje podataka iz baze podataka.

(c) Pogledi omogućuju i jedan stepen zaštite podataka, jer se neki podaci u baznim tabelama, dajući korisniku na raspolaganje samo poglede, mogu sakriti.

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

Kao što je rečeno SQL je homogeno relacioni jezik, jezik koji se može koristiti i za interaktivnu komunikaciju korisnika sa BP i unutar standardnih programskih jezika (COBOL, PL/I i C najčešće), za razvoj aplikativnih programa. U ovom drugom slučaju, naredbe programskog jezika koji se koristi, tzv. "jezika domaćina" i SQL su "pomešane". Šematski prikaz procesiranja jednog ovakvog programa dat je na Slici 1 (za IBM-ov sistem DB2).



*Slika 1. Izvršenje aplikativnog programa u DB2*

Osnovne komponente ovog sistema su:

Precompiler predprocesor koji iz aplikacionog programa "izvlači" SQL naredbe i smešta ih u DBRM (Data Base Request) modul, zamenjujući ih sa pozivima Run Time Supervisor-u.

Bind komponenta prevodi DBRM u tzv. action plan, mašinski kod za implementaciju SQL naredbi uključujući i poziv Stored Data Manager-u.

Run Time Supervisor nadgleda izvršenje SQL naredbi, prenoseći poziv za izvršenje neke naredbe od predprocesora u application plan, koji onda poziva Stored Data Manager da izvrši odgovarajuću funkciju.

Stored Data Manager, upavlja fizičkom bazom podataka, memorišući i pretražujući podatke pozivajući komponente nižeg nivoa koje vrše

baferovanje, zaključavanje, sortiranje i slično.

Način pisanja aplikacionog programa sa SQL-om, objasnićemo na primeru COBOL-a kao jezika domaćina.

Da bi predprocesor razlikovao SQL naredbe od naredbi jezika domaćina, početak i kraj svake SQL naredbe označen je na sledeći način:

EXEC SQL naredba END-EXEC

Samo jedna SQL naredba može stajati između ovih ključnih reči.

SQL naredbe se koriste u deklarativnom (DATA DIVISION) i u proceduralnom (PROCEDURE DIVISION) delu programa. U deklarativnom delu programa definišu se sve tabele (bazne i pogledi) koji se koriste u programu preko naredbi jezika domaćina i na taj način se obezbeđuje komunikacija aplikacionog programa sa bazom podataka. Pored toga, u deklarativni deo programa prenose se i sistemski podaci, specijalne promenljive (registri) preko kojih se u program prenose statusi izvršenja pojedinih SQL naredbi nad bazom podataka (korektno izvršenje ili kod neke greške).

Na primer,

.....  
.....

DATA DIVISION.

FILE SECTION.

\* Definišu se datoteke van baze podataka koje se koriste u programu.

WORKING-STORAGE SECTION.

.....  
.....

EXEC SQL BEGIN DECLARATION SECTION END-EXEC.

01 REKRADN.

05 ŠRADN PICTURE X(7).

05 MATBR PICTURE X(13).

05 IMER PICTURE X(20).

05 STAR PICTURE 99.

05 MJESTOR PICTURE X(20).

05 RADIU PICTURE X(4).

EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL INCLUDE SQLCA END-EXEC.

U IBM-ovom relacijskom sistemu DB2 SQLCA je skraćenica za SQL Communication Area u kojoj se nalaze promenljive sa statusima izvršenja SQL naredbi. To su, na primer:

- SQLCODE koja ima vrednost nula ako je operacija nad bazom uspešno izvršena, pozitivnu vrednost ako operacija nije izvršena, a pojavila se neka normalna situacija (kraj podataka) i negativnu vrednost ako je došlo do neke greške.
- SQLERRM sadrži poruku ako je SQLCODE manji od nule. Naredba INCLUDE SQLCA stavlja aplikacionom programu na raspolaganje ove sistemske promenljive.

U proceduralnom delu programa osnovni problem u usaglašavanju jezika domaćina sa SQL-om leži u tome što SQL operiše nad tabelama, a jezik domaćin nad rekordima (vrstama tabele). U slučajevima kada je rezultat upita SELECT ... FROM ... WHERE ... samo jedna n-torka (vrsta), kao i u slučajevima naredbi UPDATE, DELETE (osim tzv. CURRENT oblika) i INSERT, komunikacija SQL i programskog jezika je direktna. Na primer,

EXEC SQL

SELECT IME, STAROST

INTO :IMER, :STAR

FROM RADNIK

WHERE RADN# = 'r3'

END-EXEC.

MOVE 'r4' TO ŠRADN.

MOVE 20 TO STAR.

MOVE 'Ub' TO MJESTO.

EXEC SQL

UPDATE RADNIK

SET STAROST = :STAR, MJESTO\_ROĐ = :MJESTO

WHERE RADN# = :ŠRADN

END-EXEC.

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

U slučajevima kada upitni blok SELECT ... FROM ... WHERE ... daje više

n-torki kao odgovor, neophodno je definisati, za svaki takav upit koji se u programu koristi, jedan specijalni "bafer" koji se naziva CURSOR, u koga će se smestiti rezultat takvog upita i iz koga će biti moguće preuzimati n-torku po n-torku, kako to jezik domaćin zahteva. CURSOR se definiše na sledeći način:

EXEC SQL DECLARE ime-kursora CURSOR

FOR select-naredba

END-EXEC.

Na primer,

EXEC SQL

DECLARE XYZ CURSOR

FOR SELECT RADN#, IME, STAROST

FROM RADNIK

WHERE MJESTO\_ROĐ = :MJESTO

END EXEC.

Deklaracija kursora nije izvršna naredba, mada se pojavljuje u proceduralnom delu programa (u PROCEDURE DIVISION). Izvršne operacije nad kurzorom su:

- EXEC SQL OPEN ime-kursora END-EXEC.

Ova naredba "aktivira" kurzor, odnosno izvršava upit preko koga je kurzor definisan i smešta rezultat u odgovarajući "bafer". Kurzor ima i ulogu pokazivača i posle OPEN naredbe pokazuje na položaj ispred prve n-torce u "baferu".

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

Ova naredba pomera kurzor koji je "otvoren" za jedno mjesto unapred i pridružuje vrijednosti atributa promenljivim programa. Kada kurzor dospe do kraja "bafera" vrednost sistemske promenljive SQLCODE postaje +100.

- EXEC SQL CLOSE ime-kursora END-EXEC.

Ova naredba zatvara kurzor koji se kasnije može opet otvoriti, da bi se, na primer isti upit ponovo izvršio sa novim tekućim vrijednostima programskih promenljivih.

Primer:

PROCEDURE DIVISION.

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

```
.....  
.....  
MOVE 'Beograd' TO MJESTO.  
EXEC SQL OPEN XYZ END-EXEC.  
EXEC SQL  
FETCH XYZ INTO : ŠRADN, : IMER, : STAR  
END-EXEC.
```

```
.....  
.....  
.....  
EXEC SQL CLOSE XYZ END-EXEC.  
MOVE 'Zagreb' TO MJESTO.  
EXEC SQL OPEN XYZ END-EXEC.  
EXEC SQL  
FETCH XYZ INTO : ŠRADN, : IMER, : STAR  
END-EXEC.
```

Ovde su dati samo osnovni koncepti korišćenja SQL-a u okviru jezika domaćina za izradu aplikacionih programa. Mada su principi ovakve implementacije isti za sve sisteme i jezike domaćine, ipak postoji i niz detalja vezanih za konkretan sistem i jezik. Značajan aspekt u aplikacionim programima, kontrolne (upravljačke) naredbe SQL (CQMMIT i ROLLBACK), biće objašnjeni docnije, posle diskusije problema konkurentne obrade i oporavka baze podataka.

## Izračunate kolone (Computed values)

SQL dozvoljava formiranje novih stupaca u select izrazu, čije se vrijednosti izračunavaju na temelju podataka koji se nalaze u numeričkim stupacama postojećih tablica.

Primjer: Prikaz upisnih listova vezanih za pojedine studente uz označenu godinu studiranja.

```
SELECT IME,PREZIME, SEM, GODINA_STUDIJA= SEM/2+ SEM%2  
FROM STUDENT INNER JOIN UPISNI_LIST ON  
STUDENT.STUDENT_ID=UPISNI_LIST.STUDENT_ID WHERE  
SK_GOD='1999/00' AND SEM IN (1,3,5)
```

Pri izračunavanju vrijednosti u pojedinim stupacama mogu se koristiti vrijednosti postojećih stupaca za koje su definirani numerički tipovi podataka i neki od numeričkih operatora +,-,\*/, % (Modulo operator – ostatak pri cijelobrojnom dijeljenju).

### 9.6.3. SORTIRANJE IZLAZNIH REZULTATA

Za sortiranje izlaznih rezultata koristi se izraz **ORDER BY**, kao dio select instrukcije.

```
SELECT [ALL | DISTINCT] nazivi_stupaca
```

```
FROM nazivi_tablica|logičke veze  
WHERE uvjetni izraz  
ORDER BY kolone_sortiranja
```

Sintaksa: *ORDER BY naziv\_stupca tip\_sortiranja, naziv\_stupca tip\_sortiranja, ...*

Sortiranje može biti rastuće (ascending ASC) ili padajuće (descending DESC). Ako nije naveden tip sortiranja podrazumijeva se rastuće (ASC) sortiranje.

head(STUDENT)={student\_id, ime, prezime,jmbg,spol, dat\_rođenja, mjesto\_id}

Primjer: *SELECT \* FROM STUDENT ORDER BY PREZIME, IME*

Umjesto naziva stupaca u izrazu order by može se koristiti redni broj kolone u select listi.

Primjer: *SELECT \* FROM STUDENT ORDER BY 3 ASC , IME*

## 9.6.4. OGRANIČAVANJE ISPISA REZULTATA

Upit tipa

```
SELECT [ALL | DISTINCT] imena_stupaca FROM  
imena_tablica|logičke veze WHERE uvjetni_izraz
```

prikazuje sve rezultate iz izabranih tablica koji zadovoljavaju uvjetni izraz.

U slučaju potrebe za prikazom samo dijela izlaznih rezultata ispis se može ograničiti primjenom sintakse

```
SELECT [ALL | DISTINCT] [TOP n [PERCENT]] imena_stupaca  
FROM imena_tablica|logičke veze WHERE uvjetni_izraz ORDER BY  
kolone_sortiranja
```

pri čemu je select izraz proširen modifikatorom TOP n ili TOP n PERCENT. Ovaj modifikator označava da će u ispisu rezultata biti prikazano samo prvih n redova (TOP n), odnosno samo n% (TOP n PERCENT) od ukupnog broja redova koji zadovoljavaju postavljeni uvjet.

Primjer:

```
SELECT TOP 50 PREZIME,IME,JMBG FROM STUDENT ORDER BY  
PREZIME,IME.
```

Iz ukupnog skupa studenata prikazuje samo prvih 50 sortirano po prezimenu i imenu.

```
SELECT TOP 50 PERCENT PREZIME,IME,JMBG FROM STUDENT  
ORDER BY PREZIME,IME.
```

Iz ukupnog skupa studenata prikazuje prvu polovicu liste u uređenom sortiranju po prezimenu i imenu.

**ADD USER ime\_korisnika: šifra**

*ime\_korisnika (login)* – korisničko ime koje mora biti jedinstveno u bazi podataka, tj. nije dozvoljeno da postoje dva različita korisnika sa istim imenom

*šifra (password)* – sigurnosna (tajna) šifra, kojom svaki korisnik potvrđuje svoj identitet

## Dodjeljivanje privilegija (prava)

Prava pristupa pojedinim korisnicima dodjeljuju se naredbom

**GRANT nazivi\_prava ON naziv\_objekta TO ime\_korisnika**

*nazivi\_prava* – lista prava pristupa podacima

SELECT – pravo čitanja podataka

UPDATE – pravo promjene (ažuriranja) podataka

DELETE – pravo brisanja podataka

INSERT - pravo unosa podataka

*naziv\_objekta* – ime objekta (tablice ili pogleda) u bazi na koji se prava odnose

*ime\_korisnika* – naziv korisnika kojem se prava dodjeljuju

Primjeri:

**GRANT SELECT, UPDATE ON STUDENT TO K1**

Korisniku K1 dodjeljuje se pravo čitanja i promjene podataka u tablici STUDENT

**GRANT SELECT, DELETE ON STUDENT TO K2**

Korisniku K2 dodjeljuje se pravo čitanja i brisanja podataka u tablici

## Sužavanje pojedinih prava na dijelove tablica i pogleda

Prava promjene i unosa podataka moguće je suziti na pojedine atribute (kolone) u tablici

**INSERT (lista\_atributa)** – pravo unosa podataka u samo određene kolone

**UPDATE (lista\_atributa)** – pravo promjene podataka u navedenim stupacama tablice

Primjer:

**GRANT SELECT, UPDATE(Datum, MjestoId) ON STUDENT TO K3**

Korisniku K3 dodjeljuje se pravo čitanja i pravo promjene datuma i mjesta rođenja u tablici STUDENT

## Kaskadno dodjeljivanje privilegija

Administrator baze podataka može pojedinim korisnicima, uz dodjeljivanje prava dati mogućnost proslijeđivanja tih prava drugim korisnicima.

**GRANT nazivi\_prava ON naziv\_objekta TO ime\_korisnika WITH GRANT OPTION**

Dodjeljivanjem prava uz opciju WITH GRANT OPTION, korisniku se daje mogućnost da po potrebi ta prava proslijedi drugima.

Primjer:

**DBA: GRANT SELECT, UPDATE ON STUDENT TO K1 WITH GRANT OPTION**

**K1 : GRANT SELECT ON STUDENT TO K2**

Korisnik K1 proslijeđuje svoje pravo čitanja podataka korisniku K2

## Opoziv (uklanjanje prava)

Prethodno dodjeljena prava korisnicima mogu se ukloniti naredbom

***REVOKE nazivi\_prava ON naziv\_objekta FROM ime\_korisnika***

Primjer: *REVOKE UPDATE ON STUDENT FROM K1*

Prilikom uklanjanja prava moguće je korištenje dodatnih opcija  
CASCADE ili RESTRICT

Primjer: *REVOKE UPDATE ON STUDENT FROM K1 CASCADE*

Pravo promjene podataka u tablici STUDENT oduzima se korisniku K1,  
ali i svima onima kojima je korisnik K1 to pravo proslijedio.

Primjer: *REVOKE UPDATE ON STUDENT FROM K1 RESTRICT*

Pravo promjene podataka u tablici STUDENT oduzima se korisniku K1  
samo u slučaju da on to pravo nije nikome proslijedio. Ako je K1  
proslijedio pravo UPDATE prikazuju se podaci o korisnicima koji su to  
pravo naslijedili.

## Poništavanje prava naslijeđivanja

Pravo daljnog dodijeljivanja privilegija, koje se daje sa opcijom WITH  
GRANT OPTION moguće je poništiti naredbom

***REVOKE GRANT OPTION FOR nazivi\_prava ON naziv\_objekta  
FROM ime\_korisnika***

Primjer:

**REVOKE GRANT OPTION FOR UPDATE ON STUDENT FROM K1**

Korisniku K1 se oduzima pravo da dijeli UPDATE privilegiju drugima, a ujedno se brišu sva UPDATE prava koje je K1 prethodno dodijelio. Pri tome UPDATE pravo samog korisnika K1 ostaje sačuvano.

## **Grupiranje korisnika**

Radi lakšeg i bržeg dodjeljivanja prava korisnicima koji rade na istim poslovima i imaju slične potrebe za pristup pojedinim podacima, korisnici se povezuju u grupe

**CREATE GROUP naziv\_grupe FOR USERS (lista\_korisnika)**

Primjer:

**CREATE GROUP KORISNICI FOR USERS (K1, K2, K3)**

Dodjeljivanjem prava grupi, prava se dodijeljuju svim korisnicima koji su članovi te grupe

**GRANT SELECT, UPDATE ON STUDENT TO KORISNICI**

Grupa korisnika može se naknadno mijenjati dodavanjem novih ili isključenjem nekog od postojećih članova te grupe

**ALTER GROUP naziv\_grupe ADD|DROP USERS (lista\_korisnika)**

Primjer: **ALTER GROUP KORISNICI ADD USERS (K6)**

**ALTER GROUP KORISNICI DROP USERS (K2,K3)**

## 9. 18. OBRADA SQL UPITA

Kada baza podataka primi SQL zahtjev za obradu određenih podataka, mehanizam za izvršenje upita (Database Engine) prolazi kroz niz koraka obrade zahtjeva (query compilation steps), prije neposrednog izvršenja. U prvoj fazi obrade (syntax checking phase), sistem prima upit i provjerava ispravnost sintakse SQL izraza. Izraz mora biti formiran u skladu sa pravilima SQL sintakse, te objekti koji se pozivaju u izrazu moraju postojati u bazi. Druga faza podrazumijeva provjeru prava pristupa pojedinih korisnika, tj. provjerava se da li korisnik koji je postavio upit ima valjano pravo pristupa na objekte koji se pozivaju u SQL upitu. Pošto su prve dvije faze završene, nastupa faza optimizacije upita (query optimization phase). Optimizacija SQL upita podrazumijeva odabir najboljeg i najbržeg načina za izvođenje SQL upita, bilo da se radi o dohvatu podataka ili njihovoj promjeni (ažuriranju). U suvremenim relacijskim sustavima postoji nužna potreba za optimiranjem SQL upita. Prepostavimo upit:

```
SELECT IME, PREZIME FROM STUDENT , UPISNI_LIST WHERE  
STUDENT.STUDENT_ID =UPISNI_LIST.STUDENT_ID AND SEM=1  
AND SK_GOD='1999/00'
```

Prepostavka je da se u tablici STUDENT nalazi 1000 studenata, a u tablici upisni list 3000 upisnih listova, od čega ih je 200 u 1. semestru školske godine 1999/00.

Neka se upit izračunava bez ikakve optimizacije. Mogući tok izvođenja ovakvog upita bio bi slijedeći:

1. Izračunava se Kartezijev produkt tablica STUDENT i UPISNI\_LIST, tj. svaki student sa svakim upisnim listom. Dobiveni međurezultat sadrži 3.000.000 redova. S obzirom da je veličina radne memorije ograničena, sustav ne može držati toliku količinu podataka u radnoj memoriji, već ih pohranjuje na disk.
2. Dobiveni Kartezijev produkt reducira se WHERE uvjetom. Ponovno je potrebno učitati 3.000.000 redova sa diska u radnu memoriju i nad svakim redom provesti WHERE uvjet. Sada rezultat sadrži samo 200 redova, koji se projiciraju nad atributima IME i PREZIME.

Optimizacijom upita tijek izvršavanja bio bi slijedeći.

1. Najprije se izvršava restrikcija tablice UPISNI\_LIST i izdvajaju se samo oni upisni listovi koji zadovoljavaju uvjetni dio izraza SEM=1 AND SK\_GOD='1999/00'. Dobiveni međurezultat sadrži 200 redova i može se privremeno držati u radnoj memoriji.
2. Provodi se spajanje dobivenog međurezultata sa tablicom student na temelju izraza STUDENT.STUDENT\_ID  
=UPISNI\_LIST.STUDENT\_ID. Iz tablice STUDENT dohvata se 1000 redova a dobiveni rezultat spajanja sadrži 200 redova.
3. Rezultat se projicira nad atributima IME i PREZIME

Očito je da se jedan upit može obraditi na više mogućih načina. Pri tome pojedine metode obrade upita troše više računarskih resursa (memorije, ulazno/izlaznih operacija itd.). Radi brzine izvođenja upita i racionalnog korištenja resursa pri izvođenju upita bitna je optimizacija tj. odabir najpovoljnijeg načina. Suvremeni sustavi baza podataka rješavaju problem optimizacije upita automatski, tj. optimizacijski mehanizam ugrađen je u sustav baze podataka.

## 9.19. METODE POVEZIVANJA TABLICA

Povezivanjem (join) dvaju tablica podrazumijevamo proces kombiniranja tj. povezivanja redova jedne tablice sa redovima druge tablice, da bi se dobili rezultati postavljenog SQL upita. Bez obzira da li je upitni izraz formuliran u obliku Kartezijevog produkta ili join veze, interni mehanizmi baze podataka aktiviraju istovjetne postupke. Proces povezivanja dvaju tablica odvija se kroz dva koraka. U prvom, pristupa se redovima u jednoj od zadanih tablica, koja se uobičajeno naziva vanjska (outer). Drugi korak podrazumijeva pristup drugoj tablici (unutrašnjoj – inner), pri čemu se redovi druge tablice kombiniraju sa dobivenim međurezultatima iz prvog koraka. Rezultat koji se dobije je složena tablica, koja sadržava sve redove proizašle iz ovakvog povezivanja.

## METODA UGNJEŽĐENE PETLJE (NESTED LOOP JOIN)

Pogledajmo primjer slijedećeg upita

```
SELECT T1.C1, T1.C2, T2.C3, T2.C4 from T1,T2 where T1.C1=5 and  
T1.C2=T2.C3
```

Što je isto kao i izraz

```
SELECT T1.C1, T1.C2, T2.C3, T2.C4 from T1 INNER JOIN T2 ON  
T1.C2=T2.C3 where T1.C1=5
```

Za realizaciju veze između ovih tablica smatrajmo tablicu T1 vanjskom (outer table). Prvi korak ovog upita podrazumijeva izdvajanje redova tablice T1 koji zadovoljavaju uvjetni dio izraza – T1.C1=5. Izdvajanjem ovih redova dobija se međurezultat koji se pohranjuje u privremenu tablicu (IT1). Nakon što je dobiven međurezultat, za svaki tako izdvojeni redak, pristupa se podacima u tablici T2.

IT1		T2	
C1	C2	C3	C4
5	19	17	22
5	13	19	24
5	15	15	28
5	17	19	27
5	12	12	29
5	19	14	31
5	18	13	25

Interni pokazivač u tablici IT1 nalazi se na prvom redku, te se izdvaja vrijednost atributa po kojem se vrši povezivanje (C2). Ta vrijednost je 19. Zatim se redom pretražuju redovi u tablici T2, te se izdvajaju oni redovi za koje vrijedi da je T2.C3=19. Kada se završi pretraživanje T2, pokazivač u tablici IT1 pomiče se na slijedeći redak, ponovno se utvrđuje

vrijednost atributa C2 (13), te se ponovno pretražuje tablica T2 i traže oni redovi za koje vrijedi da je T2.C3=13 itd.

U općem slučaju algoritam povezivanja metodom ugnježđene petlje može se prikazati

```
SELECT * INTO IT1 FROM T1 WHERE T1.C1=5 (m redova)
SELECT m=COUNT(*) FROM IT1
SELECT n=COUNT(*) FROM T2
FOR I=1 TO m
    P1=I
    FOR J=1 TO n
        P2=J
        IF P1->C2 = P2-> C3 THEN ISPIS IT1.C1,IT1.C2, T2.C3,T2.C4
        NEXT J
    NEXT I
```

gdje je P1 pokazivač u tablici IT1, a P2 pokazivač u tablici T2.

## MERGE JOIN METODA

Izvršavanje ove metoda sastoji se od dva koraka. Prvi korak u realizaciji podrazumijeva izdvajanje redova koji zadovoljavaju uvjetni izraz, ali u točno uređenom redoslijedu, sortirano po vrijednostima u stupacama preko kojih se vrši spajanje. Dakle nakon izdvajanja redova koji zadovoljavaju uvjetni izraz, obje tablice se sortiraju po vrijednosti join stupaca .

IT1		IT2	
C1	C2	C3	C4
5	12	12	29
5	13	13	25
5	15	14	31
5	17	15	28
5	18	17	22
5	19	19	24

Za realizaciju veze između tablica, svakoj tablici dodijeljuje se interni pokazivač, koji je na početku postavljen na prvi redak u tablici. Za tekući položaj pokazivača u svakoj tablici uspoređuju se vrijednosti u stupacima po kojima se vrši spajanje, te se pokazivači pomiču prema dnu tablica.

```
SELECT * INTO IT1 FROM T1 WHERE C1=5 ORDER BY C2
```

```
SELECT * INTO IT2 FROM T2 ORDER BY C3
```

```
P1=IT1(1)
```

```
P2=IT2(1)
```

```
PETLJA: WHILE (TRUE) {
```

```
    WHILE (P1→C2 > P2→C3) {
```

```
        P2++
```

```
        IF (P2 OUT ) EXIT PETLJA;
```

```
}
```

```
    WHILE (P1→C2 < P2→C3) {
```

```
        P1++
```

```
        IF (P1 OUT ) EXIT PETLJA;
```

```
}
```

```
        IF (P1→C2 = P2→C3)
```

```
{
```

```
            MEMP=P2;
```

```
            WHILE (P1→C2 = P2→C3)
```

```
            { ISPIS IT1.C1.IT1.C2, IT2.C3, IT2.C4 ;
```

```
                P2++;
```

```
}
```

```
}
```

```
P1++;  
IF( P1 OUT) EXIT PETLJA;  
IF (P1→C2= MEMP→C3) P2=MEMP;  
}
```