dr Pavle Kaluđerčić *dr Slobodan Obradović*

Projektovanje informacionih sistema relacione baze podataka

Visoka škola elektrotehnike i računarstva
Beograd 2012

Autori: *dr Pavle Kaluđerčić, dr Slobodan Obradović*

Recenzenti: *dr Dragoslav Perić, dr Petar Bošnjaković*

Izdavač: *Visoka škola elektrotehnike i računarstva u Beogradu*

Lektor *Anđelka Kovačević*

Korice: *Kuk Kristijan i Dimić Gabrijela*

Prelom: *dr Slobodan Obradović*

Tiraž: *150*

Štampa: *MST Gajić*

ISBN 978-86-7982-143-0

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.652.4(075.8)

КАЛУЂЕРЧИЋ, Павле, 1935-
Projektovanje informacionih sistema - relacione baze podataka / Pavle Kaluđerčić, Slobodan Obradović. - 5. izd. - Beograd : Visoka škola elektrotehnike i računarstva strukovnih studija, 2012 (Beograd : MST Gajić). - 174, III str. : ilustr. ; 24 cm

Tiraž 50. - Bibliografija: str. 173-174.

ISBN 978-86-7982-143-0
1. Обрадовић, Слободан, 1955- [аутор]
а) Релационе базе података

Predgovor četvrtom izdanju

U posljednjih petnaest godina, u pet prethodnih izdanja knjige »Projektovanje informacionih sistema - relacione baze podataka« bilo je mnogo izmjena zbog promjena koje su se dešavale na tržištu kako hardvera tako i softvera ove danas vrlo propulzivne oblasti. Odlučili smo stoga napisati, nov i savremen tekst – novi udžbenik pod naslovom Baze podataka.

*Udžbenik »**Baze podataka**« ima težište na sintezi i eksploataciji relacionih baza podataka. Oblast projektovanja informacionih sistema je samo kratko dotaknuta u mjeri koliko je potrebno da bi se neka baza podataka uspješno mogla inkorporirati u informacioni sistem.*

Primjena računara u informacionim tehnologijama na našim prostorima je dominantna i široko rasprostranjena. Već i najmanja privatna firma, prodavnica, škola, apoteka ili ugostiteljski objekat, ako žele da budu konkurentni, moraju svoje poslovanje da »prepuste« računaru.

Naš osnovni cilj bio je stoga da omogućimo studentu da stekne praktično znanje o projektovanju informacionih sistema, te organizaciji i eksploataciji relacionih baza podataka kako bi to mogao odmah u praksi i da primjeni. Nismo se upuštali u ozbiljna teorijska razmatranja.

Primjeri u knjizi su jednostavni i mogu da posluže kao dobar kostur za nadgradnju nekog realnog sistema.

Za vježbe iz ovoga predmeta, na kojima svaki student u toku studija treba da projektuje i »eksploatiše« svoju malu bazu podataka, štampan je poseban »Priručnik« grupe autora na čelu sa dr Slobodanom Obradovićem koji sa ovim udžbenikom predstavlja zaokruženu cjelinu.

Student koji uspješno savlada materiju iz ove knjige nadamo se da je dobio sliku o projektovanju i upotrebi relacionih baza podataka, a inženjeru u praksi ova knjiga može korisno da posluži kao prvi korak koji ga uvodi u »tajne« ove oblasti.

U Beogradu, u jesen 2012. godine

Autori

Logički modeli informacionih sistema

Glava 1

1.0 Uvod

Obrada podataka, pored numeričkih proračuna, bila je i ostala sve do danas jedna od najvažnijih oblasti primjene elektronskih računara. Istina, ljudi su i mnogo prije nego što su raspolagali računarima, praktično od onda kada su postali svjesna bića i shvatili da nisu besmrtni, nastojali da ostave "pisane" tragove, neke "podatke", kako bi sljedećim generacijama ostao bilo kakav zapis o njima.

Prvi problem sa kojim se čovjek, u želji da “iza njega nešto ostane”, susreo, bio je nepostojanje pisma. Pećinski čovjek je ostavljao tragove crtežima na zidovima pećina u kojima su živjeli (Altamira, Zavala, itd.).

Sa pojavom prvog pisma, u početku veoma komplikovanog (*klinasto pismo, hijeroglifi*), "poruke" su se ostavljale na kamenim pločama (nadgrobni spomenici, 10 Božijih zapovjesti,..) jer je za *arhiviranje* podataka nedostajao pogodan *memorijski medijum.*

Osjetan napredak u tehnici arhiviranja nastupio je kada su se kao "memorijski medijum" počele koristiti posebno pripremljene kože životinja, a pronalazak *papirusa i pergamenta* bio je prvi veliki korak naprijed.

Konačno*,* pronalaskom *papira*, i *tehnike štampe*, izgledalo je da je problem memorisanja i arhiviranja podataka u potpunosti i definitivno uspješno riješen.

Međutim, ubrzo se počeo povećavati broj arhiviranih podataka pa se pojavio *novi problem – organizacija podataka u arhivama i čuvanje papirnih dokumenata.* Pri tome, broj papirnih dokumenata rastao je vrlo brzo što je stvorilo novi, dodatni, problem - *problem obrade podataka* – to jest njihova

efikasna eksploatacija s obzirom na težak pristup konkretnim podacima u papirnim arhivama.

Pronalaženje podatka, u velikim arhivama je mukotrpan i dugotrajan posao, pa je mogućnost poređenja i obrade više podataka u svrhu dobijanja novih informacija ograničena neki puta i ljudskim vijekom istraživača. Tako su pojedinci provodili čitav radni vijek "kopajući" po prašnjavim arhivama za podacima kojima bi potvrdili ili opovrgli neke njihove stavove ili teorije.

Sa kakvim problemima se suočavamo kada iz tako, na papiru, arhiviranih podataka treba dobiti neku informaciju, najbolje ilustruje jedan jednostavan primjer:

> *Pretpostavimo da rektor nekog univerziteta sa dugogodišnjom tradicijom hoće da sazna ime studenta koji je na tom univerzitetu, od njegovog osnivanja, kao najmlađi završio studije, u najkraćem roku, a sa najvećom prosječnom ocjenom.*
>
> *Pod pretpostavkom da u arhivi univerziteta postoje pisani tragovi o svim studentima koji su studirali na tom univerzitetu. do tražene informacije može se doći ako se pregledaju svi studentski dosijei i za svakog studenta:*
>
> - *zapiše datum rođenja*
> - *zapiše datum upisa na fakultet,*
> - *zapiše datum diplomiranja,*
> - *izračuna vrijeme studiranja*
> - *izračuna starost studenta*
> - *izračuna srednja ocjena,*

Nakon toga treba*:*

- *poređati studente po dužini studentskog staža,*
- *poređati studente po visini srednje ocjene*
- *poređati studente po starosti, i*
- *konačno naći među njima onoga najmlađeg koji ja za najkraće vrijeme postigao najbolji uspjeh.*

Nije potrebno podrobno objašnjavati koji je, i koliki je, to posao ako se mora obaviti ručno za veliki broj studenata.

Problem obrade i eksploatacije velikog broja podataka riješen je efikasno tek pojavom digitalnog računara i odgovarajuće magnetne, odnosno elektronske, memorije u drugoj polovici XX vijeka.

Paralelno sa brzim razvojem računarskog hardvera u posljednjih 30 godina, (prije svega pojavom moćnih personalnih računara), korisnici su postajali sve zahtjevniji i u pogledu softvera koji bi im omogućio jednostavan, brz i efikasan pristup te obradu velikog broja podataka.

U tu svrhu računari su se počeli povezivati u lokalne, a zatim i, globalne, fleksibilne mreže, koje omogućavaju brz pristup i efikasnu obradu velikog broja različitih podataka lociranih na raznim mjestima, a što je do nedavno bila privilegija samo velikih računskih centara.

Prvi modeli za *organizaciju, memorisanje,* manipulaciju *i obradu* podataka u tada još skromnim bazama podataka pojavili su se šezdesetih godina XX vijeka. U centru ovih modela nalaze se dva pojma, pojam:

- *podatka,*

i

- *informacije*.

Definišimo stoga šta se danas podrazumijeva pod tim pojmovima.

1.1 Osnovni pojmovi

Pod pojmovima *podatak* i *informacija***,** u stručnoj literaturi, danas se podrazumijeva:

- *podatak* je iskaz definisan jednom *prostom izjavnom rečenicom*,
- *informacija* je novi *podatak* koji posjeduje *relevantnu novinu*, neko novo *saznanje*, a rezultat je *obrade poznatih podataka*.

Dobijanje nove, relevantne informacije je nemoguće bez posjedovanja odgovarajućih ključnih podataka koji je definišu. Ako nemamo mogućnosti da ih saznamo, osuđeni smo na neuspjeh.

I mi u svakodnevnom životu koristimo podatke - "eksploatišemo" naše kućne baze podataka, kako bismo mogli uspješno da riješimo naše životne probleme.

Na primjer, ako u gradu u trgovinama vlada nestašica nekog artikla, ukoliko ne dobijemo pravovremeno pouzdan podatak gdje i kada će se on moći nabaviti, ostaćemo bez informacije gdje se može nabaviti, a to znači i bez njega.

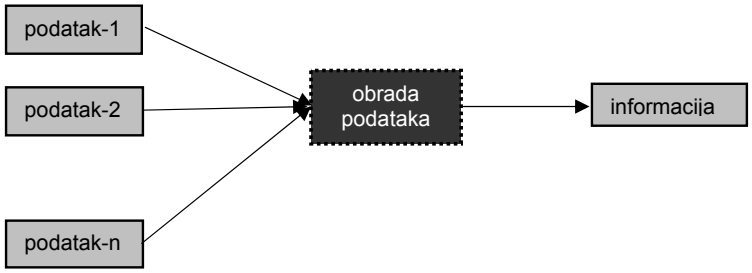
Odnos podatka, obrade podataka i nove informacije vidi se na sljedećem primjeru - ocjenjivanju učenika u školi.

Podaci su ocjene iz pojedinih predmeta (na primjer: biologija-5, matematika-5, fizika-4, maternji jezik-5, strani jezik-4, itd.).

Obradom, izračunavanjem srednje vrijednosti, dobija se prosječna ocjena svakog učenika (4,6) što je nova informacija, novo saznanje, novi podatak, koji se ne može saznati bez obrade prethodno poznatih podataka.

Na kraju, na bazi ove nove informacije dolazimo i do novog saznanja – podatka kakav je uspjeh postigao neki učenik.

Postupak obrade podataka (slika 1.1), dobijanja novih informacija i sticanja novih saznanja, može se na osnovu svega do sada rečenog podeliti u četiri faze:



Slika 1.1 Dobijanje nove informacije iz niza poznatih podataka

- *prikupljanje* podataka,
- *memorisanje i organizacija* prikupljenih podataka,
- *obrada* podataka (računanje, sortiranje, grupisanje itd.),
- *dobijanje* novih informacija - sticanje novih saznanja.

1.2 Objekat posmatranja - entitet

Podaci koje *sakupljamo, memorišemo, organizujemo i obrađujemo* nalaze se u svijetu oko nas i *vezani su za neki proces* koji se odvija u dijelu realnog svijeta iz našeg okruženja, dijelu koji želimo da bliže upoznamo na osnovu *podataka* iz njega.

Proces po definiciji predstavlja *promjenu jedne ili više veličina u vremenu* (na primjer: promjena temperature vazduha, kretanje nataliteta u nekom regionu, varijacija bruto nacionalnog dohotka, promjena opterećenosti elektroenergetskog sistema, itd.).

Podaci kojima se opisuje proces (koji je kontinualan u vremenu) *su diskretne veličine* **(**poznajemo ih samo u određenim vremenskim trenucima) pa je postupak analize i praćenja procesa preko podataka neka vrste analogno-digitalne (A/D) konverzije.

Iz diskretne baze podataka nova *informacija* dobija se *obradom diskretnih podataka* **–** pa je i nova *informacija diskretna veličina*.

Da bismo izveli pomenutu “A/D konverziju”, i *kontinualni proces* opisali *ograničenim brojem diskretnih podataka*, prisiljeni smo da definišemo naše “viđenje” za nas interesantnog dijela realnog svijeta. Dobijeni rezultat naziva se kratko *model-objekat, entitet,* ili samo *objekat*.

Svojstva objekata opisuju se preko *atributa* (koje moramo odabrati u fazi modelovanja objekta) a skup dozvoljenih vrijednosti koje neki atribut može uzeti naziva se *domen*.

Na primjer, objekat UČENIK *opisuje se atributima – ocjenama čiji domen je skup prirodnih brojeva od jedan (1) do pet (5)*

Broj atributa (*n*) koji su od značaja za opis nekog entiteta zavisi od procesa i obrade podataka koju treba obaviti. Koji su to *relevantni atributi* kojima će se opisati neki entitet u svakom konkretnom slučaju mora da definiše *kompetentna osoba*, jer će od toga zavisiti upotrebljivost i vjerodostojnost obradom dobijenih informacija.

Ako odaberemo premalo atributa, ili atribute koji nisu relevantni za taj proces, model će biti jednostavan za obradu i analizu, ali će mu vjerodostojnost biti mala, pa će i broj korisnih i upotrebljivih informacija koje on može da prezentira biti ograničen, ili ih uopšte neće ni biti.

A ako se model opiše pretjerano velikim brojem atributa, njegovoj vjerodostojnosti se neće moći prigovoriti, ali manipulacija podacima postaje teška - a dobijene informacije najčešće konfuzne**.** Prema tome:

prepoznavanje mjere pri modelovanju procesa (pri izboru broja atributa) jedan je od osnovnih zadataka projektanta informacionog sistema.

Ako je, na primjer, predmet našeg interesovanja član radnog kolektiva posmatran sa aspekta ličnog dohotka, onda atribut "broj cipela" nije relevantan.

Međutim, ako nam je potrebna informacija za nabavku HTZ opreme koju ta firma nabavlja za svoje radnike, onda su broj cipela i veličina radnog odijela i te kako važni podaci.

Entitet ili objekat, po prirodi može biti veoma različit kao na primjer:

- *dio okruženja* (član kolektiva, aparat, zgrada…..)
- *apstraktni pojam* (neka mjera, nečije zvanje, boja,…..)
- *događaj* (udes, postupak upisa studenata*,…...*)
- *asocijacija* (polaznik-kurs, predmet-nastavnik*,) itd.*

Kako je *proces* po definiciji *dinamička kategorija* (njegovi pokazatelji se mijenjaju u vremenu) to se i podaci kojima se proces opisuje moraju ažurirati, odnosno "osvježavati" u vremenu. Kada, kako često, i ko će ažurirati podatke sljedeći je važan faktor u projektovanju informacionog sistema pa i to mora biti precizno definisano.

Veličine koje se ne mijenjaju u vremenu kao što su; broj π, ubrzanje zemljine teže *g*, dielektrična konstanta vakuma ε_0 i tome slično dovoljno poznavati samo jedanput.

Posmatrajmo postupak modelovanja objekta na jednom jednostavnom primjeru:

Pretpostavimo da u sektoru za urbanizam neke opštine žele da imaju informacije o ulicama u toj opštini. Entitet (objekat), je prema tome ULICA*, a relevantni atributi kojima se opisuje ulica mogli bi biti:*

- *naziv,*
- *dužina,*
- *širina,*
- *vrsta kolovoza,*
- *godina izgradnje,*
- *broj kuća*

dok podatak o promjeni temperature asfalta u toku godine, u ovome slučaju, nećemo smatrati relevantnim. Entitet ULICA *definisan preko atributa možemo predstaviti kao:*

ULICA < *naziv, dužina, širina, vrsta_kol., god_ izg., broj_kuća…* >

Za svaku ulicu vrijednosti nabrojanih atributa, dakle podaci kojima se jedna ulica pobliže opisuje, su različiti, a mogu se vremenom i promijeniti (na primjer: promjena naziva ulice, dužine, broja kuća itd.) što zahtijeva pomenuto, povremeno, "osvježavanje" podataka.

Veličina, odnosno dužina, tabele u kojoj će se nalaziti podaci o svim ulicama u opštini zavisi od broja ulica pa je broj redova u tabeli, broj takozvanih slogova, ili *n-torki*, jednak broju ulica u toj opštini. Na primjer:

ULICA

Naziv	*Dužina (km)*	*Širina (m)*	*Vrsta kolovoza*	*Godina izgradnje*	*Broj kuća*
Sime Milutinovića	*0.56*	*12.3*	*asfalt*	*1908*	*231*
Jove Jovanovića	*0.20*	*7.5*	*granitne kocke*	*1898*	*56*
Vuka Karadžića	*1.75*	*6.00*	*asfalt*	*1864*	*442*
Zeke Buljubaše	*0.08*	*5.50*	*kaldrma*	*1888*	*12*

Slika 1.2 Primjer tabele "ULICA"

Izgradnjom nove ulice, njeni podaci se onda samo dopisuju u već postojeći spisak, u postojeću *tabelu*. Prema tome; svaka tabela mora da ima definisano:

- *ime ili naziv tabele,*
- *spisak atributa i*
- *niz vrijednosti atributa, to jest podatke.*

Da rezimiramo; *tabela* se sastoji od *polja* u koja se upisuju *podaci*. Slaganjem polja u jednome redu tabele dobijamo jedan:

- *slog, red, ili n-torku,*

a skup svih *redova (slogova), svih n-torki* neke tabele čini:

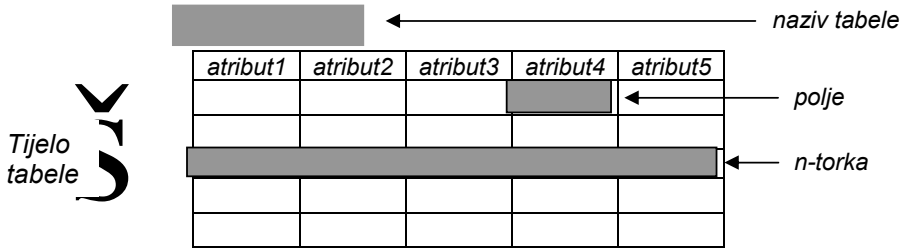
- *tijelo tabele*

kao što je to prikazano na slici 1.3.

Neka druga tabela, u istom sektoru pomenute opštine, može da sadrži podatke o nekom drugom entitetu - objektu, na primjer, stambenim zgradama u toj opštini. Nazovimo tu tabelu ZGRADA, a skup za nju relevantnih atributa mogao bi biti:

ZGRADA < *ulica, kućni_br., god_izg., br_sprat., br_stanova, itd. .>*

U tabeli (umjesto termina TABELA neki autori koriste termin DATOTEKA a često, posebno u literaturi o relacionim bazama podataka, uz još neka ograničenja, i termin RELACIJA) ZGRADA *ulica* je atribut, dok je u prethodnom primjeru tabela imala taj naziv. Naziv atributa u jednoj tabeli može prema tome biti naziv neke druge tabele. Izbor imena atributa i tabela je proizvoljan a diktiran je prije svega našim željama, interesima i "*pogledima*" koje na realni svijet hoćemo da "*bacimo*" preko podataka koje posjedujemo, odnosno koje informacije očekujemo da iz njih možemo dobiti.



Slika 1.3 Elementi tabele

Preporučuje se da ime *tabela* kao i *atributa* treba da asociraju na prirodu procesa koji se u toj tabeli prati podacima. Korišćenje istih naziva treba u principu izbjegavati jer to najčešće dovodi do grešaka koje se tek kasnije u toku rada otkrivaju. Kod izbora imena atributa posebno treba biti obazriv, jer osnovno pravilo kaže da:

atribut mora biti tako odabran (definisan) da se može iskazati samo jednom izričnom rečenicom, to jest definisati samo jednim elementarnim (atomarnim) podatkom.

Ako je za opis atributa potrebno više podataka, onda nije više u pitanju *atribut*, nego po svoj prilici novi entitet sa svojim atributima.

Tako u navedenim primjerima, kada je od interesa bio entitet - stambena ZGRADA - *naziv ulice u kojoj se ta zgrada nalazi je u tabeli* ZGRADA *atribut za koji postoji "atomaran" podatak – naziv te ulice.*

Ali ako nam pored naziva ulice treba još podataka o ulici (na primjer, dužina, širina, itd.) ULICA *postaje novi objekat – entitet sa svojim atributima a atribut "naziv ulice" u tabeli* ZGRADA *treba brisati.*

U toku eksploatacije pomenutih podataka o ulicama i zgradama može se ukazati potreba da jednovremeno posmatramo i ULICE i ZGRADE. Pristup rješavanju problema se tada komplikuje. U tom slučaju se od tih dvaju tabela, koje su očigledno u prirodi međusobno povezane (jer zgrade se nalaze u ulicama), formira *baza podataka* odnosno *informacioni sistem* koji *mora da sadrži u sebi tu vezu među objektima* kako bi bio vjerodostojna slika realnog procesa koga posmatramo i kako bi o njemu mogli da dobijamo relevantna nova saznanja i informacije**.**

1.3 Veze među objektima

Svijet u kome živimo je veoma kompleksan pa su tako i informacioni sistemi koji ga opisuju po svojoj prirodi kompleksni, ma koliko se mi trudili da model objekta pojednostavimo. Izdvojeni model – objekti (kada ih ima više) su redovito međusobno povezani vezama koje su refleksija veza koje postoje među objektima i u realnom svijetu. Jer, ako to ne bi bio slučaj, informacioni sistem ne bi bio realna slika dijela svijeta koji opisujemo, pa ne bi imao nikakvu praktičnu vrijednost.

Prirodu veza među objektima najčešće diktira čovjek, rjeđe su te veze poslijedica neke prirodne zakonitosti.

U osnovi veza među objektima su najčešće zakoni, statuti, propisi, dogovori itd., a koji su rezultat ljudskih aktivnosti. Danas razlikujemo tri tipa veza među objektima i to:

- *veza tipa 1 : 1*
- *veza tipa 1 : N*
- *veza tipa N : M*

1.3.1 Veza tipa 1 : 1

Pretpostavimo da je na nekom univerzitetu uspostavljen informacioni sistem u kome između ostaloga postoje i dva objekta:

- FAKULTET
- DEKAN

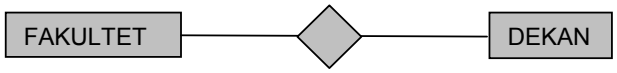
Prva tabela FAKULTET sadrži atribute kojima se opisuju fakulteti toga univerziteta na primjer:

FAKULTET <*naziv, adresa, telefon*, ...*itd.* >

a druga, DEKAN, sadrži atribute koji pobliže definišu dekane fakulteta, na primjer:

DEKAN <*šifra_dekana, ime, prezime, adresa, telefon, itd*,...>

Ako je zakonom određeno *da svaki fakultet može da ima samo jednog dekana, a da samo jedan profesor (jedna osoba) može biti dekan,* onda je veza među tabelama FAKULTET i DEKAN definisana –veza je tipa 1 : 1 - a može se grafički predstaviti u vidu romba kao:



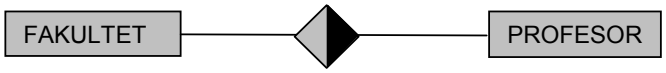
1 : 1

1.3.2 Veza tipa 1 : N

U informacionom sistemu univerziteta može da postoji i objekat PROFESOR čiji atributi treba da pobliže opišu profesore. Na primjer:

PROFESOR <*šifra_profesora, ime, prezime, zvanje, adresa, itd*,...>

Ako zakonski propisi propisuju da jedan profesor *može biti u radnom odnosu samo na jednom (1) fakultetu,* a da *svaki fakultet angažuje više (N) profesora,* onda je veza između objekata FAKULTET i PROFESOR tipa 1 : N.



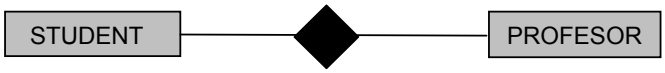
1 : N

1.3.3 Veza tipa N : M

Proširenje ovog univerzitetskog informacionog sistema može se odnositi i na studente. O svakom studentu treba imati neke podatke, na primjer:

STUDENT < *Br._ind, ime, prezime, god._rođ, adresa, itd,....*>

Kako u toku studija *svaki student dolazi u kontakt sa više (M) profesora, ali i svaki profesor drži predavanja većem broju (N) studenata* to je veza među objektima PROFESOR i STUDENT tipa M : N.



M : N

1.4 Modeli podataka

Pedesetih godina prošlog vijeka pojavio se problem kako modelirati i objediniti objekte u jedan cjeloviti informacioni sistem pa su od tada do danas razvijeni sljedeći modeli:

- *model prve generacije – bazirani na programskim jezicima,*
- *modeli druge generacije – hijerarhijski i mrežni,*
- *model treće generacije – relacioni model, te*
- *model četvrte generacije – objektno orjentisani model.*

Danas je u fazi projektovanja dominantan model *objekat-veze* (*MOV*), (*Entity-Relationship* ili *E-R model*), dok se pri implementaciji na računar još uvek dominantno koristi *relacioni*.

Relacioni model je pokazao takvu superiornost u primjeni da su ostali modeli praktično napušteni.

Međutim treba naglasiti da, bez obzira koji model je u pitanju, u svakom od njih mora uvijek postojati mogućnost:

- *definisanja podataka,*
- *definisanja pravila za očuvanje podataka,*
- *definisanja pravila manipulacije podacima.*

1.4.1 Hijerarhijski model podataka

Modeli, bazirani na programskim jezicima, održali su se kratko, takoreći nije ni bilo vremena da se implementiraju iz razvojne u eksploatacionu fazu.

Prvi model koji je našao primjenu i u praksi bio je *hijerarhijski model* a pojavio se šezdesetih godina prošlog vijeka. Osnovni razlog zašto je hijerarhija bila osnova za modeliranje informacionih sistema je činjenica da je hijerarhija prisutna u svakodnevnom životu gdje smo mi, od malih nogu, od autoriteta roditelja u porodici, preko firme u kojoj radimo, vojske u kojoj smo služili vojni rok, preduzeća u kome smo zaposleni, crkve ili neke druge institucije kojoj pripadamo itd. naviknuti na hijerarhiju.

Osnovni nedostatak hijerarhijskog modela je nepostojanje egzaktne matematičke teorije koja ga prati i koja i omogućila njegovu punu implementaciju na računar.

To je i bio podstrek *E.F.Codd-u* da početkom sedamdesetih godina prošlog vijeka definiše, dizajnira i prezentira danas najpopularniji *relacioni model*, koji će biti detaljnije opisan u sledećim poglavljima.

Analizirajmo nekoliko primjera hijerarhijskog modela kako bi sagledali sve njegove karakteristike – prednosti i mane.

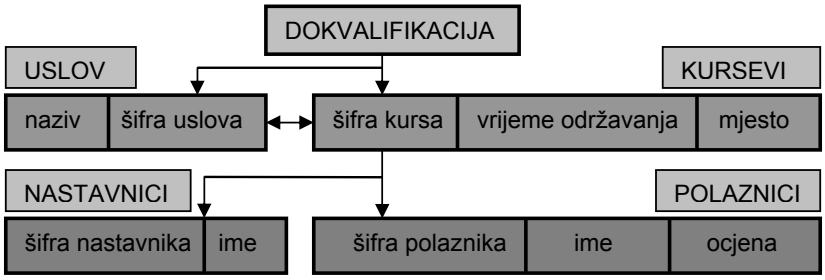
Primjer I

Pretpostavimo da je u nekoj firmi potrebno projektovati informacioni sistem (*bazu podataka*) internog školovanja njihovih službenika. Preduzeće drži niz različitih kurseva na različitim lokacijama. Neki od kurseva se nastavljaju jedan na drugi, pa je uspješno završen prethodni kurs, uslov za upis narednog.

Odgovarajuća baza podataka, nazovimo je DOKVALIFIKACIJA*, za svaki kurs treba da sadrži sljedeće informacije:*

- *broj kursa, naziv kursa, mjesto i vrijeme održavanja,*
- *broj kursa koji je uslov za uspješno pohađanje slijedećeg,*
- *detalje o predavačima (ime, prezime, adresa, itd…..),*
- *detalje o polaznicima (ime, prezime, ocjena, itd……).*

Struktura hijerarhijskog modela predstavlja se grafički jer je pregledna u grafičkoj formi (što je jedna od malobrojnih prednosti hijerarhijskog modela nad relacionim), kako se to vidi na priloženoj slici 1.4.



Slika 1.4 Primjer hijerarhijske strukture informacionog sistema

*Sa skice je vidljivo da hijerarhijska struktura ima svoj naziv, svoju osnovu ili korijen (*DOKVALIFIKACIJA*), koja se u zavisnosti od strukture dalje grana i to uvijek "prema dole".*

U konkretnom slučaju osnova, DOKVALIFIKACIJA**,** *ima svoje dvije pod-tabele koje joj slijede. To su:*

- USLOV *i*
- KURSEVI**,**

pri čemu tabela USLOV *mora biti logički povezana sa tabelom* KURSEVI *(mora se znati šta je preduslov za pohađanje narednog kursa). Tabela* USLOV *nema svojih pod-tabela – nema "nasljednika", dok tabela* KURSEVI *ima dva "nasljednika", dvije pod-tabele i to:*

- NASTAVNICI
- POLAZNICI.

Tabele USLOV i KURSEVI *su prema tome "nasljednici" u bazi* DOKVALIFIKACIJA*, a tabela* KURSEVI *je istovremeno "roditelj" za tabele* NASTAVNICI i POLAZNICI*.*

Precizno definisana međusobna povezanost pojedinih tabela definiše pomenutu hijerarhiju u ovom primjeru:

- KURSEVI - POLAZNICI,
- KURSEVI - NASTAVNICI,
- DOKVALIFIKACIJA - USLOV

a što je na slici 1.4. jasno vidljivo.

Hijerarhijskim modelom sve veze među tabelama, odnosno među atributima pojedinih tabela, moraju biti precizno definisane.

Hijerarhijski model podataka bio je prvi ozbiljniji pokušaj modelovana informacionog sistema. Međutim, on je ubrzo napušten jer za uspješan rad sa njim *svi korisnici informacionog sistema, a ne samo njegov administrator, morali su poznavati mnoga, ne baš jednostavna pravila i ograničenja u primjeni, kao i strukturu, zavisnost i prirodu veza između pojedinih polja – atributa međusobno povezanih tabela.*

Priroda i vrsta veza među objektima određena je, kao što to vidjeli, spoljnim, često nametnutim, uslovima (*zakoni, pravilnici, statuti, dogovori itd*.), pa ako i ta kategorija mora biti poznata i projektantu i administratoru, a i korisniku baze podataka, za širu primjenu hijerarhijskog informacionog sistema to sigurno nije neka ozbiljna referenca.

Ima slučajeva za koje je u hijerarhijskom modelu nema rješenja. To se odnosi na slučajeve kada "*jedan roditelj može imati više nasljednika, ali i svaki nasljednik može imati više roditelja*". To su pomenute veze tipa M : N. Definicija je u bukvalnom smislu riječi apsurdna, ali u prenosnom smislu, u informacionim sistemima moguće je da se pojavi. Pokažimo to na jednom primjeru:

Primjer II

Neka u dijelu hijerarhijski organizovanog informacionog sistema nekog univerziteta (nazovimo ga UNIVERZITET***), postoje tabele sa slijedećim atributima:***

- FAKULTET < *sifrafak, naziv, adresa, telefon, ...……..>*
- REKTOR < sifrarek, ime, prezime, adresa, telefon.....>
- DEKAN < *matbr, ime, prezime, adresa, zvanje, sifrafak,.....>*
- PROFESOR < *sifraprof, ime, prezime, adresa, sifrafak, ...>*
- STUDENT < *broj indexa, ime, prezime, adresa, sifrafak,.....>*

Organizaciona struktura univerziteta nam kazuje da su tabele FAKULTET *(F), koje "izlaze" iz korijena tabele* UNIVERZITET *(a koja*

ima u istom hijerarhijskom nivou i tabelu REKTOR*,) "roditelji" za tabele* PROFESOR *(P),koju nasljeđuje tabela i* STUDENT *(S). Tabele* DEKAN (D) *i* FAKULTET *su dodatno međusobno povezane, i nalaze se u istom hijerarhijskom nivou – slika 1.5.*

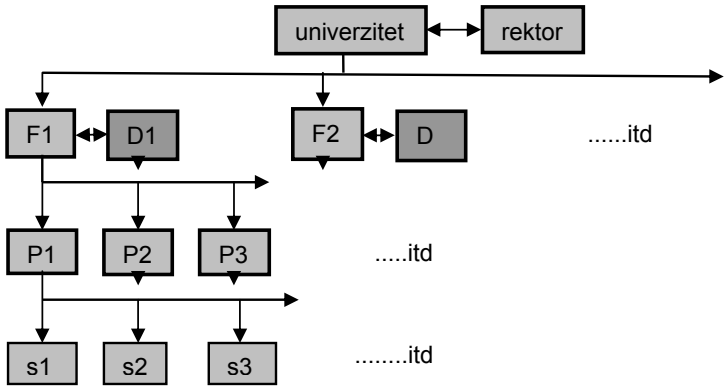
Ako zakonska praksa ne dozvoljava profesoru da jednovremeno radi na dva fakulteta, onda jednom fakultetu pripada N profesora, ali svaki profesor pripada samo jednom fakultetu.

Ovaj tip veze 1:N (*jedan prema više*) u hijerarhijskoj strukturi se lako prestavlja, jer jedan "roditelj" (FAKULTET), logično, može imati više "naslje-dnika" (PROFESOR-a).

Između tabela STUDENT i PROFESOR *veza je drugačija, jer profesor predaje većem broju (N) studenata, a student sluša predava-nja više (M) profesora. Veza je tipa N : M i ne može se na slici 1.5 gra-fički predstaviti*

Veze M:N mora biti transformisana u dvije veze tipa 1 : N. Na taj način se odstupa od hijerarhijske strukture i model se ne može više smatrati hije-rarhijskim. Istina, i u relacionim bazama postoji isti problem, ali je lakše rje-šiv a model pri tome ostaje relacioni.

Zbog teškoća pri rješavanju problema veza tipa M:N, i još nekih nedo-stataka, hijerarhijske baze podataka su danas prava rijetkost. Međutim, mo-ra se priznati da je hijerarhijski način memorisanja podataka u prednosti u pogledu brzine pristupa podacima, jer su "*putevi*" pristupa podacima tačno definisani hijerarhijskim vezama među tabelama, što kod relacionog mode-la, u tako eksplicitnoj formi, nije slučaj. Hijerarhijski sistem ima još nedosta-taka. To su prije svega:



Slika 1.5 Hijerarhijska struktura informacionog sistema univerziteta

- *nekontrolisan gubitak informacija (brisanjem tabele "RODITELJA" gubi se i tabela "NASLJEDNIK", svi podaci u njoj, kao i sve niže rangirane tabele sa podacima u njima), te*
- *pojava redundance podataka (potreba za memorisanjem istog podatka na više mjesta) što komplikuje ažuriranje podataka, uz jednovremeno smanjenje pouzdanosti informacija koje se iz takvog informacionog sistema dobijaju.*

Do kakve greške zbog pojave redundance može doći ilustrujmo opet jednim primjerom.

Primjer III

Pretpostavimo da je klinički centar uspostavio informacioni sistem koji se bazira na hijerarhijskom modelu. Pošto je u organizaciji rada u ustanovama ovakvog tipa hijerarhijska struktura, zbog načina poslovanja, izrazito prisutna u svim segmentima rada, to će "slika" poslovanja preslikana u hijerarhijsku strukturu biti sigurno realna.

Pretpostavimo da je projektant hijerarhijskog informacionog sistema predvidio, između ostaloga, slijedeće objekte i veze među njima:

Korijen stabla je KLINIČKI CENTAR*. Prvi niži nivo su* KLINIKE *(sa svim relevantnim podacima o njima), zatim* LJEKARI *(sa podacima o njima),* MEDICINSKO OSOBLJE**,** POMOĆNO OSOBLJE, LABORATORIJE, KABINETI, PACIJENTI *itd. - sa svim podacima o njima.*

Organizacionom strukturom rada zna se koji ljekari pripadaju kojoj klinici, te koje osoblje i pacijenti, "pripadaju" nekom ljekaru i klinici. Sistem na prvi pogled treba da funkcioniše besprijekorno.

Pretpostavimo sada slijedeći hipotetički slučaj. Pacijent **xy** *dolazi zbog zdravstvenih problema na ispitivanje. On bude primljen na kliniku K1 (na primjer interna klinika) i biva "dodijeljen" ljekaru te klinike L1. Na prijemnom odjeljenju klinike K1 uzimaju se svi potrebni podaci od pacijenta i unose u njihovu tabelu* PACIJENTI**,** *koja stoji na raspolaganju svima zainteresovanim ljekarima i osoblju – ali samo na klinici K1. Opšte podatke (prezime, ime, adresu, i sl.) uzima službenik na šalteru, a podatke o zdravstvenom stanju: anamnezu (ličnu i porodičnu) i nakon pregleda dijagnozu, ljekar u prijemnom odjeljenju. Na osnovu dijagnoze bolesnik se upućuje u odjeljenje na liječenje.*

Na ispitivanju, na klinici K1, utvrdi se da je za pacijenta **xy** *potreban hirurški zahvat. Pacijent biva prebačen na kliniku K2 (hirurgija) sa svom pratećom (papirnom) dokumentacijom (anamneza, dijagnoza, terapija, eventualne alergije itd). Na prijemnom odjeljenju klinike K2, njihov službenik unosi ponovo sve podatke o pacijentu u tabelu* PACIJENTI *(ali sada klinike K2)**,** iako ti podaci već postoje na klinici K1 (ovo je redundanca - ponavljanje podataka).*

Nakon hirurškog zahvata, u post-operativnom toku, dolazi do komplikacija i pacijent bude podvrgnut dodatnoj terapiji (na primjer antibioti-

cima). U toku te terapije pacijent doživi šok i nakon toga postaje alergičan na upotrebljeni antibiotik A1. Intervencijom dežurnog osoblja pacijent bude spasen, a u tabelu PACIJENTI *na klinici K2 (ali ne i K1) unosi se dodatni podatak "alergičan na antibiotik A1". Kada se stanje potpuno normalizuje pacijent bude otpušten kući.*

Nakon nekog vremena isti pacijent oboli od gripe koja se iskomplikuje i pređe u upalu pluća tako da je pacijentu opet potreban klinički tretman. On dolazi ponovo na kliniku K1 (interna) gdje, na prijemnom odjeljenju, nakon unošenja njegovog matičnog broja, službenik vidi da je pacijent **xy** *već bio hospitalizovan na toj klinici, i zato ne uzima opšte podatke od njega, nego ga odmah prosljeđuje ljekarima. Ljekari uzimaju novu anamnezu, postavljaju novu dijagnozu i upućuju pacijenta na odjeljenje te klinike na liječenje.*

Ljekar na odjeljenju, koji ima uvid u podatke o pacijentu iz tabele PACIJENTI *klinike K1, vidi da nema smetnje da se uključi terapija antibioticima (taj podatak postoji samo na klinici K2), propisuje terapiju antibiotikom A1 koja dovodi do fatalnog ishoda.*

Činjenica da su različiti podaci o istom pacijentu bili upisani na dva mjesta (klinike K1 i K2), da nisu bili jednovremeno ažurirani, dovela je u ovom slučaju do pogrešne informacije koja je imala fatalan ishod.

Svi pomenuti nedostatci hijerarhijskog modela učinili su da se ova tehnologija danas definitivno smatra *zastarjelom i odbačenom*.

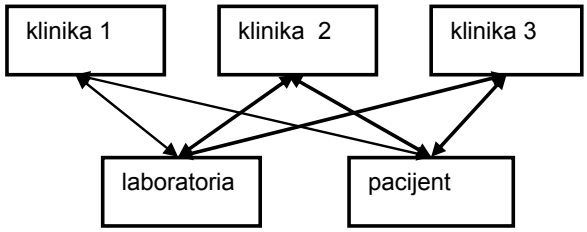
1.4.2 Mrežni model

Mrežni model nastao je kao proširenje i pokušaj poboljšanja hijerarhijskog. Osnovna razlika, u odnosu na hijerarhijski model, leži u činjenici da se dozvolilo da "nasljednik" ima više "roditelja"**,** što znači da ovakav model prihvata i veze tipa M:N**.**

Na taj način veze među pojedinim tabelama, grafički interpretirane, liče na *paukovu mrežu* odakle je model i dobio ime – slika 1.6.

U primjeru hijerarhijski organizovanog informacionog sistema kliničkog centra, ako hoćemo da otklonimo mogućnost pojave greške uslijed redundance onda tabele PACIJENT *(i eventualno* LABORATORIJA*) moraju biti dostupne ljekarima i osoblju na svim klinikama. To se može postići povlačenjem "veza" od* KLINIKA *do svih* PACIJENATA *i* LABORATORIJA*, pa dobijamo "paukovu mrežu" kako se to vidi na priloženoj skici 1.6.*

Povezivanje tabela međusobnim vezama u mrežnom modelu nije ničim limitirano osim logikom na kojoj se informacioni sistem bazira, a dijelovi mrežne strukture mogu biti i hijerarhijske pod-strukture.



Slika 1.6 Primjer mrežnog modela

Inače, i za mrežne strukture informacionih sistema važi većina nedostataka navedenih kod hijerarhijskih, pa se zbog toga rjeđe nalaze u praksi i gotovo je sigurno da će za kratko vrijeme potpuno iščeznuti.

1.4.3 Relacioni model

Relacioni model je danas najrasprostranjeniji model informacionog sistema zahvaljujući pre svega sledećim osobinama:

- *struktura modela je jednostavna a baza podataka je predstavljena samo skupom tabela i funkcionalnim vezama među njima*,
- *razvijena je matematička teorija koja omogućava jednostavnu interpretaciju ovog modela na računaru.*

Kao što mu i samo ime govori, ovaj model se zasniva na *tabelama* i *relacijama među njima* koje nisu određene ni *hijerarhijski*, ni *mrežno*, nego *funkcionalno*.

1.4.4 Objektni model

Objektni model je danas najrasprostranjeniji model informacionog sistema pre svega u fazi projektovanja. Postoji čitav niz oblasti primene računara u kojima relacioni model nije pogodan za primenu. To su pre svega računarom podržano projektovanje, multimedijalni sistemi i baze znanja.

Objektni model se zasniva na idejama objektno-orijentisanih programskih jezika i semantičkih modela podataka. Ovaj model je još u razvoju i nije unificiran. No, mnogi isporučioci softvera već nude svoje proizvode i u ovoj oblasti.

Glava 2

2.0 Relacioni model - uvod

Principe i strukturu relacionog modela objavio je *E. F. Codd 1970. godine* u svome radu:

"A Relation Model of Data for Large Shared Data Banks" Communications of the ACM, Volume 13, Number 6, (1970.), pages 377-387.

Od tada se ovaj model stalno usavršava i danas je za organizaciju, manipulaciju i obradu podataka opšteprihvaćen u svijetu.

Njegova osnovna prednost, u odnosu na hijerarhijski i mrežni model, je u tome što se u potpunosti "oslanja" na matematičku disciplinu, takozvanu *relacionu algebru***,** čime je omogućena računarska podrška, razvoj specifičnog softvera, i obrada uz zagarantovanu konzistentnost podataka i rezultata a objektno orjentisani programski jezici (na primjer Visual Basic) u potpunosti zadovoljavaju sve zahtjeve za implementaciju operacija pomenute relacione algebre.

2.1 Osnovne definicije

Na prvi pogled izgleda da se sistem, formiran kao relaciona baza podataka, sastoji od nasumice "nabacanih" i međusobno nevezanih, *tabela* u kojima za svaki objekat sistema koji je predstavljan tabelom:

- *kolone predstavljaju atribute (sa podacima),*

a

- *redovi n-torke ili slogove (sa podacima).*

Međutim, kada se malo bolje sagleda suština relacione organizacije, tek tada prednosti ovog modela postaju evidentne. Stoga je potrebno objasniti osnovne principe na kojima se bazira relacioni model i definisati osnovne pojmove u njemu. To su:

- *relacija,*
- *entitet,*
- *atribut,*
- *domen,*
- *kardinalnost relacije,*
- *primarni*
- *sekundarni ključ.*

2.1.1 Pojam relacije u relacionom modelu

Relacija, osnovni element relacionog modela ustvari je *sinonim, uz neka ograničenja, za tabelu ili datoteku*. E. F. Codd je u svom radu, kojim je po prvi puta "promovisao" relacioni model, pod relacijom definisao pravougaono područje - tabelu koje se sastoji od kolona (*atributa* i vrijednosti atributa - *podataka*) te redova (*n-torki*[1], odnosno *slogova sa podacima*). Ovako definisana tabela predstavlja skup vrijednosti ali postoje i određeni uslovi koje neka tabela mora da zadovolji da bi bila i *relacija*. Ti uslovi su:

a. Sve vrijednosti podataka jednog atributa moraju biti istog tipa.

> *Na primjer, sve vrijednosti atributa "dan_mjesec_godina" moraju biti datumi, sve vrijednosti atributa "plata" moraju biti numeričkog, dakle brojnog karaktera, vrijednosti atributa "ime i prezime" moraju imati slovni (alfanumerički) karakter itd.*

Unutar jedne relacije, svaki atribut može biti drugog tipa, što je sa stanovišta računarskog softvera kvalitativna novina koja je tražila doradu i proširenje do tada poznatih programskih jezika (Fortran, Pascal**,** itd.), kod kojih je, u njihovim verzijama, matrična varijabla (dakle tabela) morala imati sve vrijednosti varijable (podatke) istog tipa (REAL, INTEGER, LOGICAL itd.).

Svaki podatak u tabeli-relaciji predstavlja samo skup znakova i ništa više. Na osnovu jednog podatka u relaciji ne može, i ne smije se, doznati niti zaključivati ništa o vrijednostima drugih atributa u istoj ili drugoj n-torci te relacije. Drugim riječima:

[1] *U svetskoj literaturi i kod nas, odomaćio se novi izraz "torka" (engl. tuple) koji označava jedan zapis ili slog.*

u relacionoj bazi podataka ne smiju postojati funkcionalne zavisnosti među atributima.

b. Unutar jedne relacije ne smiju postojati dvije identične n-torke, dvije n-torke sa identičnim vrijednostima atributa – identičnim podacima. Ovo je logičan zahtjev, jer dvostruko memorisanje podataka može biti štetno (redundanca podataka), a nije ni potrebno.

c. Redoslijed n-torki u relaciji je proizvoljan.

d. Svi atributi unutar jedne relacije moraju imati različita imena, dok je redoslijed njihovog navođenja također proizvoljan.

2.1.2 Entitet

Entitet je model dijela realnog svijeta opisanog ograničenim brojem atributa. Entitet je nešto što postoji i što se na jedinstven način može identifikovati. Svaki entitet mora imati jedno ili više svojstava (identifikator) koja ga jasno razlikuju od svih drugih entiteta.

2.1.3 Atribut

Atribut je jedno od svojstava *entiteta (objekta)* kojeg posmatramo, i o kojem sakupljamo podatke. Svi podaci u jednom redu, jednom slogu, odnosno jednoj n-torci, definišu jednu *jedinku* datog objekta, dok jedna *kolona* opisuje jedno *svojstvo svih jedinki*.

Na primjer, u objektu:

SLUŽBENIK < *JMBG, ime, prezime, dat_rođ, adresa, tel…,* >

svaka n-torka sadrži gore navedene podatke o jednom službeniku, dok kolona "JMBG", sadrži jedinstvene matične brojeve svih službenika tog preduzeća.

Atributi u trenutku skupljanja podataka u relacionoj bazi podataka ne moraju obavezno imati poznate vrijednosti i u tom slučaju ih nazivamo *Null* - vrijednostima a treba ih, posebno ako ih u nekoj relaciji ima više, izbjegavati.

Tako ne bi bilo racionalno u tabelu SLUŽBENIK *uvesti i atribut "odlikovanja", jer se odlikovanja ne dijele masovno i nasumice (bar u civilizovanim zemljama), pa će za većinu službenika u tom polju ostati prazno mjesto, to jest Null-vrijednost. Rješenje problema bilo bi u kreiranju nove tabele, nazovimo je:*

ODLIKOVANJA < *JMBG, medalja* >

a u kojoj bi se registrovali samo odlikovani službenici sa njihovim matičnim brojem (JMBG) i nazivom medalje koju su dobili, pa bi shodno tome relacija ODLIKOVANJA *bila potpuno popunjena, a preko matičnog broja JMBG bila bi povezana (u relaciji) sa osnovnom tabelom* SLUŽBENIK i ostalim podacima o njemu*.*

Pri projektovanju informacionog sistema, u skladu sa potrebama, treba veoma pažljivo i studiozno iz mnoštva mogućih atributa odabrati i definisati samo one koji opisuju objekat na, za nas, zadovoljavajući i prihvatljiv način. Na primjer:

U slučaju objekta SLUŽBENIK *kao atribut odabran je datum rođenja ("dat_rođ"**)** sa namjerom posjedovanja podataka o starosnoj dobi svakog službenika u preduzeću, a vjerovatno ne samo zbog mogućnosti čestitanja rođendana službenicima od strane direkcije firme.*

Nije usvojen atribut "godine_starosti" jer bi to bilo loše rješenje. Baza podataka morala bi se u tom slučaju svaki dan "osvježavati" s obzirom da su svi službenici svakog dana jedan dan stariji i uvijek postoji mogućnost da baš toga dana neko od službenika ima rođendan, da postaje i godinu dana stariji, što zahtijeva stalno ažuriranje baze, to jest izmjenu podatka u njoj.

U slučaju kada je atribut "datum rođenja", taj podatak je konstantan, a godine starosti se bez problema mogu, za svakoga, u svakom trenutku, izračunati iz poznatog tekućeg datuma.

2.1.4 Domen atributa

Domen atributa je skup vrijednosti koje on može da poprimi:

*U relaciji SLUŽBENIK**,** atribut "pol" može da poprimi samo dvije vrijednosti, "muško" ili "žensko".*

Domen za atribut "zvanje" ima onoliko vrijednosti koliko ima različitih zvanja službenika angažovanih u toj firmi.

Domen atributa "telefon" jednak je broju n-torki u relaciji (izuzev ukoliko dva službenika ne koriste isti telefonski broj, ili ako ima službenika bez telefonskog priključka).

2.1.5 Primarni ključ

Primarni ključ, ili često kraće samo ključ, je atribut (ponekad, ako je to neophodno, i skup, odnosno kombinacija atributa) čija vrijednost jednoznačno definiše samo jednu n-torku, samo jedan slog, u nekoj relaciji - tabeli, što znači da jednoznačno "izdvaja" samo jedan red - jedan slog, u njoj (jedinstvenost). Pri tome ne se postojati nijedan podskup tog skupa

atributa koji obezbeđuje jedistvenost (ova osobina primarnog ključa naziva se minimalnost).

Prema tome:

U jednoj relaciji ne smiju postojati dvije različite n-torke sa istom vrijednošću ključnog atributa.

Ukoliko ne postoji atribut koji zadovoljava uslove za ključni atribut, ključ se može kreirati kombinacijom dva ili više atributa.

Primarni ključ mora biti: jedinstven, nepromjenljiv i uvijek raspoloživ.

Izbor primarnog ključa je veoma važan jer se jedino preko njega može pristupiti jednoj, i samo jednoj, n-torci u nekoj relaciji, pa uz poznati naziv atributa možemo tako pristupiti i jednom, i samo jednom, podatku posmatranog objekta.

Primarni ključ se često obilježava (markira) prilikom definisanja relacije. U literaturi se ključni atribut najčešće označava *znakom* **#** isključivo iz razloga da bi tabela – relacija bila preglednija za korisnika. Neki savremeni softverski paketi koji se koriste za upravljanje relacionom bazama podataka (ACCESS, na primjer) kao oznaku za ključ koriste i doslovno znak ključa **(** 🔑 **)**.

Na primjer, u relaciji SLUŽBENIK**,** *kandidat za primarni ključ na našim prostorima je matični broj građanina (JMBG), s obzirom da od svih navedenih atributa u relaciji* SLUŽBENIK *samo za matični broj možemo biti sigurni da je jedinstven, to jest da ne postoje dvije osobe sa istim matičnim brojem.*

Takvu tvrdnju ne možemo, na primjer, izreći za ime, prezime, adresu ili telefonski broj. Ali ako u tom preduzeću može biti angažovan i neki stranac (koji nema naš JMBG) onda se za ključni atribut mora tražiti neko drugo rješenje.

Relacija SLUŽBENIK sa obilježenim ključem glasi:

SLUŽBENIK < *JMBG#, ime, prezime, dat_rođ, adresa, telefon…* >

Ukoliko u nekoj relaciji nismo u stanju da "izdvojimo" kandidata za ulogu primarnog ključa, problem se rješava na dva načina:

- *ili definisanjem grupe atributa (složeni ključ) koji onda imaju jedinstvenu vrijednost za svaki slog, ili*
- *uvođenjem novog, dodatnog, atributa, nazovimo ga jednostavno "šifra"**,** pri čemu se onda pogodnim izborom te šifre (na primjer, redni brojevi) obezbjeđuje njena jednoznačnost.*

Na kraju, napomenimo da neki savremeni softverski paketi (ORACLE, na primjer) dopuštaju da relacija sadrži i identične n-torke,

dakle da u relaciji ne mora biti definisan primarni ključ. Razlog za to je unos podataka iz drugih izvora, na primjer iz programa za rad sa tabelama (EXCELL, na primjer), koji po pravilu nemaju mehanizme koji bi spriječili pojavu duplikata (identičnih vrijednosti ključnih atributa ili čitavih n-torki).

Naravno, postoje postupci pomoću kojih se ovakvi zapisi mogu eliminisati iz tabela ako je to potrebno. Međutim, jedno je sigurno:

nepostojanjem primarnog ključa gubi se mogućnost direktnog pristupa u nekoj relaciji jednoj i samo jednoj n-torci.

2.1.6 Spoljni (strani) ključ

Spoljni ključ (na primjer u relaciji A) je atribut (skup atributa) koji u drugoj relaciji (na primjer B) ima ulogu primarnog ključa. Osnovna uloga spoljnih ključeva je uspostavljanje veza (relacija) među tabelama, a ne identifikacija n-torki. Na primjer u relacijama:

BOLNICA < šifrabol#, naziv_bolnice, adresa, telefon,...>

LJEKAR < šifraljek#, šifrabol, ime, prez, adresa,.....>

"šifrabol#" je u relaciji BOLNICA *primarni ključ, a u relaciji* LJEKAR *atribut, i služi za povezivanje relacija. Na taj način možemo, na primjer, za ljekara čiju šifru poznajemo, dobiti telefonski broj bolnice u kojoj radi, unoseći poznatu šifru bolnice (šifrabol#) iz njegove n-torke u relaciji* LJEKAR*, u relaciju* BOLNICA *(gdje je šifrabol# primarni ključ), očitavajući nakon toga vrijednost atributa "telefon" u n-torci relacije* BOLNICA*.*

Spoljni ključ (naziva se još i *strani ključ*), može, ali i ne mora se, posebno obilježiti (sa ! ili $). I to se opet koristi samo u tekstu, jer to obilježavanje isključivo doprinosi vizuelnoj preglednosti modela. Jedino ako je strani ključ ujedno i dio nekog primarnog složenog ključa, onda ga treba obilježiti uobičajenom oznakom (#) za primarni ključ.

2.2 Codd-ova pravila

E.F. Codd je u svojim radovima definisao 12 pravila od kojih najmanje 6 mora biti ispunjeno da bi se informacioni sistem mogao smatrati relacionim. Ta pravila su rezultat teorijskog pristupa ovome problemu. S obzirom na obim ove knjige ograničimo se na definiciju 6 osnovnih pravila, ne upuštajući se i u njihovo dokazivanje.

Osnovno ili nulto pravilo koje predstavlja “conditio sine qua non“ u relacionim bazama podataka glasi:

Svaki sistem za upravljanje bazama podataka, koji se smatra, ili koji jeste, relacioni, mora imati mogućnost upravljanja bazom podataka na relacioni način i relacionim metodama.

Ostala pravila su:

1. *Predstavljanje informacija*

   Sve informacije u relacionoj bazi podataka moraju logički biti predstavljene na isti način: vrijednostima podataka u tabelama - relacijama.

2. *Logička dostupnost podacima*

   Svaki podatak mora imati "atomarnu", nedjeljivu vrijednost, koja logički mora biti dostupna korisniku uz pomoć:

   - *imena relacije u kojoj se nalazi,*
   - *vrijednosti primarnog ključa te relacije i*
   - *imena atributa u toj relaciji.*

3. *Mogućnost prikazivanja nepostojeće informacije*

   Relaciona baza podataka podržava koncept nultog podatka, (Null). Pod pojmom nultog podatka podrazumijeva se vrijednost atributa koja u datom trenutku nije poznata. To, dakle, nije vrijednost koja je predstavljena nulom, ili nizom nula, nizom praznih (blank) mjesta, ili nizom bilo kojih drugih znakova. To je jednostavno, trenutno nepoznata vrijednost podataka, i predstavlja specifikum i kontroverzniji aspekt relacionog modela informacionog sistema.

   Tako, na primjer, atribut "telefon" u relaciji SLUŽBENIK ima Null-vrijednost u trenutku primanja nekog službenika u radni odnos ukoliko taj službenik nema telefonski broj.

4. *Dinamički katalog*

   Relaciona baza podataka mora biti tako prezentirana da autorizovani korisnici mogu primijeniti neki od "relacionih jezika" na podatke u njoj. Pod ovim se podrazumijeva:

   - *pristup sistemskim katalozima,*
   - *pristup relacijama koje su u toku rada za stalno, ili samo na ograničeni vremenski rok prisutne i*
   - *pristup relacijama koje se programski kreiraju (tzv. dinamički katalozi) a koji sadrže nove informacije.*

5. *Softverski paket za manipulaciju bazom podataka*

Softverski paketi omogućavaju manipulisanje podacima u relacionoj bazi podataka, i oni su manje ili više prilagođeni korisniku - manipulatoru. Svaki od njih sadrži programski jezik koji pomoću definisane sintakse, i na korisniku blizak način, omogućava:

- *definisanje tabela,*
- *definisanje atributa,*
- *unošenje podataka,*
- *definisanje proizvoljnog "pogleda"*
- *definisanje proizvoljnog upita,*
- *manipulaciju podacima,*
- *postavljanje ograničenja korisnicima,*
- *autorizaciju korisnika, te*
- *upravljanje proizvoljnim transakcijama.*

6. Nezavisnost integriteta baze

Nijedno ograničenje integriteta (sigurnosti očuvanja podataka) ne smije se pri organizaciji logičkog modela relacione baze podataka naći u aplikativnom dijelu softverskog programa dostupnom širem broju korisnika, nego isključivo u dijelovima koji su pod kontrolom administratora baze. Drugim riječima, pravila integriteta se definišu u okviru sinteze baze podataka.

Nakon objavljivanja Codd-ovog rada veliki broj autora nastavio je da sa bavi teorijom i praksom sinteze i eksploatacije baze podataka. Tako je naknadno, nakon 12 Cood-ovih pravila definisano još šest dodatnih a mogu se naći u stručnoj literaturi.[2]

2.3 Ograničenja i pravila pri projektovanju

U relacionom modelu potrebno je u pojedinim relacijama ograničiti vrednosti nekih atributa. Već postojanje domena atributa predstavlja određeno ograničenje (starost, cijena, itd...). Ali postoje i tzv. opšta ograničenja koja važe za svaki relacioni model i koja se nazivaju *pravilima integriteta relacionog modela.* To su:

1. Integritet entiteta: Nijedan atribut koji je primarni ključ ili dio primarnog ključa, ne smije nikada da uzme *null* vrijednost.

2. Referencijalni integritet: Skup vrijednosti spoljnjeg ključa relacije R1 mora biti podskup skupa vrijednosti primarnog ključa relacije R2 sa kojom se povezuje R1. Relacije R1 i R2 moraju biti različite. Pored ogra-

[2] *Ratko Vujnović: SQL Relacijski model podataka, Znak, Zagreb 1996.*

ničenja koja su zadana strukturom modela, mogu se, za bolju specifikaciju semantike, iskazati dodatna, eksplicitna ograničenja. Na primjer:

Neka jednostavna ograničenja mogu se iskazati preko definicije domena atributa (ocena na ispitu mora biti veća ili jednaka 5 a manja ili jednaka od 10).

Neka složenija ograničenja označavaju "poslovni integritet". Na primjer, student da bi upisao narednu godinu, na nekim fakultetima mora položiti sve ispite iz prethodne godine, a na nekim može prenijeti dva ispita.

Ostala pravila kojima se definiše kvalitet, i koja mora da ispuni svaki sistem za upravljanje bazama podataka da bi se smatrao relacionim, mogu se ukratko sažeti kao:

a. programskim jezikom treće generacije ne smiju se zaobići pravila integriteta definisana relacionim jezikom.

b. mora postojati mogućnost kreiranja "pogleda" na bazu podataka i definisanja operacija nad njima te izvršavanja operacija održavanja baze podataka.

c. aplikacioni programi[3] *moraju ostati nepromjenjeni ako se promjene bazne relacije u sistemu (sem u slučaju promjene strukture tabela);*

d. aplikacioni programi moraju ostati nepromjenjeni i ako se promjeni fizička organizacija baze podataka ili fizički metod pristupa;

Nažalost, danas je situacija takva da većina sistema za održavanje baza podataka ne zadovoljava sve navedene kriterijume. Problemi koji se javljaju uvijek su u vezi održavanja integriteta baze.

Da rezimiramo*: relacioni model je danas dominantan zahvaljujući strogoj matematičkoj teoriji na kojoj se bazira.* Softverski paketi, koji se koriste u manipulisanju relacionim bazama podataka međusobno su slični pa korisnik, koji je savladao jedan, sa malo truda može da se "prebaci" i na neki drugi.

Uvođenjem tehnike objektnog programiranja, dizajn aplikativnih programa postao je "blizak" korisniku, jer većina novih paketa sadrži podprograme prilagođene manipulaciji podacima, kao i gotove "generatore" za razne obrasce, izvještaje i aplikacije. Hardverska konfiguracija savremenih personalnih računara omogućava postavljanje i najvećih softverskih paketa (ORACLE, na primjer) na danas već praktično standardnim konfiguracijama personalnih računara.

[3] *Programi kojima se korisniku omogućava jednostavno korišćenje nekog informacionog sistema.*

2.4 Distribuirane baze podataka

Pod distribuiranom bazom podataka smatra se informacioni sistem koji radi u računarskoj mreži i koji ima najmanje dva dislocirana računara, sa dislociranim tabelama, u kojima se nalaze podaci.

Distribuirane baze podataka zbog svoje kompleksnosti donose i niz problema, kako u njihovom projektovanju, tako i u eksploataciji. Pri tome, do danas još nije u potpunosti razvijena matematička teorija koja bi "*podržavala*" ovakve sisteme što donekle otežava njihovu sintezu posebno u smislu "*tajnosti*" i "*privatnosti*". Osnovni problemi na čijem rješavanju se danas radi su:

- *pravo pristupa (ko, kada i kojim podacima može pristupiti),*
- *očuvanje podataka (ko ima pravo promjene podataka), i*
- *potpuna tajnost podataka.*

Organizacija distribuiranih baza podataka je tako postavljena da se globalni informacioni sistem sastoji od više lokalnih baza podataka, a uvijek postoje najmanje dva upravljačka nivoa, sistema, i to:

- *lokalni upravljački sistem (*LDBMS - *Local Database Management System), i*
- *globalni sistem (*DDBMS - *Distributed Database Management System).*

Pomenuti sistemi su isključivo logički, dakle međusobno su samo softverski povezani, tako da korisnik distribuirane baze podataka sa svog radnog mjesta, sa svog računara, "*ne vidi*" tu distribuiranost i ima osjećaj da je povezan samo sa jednim centralizovanim informacionim sistemom. Evolucija načina obrade zajedničkih podataka tekla je od obrade na jednom centralnom računaru, preko obrade na nepovezanim računarima, do obrade podataka u mrežnom okruženju, u početku sa serverom[4] datoteka, a danas sa distribuiranom obradom.

Kod prvog modela sa centralnim računarom – serverom, podaci i aplikacije su se nalazili na jednom centralnom računaru (*host*), a zaposleni su mu pristupali preko ''*glupih*'' terminala na kojima nije postojala mogućnost obrade.

Aplikacije su pri tome obično podjeljene na dva dijela:

- *čeoni dio (front end) odgovoran za komunikaciju sa korisnikom i*
- *pozadinski dio (back end) odgovoran za upravljanje podacima.*

Prednosti ovakvog rješenja su:

- *lako administriranje,*

[4] *Centralni, glavni računar koji upravlja računarskom mrežom*

- *pouzdanost podataka*
- *brz pristup,*
- *zaštita podataka (rezervne kopije) i*
- *zajedničko korišćenje skupih periferijskih uređaja.*

Mane ovoga rješenja su:

- *potreba za inoviranjem centralnog računara te,*
- *mali broj proizvođača namjenskih računara (visoke cijene).*

Obrada podataka na nepovezanim personalnim računarima počinje pojavom personalnih računara na tržištu i operativnih sistema DOS, Unix i Windows kada je veliki broj proizvođača ovakvih računara oborio cijene i učinio ih dostupnim velikom broju korisnika.

Rad na samostalnim radnim stanicama ima mnogo prednosti:

- *jeftine su i jednostavne za upotrebu,*
- *korisnik sam prilagođava radnu stanicu svojim potrebama,*
- *dostupno je mnoštva "alata" i aplikacija od raznih proizvođača.*

Naravno, ima i nedostataka – mana. To su:

- *podaci su raspodijeljeni na više računara,*
- *podaci na jednoj stanici su autonomni i korisnik je odgovoran za upravljanje podacima,*
- *podaci na jednom računaru nisu uvijek dostupni svim korisnicima kojima mogu zatrebati jer se ne mogu koristiti zajednički skupi uređaji.*

U posljednjih desetak godina trend razvoja računarske tehnike ide u pravcu stvaranja većih, globalnih računarskih mreža. U prvo vrijeme su to bile lokalne mreže (u okviru fakulteta, ustanove, preduzeća itd.), zatim je došlo do njihovog povezivanja (na nivou univerziteta, industrijskog koncerna, itd.), da bi se konačno došlo i do globalnih svjetskih mreža (danas je najpoznatija od njih - *Internet*). Da bi mogli da koriste zajedničke podatke u lokalnoj mreži datoteke se smještaju na *server datoteka* (*file server*). To je samostalan računar koji je obično centralno čvorište preko kojeg se koriste zajednički resursi (na primjer diskovi i štampači). Program za upravljanje bazom podataka, izvršava se na svakoj radnoj stanici u mreži, a uzima podatke sa servera i kasnije ih vraća i kopira opet na server. Ovaj koncept nije dobar kod višekorisničkih aplikacija jer sistem sa serverom ne dopušta istovremeno višestruko pristupanje istom skupu podataka (*data concurrency*), veliki je "saobraćaj" na mreži kada mnogo korisnika radi, i zato lako dolazi do zagušenja.

Činjenica da je korisnik računara koji je u mreži u stanju da, uz poznavanje odgovarajućih lozinki i ključnih riječi, pristupi praktično svakom računaru i svakom podatku u toj mreži, odrazila se i na daljni razvoj

informacionih sistema jer se podaci više ne nalaze na jednom mjestu, oni su distribuirani, a korisnik treba da zna kako i gdje da ih pronađe. Dakle dolazi se do obrade po modelu klijent/server, odnosno do distribuirane obrade aplikacija (*distributed application processing*) koja objedinjuje dobre strane obrade u mrežnom okruženju sa visokim performansama centralizovanih sistema. Ovi sistemi su višeslojni i sadrže tri komponente:

- *server baze podataka,*
- *aplikacije klijenata i*
- *mrežu.*

Serveri (*back end*) obezbjeđuju efikasno upravljanje resursima posebno u uslovima kada više klijenata istovremeno zahtijeva isti resurs. Oni obezbjeđuju:

- *upravljanje bazom podataka,*
- *kontrolu pristupa (i bezbjednost podataka) i*
- *centralizovano zadovoljenje pravila integriteta.*

Aplikacija - klijent (*front end*) omogućava svim korisnicima komforan rad sa podacima a obezbjeđuje:

- *najpogodniji interfejs prema korisniku za određenu vrstu obrade,*
- *upravlja načinom prikaza podataka korisniku,*
- *izvršava logiku aplikacije,*
- *proverava ispravnost ulaznih podataka i traži i prihvata podatke od servera*.

Dio ovih aktivnosti u distribuiranoj obradi (poslednje tri) mogu se realizovati i kao poseban sloj u obradi podataka pa mogu biti locirane bliže serveru, takozvani aplikacioni server (*application server*). Ovo je srednji sloj (*midle tier*), višeslojne arhitekture, pa ovakvi sistemi imaju i određenih prednosti. To su rije svega:

- *svaki računar u sistemu može biti izabran posebno, tako da najbolje ispunjava zahteve obrade,*
- *sistem je fleksibilan i otvoren u pogledu izmena hardvera i softvera i sistem se lako može proširivati.*

Na kraju, treba istaći i mogućnost specijalizacije i zasebnog razvoja svake funkcionalne komponente kao i upotrebu grafičkog radnog okruženja i objektno orijentisanih alata.

Glava 3

3.0 Osnove relacione algebre - uvod

Za manipulisanje podacima i tabelama u relacionim bazama podataka potrebna su osnovna znanja iz *relacione algebre*. Relaciona algebra spada u matematičku oblast teorije skupova, relativno je nova disciplina, i na njoj se bazira relacioni model baze podataka.

Operatori relacione algebre dijele se u dvije grupe i to:

- *osnovni operatori,*
- *operatori pridruživanja.*

Relacioni operatori su sa stanovišta matematičke teorije operatori visokog nivoa jer operišu sa relacijama, dakle sa skupovima vrijednosti (tabelama), a ne samo sa jednom, i kao rezultat daju opet relaciju – skup vrijednosti – novu relaciju.

E.F.Codd je relacione operatore podijelio u dvije grupe:

- *operatore koji su pogodni za ažuriranje*
- *operatore koji su pogodni za izvještavanje.*

3.1 Operatori pogodni za ažuriranje

Tradicionalni operatori izvode se nad minimum dvije relacije. To su:

- *unija (UNION),*
- *presjek (INTERSECT),*
- *razlika (DIFFERENCE),*
- *proizvod (CARTESIAN PRODUCT).*

3.1.1 Unija

Unija dva skupa, dvije relacije A i B, je relacija koja se sastoji od svih elemenata koji pripadaju relacijama ili A ili B.

Svaka relacija je po definiciji skup n-torki, pa je i unija dvije relacije skup n-torki, ali ne mora u opštem slučaju biti i relacija. Relacija, naime, ne smije sadržavati različite tipove n-torki pa se teoretski može napraviti unija od dvije relacije koja ima različite atribute. Rezultat je u tom slučaju tabela, ali nije i relacija.

Da se ovo ne bi desilo definišu se i ograničenja koja moraju biti zadovoljena kako bi nad dvije relacije bila izvodljiva operacija *unija*, a da rezultat pri tome opet bude relacija. Za takve relacije se kaže da su *union-kompatibilne*. Ta ograničenja su:

1. *obje relacije moraju imati iste atribute,*
2. *isti atributi moraju biti definisani nad istim domenom.*

Operacija unija nad relacijama A i B simbolički se označava sa: A∪B.

Primjer: Unija dvije relacije A i B:

A

ŠIFRA #	*PREZIME*	*IME*	*TEL. BROJ*
3244	*Aksentijević*	*Petar*	*0710 334 952*
1772	*Maksimović*	*Ilija*	*015 723 543*

B

ŠIFRA #	*PREZIME*	*IME*	*TEL. BROJ*
3244	*Aksentijević*	*Petar*	*0710 334 952*
2345	*Petrović*	*Dara*	*081 17 318*

je relacija (C=A∪B) sa istim atributima i eliminisanim višestrukim, identičnim, n-torkama dakle:

C=A∪B

ŠIFRA#	*PREZIME*	*IME*	*TEL:BROJ*
3244	*Aksentijević*	*Petar*	*0710 334 952*
1172	*Maksimović*	*Ilija*	*015 723 543*
2345	*Petrović*	*Dara*	*081 17 318*

3.1.2 Presjek

Presjek dvije relacije A i B (označava se sa A∩B) je nova relacija koja sadrži sve n-torke koje su zajedničke za obje relacije.

U prethodnom primjeru to je telefonski pretplatnik sa šifrom 3244, jer je on prisutan u obje relacije:

$C = A \cap B$

ŠIFRA #	PREZIME	IME	TEL. BROJ
3244	Aksentijević	Petar	0710 334 952

3.1.3 Razlika

Razlika A - B dvaju relacija (razlika se označava i sa A/B) je nova relacija koja ima iste atribute kao i relacije A i B**,** a tijelo se sastoji samo od onih n-torki koje se nalaze u relaciji A**,** a ne nalaze u B. Prema tome za razliku važi pravilo:

$$A - B \neq B - A.$$

U prethodnom primjeru rezultat razlike bio bi shodno tome:

$A - B$ *ili* (A / B)

ŠIFRA #	PREZIME	IME	TEL. BROJ
1772	Maksimović	Ilija	015 723543 543

a

$B - A$ *ili* (B / A)

ŠIFRA #	PREZIME	IME	TEL. BROJ
2345	Petrović	Dara	081 17318

3.1.4 Proizvod

Pojam proizvoda u relacionoj algebri je nešto širi od pojma prostog Dekartovog, odnosno Kartezijevog proizvoda. Naime, Dekartov proizvod dva skupa A i B, definiše se kao skup uređenih parova u kojem prvi element pripada skupu A, a drugi skupu B.

U relacionoj algebri, međutim, uvijek želimo da dobijemo uređen skup n-torki, a ne uređen skup parova, pa se stoga definicija Dekartovog skupa proširuje na taj način što se umjesto skupa elemenata uzima skup n-torki, pri čemu je svaka tako novodobijena n-torka rezultat spajanja uređenog para n-torki.

Treba napomenuti da kod izvođenja proizvoda dvije relacije postoji opasnost da dođe do greške ukoliko te dvije relacije imaju atribute sa istim imenima, a nemaju isto značenje.

Ilustracije radi pogledajmo primjer proširenog Kartezijevog proizvoda relacija ALFA i BETA

Ali ako bi željeli da napravimo proizvod relacija:

PROFESOR <*šifra#, ime, prezime, zvanje, adresa, ... >*

i

STUDENT <*šifra#, ime, prezime, adresa,.....>*

to ne bi bilo moguće, jer se imena atributa (ime, prezime i adresa) ponavljaju. Rješenje u ovakvim slučajevima je u preimenovanju atributa u jednoj od relacija, na primjer u relaciji STUDENT:

STUDENT <*šifrast#, imest, prezimest, adresast,.....>*

ALFA

A	B
a_1	b_1
a_2	b_2
a_3	b_3

*

BETA

C	D	E
c_1	d_1	e_1
c_2	d_2	e_2

=

*ALFA*BETA*

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_2	d_2	e_2
a_2	b_2	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_1	d_1	e_1
a_3	b_3	c_2	d_2	e_2

3.2 Operatori pogodni za izvještavanje

U specijalne operatore spadaju:

- *selekcija,*
- *projekcija,*
- *spajanje,*
- *dijeljenje.*

3.2.1 Selekcija

Selekcija, ili kako se još naziva *ograničenje ili restrikcija*, izdvaja iz relacije samo one n-torke koje zadovoljavaju zadani kriterijum (uslov), koji je definisan logički. *N-torke* u kojoj je taj logički uslov zadovoljen, definišu onda novu relaciju.

Na primjer, nad relacijom ROBA*:*

ROBA < *šifra#, naziv, proizvođač, datum, adresa,....>*

možemo napraviti selekciju po atributu "adresa", i tako iz relacije ROBA *izdvojiti samo one n-torke za koje je vrijednost atributa "adresa" neka unapred zadana, na primjer:*

adresa = "Trebinje"

Na taj način dobijamo novu, izvedenu relaciju, koja onda mora imati i novo ime.

Treba naglasiti da upit kojim se vrši selekcija mora uvijek biti logičan i izvodljiv. U protivnom se selekcija ne može provesti.

3.2.2 Projekcija

Projekcija relacije daje novu relaciju koja se sastoji samo od određenih (ili samo jednog) atributa zadane relacije. Rezultat operacije projekcija je podskup izabranih atributa neke relacije sa svim njenim n-torkama. Na primjer:

Projekcija relacije ROBA *iz malopređašnjeg primjera po atributima šifra#, naziv i adresa proizvođača bila bi:*

ROBA1 < *šifra#, naziv, adresa* >

a imala bi isti broj n-torki kao i relacija ROBA, *s obzirom da ne mogu postojati dvije n-torke u relaciji* ROBA *sa istom šifrom.*

Često se dešava da se primjenom *projekcije*, iz nesmotrenosti, mogu izgubiti neki podaci jer u novonastaloj tabeli mogu da se pojave identične n-torke (što u slučaju selekcije nije moglo da se desi), koje onda u nekim softverskim paketima bivaju bez upozorenja brisane, kako bi dobijeni rezultat ponovo bila relacija.

Na primjer, projekcija relacije:

STUDENT < *broj_ind#, ime, prezime, ime_oca, dat_rod,...* >

po atributu "broj_ind#" ima sigurno isti broj n-torki kao i relacija STUDENT *s obzirom da ne postoje dva studenta sa istim brojem indeksa.*

Ali, projekcija po atributu "ime" imaće po svoj prilici manji broj slogova jer postoji velika vjerovatnoća da će se pojaviti dva ili više studenta sa istim imenom, pa će u projektovanoj relaciji ostati samo jedno od njih jer se identične n-torke eliminišu.

3.2.3 Spajanje (Join)

Operacija spajanja ima više podvrsta od kojih su dvije najvažnije:

- *prirodno spajanje,*
- *spajanje pod nekim uslovom.*

Prirodno spajanje relacija A i B daje relaciju AB koja ima sve atribute relacije A, i one atribute relacije B koje nema relacija A.

Na primjer, relacije A i B

A < x1, x2, x3,....xn, y1, y2,....ym >
B < y1, y2,...,ym, z1, z2,....zp >

spojene prirodno daju relaciju AB:

AB < x1, x2,.....xn, y1, y2,.....ym, z1, z2,.....zp >

ili, relacije ALFA i BETA:

ALFA

ŠIFRAD#	NAZIV	MJESTO
d001	Comex	Toronto
d002	Unita	Vancouver
d003	Dual	Beograd

BETA

ŠIFRAD#	ŠIFRAP#	BROJ KOM.
d001	p991	324
d002	p678	23
d003	p007	12564

spojene prirodnim spajanjem daju kao rezultat relaciju GAMA

GAMA

ŠIFRAD#	NAZIV	MJESTO	ŠIFRAP#	BROJ KOM.
d001	Comex	Toronto	p991	324
d002	Unita	Vancouver	p678	23
d003	Dual	Beograd	p007	12564

Ako relacije koje se prirodno spajaju nemaju nijedan zajednički atribut, onda operacija prirodnog spajanja prelazi u Kartezijev-Dekartov proizvod.

Spajanje pod nekim uslovom (Ψ) izvodi se nad relacijama samo onda kada one nemaju nijedan isti atribut. Rezultat spajanja je u tom slučaju Kartezijev proizvod tih relacija koji sadrži samo one n-torke koje zadovoljavaju logički uslov definisan izrazom (Ψ), pa se ovakav način spajanja zato i naziva *Ψ-spajanje***.**

3.2.4 Operacija dijeljenja

Dijeljenje se ne može izvesti sa proizvoljnim relacijama - tabelama. Da bi operacija A podijeljeno sa B (A:B) bila izvodljiva, potrebno je da se svi atributi relacije B nalaze i u relaciji A.

Na primjer, ako imamo dvije relacije A i B:

$$A < x1, x2,....,xn, y1, y2,..., ym > ;$$

$$B < y1, y2,..., ym >$$

(koje zadovoljavaju postavljeni uslov), rezultat dijeljenja će biti relacija C koja ima samo x-atribute, a tijelo joj se sastoji od onih n-torki relacije A za koje se vrijednosti y-atributa pojavljuju u relaciji B. Dakle*:*

A

X#	Y#
017	a22
033	a43
077	a86
061	a43
044	a43

•
•

B

Y#
a43

=

C

X#
033
061
044

3.3 Dodatni operatori

Pored navedenih operatora u modernoj relacionoj algebri postoji još nekoliko dodatnih, izvedenih, operatora koje su definisali autori poslije E.F.Codda jer se pokazalo da osam osnovnih nije uvijek moglo zadovoljiti sve zahtjeve. Tako se danas koriste još i operatori:

- *proširenja,*
- *agregacije,*
- *uopštenog dijeljenja,*
- *spoljnjeg spajanja*
- *uslovni operator (MAYBE).*

Najinteresantniji od njih je operator *MAYBE* koji se koristi za manipulisanje *Null* vrijednostima, i predstavlja proširenje klasične logičke algebre. Naime, u klasičnoj logičkoj algebri postoje samo dvije moguće vrijednosti, dva stanja, koje logička varijabla, ili izraz, mogu uzeti. To su:

- *istina (TRUE)*
- *laž (FALSE).*

Uvođenjem *Null* vrijednosti definisana je još jedna, treća, mogućnost, označimo je sa U (*unknow - nepoznato*) pa logičke operacije AND, OR i NOT rezultiraju sada sa tri stanja i to:

- *T (TRUE),*
- *F (FALSE)*
- *U (UNKNOWN).*

Definicija logike tri stanja nije još opšteprihvaćena, ali se najčešće koristi ona po kojoj svaki operator daje rezultat *U (Null)* uvijek onda kada je ž neke operand nepoznat. Priloženi grafički prikaz daje definiciju najvažnijih operatora.

AND

	T	F	U
T	T	F	U
F	F	F	U
U	U	F	U

OR

	T	F	U
T	T	T	U
F	T	F	U
U	T	U	U

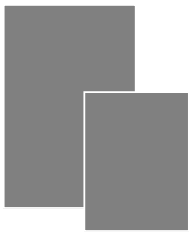
MAYBE

T	F
F	T
U	U

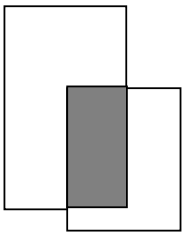
NOT

T	F
F	F
U	U

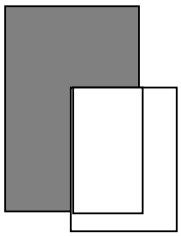
Unija



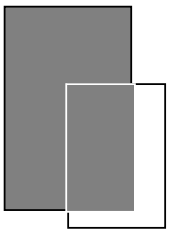
Presjek



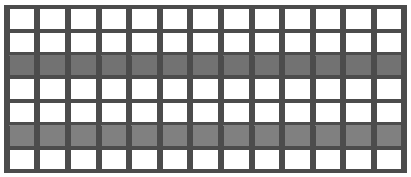
Razlika A-B



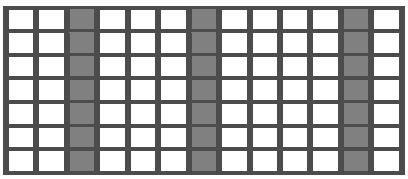
Razlika B-A



Restrikcija



Selekcija



Prirodno spajanje

A 1	B 1
A 2	B 2
A 3	B 3
A 4	B 4

+

B 1	C 1
B 2	C 2
B 3	C 3
B 4	C 4

=

A 1	B 1	C 1
A 2	B 2	C 2
A 3	B 3	C 3
A 4	B 4	C 4

Glava 4

4.0 Sinteza relacionog modela - uvod

Relacioni model baze podataka sastoji se od dva dijela i to:

- *logičkog*

i

- *fizičkog modela.*

Logički model baze podataka nastaje kao rezultat:

- *analize* postojećih i / ili dobijenih *novih podataka*,
- *definisanja* tabela (*relacija*) *i veza* među njima, te
- *dovođenja modela na relacioni oblik,*

uz prethodno definisanu:

- *strukturu i oblik* podataka koji će činiti *sadržaj* baze..

Fizički model baze podataka određuje kako su podaci memorisani na memoriji računara i kako se njima manipuliše - danas najčešće na disku.

Problemu sinteze logičkog modela pristupa se na dva načina i to:

- *preko modela objekat-veze* (MOV, Entity-Relationship Model, skraćeno E-R model),

ili

- postupkom *normalizacije tabela*,

pri čemu treba naglasiti da jedan pristup ne isključuje drugi, nego se naprotiv, često, međusobno i dopunjuju.

4.1 E-R model

Ovaj pristup sintezi relacione baze podataka prikazan je po prvi puta u radu: CHEN, P. P. *The Entity Relationship* objavljenom 1976. godine. Najkraća definicija ovog postupka bi bila**:**

dobijanje saznanja o objektima, vezama među njima, te njihovim svojstvima.

Chen je predložio da se model nazove *E-R model* (***E**ntity – **R**elation*, (*entiteti – relacije*) to jest, *objekti i veze* među njima).

U našoj literaturi ovaj model se naziva i *MOV* (skraćenica od ***M**odel **O**bjekat-**V**eze*).

4.1.1 Osnovne definicije i pojmovi E-R modela

U pomenutom radu Chen je uveo nekoliko novih pojmova koji u suštini ne mijenjaju ništa u osnovi u Cood-ovog relacionog modela, ali umnogome olakšavaju i ubrzavaju njegovu optimalnu sintezu. Objekte opisane atributima Chen je podijelio na:

- *objekte*

u užem smislu riječi, te

- *veze*,
- *vezne objekte*.

4.1.2 Objekti

Status *objekta* u E-R modelu imaju oni entiteti koji pored identifikatora objekta (*primarnog ključa*) imaju još i neka svojstva koja se opisuju *atributima*.

Atribut je svojstvo objekta koje se opisuje jednim podatkom. Ukoliko je za opisivanje atributa potrebno više podataka, onda taj atribut predstavlja novi objekat. Na primjer:

Neka je objekat PRODAVNICA *opisan atributima:*

PRODAVNICA < *šifraprod#, naziv, djelatnost , grad* >

Objekat PRODAVNICA*, definisan na ovaj način, je entitet koji ima svoj identifikator - ključ (šifraprod#) i tri atributa (koji opisuju njegova svojstva koja su od značaja za poslovanje).*

U sljedećem koraku treba utvrditi da li svi atributi zadovoljavaju postavljeni kriterijum. Ako pretpostavimo da je:

- *šifra prodavnice jednoznačna (što mora biti),*
- *da svaka prodavnica ima samo jedno ime, te*
- *da ima definisanu djelatnost,*

onda su i atributi "naziv" i "djelatnost" atributi. Međutim, ako za atribut "grad" pored imena postoji i podatak o broju stanovnika u tom gradu

(što je sa gledišta profitabilnog rada nekog lanca trgovina i te kako važan podatak), onda "grad" ne može biti atribut jer ga opisuju dva podatka (naziv i broj stanovnika). Grad predstavlja stoga objekat koji mora biti definisan, na primjer kao:

GRAD < *šifragrada#, naziv, brojstan.* >

a u datoteci PRODAVNICA *atribut "grad" se briše. Na kraju,treba još ustanoviti i definisati vezu između tih objekata (*PRODAVNICA, GRAD*) – o čemu će još biti riječi.*

Generalno, objekti u E-R modelu mogu se podijeliti na:

- *čvrste objekte,* **-** *objekti u punom smislu te riječi, i*
- *slabe objekte koji na neki način (egzistencijalno ili identifikaciono) zavise od jednog ili više čvrstih objekata.*

Pod *čvrstim objektom* smatra se onaj koji se može potpuno definisati primarnim ključem i nizom atributa. To su, na primjer, objekti:

STUDENT < *brind#, prez, ime, datrodj, adresa, tel, ...* >

RAČUN < *brojračuna#, datum, iznos .....*>

HOTEL < *šifrahotela#, naziv, adresa, tel, ...*> .

Pod *slabim, egzistencijalno i/ili identifikaciono zavisnim objektom***,** smatra se onaj koji egzistencijalno i/ili identifikaciono zavisi od nekog drugog, čvrstog, objekta. Na primjer, objekat STAVKA nekog računa:

STAVKA < *brojstavke#, kolicina, cijena, ...*>

može biti prisutan samo onda ako pripada nekom računu. RAČUN je čvrst objekat, a STAVKA je slabi objekat koji može postojati samo ako postoji odgovarajući RAČUN (egzistencijalna zavisnost).

Na ovaj način definisan objekat STAVKA nije samo *egzistencijalno*, nego i *identifikaciono* zavisan od čvrstog objekta RAČUN**,** jer stavke iz raznih računa mogu imati isti broj (istu šifru), pa ih prema tome ne bi bilo moguće jednoznačno identifikovati i međusobno razlikovati bez navođenja i broja računa kojem ta stavka pripada. Korektno se objekat STAVKA stoga mora definisati uz RAČUN kao:

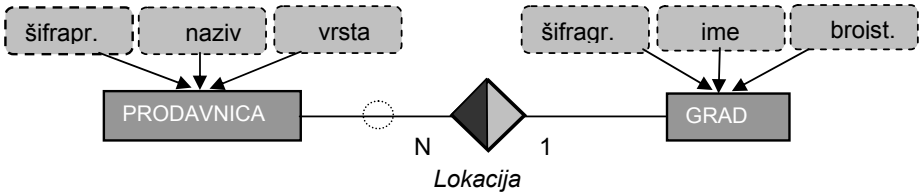
RAČUN < brojračuna#, iznos,.....>

STAVKA < *brojstavke#, brojračuna#, količina, cijena, ...* >

Zaključak: *Slabi objekti* koji identifikacijski zavise od čvrstih objekata moraju imati *složen primarni ključ* koji se sastoji od *ključa slabog objekta*, i *primarnih ključeva jakih* objekata od kojih zavise.

4.1.3 Veze

Očigledno je da u malopređašnjem primjeru datoteke PRODAVNICA i GRAD nisu nezavisne jer se svaka prodavnica odnosi na grad u kojem se ona nalazi. Neophodno je, prema tome, ova dva objekta međusobno povezati. Nazovimo tu vezu među njima u ovom konkretnom slučaju *"lokacija"* (vidi sliku 4.1).



Slika 4.1 E-R model sistema prodavnica

Vezu između GRADA *i* PRODAVNICE *čini "lokacija", nalazi se između* GRADA *i* PRODAVNICE a *tip uspostavljanja veze (moguće varijante, vidjeli smo, su 1:1, 1:N ili M:N) mora biti poznat. U konkretnom slučaju, ako u gradu može biti više prodavnica, a određena prodavnica može biti samo u jednom gradu, tip veze je 1:N.*

Ako je veza "lokacija" još i opcionalna, dakle neobavezna, (svaki grad može, ali ne mora, imati prodavnicu posmatranog lanca trgovina), onda tu činjenicu označavamo na strani veze koja je neobavezna kružićem (slika 4.1).

Treba ponovo naglasiti da su veze najčešće rezultat zakonskih propisa, dogovora, ugovora, statuta, raznih internih normi, itd. pa ih u toj oblasti treba i "tražiti" a rjeđe posljedica prirodnih zakona.

Vezu, da bi je u ER modelu razlikovali od *objekta,* predstavljaćemo grafički rombom.

Veze tipa 1:1 i 1:N, ne mogu imati i vlastite atribute kojima se bliže opisuju.

4.1.4 Vezni objekti

Da bi se veze tipa N:M u E-R modelu eliminisale, prevode se, pod određenim uslovima u novi objekat, *daju im se svojstva objekta, čime takve veze postaju vezni objekti* koji onda mogu imati i sva svojstva objekta, imaju prema tome i identifikator (ključ), a mogu, ali i ne moraju, imati i atribute.

Definisanje tipa veze je stoga osnovni preduslov da se E-R modelom realno prikaže stanje stvari.

Kada se pri sintezi E-R modela pokaže da su veze tipa N:M neminovne (jer egzistiraju u realnom svijetu), "eliminacija" ovih veza iz modela izvodi se na sljedeći način:

- *svaka veza tipa N : M zamjenjuje se novim objektom (najčešće istog naziva kao i veza N : M koja se zamjenjuje) sa složenim ključem – identifikatorom - koji se sastoji od ključeva objekata koje je veza tipa N:M povezivala.*
- *nakon toga se novi objekat povezuje vezama 1 : N sa postojećim.*

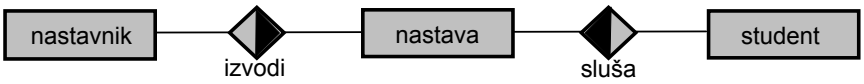
Na primjer, objekti NASTAVNIK i STUDENT vezani su N:M jer:

- jedan nastavnik komunicira sa više (N) studenata, a svaki student komunicira sa više (M) nastavnika

ER model (slika 4.2) sadrži tu vezu (nazovimo je *nastava*) koju treba eliminisati, tačnije zamijeniti vezama tipa 1:N.

Slika 4.2 ER-model veze N:M

Zamjena veze *nastava*, veznim objektom NASTAVA, izvodi se uvođenjem dvije nove veze tipa 1:N između NASTAVNIKA i NASTAVE (nazovimo je "*izvodi"*) i STUDENT-a i NASTAVE (nazovimo je "*sluša*"). Tako dobijamo nov oblik ER-modela (bez N:M veze) – slika 4.3.



Slika 4.3 Transformisani ER-model veze N:M

Novouvedeni objekat (NASTAVA) mora imati složen ključ koji se sastoji od ključeva objekata koje povezuje. Pored toga, vezni objekat može, ali ne mora, imati i još neke atribute koji ga opisuju pobliže. Na primjer, objekat NASTAVA može imati atribut koji daje termin kada se nastava (nastavnika sa šifrom šifran# i studenta sa šifrom broj_indeksa#) izvodi. Prema tome relacioni model može imati sljedeći oblik:

NASTAVNIK < šifran#, ime, prezime, adresa, telefon.....>
STUDENT < broj_indeksa#, ime, prezime, adresa, telefon.....>
NASTAVA <šifran#, broj_indeksa#, termin_predavanja,.....>

Objektu NASTAVA može se dodijeliti (po želji) i vlastiti ključ, ali dva spoljna ključa moraju i tada ostati kao atributi. Na primjer:

NASTAVA <*šifranastave#, šifran, broj_indeksa, termin_predavanja,..*>

4.1.5 Osobine veza

Pored *tipa veze*, svaku vezu karakterišu još dvije osobine i to:

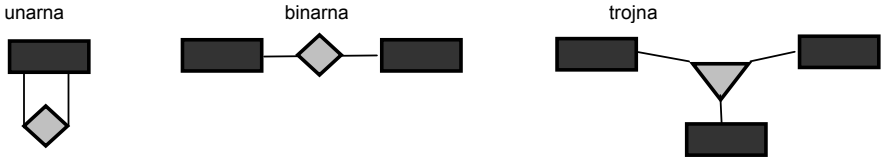
a. Red veze,
b. Kardinalnost veze,
c. Način uspostavljanja veze.

a. Red veze

Red, odnosno stepen veze određuje broj objekata koji čine neku vezu. U praksi se najčešće javljaju:

- *unarne,*
- *binarne,*
- *trojne* veze

(slika 4.4), ali se mogu javiti i veze reda višeg od tri (n > 3) koje se zbog nepreglednosti E-R modela u praksi izbjegavaju uvijek kada je to moguće svođenjem na veze nižeg reda.



Slika 4.4 Unarna, binarna i trojna veza

a1. Unarne (ili unutrašnje) veze su relativno rijetke u praksi, a uspostavljaju se unutar jedne tabele, jednog objekta. Na primjer:

- *Neka osoba u opštinskoj datoteci* GRAĐANIN *(koja sadrži uobičajene podatke kao sto su ime, prezime, godina rođenja, mjesto rođenja, školska sprema, bračno stanje itd.) može (veza je opcionalna – nije obavezna) da se nalazi u bračnoj vezi sa drugom osobom u istoj datoteci,*
- *Neki takmičar, u disciplini takmičenja parova (na primjer u umjetničkom klizanju, ili tenisu) u datoteci* TAKMIČAR*, nalazi se u unarnoj vezi (ova veza je obavezna jer u parovima ne može da nastupi jedan takmičar sam) sa partnerom iz iste datoteke.*

a2. Binarna veza je veza između dva objekta i najčešće se susreće u praksi. Takva veza postoji, na primjer, između objekata*:*

- GRAD i PRODAVNICA*,*
- NASTAVNIK i *PREDMET,*
- VOZAC i VOZILO itd*.*

a3. Trojna veza nastaje proširenjem binarne veze. Na primjer, binarna veza NASTAVNIK i PREDMET mora se nekada proširiti i literaturom koju nastavnik koristi na svom predmetu. U tom slučaju potrebno je uvesti i treći objekat LITERATURA**.**

Novonastala trojna veza (nazovimo je DISCIPLINA) iskazana riječima glasi: *Nastavnu* DISCIPLINU *čine:*

- PREDMET koji se predaje
- LITERATURA po kojoj predaje*, i*
- NASTAVNIK koji je predaje*,*

Veze reda većeg od 3 iskazuju na sličan način odnos između više od tri entiteta. Analiza im je identična trojnim vezama, ali zbog nepreglednosti modela u praksi se izbjegavaju.

b. Način učestvovanja u vezi

Način učestvovanja (prisustva) objekata u vezi može bitidvojak i to:

- *obavezan, i*
- *neobavezan (opcionalan).*

b1. Obavezno prisustvo smatramo da postoji onda kada *jedan objekat mora biti povezan sa drugim objektom***.** Na primjer, GRAD se mora nalaziti na nekoj LOKACIJI.

b2. Opcionalna, neobavezna, veza postoji onda kada prethodni uslov nije zadovoljen. Na primjer, svaki RADNIK može biti ZADUŽEN nekim POSLOM, ali mogu postojati i takvi poslovi za koje trenutno nije ZADUŽEN nijedan RADNIK. Veza ZADUŽEN između objekata RADNIK i POSAO prema tome bila bi opcionalnog karaktera.

Posmatrajmo sada detaljnije ponašanje i karakteristike objekata u binarnim i trojnim vezama.

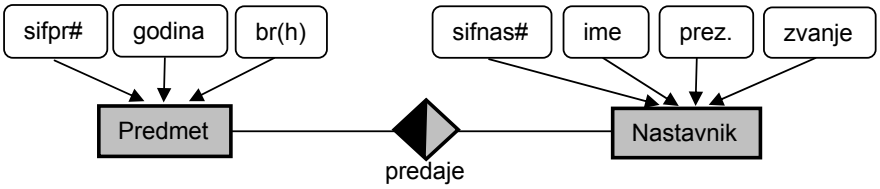
Uzmimo za primjer bazu podataka obrazovne ustanove u kojoj treba povezati objekte NASTAVNIK i PREDMET a pod uslovom koji je definisan pravilnikom škole i postojećim statutom koji glasi:

- *jedan nastavnik može da predaje više predmeta (NN može da predaje matematiku i fiziku) ali, jedan, određeni predmet (PP) u nekom određenom odjeljenju može da predaje samo jedan, određeni, nastavnik (na primjer fiziku predaje u odjeljenju V_v samo NN iako škola ima više profesora fizike). Tip veze je 1 : N.*

Grafički se E-R model, može prikazati na razne načine. Chen je u njegovom radu koristio simbole kako je to pokazano na slici 4.5.

Dopunimo sada ovaj model i spiskom literature koju na pojedinim predmetima koriste nastavnici, uz uslov da:

jedan udžbenik može da se koristi za više predmeta, ali na jednom predmetu može, fakultativno, da se koristi više različitih udžbenika.



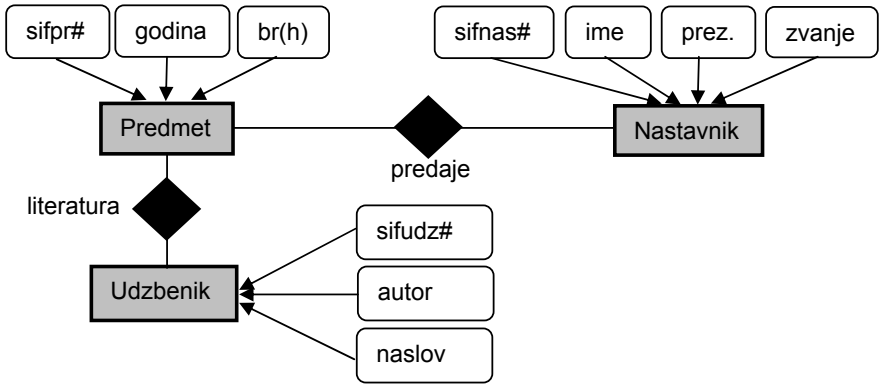
Slika 4.5 E-R model dijela informacionog sistema obrazovne ustanove

Postojećim datotekama NASTAVNIK i PREDMET, te vezi "*predaje*" moramo pridodati novu datoteku UDŽBENIK, kao i novu vezu koja će je povezati sa relacijom PREDMET**.**

Nazovimo tu novu vezu "*literatura*" i definišimo nova "pravila poslovanja". Neka su to:

1. *jedan predmet može predavati više nastavnika,*
2. *jedan nastavnik može predavati više predmeta,*
3. *za jedan predmet može se koristiti više udžbenika,*
4. *jedan udžbenik može se koristiti za više predmeta.*

Veze *literatura* i *predaje* su prema tome obje tipa N:M, a grafička interpretacija E-R modela sa takve dvije binarne veze, vidi se na slici 4.6.



Slika 4.6 Proširen model informacionog sistema obrazovne ustanove

Dvjema binarnim vezama N:M nije, nažalost, eksplicitno definisan trojni odnos između PREDMETA, NASTAVNIKA i UDŽBENIKA, jer iz premisa:

- *jedan predmet izvodi više nastavnika* , i

- *za jedan predmet koristi se više udžbenika*,

ne slijedi obavezno i zaključak da:

svaki nastavnik koristi za svoj predmet sve udžbenike

li negacija te iste tvrdnje, jer se polazne pretpostavke ne baziraju na jednoj trojnoj, nego na dvije binarne veze. Tako odgovor na pitanje:

- *Koje udžbenike za svoj predmet koristi neki nastavnik*?

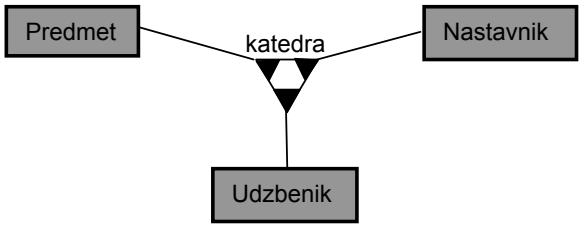
ovako koncipiran model ne može dati.

Da bi i ovo postalo moguće binarne veze moraju se predefinisati u trojnu vezu kao što je to predstavljeno na modelu trojne veze KATEDRA (slika 4.7).

Riječima iskazana "Pravila poslovanja" u trojnoj vezi KATEDRA glase:

1. *jedan predmet prema jednom udžbeniku izvodi više nastavnika,*
2. *jedan nastavnik za jedan predmet koristi više udžbenika,*
3. *jedan udžbenik, jedan nastavnik, koristi za više predmeta.*

Trojna veza (sve tri veze su tipa 1:N) KATEDRA zamjenjuje dvije binarne veze (tipa M:N) i omogućava da se izrazi svaki odnos između objekata i na taj način dobije i odgovor na svako postavljeno pitanje.



Slika 4.7 Trojna veza KATEDRA među objektima

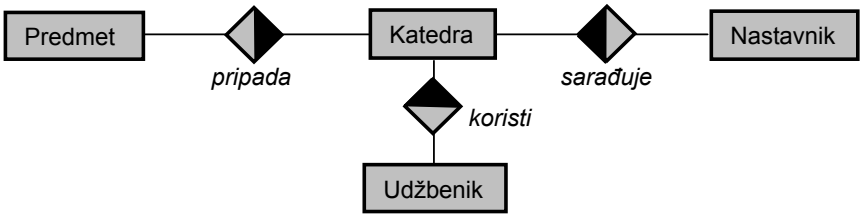
Ima autora koji se bave ovom problematikom koji smatraju da trojne, i veze višeg reda, “zamagljuju” prirodu odnosa među objektima i time smanjuju preglednost modela. Zbog toga se višestruka veza često ponovo zamjenjuje sa binarnim vezama. Ovaj postupak zamjene može se izvesti na sljedeći način: Prvo se:

- *Višestruka veza (u posljednjem primjeru trojne veze to je veza KATEDRA – slika 4.7) predstavlja kao novi objekat kome se pridružuje njegov identifikator – ključ.*
- *Nakon toga se svaki objekat višestruke (trojne) veze binarno povezuje vezom tipa 1:N sa tim novim objektom.*

E-R model trojne veze KATEDRA, preveden na E-R model sa tri binarne veze, vidi se na slici 4.8.

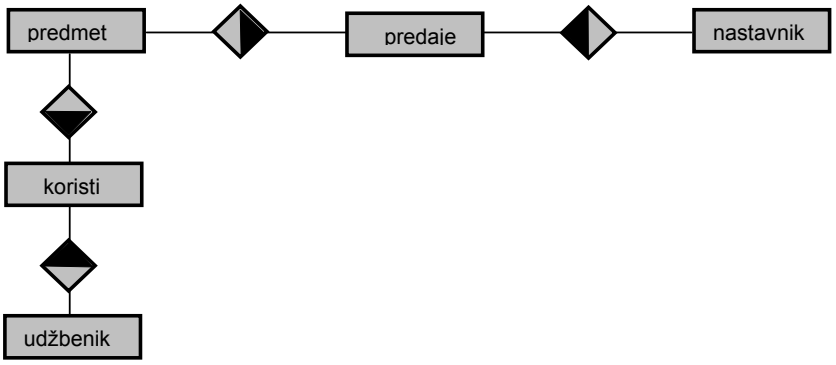
Vezni objekat KATEDRA preveden je u objekat KATEDRA**,** a uvedene su tri nove veze i to:

- *"pripada" - povezuje PREDMET i KATEDRA tipom veze 1:N,*
- *"sarađuje" - povezuje NASTAVNIK i KATEDRA, veza 1:N,*
- *"koristi" - povezuje UDŽBENIK i KATEDRA tip veze 1:N.*



Slika 4.8 Model sa tri binarne veze

Eliminacija veza tipa N:M izvodi se na isti način i u složenijim strukturama. Tako, na primer, proširen model informacionog sistema obrazovne ustanove sa slike 4.4, sa eliminisanim vezama tipa N : M, vidi se na slici 4.9.



Slika 4.9 Transformisani ER-model sa slike 4.4 sa dva nova vezna objekta,

Zaključak: Prilikom sinteze E-R modela sistema u prvoj iteraciji treba uvijek uvesti vezu onoga reda za koju projektant sistema utvrdi da realno postoji među objektima. Po završetku izrade modela, veze reda višeg od dva, i veze tipa N:M zamjenjuju se onda binarnim vezama na način kako je to pokazano. Posmatrajmo, na kraju, još jedan primjer veze jakog i slabog objekta.

Pretpostavimo da informacioni sistem nekog hotela sadrži sistem rezervacija u hotelskom lancu sa objektima:

HOTEL, TIP_SOBE i REZERVACIJA,

a pravila poslovanja dozvoljavaju da:

- *svaki* HOTEL *može imati sve tipove soba,*
- *svaki* TIP_SOBE *može da se nalazi u svakom hotelu,*

- *za svaki* HOTEL *može se napraviti više* REZERVACIJA *istovremeno, a* REZERVACIJE *mogu da bude izvršene u raznim* HOTEL-ima, *i*
- *jednom* REZERVACIJOM *rezerviše se više tipova soba, a svaki* TIP_SOBE *može biti predmet više* REZERVACIJA*.*

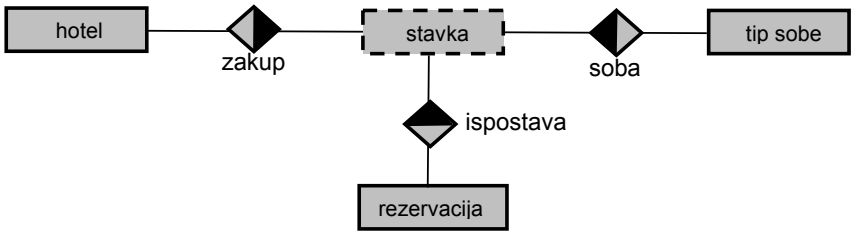
U pitanju je očito trojna veza, a sva tri tipa veze su M:N**.** Uvedimo zato, odmah u startu, još jednu datoteku STAVKA ("preskačući" postupnu fazu eliminacije trojne veze), a za koju će važiti da:

- STAVKA *pripada jednoj* REZERVACIJI*, dok jedna* REZERVACIJA *može imati više* STAVKI*,*
- STAVKA *rezerviše jedan* TIP_SOBE, *a jedan* TIP_SOBE *može biti predmet* REZERVACIJE *u više* STAVKI*, i*
- STAVKOM *se rezervišu sobe u* HOTEL-u*, dok sobe* HOTEL-a *mogu biti predmet rezervacije u više* STAVKI*.*

Sve tri ovako definisane veze, nazovimo ih:

- *zakup,*
- *soba,*
- *ispostava*

su tipa 1:N*, a datoteka* STAVKA *je uz to egzistencijalno i identifikaciono zavisna od datoteke* REZERVACIJA*. Datoteka* STAVKA *je slab objekat jer zavisi od* REZERVACIJE*. E-R model informacionog sistema ima ukupno četiri objekta (jedan od njih* STAVKA *je slabi), i tri veze "zakup", "soba" i "ispostava" – slika 4.10.*



Slika 4.10 Primjer veze slabog i jakih objekata

E-R model je prvi korak u projektovanju informacionog sistema. Njegova izrada se uz to odvija, kao što smo to vidjeli na prethodnim primjerima, uvijek po etapama, pri čemu se počinje sa definicijom objekata i njihovih svojstava, a zatim definicijom i svojstvima veza. Proces je iterativan, što znači da često saznanja do kojih dođemo u toku sinteze modela zahtijevaju da se početna koncepcija modela promijeni. Tek nakon konačne verzije modela, kao posljednji korak slijedi i prevođenje E-R modela na relacioni oblik.

4.2 Prevođenje E-R modela na relacioni oblik

Tehnika prevođenja E-R modela na relacioni oblik izvodi se tako što:

- *svaki objekat E-R modela postaje relacija,*
- *svaka veza N:M postaje objekat - vezna relacija,*
- *ime objekta postaje ime relacije,*
- *karakteristike objekta postaju njegovi atributi,*
- *identifikatori objekata postaju ključevi relacija.*

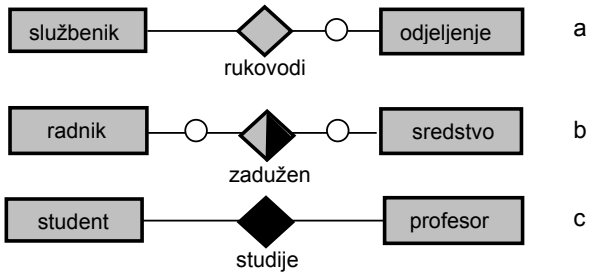
Binarne veze su zastupljene u većini informacionih sistema, a tehnika redukcije veza M : N, koja se primjenjuje u binarnim vezama, principijelno se ne razlikuje od tehnike transformacije tih veza koja se koristi u unarnim, trojnim ili vezama višeg reda. Posmatrajmo stoga prvo prevođenje binarnih veza na relacioni oblik.

4.2.1 Prevođenje binarnih veza na relacioni oblik

Pri prevođenju binarnih veza na relacioni oblik mogu nastupiti tri karakteristična slučaja – slika 4.11.

a. *Objekat* SLUŽBENIK *vezan je vezom tipa 1:1 "rukovodi" sa objektom* ODJELJENJE*, i to neobavezno, jer svaki službenik mora da pripada jednom* ODJELJENJU*, ali nekom* ODJELJENJU *može, ali ne mora, da pripada svaki službenik. Opcionalnost je prema tome na strani tabele* ODJELJENJE – slika 4.11a*.*

b. Tabela RADNIK *vezana je veznim objektom* ZADUŽEN *sa tabelom* SREDSTVO *(pri tome se misli na sredstvo za rad, na alat, itd.) vezom tipa 1:N, i to obostrano neobavezno jer, radnik može, ali ne mora, biti zadužen sa više sredstava za rad, i svaka alatka, svako sredstvo za rad može, ali ne mora, biti kod jednog radnika – slika 4.11b.*



Slika 4.11 Primjeri binarnih veza

c. Konačno tabela STUDENT *povezana je vezom "studije" sa tabelom* PROFESOR *u bazi podataka nekog fakulteta na način M:N i to*

obostrano obavezno, jer svaki profesor predaje većem broju studenata, a svaki student sluša predavanja kod više profesora – slika 4.11c.

Binarne veze tipa 1:1 prevode se na relacioni jezik tako što se u jednu od tabela koje učestvuju u vezi (teoretski svejedno koju), uvrsti primarni ključ druge tabele kao atribut.

Veza se prema tome iskazuje spoljnim ključem.

U primjeru (slika 4.11a) model veze 1:1 rezultira dvjema relacijama:

ODJELJENJE < *šifraodjelj#, šifrasluz#, ... >*
SLUŽBENIK < *šifrasluz#, ...>*

Prilikom izbora tabele kojoj treba da dodamo spoljni ključ koji ostvaruje vezu među tabelama treba obratiti pažnju na to da tabela kojoj dodajemo spoljni ključ ima što manje *Null* vrijednosti. U Prethodnom primjeru vezu bi mogli da izvedemo i ovako:

ODJELJENJE < *šifraodjelj#,.....>*
SLUŽBENIK < *šifrasluz#, šifraodjelj#, ...>.*

Međutim, u tom slučaju, pošto je rukovođenje opcionalno, dakle neobavezno, n-torka nekog službenika koji nema rukovodeću funkciju imala bi *Null* vrijednost**,** što u prvom slučaju nije bio slučaj budući da svako odjeljenje obavezno ima i svoga rukovodioca.

Binarne veze 1:N prevode se na relacionu formu slično kao i veze tipa 1:1. Veza se ponovo iskazuje spoljnim, stranim, ključem ali ne u bilo kojoj od relacija, nego u onoj *koja u vezi E-R modela predstavlja objekat tipa povezanosti mnogo (N).*

Drugačije ne može ni biti, jer bi spoljni ključ u relaciji koja predstavlja objekat tipa jedan (1), u svakoj n-torci drugog objekta (mnogo) morao poprimiti onoliko vrijednosti sa koliko se slogova (n-torki) objekta tipa mnogo nalazi u vezi - a to naravno nije moguće. Konačno, treba obratiti pažnju i na opcionalnost veze i mogućnost da spoljni ključ dobije *Null* vrijednost[1], što se ne smije dopustiti. U konkretnom slučaju (slika 4.11b) relacioni model imao bi sljedeću formu:

RADNIK < šifrarad#,..... >
SREDSTVO < šifrasred#, šifrarad#,...>.

Binarne veze tipa M:N prevode se na relacionu formu uvođenjem nove relacije (vezna relacija). Ključ te nove relacije je po pravilu složen i sastoji se od primarnih ključeva objekata koji učestvuju u vezi, a atributi su

[1] *Za vezne atribute, tj. za strani ključ, može se dopustiti pojava NULL vrijednosti na strani više ako je veza opcionalna.*

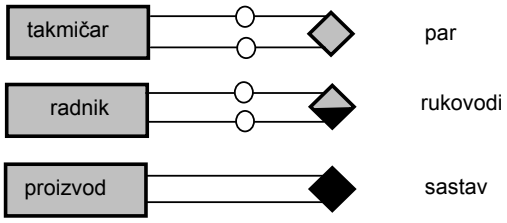
svojstva veze. U malopređašnjem primjeru (slika 4.11c) rješenje bi moglo imati oblik:

STUDENT < *brojindexa#, ime, prezime, adresa,* .....>
PROFESOR < *matbroj#, ime, prezime, adresa*,.....>
STUDIJE <*brojindexa#, matbroj#, fakultet*, ...>.

Nova, vezna, relacija je STUDIJE, koja pored složenog ključa ima i svoj atribut "*fakultet*" koji pobliže definiše STUDIJE, a broj atributa veznog objekta u principu može biti i veći ako se za tim ukaže potreba.

4.2.2 Prevođenje unarnih veza na relacioni oblik

I unarne veze prevode se na relacioni oblik različito za slučaj tipa veze 1:1, 1:N, odnosno N:M.



Slika 4.11 Tipovi unarnih veza

a. Unarne veze tipa 1:1 postoje među n-torkama unutar jedne tabele a prevode se na relacioni oblik uvođenjem šifre jedne n-torke (koja učestvuje u vezi) kao spoljnjeg ključa u drugu koja je u vezi sa njom. Pošto bi ta šifra bila identična prethodnoj (jer su obje u istoj tabeli), ova druga se mora preimenovati. Na primjer, ako tabela TAKMIČAR ima oblik:

TAKMIČAR < *šifratak#, ime, prezime, ...*>

veza PAR *u datoteci TAKMIČAR (takmičar nastupa u kategoriji parova, jer tek sa partnerom svaki od takmičara u toj kategoriji postaje takmičar u punom smislu te riječi), iskazuje se uvođenjem šifre partnera, pa tabela TAKMIČAR nakon uvođenja veze ima sljedeći oblik:*

TAKMIČAR< *šifratak#, ime, prezime, šifrapart,...>.*

Napomena: U sistem kojim se obezbjeđuje integritet podataka mora se ugraditi mehanizam koji će spriječiti da se ne dogodi da takmičar u kategoriji parova ima partnera, a njegov partner nema – a što je moguće da se pri projektovanju sistema greškom desi.

b. Unarne veze tipa (1:N) prevode se na relacioni oblik identično kao i u slučaju 1:1 uvođenjem spoljnjeg, preimenovanog ključa. Tabela RADNIK nakon prevođenja u relacionu formu ima slijedeći oblik:

RADNIK < *šifrad#, šifraruk, ime, prez, adresa, ...*>

Ovakva veza predstavlja u stvari *hijerarhijsku strukturu unutar objekta* a iskazuje se spoljnim ključem. Spoljni ključ ukazuje na nadređeni objekat u pomenutoj hijerarhiji. U konkretnom slučaju je RADNIK je "*podređen*" RUKOVODIOCU.

c. Unarna veza tipa N:M prevodi se na relacionu formu uvođenjem nove vezne tabele, vezne relacije, čiji ključ je složen - sastavljen od dva atributa. I u ovom slučaju mora se jedan od atributa u veznoj relaciji (SASTAV) preimenovati, jer ne mogu u jednoj relaciji postojati dva atributa sa istim imenom. U primjeru PROIZVOD - SASTAV (slika 4.11c) rješenje bi moglo da izgleda ovako:

PROIZVOD < *šifrapriz#, naziv, datum,.....>*
SASTAV < *šifraproiz#, šifra_sastava_proiz#, naziv, ...>*

Veze (M:N) unutar jednog objekta, zbog odnosa "mnogo prema mnogo", ne *iskazuje hijerarhijsku nego mrežnu strukturu*. Zato treba obratiti pažnju da se ne desi slučaj da, na primjer, neki od proizvoda postane sastavni dio samoga sebe, to jest da dođe do pojave "zatvorenog ciklusa", što može imati neugodne posljedice (povratna sprega koja može uticati na nestabilnost rada!) u eksploataciji.

4.2.3 Prevođenje veza reda većeg od dva

N-arne veze (n>2) mogu tvoriti slijedeće tipove (slika 4.12.):

- *povezanost svih objekata je tipa jedan (A),*
- *povezanost jednog objekta je tipa mnogo a preostalih tipa jedan (B),*
- *povezanost dvaju objekata je tipa mnogo, a preostalih tipa jedan (C),*
- *povezanost svih objekata je tipa mnogo (D).*

N-tarne veze prevode se na relacioni oblik uvođenjem dodatnih veznih relacija, koje uključuju identifikatore objekata koji tvore vezu.

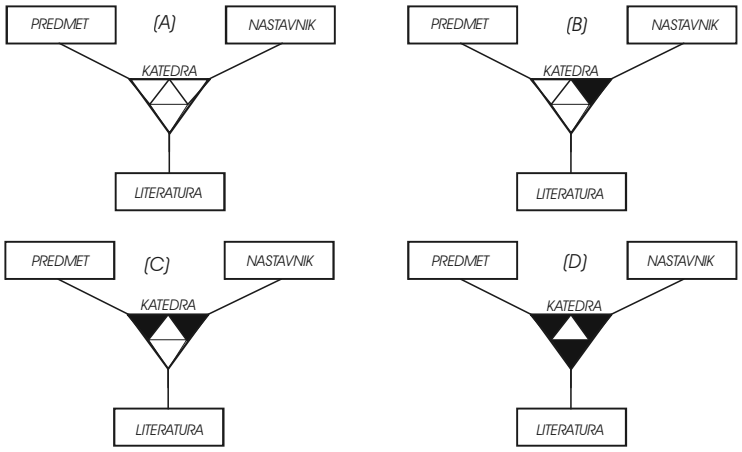
Pokažimo to na primjeru trojne veze, a veze višeg reda prevode se analogno postupku koji se primjenjuje kod trojne.

U prvom slučaju (slika 4.12) dodatni uslovi koji definišu poslovanje sistema mogu, na primjer, biti:

- *jedan nastavnik, za jedan predmet, koristi jednu knjigu,*
- *jedan predmet, po jednoj knjizi, predaje jedan nastavnik, i*
- *jedan udžbenik, koristi jedan nastavnik, za jedan predmet*,

pa relacioni model može da ima slijedeću formu:

PREDMET < *šifrapred#, naziv,.....>*
NASTAVNIK < *šifranast#, ime, prezime, adresa,.....>*
LITERATURA < *šifralit#, naziv, izdavac*,.....> .



Slika 4.12 Tipovi n-tarnih veza

Vezna relacija KATEDRA, ima tri kandidata za primarni ključ tako da može da poprimi jedan od tri slijedeća ravnopravna oblika:

KATEDRA < *šifrapred#, šifranast, šifralit*,.....>
KATEDRA < *šifrapred, šifranast#, šifralit*,.....>
KATEDRA < *šifrapred, šifranast, šifralit#*,.....>.

U drugom slučaju dodatni uslovi na sistem mogu biti:

- *jedan nastavnik za jedan predmet koristi jednu knjigu,*
- *jedan predmet, uz jednu knjigu, predaje više nastavnika, i*
- *jedan udžbenik, koristi jedan nastavnik za jedan predmet.*

Šema relacionog modela ostaje ista kao i u prethodnom slučaju, vezna relacija KATEDRA ima sada dva kandidata za primarni ključ, a s obzirom da jedan predmet, uz upotrebu jedne knjige, predaje više nastavnika, mogu se javiti dvije varijante:

KATEDRA < *šifrapred#, šifranast#, šifralit*,.....> ,
KATEDRA < *šifrapred, šifranast#, šifralit#*,.....> .

U trećem slučaju dodatni uslovi na sistem mogu da glase:

- *jedan nastavnik za jedan predmet koristi jednu knjigu,*
- *jedan predmet, uz jednu knjigu, predaje više nastavnika, i*
- *jedan udžbenik, jedan nastavnik, koristi za više predmeta.*

Šema relacionog modela je ista samo vezna relacija KATEDRA ima sada, samo jednog kandidata za primarni ključ:

KATEDRA < *šifrapred#, šifranast#, šifralit,.....*>

Konačno, u četvrtom slučaju dodatni uslovi glase:

- *jedan nastavnik za jedan predmet koristi više knjiga,*
- *jedan predmet, uz jednu knjigu, predaje više nastavnika, i*
- *jedan udžbenik, jedan nastavnik, koristi za više predmeta.*

U model, pored relacija PREDMET, NASTAVNIK i LITERATURA, uvodi se i relacija KATEDRA koja ima samo jednog kandidata za primarni ključ a koji se sastoji od tri ključna atributa relacija koje povezuje:

KATEDRA < *šifrapred#, šifranast#, šifralit#,.....*> .

4.3 Normalizacija

Projektovanju informacionog sistema može se prići i na drugi način i to na prvi pogled jednostavnije, prosto skupljanjem i registrovanjem svih raspoloživih atributa. U tom slučaju mora se posebno obratiti pažnja na to da logički model ne sadrži redundancu podataka.

Tabele sa sirovo "nabacanim" atributima rijetko kada zadovoljavaju ovaj uslov, jer se se pri njihovom koncipiranju nije vodilo računa o izboru objekata - tabela i njihovih atributa.

Optimalan izbor relacija i njihovih atributa naziva se *normalizacija sistema*.

Normalizacija je u stvari*:*

postupak izmjene (najčešće dekompozicije, rastavljanja na više relacija) prvo-koncipiranih tabela (relacija).

Dekompozicijom relacije dobija se uvijek dvije ili više novih, međusobno povezanih relacija, koje se nalaze i u nekoj od normalnih formi.

U slučaju kada se projektovanju sistema pristupa preko E-R modela, i ako se isti korektno izvede, onda će i relacije koje iz njega slijede već biti u nekoj višoj normalnoj formi pa mnogi tako i provjeravaju kvalitet dobijenog E-R modela.

Tehnički postupak izvođenja normalizacije svodi se na operacije relacione algebre - projekciju i spajanje, i to:

- *na dekompoziciju tabela generisanjem vertikalnih podskupova (operacija projekcije), i*
- *na generisanje novih relacija iz dvaju ili više dobijenih vertikalnih podskupova (operacija spajanja).*

Ovakav pristup sintezi informacionog sistema naziva se *vertikalna normalizacija* u kojoj postoji *pet normalnih formi***.** Za praksu su najvažnije *druga i treća***,** i na njima se često proces normalizacije i zaustavlja.

Pomenimo da pored *vertikalne* postoji i *horizontalna normalizacija***,** koja nije teoretski potpuno dorađena do kraja, i vezana je prvenstveno za distribuirane baze podataka.

Šta je to normalizovana, a šta nenormalizovana forma neke relacije (tabele), najbolje se može vidjeti na nekoliko primjera:

Pretpostavimo da neka trgovačka firma želi da vodi evidenciju o svome poslovanju. U tu svrhu skuplja i obrađuje podatke koji se nalaze u tabeli NARUDŽBA*. To su:*

- *šifra kupca*
- *ime i prezime kupca,*
- *adresa kupca,*
- *količina kupljene robe,*
- *šifra robe (artikla)*
- *naziv robe,*
- *kvalitet, i*
- *cijena robe.*

NARUDŽBA

šifkup#	*ime*	*adresa*	*količina*	*šifart#*	*naziv*	*kvalitet*	*cijena*
k1	*Nikola*	*Beograd*	*100*	*a1*	*lak*	*II*	*220*
k1	*Nikola*	*Beograd*	*200*	*a2*	*boja*	*I*	*130*
k1	*Nikola*	*Beograd*	*50*	*a3*	*gips*	*III*	*20*
k1	*Nikola*	*Beograd*	*300*	*a4*	*četke*	*II*	*70*
.....	.....	.....	.....	.....	.....	.....	.....
k17	*Mitar*	*Beograd*	*200*	*a2*	*boja*	*I*	*130*

Primjer sa nekoliko podataka ove tabele vide se u tabeli NARUDŽBA*.*

Bez dubljeg upuštanja u analizu problema model poslovanja se može predstaviti u vidu samo jednog objekta – jedne tabele NARUDŽBA *- sa nizom atributa:*

NARUDŽBA <*šk#, imekp., prezkp., kolrb, šr#, nazivrb., kv., cijena*>

Ovako koncipiran model je loš iz slijedećih razloga:

- *šifra kupca (šk#) i šifra robe (šr#) su atributi tabele* NARUDŽBA*, iako su u realnom svijetu identifikatori dvaju različitih objekata (*KUPAC, ROBA*). Tako, na primjer, kvalitet nije atribut kupca nego nekog drugog objekta koji bi mogao da se zove* ROBA*.*

- *Model sadrži redundancu jer se adresa i ime kupca javljaju onoliko puta koliko puta je neki kupac kupovao robu u toj trgovini. Isti zaključak vrijedi i za artikle. Na primjer:*
- *opis artikla sa šifrom a2 (kvalitet, naziv i cijena) javlja se dva puta, jer su dva različita kupca nabavljala taj isti artikl.*

Prisustvo redundance u bazi podataka dovodi do niza problema prvenstveno kod izmjene (ažuriranja), ali i kod korišćenja, podataka. Ti problemi se nazivaju *anomalije***.** To su prije svega:

a. Anomalije pri upisu podataka

Ova anomalija sastoji se u tome da upis podataka o kupcu nije moguć sve dok neki kupac nešto ne kupi, iako on kao potencijalni kupac postoji i marketinški je interesantan. Analogno vrijedi i za artikle. Na primjer:

> *kupac sa šifrom k2 nema još ništa naručeno, i njegovi podaci nisu uneseni u tabelu NARUDŽBA (iako je možda upravo taj kupac veoma interesantan kao potencijalni kupac), ili podatke o artiklu sa šifrom artikla **a5** ne možemo imati u bazi (iako taj artikl postoji) sve dok neko ne izvrši porudžbinu*.

b. Anomalije pri brisanju podataka

Brisanjem jedne narudžbe iz tabele može se desiti da se izgube svi podaci o kupcu ili artiklu - robi. U konkretnom primjeru:

> *brisanjem kupca sa šifrom k17 biće izgubljeni svi podaci o njemu s obzirom da je on izvršio narudžbu samo jedanput. Na sreću, slučajno neće biti izgubljeni i podaci o artiklu koji je on kupio, jer postoji još jedan kupac (sa šifrom k2) koji je kupio isti artikl.*

Ali zato, brisanjem nekoliko narudžbi i takva greška može da se desi.

c. Anomalije pri izmjeni podataka

Promjenom imena ili adrese kupca nastaje problem izmjene podataka na onoliko mjesta na koliko je kupac bio upisan. Isti zaključak odnosi se i na artikle prilikom promjene imena ili svojstva (na primjer cijene) nekog artikla.

4.3.1 Prva, druga i treća normalna forma

Svi pobrojani nedostatci mogu biti eliminisani ako se prvobitno koncipirane relacije - tabele normalizuju i dovedu u potrebnu normalnu formu. Pri tome treba samo obratiti pažnju da tokom postupka normalizacije (koji se, kako rekosmo, svodi na dekompoziciju i ponovo spajanje tabela) ne dođe i do gubitka informacija, o čemu će još biti riječi.

- *Prva normalna forma*

Relacija se nalazi u prvoj normalnoj formi (1NF) ako, i samo ako, je njen domen (podaci u tabeli) skup atomarnih vrijednosti. Budući da je ovo i uslov da bi neka tabela uopšte bila relacija, slijedi da se svaka relacija nalazi u prvoj normalnoj formi.

Pojava redundance koja je u prvoj normalnoj formi praktično skoro uvijek prisutna (kao što smo to vidjeli u posljednjem primjeru), te mogućnost svih pratećih grešaka, ukazuju na to da prva normalna forma nije ni izdale-

ka dovoljna za sintezu upotrebljive baze podataka. Neki autori ovu formu nazivaju *nenormalizovanom formom ili nultom normalnom formom***,** tako da se kod raznih autora može naći i različit ukupan broj normalnih formi.

- *Druga normalna forma*

Druga normalna forma nema nekog većeg praktičnog značaja, jer je većina pomenutih anomalija i dalje prisutna kod relacija koje su dovedene u drugu normalnu formu. Pomenimo je stoga samo kao prethodnika treće normalne forme koja za praksu ima najveći značaj.

Po definiciji, relacija se nalazi u drugoj normalnoj formi ako svaki atribut, koji nije ključni, zavisi potpuno (ne djelimično) od ključnog atributa.

Posmatrajmo opet prethodni primjer.

Relacija NARUDŽBA *nije u drugoj normalnoj formi (2NF) jer u toj relaciji je jedini mogući kandidat za ključ složeni ključ (šk#, šr#) a atribut kvalitet očigledno zavisi samo od dijela (šr#) a ne i cijeloga ključa. Prevedimo stoga relaciju NARUDŽBA u dvije relacije, od kojih prva R1 sadrži podatke o kupcu, a druga R2, o artiklu kojega je taj kupac kupio, i to na slijedeći način:*

R1 < šk#, ime, adresa >
R2 <šk#, šr#, naziv, kvalitet, cijena, količina >

Relacija R1 je u 2NF, ali R2 još uvijek nije jer ponovo atribut kvalitet zavisi samo od dijela ključa (šifart#) a ne od cijeloga ključa (šifkup#, šifart#). Relaciju R2 moramo stoga opet razbiti na dvije od kojih, prva R21 sadrži samo podatke o robi , a druga R22 o kupcu i robi koji je on kupio:

R21 < šr#, kvalitet, cijena >
R22 < šk#, šr#, količina >.

Sada su sve tri relacije R1, R21 i R22 u 2NF, jer R1 i R21 nemaju složeni ključ, a atribut količina u R22 zavisi potpuno, a ne djelimično, od složenog ključa te relacije.

- *Treća normalna forma*

Relacije R1, R21 i R22, u ovom primjeru, zadovoljavaju i uslov treće normalne forme (3NF), što nikako ne znači da je to i u svakom drugom slučaju tako, jer se može desiti da dekompozicijom 1NF postignemo samo 2NF, a ne i više od toga. Pokažimo to na drugom primjeru.

Pretpostavimo da unutar informacionog sistema nekog instituta imamo objekat - entitet LABORANT *sa atributima:*

LABORANT < *šiflab#, ime, prezime, broj_sobe, telefon* >

Relacija LABORANT *je već u 2NF jer ima samo jedan ključni atribut (šiflab# - šifra laboranta), i niz atributa. Međutim, između atributa broj_sobe i telefon može da postoji međusobna zavisnost (telefon se nalazi u sobi) pa će se stoga neki broj telefona u laboratoriji pojaviti u relaciji onoliko puta koliko laboranata radi u njoj, pod uslovom da koriste isti telefon.*

Prema tome, iako je relacija LABORANT *u 2NF, pojava redundance podataka u toj formi nije otklonjena, a time i sve anomalije pri održavanju kao na primjer:*

- *broj jednog od telefona i broj sobe neke laboratorije nije moguće unijeti u informacioni sistem sve dok se ne unesu podaci o bar jednom laborantu koji radi u njoj,*
- *brisanjem podatka o posljednjem, laborantu neke laboratorije, gubi se podatak i o broju telefona u njoj (iako laboratorija, prostorija i dalje postoji), te konačno,*
- *ako laboratorija promijeni broj telefona, izmjenu treba unijeti onoliko puta koliko laboranata radi u njoj.*

Navedene slabosti ukazuju da relacija LABORANT mora biti dalje dekomponovana i na taj način dovedena do 3NF. Dekompozicija se izvodi tako da se neključni atributi, koji su u međusobnoj funkcionalnoj vezi, izdvoje u zasebnu relaciju, te da se tako nastala relacija poveže sa ostatkom prvobitne preko jednog atributa. Konkretno, u posljednjem primjeru od relacije LABORANT treba napraviti dvije:

LABORATORIJA <*broj_sobe#, telefon* >
LABORANT < *šiflab#, ime, prezime, broj_sobe* >

U ovom primjeru funkcionalna zavisnost između atributa *šiflab#, broj_sobe i telefon* postoji i dalje. Broj telefona naime zavisi od broja sobe u kojoj se telefon nalazi, a broj sobe je u vezi sa laborantom koji sjedi u njoj. Ne bi bilo dobro da smo dekompozicijom tabele izgubili tu zavisnost, jer model sistema ne bi više bio realna slika stvarnosti. Relacija LABORANT nije, prema tome, prije dekompozicije bila u trećoj normalnoj formi, jer je postojala veza među atributima.

Svaka relacija koja se nalazi u trećoj normalnoj formi nalazi i u drugoj, dok obrnuto ne važi.

Relacija se nalazi u trećoj normalnoj formi ako, i samo ako, neključni atributi nisu tranzitivno[2] zavisni od bilo kog ključa te relacije.

4.3.2 Problem gubitka informacija

Pomenute međusobne zavisnosti atributa (funkcionalna i tranzitivna) nisu vještačke veze među atributima, nego veze koje postoje i u stvarnosti, i zato se, preslikavaju iz realnog svijeta u informacioni sistem. Dekompozicijom relacije ne smijemo takve veze potpuno raskinuti, jer će model u tom slučaju izgubiti svoju vjerodostojnost. To praktično znači da svaka zavisnost unutar neke relacije mora logički slijediti i iz relacija koje su generisane dekompozicijom, odnosno iz njih se mora moći dobiti operacijom prirodnog spajanja. U protivnom može doći do gubitka informacija. Pravilo prilikom dekompozicije kojega se treba držati glasi:

Relacija se dekomponuje bez opasnosti gubitaka funkcionalnih zavisnosti samo ako se dekompozicija vrši prema funkcionalnoj zavisnosti koja ne ide od kandidata ključa (kao što je urađeno sa relacijama koje povezuju laboratorije i laborante).

4.3.3 Ostale normalne forme

Pored pomenutih normalnih formi postoje još i:

- *Boyse - Codd-ova normalna forma (BCNF)* koja je u stvari stroži oblik treće normalne forme i relevantna je za one šeme relacija koje imaju više kandidata ključa, koji se pri tome još i međusobno prekrivaju. U praksi su ovakvi slučajevi rijetki.
- četvrta normalna forma eliminiše redundancu koja je u trećoj normalnoj formi moguća ako među atributima postoji višeznačna zavisnost,[3] čime se eliminišu i anomalije pri održavanju.

Posmatrajmo, na kraju, još jedan primjer kako bi i ova definicija postala razumljivijom.

Pretpostavimo da relacija NASTAVA ima tri atributa i to: šifrapredmeta#, šifranastavnika#, šifraudzbenika#

NASTAVA < *šifrapredmeta#, šifranastavnika#, šifraudzbenika#,.. >

---

[2] *Ako između atributa unutar jedne n-torke postoje sledeće funkcionalne zavisnosti: A→B, B→C i A→C, onda se kaže da je atribut C tranzitivno i funkcionalno zavistan od atributa A.*

[3] *Višeznačna zavisnost među atributima neke relacije* ***R*** **<** *X, Y, Z* **>** *postoji onda kada skup vrijednosti atributa Y zavisi od X a ne zavisi od Z. Riječima iskazano; X višeznačno određuje Y, odnosno Y višeznačno zavisi od X.*

te da jedan predmet može da predaje više nastavnika, i da se za jedan predmet može koristiti više udžbenika. Tako u relaciji NASTAVA imamo dvije višeznačne zavisnosti jer jednom predmetu (na primjer fizici) pripada više nastavnika (na primjer Marković, Pavlović, Marić) i više udžbenika (na primjer "Fizika materijala" i "Atomska i molekularna fizika").

Ovako koncipirana relacija NASTAVA sadrži prema tome redundancu, jer se zapis o tome da pojedini nastavnik predaje neki predmet javlja onoliko puta, koliko različitih udžbenika on koristi za taj predmet.

I ova redundanca se može dekompozicijom izbjeći, i tada se kaže da je sistem u četvrtoj normalnoj formi. Konkretno, u ovom slučaju relaciju NASTAVA treba predstaviti dvjema relacijama:

NASTAVA1 < šifrapredmeta#, šifranastavnika#, …. >
NASTAVA2 < šifrapredmeta#, šifraudzbenika#, ….. >

- peta normalna forma je posljednji stupanj dekompozicije koja riječima iskazana glasi:

relacija se nalazi u petoj normalnoj formi ako se ne da dalje, sa nekim smislom, dekomponovati.

4.3.4 Primjeri normalizacije

Pokažimo proces normalizacije na dva konkretna primjera.

Primjer 1:

Izvor podataka za projektovanje IS-a je formular pomoću kojega nabavljač nekog hotela vrši narudžbe robe potrebne za hotelsko poslovanje. Objekat NARUDŽBA, koji treba da se modelira za potrebe obrade podataka, ima veći broj atributa.

NARUDŽBA

Broj Narudžbe: 23425142				*Datum*: 17.03.2007		
Šifra kupca: K - 302111						
Naziv i adresa kupca: Hotel "Jahorina", 71436 Pale						
Datum isporuke: 23.04.2007						
Primjedba:	Telefonski razgovor sa gospodinom Tomićem					
Proizvod	*Naziv*	*Količina*	*J.M.*	*Cijena*	*Vrijednost*	*Popust*
P-122	prašak	300	kg	5.3	1590	3
S-001	sapun	230	kom	2.0	460	2
D-123	deterdžent	200	kg	5.5	1100	5
Ukupno:					2790	

***Prva normalna forma** relacije NARUDŽBA treba da obezbijedi da se u svakom polju relacije nalazi samo jedan podatak, to jest da svaka n-torka relacije bude sastavljena samo od atomarnih vrijednos-*

ti. Takva relacija (kreirana na osnovu hotelske narudžbe) može da ima slijedeći oblik:

NARUDŽBA < *br#, dnar, šk, nk, disp, pr, šp, np, kol, jm, cpr, vp, pop, uc, up, ukv* >

br - broj narudžbe,	*dnar - datum narudžbe,*
šk - šifra kupca,	*nk - naziv i adresa kupca,*
disp - datum isporuke,	*pr - primjedba,*
šp - šifra proizvoda,	*kol - količina,*
jm - jedinica mjere,	*cpr - cijena proizvoda,*
vp - vrijednost proizvoda,	*p - popust,*
uv - ukupna vrijednost,	*up - ukupan popust,*
uc - cijena koju treba platiti.	*np – naziv proizvoda*

Druga normalna forma treba da eliminiše redundancu podataka što drugim riječima znači da*:*

- *podatke koji se ponavljaju treba izdvojiti iz relacije,*
- *formirati od svake grupe koja se ponavlja novu relaciju,*
- *i konačno svakoj tako dobijenoj relaciji dodati novi ključ.*

Dekompozicijom relacije NARUDŽBA dobijamo dvije relacije: *REL1 (koja sadrži podatke o kupcu)*

REL1 <brnar#, datnar, šifkp, nazkp, datis, prim, uknar, ukpop, ukvr >

i REL2 (sadrži podatke o naručenoj robi)

REL2 <brnar#, šifpr#, nazpr, kol, jm, cenpr, vrpr, pop >

Relacija REL2 ima složeni ključ,i treba utvrditi zavisnost svakog atributa od tog složenog ključa. Ukoliko postoje zavisnosti, relaciju treba prevesti u višu, treću normalnu formu.

U relaciji REL2, koja ima složeni ključ, atributi nazpr, jm i cenpr zavise samo od šifpr#, dakle od dijela ključa, pa ih zato treba ponovo izdvojiti u posebnu relaciju. Ovako dekomponovani sistem, koji je u II normalnoj formi dobija slijedeći izgled:

REL1 <*brnar#, datnar, šifkp, nazkp, datis, prim, uknar, ukpop, ukvr* >
REL21 <*brnar#, šifpr#, kol, vrpr, pop* >
REL22 < *šifpr#, nazpr, jm, cenpr* >

Treća normalna forma je slijedeći korak kojim se uklanjanju i eventualne tranzitivne zavisnosti. U relaciji REL1 atributi šifkp i nazkp zavise tranzitivno od ključnog atributa relacije brnar#, pa će se pojaviti redundanca podataka, to jest ponavljanje podataka o šifri, nazivu i adresi kupca prilikom svake narudžbe. Izdvajanjem ova dva atributa u zasebnu relaciju dolazi se do optimalne, 3-će normalne forme:

REL11 <*brnar#, datnar, šifkp, datis, prim, uknar, ukpop, ukvr* >
REL12 <*šifkp#, nazkp* >

REL21 <brnar#, šifpr#, kol, vrpr, pop >
REL22 <*šifpr#, nazpr, jm, cenpr* >

Primjer 2:

Objekat X1 informacionog sistema lanca prodavnica Y ima sljedeće atribute - *ime, adresu i telefon* -, a prodaje više artikala raznih proizvođača o kojima se vodi evidencija o *broju komada, cijeni i boji*.

Prva normalna forma sadrži sve raspoložive podatke dakle:

X1 < *šrad#, naziv, adr., tel, šart#, nazart, firma, boja, cijena, kom.>.*

Druga normalna forma treba da eliminiše podatke koji se ponavljaju. To su očigledno podaci o prodavnici koji će se ponavljati za sve artikle koji se nalaze u njoj. Izdvojimo stoga podatke o prodavnici u posebnu relaciju X2*:*

X2< *šrad#, naziv, adresa, tel, >*

Relacija X1 *glasi sada:*

X1 < *šrad#, šart#, nazart, firma, boja, cijena, kom.>.*

i ima složeni ključ, ali atribut "nazart" zavisi samo od dijela toga ključa (šart#). Izdvojimo ga u posebnu relaciju X3*:*

X3 < *šart#, nazart >*

pa se tako dobija 3NF sa slijedećim relacijama:

X1< *šrad#, naziv, adresa, tel, >*
X3< *šart#, nazart >*
X2< *šrad#, šart#, firma, boja, cijena, komada >.*

Optimalni normalni oblik (obično 3NF ili više) na najbolji mogući način opisuje relevantna znanja o objektima i vezama među njima.

Normalizacija ima i svoju slabu stranu što se manifestuje u smanjenoj efikasnosti modela. Pokazalo se, naime, da kontrolisana prisutnost redundance podataka (što se ER modelom može ostvariti) ponekad može da bude tolerantna i korisna, a potpunom normalizacijom tabela svaka redudanca se isključuje pa model neki puta bude komplikovan.

Na primjer, u slučaju sekretarijata za saobraćaj u kome se vodi evidencija o VOZAČ-ima i VOZILIMA, ER-model sadrži dva objekta (VOZAČ i VOZILO) i vezu 1:N među njima (jer jedan vozač može posjedovati više vozila). Relacioni model ima prema tome sljedeći oblik:

VOZAČ < *JMBG#, ime, prezime, adresa, tel. datum_rođenja...>*
VOZILO< *Reg_br#, marka, tip, boja, br_mot., snaga,... JMBG.....>*

i pogodan je za eksploataciju.

Ako se problemu sinteze priđe preko normalizacije onda je *prva normalna forma* skup svih atributa, dakle:

R1<*JMBG, ime, prezime, adresa, tel. Reg_br., marka, tip, boja, snaga,.....*>

Druga i treća normalna forma razdvaja atribute prema zavisnosti pa dobijamo:

R11< *JMBG#, ime, prezime, adresa, tel., ...*>
R22< *Reg_br#, marka, tip, boja, snaga, br_mot., br_šas.,... JMBG ...*>

Daljnu dekompoziciju možemo izvesti ako se pokaže da se neki podaci ponavljaju. Na primjer, u tabeli R22 pojavljuje se više puta:

"marka" = "Zastava" *"tip" = YUGO45*
"boja" = plava *"snaga" = 42kW*

odnosno neke druge vrijednosti za neke druge popularne modele, pa se ovi atributi mogu izdvojiti kao posebna relacija, nazovimo je R221

R221<*sifra#, marka, tip, boja, snaga*>

koja je u vezi sa relacijom R22 tipa 1:N jer postoji više vozila različitih registarskih brojeva sa ovakvim karakteristikama. Relacija R22 glasi sada:

R22 < *Reg_br.#, šifra, br_mot, br_šasije, JMBG.....*>

a E-R model u ovom slučaju ima izgled:

Glava 5

5.0 Osnove projektovanja

Projektovanjem informacionog sistema (IS) treba ili:

- *poboljšati postojeći informacioni sistem*
- *rekonstrukcijom postojećeg unaprijediti njegovo poslovanje.*

Da bi se to uspješno ostvarilo, mora se u oba slučaja upoznati i razumjeti funkcionisanje sistema i definisati način kako da se računar upotrebi što je moguće efikasnije.

Proces projektovanja grubo se može podijeliti na tri koraka:

- *analiza sistema (preliminarna i detaljna istraživanja),*
- *projektovanje baze podataka,*
- *izrada aplikacija.*

a počinje kada se utvrdi da je neophodno preći na moderniji način poslovanja, ili kada postojeći IS mora da se poboljša, jer ne zadovoljava.

Analiza sistema predstavlja proces sakupljanja i interpretacije podataka, dijagnosticiranje problema i korišćenje podataka za unapređenje poslovanja. Taj posao treba da obavlja *sistem analitičar*.

Prvi korak u analizi sistema je određivanje *domena*, odnosno oblasti rada za koju se IS projektuje (računovodstvo, pravosuđe, proces u hemijskoj industriji, itd.). Često analiza sistema mora da obuhvati i procjenu razvoja sistema, tj. da predvidi koje bi se sve situacije mogle pojaviti u bližoj pa i daljnoj budućnosti. Međutim, sistem analiza ne smije da se pretvori u određivanje šta računar može, a šta ne može da uradi. Takođe, modifikacija sistema treba da bude *rezultat*, a ne *cilj* analize. Ne treba po svaku cijenu uvoditi atraktivna rješenja koja ne doprinose efikasnosti sistema.

Razlozi uvođenja (ili modernizacije) računarske opreme pri rekonstrukciji odnosno uvođenju novog informacionog sistema su:

- *veća brzina rada računa*
- *veća tačnost u radu*
- *obrada uvijek na isti način,*
- *brži pristup podacima*
- *mogućnost dobijanja novih informacija,*
- *povećanje sigurnosti rada,*
- *povezivanje, više informacionih sistema,*
- *smanjenje troškova poslovanja.*

Projektovanje informacionog sistema sastoji se od više faza i to:

- *definisanje izvora informacija - podataka,*
- *izbor metoda i projektovanje sistema,*
- *izbor modela sistema,*
- *izbor načina izrade dokumentacije,*
- *izbor načina održavanja dokumentacije,*
- *definisanje načina vođenja projekta,*
- *vrednovanje kvaliteta projekta.*

5.1 Izvori podataka - informacija

Prikupljanje informacija je jedna od najvažnijih karika u razvoju informacionog sistema. Informacije se prikupljaju iz više izvora, a prije svega od:

- *korisnika, odnosno investitora projekta informacionog sistema,*
- *postojeće dokumentacije koja "kruži" u poslovnom sistemu, te*
- *spoljnih izvora (literatura i eksperti za određene oblasti).*

Postupak prikupljanja informacija izvodi se:

- *u fazi analize postojećeg informacionog sistema,*
- *u fazi utvrđivanja ciljeva koji se žele postići.*

1. Osnovni izvor informacija su korisnici sistema. Od njih se saznaje koje aktivnosti, odnosno koji se procesi i postupci izvode u realnom sistemu, sa posebnim osvrtom na ciljeve koji se žele dostići novim informacionim sistemom.

Informacije se od korisnika prikupljaju pomoću intervju-a, upitnika i posmatranjem.

Intervjui se koriste za sakupljanje podataka u direktnom razgovoru sa korisnicima, postavljanjem određenih pitanja od strane analitičara. Intervju se može obavljati grupno ili pojedinačno, a najbolje je koristiti ga tek na-

kon što je iskorišćen neki drugi metod za analizu, kada se već imaju neka saznanja o sistemu. *Intervju treba da bude razgovor a ne ispitivanje*.

Projektant IS-a koji namjerava dobiti intervju od korisnika mora se prethodno dobro pripremiti (najbolje unapred spremiti upitnik koji se na licu mjesta popunjava), a predmet intervjua kreće se od neformalnog upoznavanja sistema, do detaljne analize kako globalne strukture tako i pojedinačnih segmenata i elemenata sistema.

Intervjui se održavaju kontinualno i sa raznim osobama a u cilju dobijanja cjelovitog uvida u problem. Ukoliko su odgovori korisnika kvantitativni i prepušteni subjektivnom osjećaju korisnika, kao podatak se može uzeti i aritmetička sredina dobijenih odgovora.

Na primjer, ako se od korisnika traži podatak koju temperaturu u prostoriji smatra ugodnom za rad, kao rezultat treba uzeti aritmetičku sredinu dobijenih rezultata

Uspješnost intervjua umnogome zavisi i od izbora sagovornika i načina vođenja intervjua. Izbor sagovornika mora biti takav da garantuje maksimalnu pouzdanost podataka. Zato u prvoj fazi rada treba intervjue provesti sa rukovodećim kadrom jer se u toj fazi obično donose dalekosežne odluke o koncepciji informacionog sistema. Od rukovodećeg kadra treba pri tome pokušati dobiti garanciju da će i ostali sagovornici biti spremni na dodatne napore koji obavezno prate razvoj i implementaciju novog informacionog sistema.

Intervju mora biti dobro isplaniran. Analitičar koji planira intervju mora znati koje informacije može i smije tražiti od sagovornika. Najbolje je intervju započeti kratkim upoznavanjem sagovornika o rezultatima prethodnih razgovora, kao i o do tada stečenima saznanjima. Zatim treba sagovorniku precizno definisati problem koji treba da bude riješen, ali ni na koji način ne treba nametati rješenje nego samo sugerisati moguća. Posebno treba izbjegavati računarsku terminologiju koju sagovornik eventualno ne razumije a neće to javno da kaže, pa često daje neadekvatne i netačne odgovore.

Sve informacije nije moguće dobiti samo od jedne osobe i u toku jednog intervjua. Zato treba neki put na vrijeme prekinuti intervju i ugovoriti vrijeme nastavka. Isto tako sve informacije koje se dobiju treba pismeno dokumentovati, jer u suprotnom najčešće dolazi do nesporazuma; "*ko je kome kada nešta rekao*", odnosno "*šta je tada htjeo da kaže*".

Upitnik (anketa) je tehnika koja omogućava da analitičar kontaktira veliki broj ljudi i da uporedi njihove odgovore na ista pitanja. Upitnik može da bude i anoniman, što ima svojih prednosti. Naravno ima i svojih nedostataka, obično upitnik popuni samo 30-40% ljudi kojima je upitnik upućen. Mnogi ga popune tek toliko da ga ne vrate praznog ne razmišljajući kakve odgovore daju. U upitniku se obično postavljaju pitanja sa ponuđenim odgovorima, ali i ona koja zahtijevaju opširne odgovore.

Na primjer, ako projektujemo informacioni sistem biblioteke neka od takvih pitanja bi mogla biti:

- *1. Mislite li da se često dešavaju greške prilikom izdavanja knjiga čitaocima?*

  Da [ ] Ne [ ]

- *2. Koje su najčešće greške?*_____
- _____

2. Izvor informacija je i postojeća dokumentacija.

Osim intervjua i upitnika informacije se mogu dobiti i iz pisane dokumentacije, uglavnom iz "*ulaznih i izlaznih*" papira koji cirkulišu u toku radnog procesa. Ovi podaci samo upotpunjuju znanja dobijena intervjuima, i nisu sami za sebe dovoljni za meritorno odlučivanje. Svako preduzeće ima veliki broj dokumenata koja regulišu njihovu unutrašnju organizaciju i međusobne odnose. Uvidom u ažurnost raznih dokumenata može se doći i do novih pitanja na koja treba naći odgovore upravo u novom informacionom sistemu.

Ukoliko je postojeći (stari) informacioni sistem već implementiran na računar, onda je dio dokumentacije već sređen na način koji se može iskoristiti za projektovanje novog sistema. Analizom postojećeg softvera, kao i priručnika za njegovo korišćenje, mogu se steći dragocjena znanja o postojećem sistemu. Pri tome treba paziti da se ne "oslonimo" i suviše na podatke koje je prikupljao neko drugi, jer se često dešava da se upravo u tim podacima kriju greške koje su dovele do potrebe za izradom novog sistema. Konačno u dokumentaciju ulaze svi pisani dokumenti koji se u poslovanju pojavljuju (narudžbenice, računi, magacinske liste, itd).

Posmatranje je tehnika koja često pomaže da se otkriju činjenice koje se ne mogu utvrditi drugim tehnikama. U preduzeću (radnoj organizaciji) dolazi nekada do niza konfliktnih situacija koje se ne mogu videti kroz dokumente, ili o kojima sagovornici nisu spremni da govore tokom intervjua (na primjer, veliki broj telefonskih poziva između prodavnice i skladišta da bi se utvrdilo da li neki artikl postoji, i sl.).

3. Spoljni izvori informacija podrazumijevaju spoljnje saradnike, savjetnike, eksperte, literaturu kao i sisteme koji djeluju u bližoj ili daljnoj okolini. Spoljni izvori su naročito važni onda kada se informacioni sistem projektuje prvi put, pa korisnici sistema nisu sasvim sigurni šta hoće, i kako bi njihov sistem trebalo da izgleda. Vrlo često je sistem koji se projektuje podsistem nekog većeg sistema. U tom slučaju je i nadređeni sistem izvor spoljnih korisnih podataka i informacija.

U fazi prikupljanja informacija koriste se svi raspoloživi izvori, neki manje neki više, što prvenstveno zavisi od specifičnosti svakog slučaja. Tako na primjer, analitičar može na osnovu ulazne i izlazne dokumentacije da pripremi intervjue za korisnike, a kada neki korisnik nije u stanju da odgovo-

ri jasno i precizno, onda se konsultuje odgovarajući spoljni izvor sa kojim je korisnik u vezi.

Metodologija prikupljanja podataka treba konačno da nam obezbijedi izbjegavanje redundance u podacima, to jest ponavljanja sličnih postupaka, i dobijanje istih podataka na različite načine.

Prikupljanje informacija treba organizovati po tzv. "*top - down*" metodi tako da se sistem oblikuje postepeno, počev od osnovnih elemenata i ciljeva, koji onda nakon detaljne razrade pojedinih elemenata oblikuju konačne cjeline.

Na kraju treba provesti i verifikaciju predloženog modela čime se konačno potvrđuje da je analitičar detaljno i vjerodostojno upoznao sistem koji projektuje.

I optimizacija dobijenog rješenja je poželjna, ali nije uvijek i moguća. Zato treba odvojiti pojam optimizacije modela informacionog sistema, od optimizacije tehnološkog postupka za koji se informacioni sistem projektuje.

Analitičari informacionog sistema vrlo često prave grešku smatrajući svojom dužnošću da u sklopu projektovanja informacionog sistema poboljšaju tehnološki proces.

Ali ako informacioni sistem u eksploataciji posluži inženjerima i tehnolozima za optimizaciju tehnološkog postupka, onda je to najbolji pokazatelj da je projekat informacionog sistema uspio.

5.2 Izbor metoda

Izbor metoda za razvoj IS-a (linearni, nelinearni, evolucioni, itd.), kao i izbor odgovarajućeg softvera kojim će projektovani sistem biti podržan, veoma je važan faktor pri projektovanju. S obzirom na relativno veliki broj raspoloživog softvera danas, u praksi se sreće čitav niz različitih modela, pa je i izbor i odgovarajućih metodologija rada velik.

Uobičajeno je da se u okviru *jedne* radne organizacije ili *jednog* projektnog tima, koristi *jedna detaljno razrađena metodologija*, jer se time postiže standardizacija procesa projektovanja i dokumentovanja sistema. Uvođenje stalno "novih" i boljih metodologija ima više loših nego dobrih efekata na efikasnost i kvalitet rada projektnog tima. Svaki informacioni sistem određen je sa tri osnovna elementa:

- *tokovima podataka*
- *podacima,*
- *procesima u informacionom sistemu,*

pa stoga svaka metodologija projektovanja informacionih sistema mora da sadrži i *sredstva koja omogućavaju definisanje tokova podataka, procesa obrade i organizacije podataka***.** Redoslijed projektovanja i analize pojedinih

elemenata kod raznih metodologija je različit pa se danas govori o metodologijama:

- *modela tokova podataka,*
- *modela podataka,*
- *modela procesa.*

U prvom slučaju projekat otpočinje analizom tokova podataka, u drugom dekompozicijom sistema na podsisteme, i konačno u trećem slučaju - od definicije logičkog modela podataka u sistemu.

Teško je reći koji je od ova tri pristupa najbolji i danas se smatra je *da ne postoji jedan generalni, najbolji metod projektovanja* nego se za svaki slučaj traži i optimalno rješenje.

5.3 Izrada modela

Modela se izvodi u dva koraka i to:

1. Izradom logičkog modela sa definisanjem logičke strukture:

- *sistema,*
- *podsistema,*
- *tokova procesa i*
- *modela podataka.*

Model procesa i model podataka su konačni rezultat faze istraživanja, koja najčešće započinje analizom stanja postojećeg sistema.

2. Izradom fizičkog modela informacionog sistema sa:

- *projektovanjem baze podataka (na osnovu modela podataka)*
- *izradom aplikacija (na osnovu modela procesa),*
- *definisanjem podsistema, tokova, procedura, modela i podataka, te*
- *definisanjem zahtjeva koji se postavljaju pred sistem.*

5.4 Analiza postojećeg sistema

Uvijek kada je to moguće proces projektovanja novog sistema započinjemo analizom postojećeg sistema. Ova analiza može biti izvedena preciznije (sistem nam stoji na raspolaganju!) nego što će to biti slučaj sa početnim fazama novog sistema. Zato se analiza logičkog modela postojećeg sistema izvodi u više faza i to:

- *analiza i prikaz globalne strukture postojećeg sistema,*
- *dekompozicija sistema na njegove podsisteme,*
- *izrada dijagrama toka podataka za svaki podsistem, te*
- *izrada i detaljan opis potrebnih procedura za obradu.*

Analiza logičkog modela postojećeg sistema nastavlja se analizom i izradom fizičkog sistema u kojem su, u posljednjoj fazi, opisane procedure koje se odvijaju u sistemu.

Logički model sistema treba prema tome samo da prikaže logičku strukturu pojedinih elemenata ne ulazeći u način izvođenja pojedinih procedura, tako da se može generalno reći da logički model *ističe procese, a zanemaruje postupke*.

5.5 Projektovanje novog sistema

U toku rada na projektu novog informacionog sistema potrebno je proći tri osnovne faze i to:

1. Definicija problema:
   - *definisanje ciljeva i spoljnih ograničenja na sistem,*
2. Izrada okvirnog projekta:
   - *definicija, izrada i oblik logičke strukture novog sistema,*
   - *izrada i usvajanje prijedloga fizičkog modela novog sistema,*
   - *specifikacija računarske opreme,*
3. Izrada detaljnog projekta:
   - *definicija korisničkih procedura,*
   - *izrada prijedloga logičkog modela, te*
   - *definicija baze podataka i programa obrade podataka.*

Ocjena izvodljivosti novog sistema ne može se odmah precizno definisati u opštem slučaju. Prilikom ocjene izvodljivosti sistema treba voditi računa o tome:

- *kako investitoru predstaviti novi sistem,*

a tom prilikom treba:

- *izložiti dinamiku realizacije (bar u okvirnoj formi),*
- *koliko će sve to koštati, i što je najvažnije,*
- *uvjeriti investitora da će sistem funkcionisati.*

Da bi se mogli utvrditi svi nabrojani elementi ocjena izvodljivosti treba da sadrži:

- *prikaz strukture informacionog sistema, te*
- *blok dijagrame pojedinih podsistema,*

a za utvrđivanje visine potrebne investicije mora se (okvirno) utvrditi:

- *koju i koliku opremu treba nabaviti,*
- *procijeniti troškove opreme, te*
- *dinamiku nabavke i održavanja.*

Na osnovu ovako dobijene ocjene izvodljivosti sistema definiše se prijedlog novog sistema. Naravno, ne mora svaki prijedlog biti i prihvaćen, zapravo, vrlo često prijedlog bude odbačen. Da ne bismo gubili i suviše mnogo vremena na izradi velikog broja prijedloga, od kojih će većina biti

odbačena, u praksi je uobičajeno da se naprave tri okvirna prijedloga koji se razlikuju po intenzitetu tehnološkog i organizacionog zahvata u postojeći sistem.

- *Prvi prijedlog treba da definiše moguću ali ostvarivu granicu automatizacije novog sistema.*
- *Drugi prijedlog definiše minimum zahvata u postojeći sistem koji donose kvalitativne novosti.*
- *Treći prijedlog treba da bude kompromis želja i mogućnosti.*

Svi navedeni prijedlozi vrednuju se pomoću tri osnovna kriterijuma i to prema:

- *tehničkoj izvodljivosti,*
- *operativno-organizacionoj prihvatljivosti, i*
- *ekonomskoj opravdanosti predložene investicije.*

Sistem se smatra tehnički izvodljivim i prihvatljivim ako u organizaciji postoji oprema, odnosno ako postoji mogućnost nabavke opreme za njegovu realizaciju.

Operativno-organizacijski je sistem prihvatljiv onda kada na svom izlazu daje kvalitetne informacije za potrebe tehnološkog procesa i upravljanja radnom organizacijom.

Ekonomska opravdanost razvoja i uspostavljanja novog informacionog sistema određuje se poređenjem troškova za njegovu realizaciju, i dobiti koju taj sistem donosi. Ovo poređenje nije moguće izvesti do u detalje, jer mnogi parametri u startu, kao na primjer troškovi razvoja, rada i održavanja sistema nisu neposredno mjerljivi, dok je dobit u fazi razvoja samo djelimično kvantitativno mjerljiva.

Detaljno izvedena analiza postojećeg logičkog modela informacionog sistema najbolja je osnova za izradu novog projekta, jer se rad na izradi novog sistema uvijek započinje projektovanjem strukture i to:

- *definisanjem novog logičkog modela, i*
- *definisanjem fizičkog modela novog sistema.*

Logički model novog sistema je po formi uvijek sličan logičkom modelu koji je generisan u procesu analize postojećeg sistema. Stoga se preporučuje u procesu projektovanja logičkog modela novog sistema korišćenje istih jezika, istih softverskih paketa (po fazama), kao i u procesu analize prethodnog sistema.

U fazi izrade *fizičkog modela* novog sistema, donosi se odluka o tome koji će se sistem za upravljanje bazama podataka koristiti u implementaciji, kreira se baza podataka i utvrđuju se procesi definisani logičkim mo-

delom koji će se obavljati ručno, a koji će se automatizovati. U skladu sa tim definiše se i potrebna računarska oprema koja je određena prije svega:

- *brojem podataka,*
- *potrebnim vremenom pristupa na diskovima, te*
- *frekvencijom i brzinom izvođenja procesa obrade.*

Projektovanje novog sistema je složeniji postupak od analize postojećeg, jer se analizom opisuje postojeći sistem, dok se projektom daje nešta novo. Ocjena uspješnosti analize postojećeg modela uvijek je moguća, jer se kriterijum zna - koliko vjerno dobijeni model opisuje postojeći sistem - dok se za novi sistem to ne može reći. Postupak projektovanja novog sistema polazi od:

- *logičkog modela postojećeg sistema,*
- *ciljeva koji se žele dostići a definisanih u prethodnim fazama,i*
- *zahtjeva postavljanih u toku analize postojećeg sistema.*

a izvođenje se realizuje obično u dvije faze i to:

- *izrada okvirnog projekta,*
- *izrada detaljnog projekta.*

Izrada okvirnog projekta počinje definisanjem problema, te ocjenom mogućnosti i načina za njegovo rješavanje.

Način definisanja problema kao i ciljeva novog informacionog sistema nije u početku precizno određen, jer se projektovanje radi u vrlo širokom spektru raznih situacija. Međutim, neke preporuke, proizašle iz prakse, ipak postoje i sastoje se u slijedećem:

- *svi ciljevi i problemi treba da budu dokumentovani u pismenoj formi,*
- *ciljevi treba da su tako definisani da ne budu neostvarivi (ne smiju biti prosti spisak želja),*
- *procjenu realnosti ciljeva treba izvesti uzimajući u obzir opšti nivo razvijenosti tehnologije u firmi i organizacije poslovanja,*
- *kao najbolja polazna osnova za definisanje ciljeva novog informacionog sistema služe "uska grla" postojećeg,*
- *ciljevi se oblikuju vrlo često i po uzoru na druge postojeće informacione sisteme iz bliže ili daljnje okoline.*

Izrada detaljnog projekta naslanja se na prijedlog logičkog modela novog sistema definisanog u prethodno opisanoj fazi. Projektovanje počinje najčešće definisanjem korisničkih procedura koje određuju kako sistem treba da izvede pojedine funkcije, i šta u tom smislu treba korisnik da zna i da uradi. Nakon toga slijedi definisanje fizičkog modela baze podataka, koji se naslanja na E-R model cijelog sistema, odnosno na relacioni model izveden iz E-R modela, ili je pak dobijen postupkom normalizacije, a najčešće njihovom kombinacijom.

Na kraju, na osnovu korisničkih procedura i fizičkog modela baze podataka, definišu se procesi obrade podataka, to jest projektanti i programeri pišu detaljne računarske programe po pravilima oblikovanja kompleksnih programskih struktura (modularno, koristeći tehnike strukturiranog i objektno-orijentisanog programiranja).

5.6 Realizacija sistema

Realizacija sistema počinje fizičkom realizacijom usvojenog modela baze podataka, a sastoji se od:

- *inicijalizacije baze podataka,*
- *izrade programa i testiranje u realnom okruženju,*
- *izrade konačne dokumentacije, te*
- *uvođenja sistema u eksploataciju.*

Unutar navedenih faza treba riješiti razne probleme koji mogu nastati, jer se tek u ovoj fazi vide pravi rezultati rada u prethodnim fazama.

- *Inicijalno punjenje* projektovane baze podacima je prva faza realizacije. Tek nakon toga, kada imamo određenu količinu podataka u sistemu, pristupa se izradi i testiranju programa za realizaciju tokova i procesa obrade podataka.

- *Izrada i testiranje pojedinih programskih komponenti* sistema izvodi se neposredno po realizaciji pojedinih modula programa i to po principu “što prije - to bolje“, kako bi se spriječilo prenošenje grešaka iz modula u modul. Na kraju slijedi testiranje sistema kao cjeline, konačno punjenje baze svim podacima, te probni rad i uhodavanje sistema. Testiranjem IS u realnoj situaciji provjeravaju se programi i naravno model procesa. Najčešće se paralelno koristi stari i novi informacioni sistem i upoređuju se rezultati. Problemi koji se mogu javiti posljedica su grešaka u izradi modela procesa, modela podataka ili njihove implementacije.

- *Izrada i održavanje dokumentacije* sistema dijeli se na:

  - *izvještaje (po fazama rada), i*
  - *kompleksnu dokumentaciju.*

- *Izvještaji* su namijenjeni informisanju rukovodstva investitora projekta i treba da sadrže slijedeće elemente:

- *specifične podatke o radu i rezultatima za svaku fazu rada,*
- *opšta saznanja o projektu stečena u svakoj fazi rada,*
- *preporuke i prijedlog plana nastavka rada,*
- *izvještaj o utrošenim sredstvima, i*
- *dokumentovan zahtjev za sredstva potrebna za nastavak.*

- *Kompletnu dokumentaciju sistema* čine opisi i definicije pojedinih komponenti sistema, a u zavisnosti od metodologije prema kojoj je sistem razvijan. Tu su prije svega:

  - *opis strukture podataka*, unutar koga se prikazuju hijerarhijski dijagrami dekompozicije, dijagrami toka podataka, E-R model podataka i odgovarajući logički model,
  - *opis procesa sistema*, unutar kojih se detaljno daju dijagrami strukture procesa obrade, dijagrami akcija, te stabla i tabele odlučivanja,

- *Uvod i osnovne karakteristike* sistema sadrže:

  - *nekoliko riječi o autoru (autorima) sistema,*
  - *podatke o osobama ili institucijama od kojih se mogu dobiti dodatne informacije,*
  - *opis opreme na kojoj sistem radi,*
  - *ograničenja koja su uvedena u sistem,*
  - *spisak pretpostavki na kojima se sistem bazira.*

- *Ulaz i izlaz* iz sistema treba da sadrži:

  - *spisak izlaza iz sistema sa naznakom kako se koriste,*
  - *spisak ulaza u sistem sa naznakom odakle je sve moguće uvesti informaciju u sistem,*
  - *način pokretanja i zaustavljanja sistema,*
  - *primjer sa korišćenjem ulaza, izlaza i najvažnijih operacija.*

- *Osnovne funkcije* sistema su:

  - *opis svake funkcije u sistemu,*
  - *namjena (kada i gdje se koristi) svaka funkcija,*
  - *efekti koji se postižu sa pojedinim funkcijama,*
  - *moguće jednostavnije greške, njihov opis i način eliminacije,*
  - *ilustrativni primjer.*

- *Detaljan opis* instalacije sistema sadrži:

  - *detaljan opis instalacije sistema, a*
  - *sadrži opis programa sa opisom procedura i varijabli u njemu.*

- *Korisnički priručnik*

  - *Korisnički priručnik, odnosno uputstvo za upotrebu, treba korisnika da uputi u mogućnosti i način korišćenja informacionog sistema.*

5.7 Vođenje i vrednovanje projekta

Poslovi vođenja nekog projekta zavise prvenstveno od opsega rada i pristupa razvoju informacionog sistema. Uspješnost vođenja projekta zavisi od više faktora od kojih su najvažniji:

1. Sastavljanje projektnog tima što je često i presudan faktor za uspješnost realizacije svakog, pa i projekta informacionog sistema. Svaki projektni tim treba da se sastoji od:

- *analitičara,*
- *tehnologa,*
- *projektanta,*
- *korisnika,*
- *rukovodioca radne organizacije, i*
- *predstavnika investitora.*

2. *Vrednovanje projekta* izvodi se na osnovu niza kriterijuma kao:

- ispravnost, da li sistem daje tačne informacije ili su moguće i lažne,
- potpunost, to jest da li sistem može da generiše sve potrebne informacije,
- robusnost, se ogleda u reakcijama sistema na nepredvidivo rukovanje,
- pouzdanost, koja je prije svega određena kvalitetom angažovane opreme,
- optimalnost, da li je sistem mogao biti realizovan i sa manje sredstava,
- jednostavnost u korišćenju, koliko je sistem "naklonjen" korisniku,
- jednostavnost održavanja, upoznavanja, održavanje i izmjena dijelova,
- mogućnost proširenja i povezivanja sa drugim sistemima, te konačno,
- prenosivost, koja se ogleda u fleksibilnosti i prenošenju u nove uslove.

Iskustvo je pokazalo da informacioni sistemi nikada u potpunosti ne zadovoljavaju sve postavljene zahtjeve, odnosno nemaju željene karakteristike. Razlog tome leži u činjenici da se podaci i procesi u realnom sistemu stalno mijenjaju i zato informacioni sistemi moraju biti sposobni da se prilagođavaju i nadograđuju.

Sinteza fizičkog modela

Glava 6

SQL

Sastavni deo svakog sistema za upravljanje bazama podataka čine i specijalni jezici za opis i korišćenje baza podataka koji sadrže sledeće sastavne delove: *jezik za opis podataka* (DDL – Data Description Language), *jezik za opis fizičke strukture podataka* (DMCL – Device Media Control Language), *jezik za rad sa podacima* (DML – Data Manipulation Language) i *jezik upita* (QL – Query Languages).

Jezik za opis podataka može biti poseban neproceduralni jezik ili proširenje postojećeg programskog jezika (Cobol, C, Pascala ili C++). Njegova osnovna funkcija je specifikacija logičke strukture podataka time što se definišu: objekti i/ili tabele, logičke veze između objekata, atributi i dozvoljeni interval vrijednosti za svaki atribut.

Jezik za opis fizičke strukture podataka definiše: način memorisanja i dodjele memorijskog prostora na disku, redosled i fizičku organizaciju podataka, dodjelu privremene memorije i načine adresiranja i pretraživanja podataka.

Jezik za rad sa podacima obezbeđuje vezu između podataka i aplikacije (programa za rad sa podacima: unos, obradu, prikaz i sl.), a njegove osnovne funkcije su: otvaranje i zatvaranje datoteka (naredbe OPEN i CLOSE), pronalaženje željenog sloga (FIND), izmjena sadržaja nekog polja (MODIFY), dodavanje sloga (INSERT), brisanje sloga (DELETE, REMOVE) itd.

Jezici upita omogućavaju realizaciju proizvoljnih funkcija – upita nad relacijama. Postoje dve klase ovih jezika zavisno da li se zasnivaju na relacionoj algebri ili na predikatskom (relacionom) računu. U relacionoj algebri definisane su operacije pomoću kojih je moguće dobiti željenu relaciju (tabelu) iz skupa datih relacija (tabela). Relacionim računom definišu se osobine relacija koje se žele dobiti.

U relacionoj algebri definisane su sledeće operacije: unija, diferencija, presjek, Kartezijev (Dekartov) proizvod, projekcija, selekcija (restrikcija), spajanje (join), djeljenje. Postoje još neke izvedene operacije uslovljene Null vrijednostima.

Relacioni račun je neproceduralni jezik i programer umjesto da definiše proceduru pomoću koje će se dobiti željeni rezultat, samo specificira željeni rezultat. Postoje dva oblika relacionog računa: relacioni račun n-torki i relacioni račun domena.

Primjer relacionog računa n-torki:

```
x ∈ RADNIK
x.SIFRA, x.IME   GDE_JE   x.PLATA > 2000
AND   x.KVALIF = 'VKV'
```

Primjer relacionog računa domena:

```
x,y GDE_JE ∃ z > 2000 AND
RADNIK (SIFRA: x, IME: y, PLATA: z, KVALIF = 'VKV')
```

Zajednička karakteristika im je da su njihove konstrukcije mnogo sličnije prirodnim jezicima nego što je to slučaj sa jezicima treće generacije. Najpoznatiji primjeri ovih jezika su SQL, Quel i QBE. Oni se zasnivaju na raznim principima: QBE se zasniva na relacionom računu domena, Quel se zasniva na relacionom računu (ne podržava relacionu algebru), a SQL se zasniva na relacionom računu n-torki tj. na kombinaciji relacione algebre i relacionog računa.

Ova tri jezika nisu značajna samo u oblasti istraživanja baza podataka već imaju značajne komercijalne implementacije. Iako smo ih označili kao jezike upita to je u stvari netačno jer ovi jezici imaju ugrađene i sve druge mogućnosti: definisanje strukture podataka, modifikacija podataka u bazi kao i mogućnosti za definisanje bezbjednosnih ograničenja. Iako postoje komercijalni proizvodi zasnovani na sva tri jezika (Ingres je zasnivan na Quel-u) detaljnije ćemo obraditi SQL jer je on našao najširu primjenu, a u narednom poglavlju biće opisane mogućnosti QBE jer je njegova primena vrlo prosta i implementiran je u sve moderne RDBMS i alate za rad sa bazama podataka.

SQL je programski paket koji se zasniva na relacionoj algebri i relacionom računu. Skraćenica SQL dolazi iz engleskog **Structured Query**

Language, što bi u slobodnom prevodu značilo **Strukturni jezik za postavljanje upita.** Teorijske osnove, kao i koncept prvog SQL programskog paketa, dao je E.F. Codd utvrdivši u svom, već pomenutom, radu:

"...***da koncept relacionog modela dopušta razvoj univerzalnog jezika za manipulisanje podacima baziranog na primjeni relacione algebre***".

Razvoj SQL programskog paketa tekao je u skladu sa tempom kojim se danas prihvataju nove ideje u tehnici, a posebno u računarskoj tehnici. Godine 1974, dakle četiri godine nakon definisanja teorijskih principa relacionih baza podataka, nastaje programski paket nazvan **SEQUEL,** da bi ubrzo došlo do njegove modifikacije i pojave paketa **SQUARE**, a potom, kako to onda obično biva dolazi **SEQUEL2**. IBM razvija sopstveni paket pod imenom **SISTEM-R**. Neka vrsta sinteze iskustava pomenutih programskih paketa ostvarena je u verziji koja se danas nalazi pod imenom **MySQL**.

Osnovna ili standardna verzija SQL programskog paketa je verzija predviđena za manipulisanje podacima u relacionim bazama i implementirana je danas praktično u svim programskim paketima za obradu podataka kao njihov sastavni dio, a u svojoj sintaksi ima za sve osnovne relacione operatore (spajanje, razliku, množenje, restrikciju i projekciju) ekvivalentne SQL naredbe. Tri preostala relaciona operatora (dijeljenje, presjek i unija) ne smatraju se osnovnim, jer se te operacije mogu izvesti kombinovanjem pomenutih, osnovnih, i za razliku od većine verzija SQL-a, upitni jezik Quel ih ne podržava. SQL je standardizovan od strane kompetentnih međunarodnih institucija kao što su **ISO** (International Standardisation Organisation) i **ANSI** (American National Standards Institute). U ovoj knjizi poštovana je sintaksa propisana po ANSI standardu.

Osnovni nedostatak SQL paketa ogleda se u činjenici da sve njegove mogućnosti i opcije ne mogu savladati korisnici - laici u smislu projektovanja, eksploatacije, obrade i neposrednog korišćenja informacionih sistema. SQL**,** naime, zahtijeva od korisnika poznavanje konfiguracije informacionog sistema (logički model baze, odnosno, spisak relacija, veza među njima kao i raspored atributa u njima) te poznavanje osnova tehnike programiranja.

SQL spada u takozvane ***no-case-sensitive*** programske pakete, što znači da se naredbe, ključne riječi, imena objekata i varijabli mogu pisati i malim i velikim slovima, a da sistem pri tome ne pravi razliku među njima. Međutim, i pored toga, zbog preglednosti napisanog programa, preporučljivo je dosledno koristiti mala i velika slova. Mi ćemo se stoga ubuduće pridržavati sledećih oznaka:

- velika slova - ključne riječi, funkcije, i imena relacija,
- mala slova - imena atributa, varijabli, itd.,
- italik mala slova - vrijednosti atributa (podaci),
- u uglastim zagradama - neobavezne opcije,
- komentari - dva znaka minus (- -).

Programski paket SQL omogućava jednostavno :

- kreiranje relacija,
- unos podataka,
- brisanje,
- ažuriranje,
- pretraživanje podataka, te
- prezentiranje novih informacija.

Korisnik može u komunikaciji sa sistemom upotrjebiti dvije opcije i to:

- interaktivni način rada (korisnik direktno komunicira sa bazom, a rezultat dobija na ekranu monitora), odnosno
- ugradnjom i povezivanjem SQL-a sa nekim programskim paketom (Delphi, Visual C++, Visual Basic itd.). Mogu se koristiti i gotove aplikacije napisane sintaksom jednog od tih jezika.

U prvom slučaju korisnik mora poznavati strukturu baze podataka (nazive tabela, relacije, odnose među njima, te imena atributa u njima), kao i sintaksu upitnog jezika.

U drugom slučaju, prilikom korišćenja unapred definisanih aplikacija, dovoljno je znati samo koju informaciju želimo dobiti (za koju naravno prije toga mora biti pripremljena odgovarajuća aplikacija) pa da njenim pozivom dođemo do rezultata. Nažalost, u ovome slučaju koji je praktičan za korisnika-laika, primjena je ograničena samo na one aktivnosti i operacije koje su unaprijed pripremljene.

Osnovne (ne i jedine) naredbe SQL-a koje se koriste u manipulisanju podacima u relacionoj bazi podataka omogućavaju definisanje, korišćenje i zaštitu podataka. Sve SQL naredbe po pravilu se završavaju interpunkcijskim znakom (**;**) i mogu se podijeliti u tri grupe.

a) ***Naredbe za definisanje podataka*** (Data Definition Statements) omogućavaju definisanje resursa i logičkog modela relacione baze podataka:
   - **CREATE TABLE** – kreiranje fizičke tabele baze podataka,
   - **CREATE VIEW** – kreiranje virtuelne imenovane tabele, “pogled”,

- **CREATE INDEX** – kreiranje indeksa nad jednom ili više kolona tabele ili pogleda,
- **ALTER TABLE** – izmjena definicije tabele, izmjena, dodavanje ili uklanjanje kolone (atributa),
- **DROP TABLE** – uklanjanje tabele iz baze podataka,
- **DROP VIEW** – uklanjanje pogleda iz baze podataka.

b) ***Naredbe za rukovanje podacima*** (Data Manipulation Statements) omogućavaju ažuriranje podataka u širem smislu (izmjenu, dodavanje i brisanje) i izvještavanje (pribavljanje novih informacija) iz baze podataka:

- **SELECT** – pristup podacima i prikaz sadržaja baze podataka,
- **INSERT** – unošenje podataka, dodavanje redova u tabelu,
- **DELETE** – brisanje podataka, izbacivanje redova iz tabele**,**
- **UPDATE** – ažuriranje, izmjena vrijednosti podataka u koloni.

c) ***Naredbe za upravljanje bezbjednošću podataka*** (Data Control Functions) omogućavaju oporavak, konkurentnost, sigurnost i integritet relacione baze podataka:

- **GRANT** – dodjela prava korišćenja tabele drugim korisnicima od strane vlasnika tabele,
- **REVOKE** – oduzimanje prava korišćenja tabele drugim korisnicima,
- **BEGIN TRANSACTION** – početak transakcije koji se može završiti jednom od dveju narednih naredbi,
- **COMMIT WORK** – prenos dejstva transakcije na bazu podataka,
- **ROLLBACK WORK** – poništavanje dejstva transakcije na bazu podataka.

Bez poznavanja navedenih komandi za definisanje podataka i rukovanje podacima ne može se ozbiljno računati na održavanje, dizajniranje i izradu elementarnih aplikacija za obradu podataka uz korišćenje moderne relacione tehnologije. Štaviše, za najveći broj standardnih zahtjeva, dovoljno je poznavanje samo ovih nekoliko naredbi, pa da se postigne željeni cilj.

Sa druge strane, ove osnovne naredbe predstavljaju uvod u šira znanja za one koji nameravaju da se profesionalno bave projektovanjem i održavanjem informacionih sistema, a kompletan spisak SQL naredbi može se naći u svakom priručniku **SQL**-a.

6.1 Naredbe za definisanje podataka

Nova relacija (tabela) kreira se pomoću programskog paketa SQL naredbom **CREATE TABLE**. Opšti oblik ove naredbe glasi:

CREATE TABLE ime_relacije, lista imena i tipova atributa

uz definisanje eventualnih ograničenja njihovih vrijednosti.

Ime relacije (ili **ime_tabele**) ne smije biti nijedna od ključnih riječi programskog paketa SQL i mora počinjati slovom engleskog alfabeta. Od specijalnih znakova može sadržavati samo znak **_**. Dužina imena nije precizirana.

Ime atributa bira se na isti način kao i ime relacije. Broj atributa jedne relacije ograničen je mogućnostima računara, a zavisi i od implementacije. U praksi su relacije sa više od 30 atributa rijetkost. Obično je taj broj, u dobro projektovanim sistemima, od 10 do 30.

Svi sistemi za upravljanje bazama podataka postavljaju neke ograničenja u pogledu broja atributa. U Access-u taj broj ne može biti veći od 255 a u SQL Serveru od 1024.

U jednoj bazi podataka, kao što smo vidjeli, može postojati više atributa sa istim imenom, ali oni ne smiju pri tome biti u istoj relaciji (tabeli). Preporučljivo je ovu opciju (ista imena atributa u raznim tabelama) radi preglednosti i eliminacije grešaka, izbjegavati kad god je to moguće.

Tip atributa služi da se pobliže opiše podatak i na taj način odredi optimalan način njegovog memorisanja - sa jedne strane, a definicijom **ograničenja vrijednosti** s druge strane, spriječavaju se moguće greške pri unošenju podatka.

Razni programski paketi omogućavaju definisanje različitih tipova podataka, ali se svi svode na nekoliko osnovnih tipova.

- **Slovni ili znakovni tip** podatka koristi se kada podatak predstavlja niz alfanumeričkih znakova. Takav podatak je na primjer ime i prezime, adresa ili zanimanje radnika, broj telefona itd. Slovni ili **string** podatak može biti svaki niz alfanumeričkih i nekih od specijalnih znakova. Broj znakova može biti ograničen na *n* (tip podatka je u tom slučaju **CHAR(n)**), ili je promjenljive dužine (**VARCHAR**).
- **Numerički (NUMERIC) tip** podatka (na primjer visina ličnog dohodka ili dužina neke ulice) može slično kao i u drugim programskim jezicima, da bude cjelobrojan (**SMALLINT,** ako je dužina 16 bita, **INTEGER**, ako je dužina 32 bit-a i **QUADWORD,** sa dužinom od 64 bita), ali i realan broj sa fiksnom ili pomičnom decimalnom tačkom razne preciznosti koja zavisi od dužine, to jest broja cifara. Tako je tip podataka **FLOAT(n)** realan broj do 6 cifara, odnosno **DOUBLE PRECISION** dužine od 7 do 15 cifara.

- **Datumski (DATE)** i **vremenski (TIME) tip** podatka često se koristi u informacionim sistemima jer je čitav niz podataka vezan za vrijeme, za neke rokove, bez obzira da li su iskazani danima, mjesecima i godinama ili satima, minutama i sekundama.

  Datumski tip DATE je jedna cjelina, jedan podatak, iako u sebi sadrži tri numerička polja (za *dan*, *mjesec* i *godinu*), međusobno odvojena tačkom, crtom, ili kosom crtom, a što zavisi od zemlje u kojoj će se koristiti. Za ovakav tip podatka važi i posebna aritmetika, koja omogućava korisniku da računa vremenske intervale.

  Za vremenski tip podatka TIME važe ista pravila kao i za datumski, s tim što se koriste različiti postupci za obradu zasnovani na različitim aritmetikama: jedna aritmetika za datumski tip, a druga za vremenski, jer godina ima 12 mjeseci, a mjesec 28 (29), 30 ili 31 dan, dok je za sat, minut i sekundu taj odnos 24:60:60, pa se i odgovarajuće aritmetike shodno tome moraju razlikovati.

- **Logički tip** podatka (**LOGICAL**) koristi se kod atributa kod kojih je domen ograničen na dvije vrijednosti.

  *Na primjer takav atribut je **prisutnost,** kod koga mogu postojati samo vrijednosti **prisutan** ili **nije_prisutan,** ili atribut **pol** kod koga mogu postojati vrijednosti **muški** i **ženski**.*

- **Memo tip** podatka je u principu slovni (alfanumerički) tip, ali sa većom dužinom (na primjer do 64 kilobajta) a koristi se za unošenje opisnih podataka (dijagnoza ili anamneza pacijenta, na primjer).

- **Bit-mapa** podatak nosi neku grafičku informaciju, dijagram ili sliku.

U tehnici definisanja podataka postoji i pomenuti slučaj **nepostojećeg podatka** kada vrijednost podatka (bilo kog tipa) nije poznata ili nije nastupio momenat njegovog prisustva u bazi, a koji u drugim programskim jezicima nije poznat, takozvana ***Null-***vrijednost. SQL dopušta svakom podatku (sem primarnog ključa) da ima i nepostojeću, Null-vrijednost.

U sledećem primjeru možemo vidjeti kako se jednostavno pomoću SQL-a kreira nova tabela. Pretpostavimo da u informacionom sistemu nekog preduzeća treba kreirati tabelu **RADNIK,** u kojoj bi se nalazili podaci o kvalifikaciji, imenu radnika, poslu koji obavlja, njegovom rukovodiocu, datumu zaposlenja, premiji i plati.

*Tabelu **RADNIK** kreiramo naredbom:*

```
CREATE TABLE RADNIK
        (idbr# INTEGER NOT NULL,
        kvalif CHAR(3),
        ime CHAR(25) NOT NULL,
        posao CHAR(10),
        rukovodilac INTEGER,
        dat_zap DATE,
        premija FLOAT(1),
        plata FLOAT(1),
        brod$ SMALLINT );
```

a kao rezultat dobijamo relaciju:

RADNIK <idbr#,kvalif, ime, posao, rukovodilac, dat_zap, premija, plata, brod$>

u kojoj atributi ***idbr#*** *i* ***ime*** *ne mogu imati vrijednosti* ***Null****.*

Za potpuniju informaciju o preduzeću, sem tabele RADNIK, potrebno je kreirati i tabelu ODJELJENJE u kome radnik radi i tabelu PROJEKAT koja sadrži informacije o poslovima kojima se preduzeće trenutno bavi. To se postiže sledećim naredbama CREATE TABLE:

```
CREATE TABLE ODJELJENJE
        (brod# SMALLINT NOT NULL,
        ime_od CHAR(15) NOT NULL,
        mesto CHAR(20));

CREATE TABLE PROJEKAT
        (brproj# INTEGER NOT NULL,
        imeproj CHAR(25) NOT NULL,
        sredstva DOUBLE(2));
```

Ovako definisana tabela RADNIK sadrži u sebi i neke relacije između pojedinih radnika – *unarne veze* (neki radnici su istovremeno rukovodioci nekim drugim radnicima). Takođe je ostvarena i jedna relacija sa jednim drugim entitetom – *binarna veza* sa tabelom ODJELJENJE, jer radnici su zaposleni u nekom od odjeljenja koja se nalaze u sastavu preduzeća. Pri tome jedan radnik pripada samo jednom odjeljenju, a u jednom odjeljenju radi više radnika.

Ovo je veza tipa 1:N (jedan prema više), a ostvaruje se tako što se u tabeli RADNIK na strani više (više radnika) uvodi kao atribut spoljni, strani, ključ *brod$* koji predstavlja primarni ključ u tabeli ODJELJENJE (na strani **1**). Spoljni ključ prepoznajemo po znaku $ iza imena atributa.

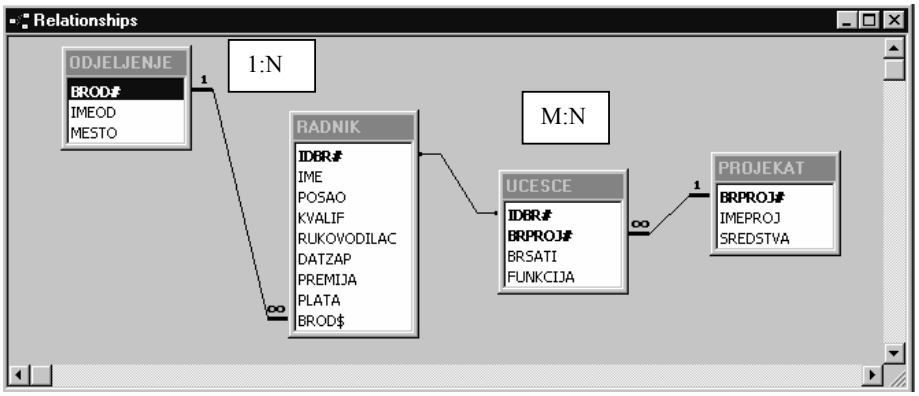
Ali, svi radnici rade na nekim konkretnim poslovima, projektima, i pri tome jedan radnik može raditi na više projekata, a istovremeno na jednom velikom projektu radi više radnika. Dakle ovo je relacija M:N (više prema više). Da bismo ostvarili ovu relaciju između dva entiteta

treba kreirati novu tabelu, nazovimo je UČEŠĆE, koja ima **složeni primarni ključ** (*idbr#,brproj#*) koji sačinjavaju primarni ključevi iz tabela RADNIK (*idbr#*) i PROJEKAT (*brproj#*).

```
CREATE TABLE UČEŠĆE
        (idbr# INTEGER NOT NULL,
        brproj# INTEGER NOT NULL,
        brojsati SMALLINT,
        funkcija CHAR(15),);
```

Dakle, bazu podataka sačinjavaju četiri fizičke tabele (slika 6.1.):

RADNIK <idbr#,kvalif, ime, posao, rukovodilac, dat_zap, premija, plata, brod$>
ODJELJENJE <brod#,, naziv, mesto>
PROJEKAT <brproj#,, imeproj, sredstva>
UČEŠĆE <idbr#, brproj#, funkcija,>



Slika 6.1 Baza podataka preduzeća – tabele i veze između njih

Moderne verzije RDBMS-a (Relational Data Base Management System) omogućavaju da se u naredbi CREATE TABLE definišu primarni i strani, spoljnji ključ, kao i da se u definiciju tabele unesu dodatna pravila, koja se odnose na očuvanje integriteta podataka pri ažuriranju baze (pri unosu i brisanju podataka-pravila referencijalnog integriteta). Ona određuju kako se operacije ažuriranja jedne tabele prenose na druge tabele, koje su u nekoj relaciji-vezi sa tabelom koja se ažurira. Moderne verzije RDBMS-a pored naredbe CREATE TABLE imaju i mogućnost kreiranja relacije bez pisanja ove naredbe korišćenjem unaprijed pripremljenih opcija, "čarobnjaka" (WIZARD) i "alata" (TOOLS), ugrađenih u sistem za manipulisanje bazom. Korisnik u tom slučaju mora samo unijeti naziv relacije, broj atributa, naziv atributa i tip podatka koji će poprimiti taj atribut, te da li atribut može poprimiti vrijednost ***Null*** ili ne.

6.1.1 Izmjena vrijednosti ili definicije atributa i dodavanje novog atributa u postojeću tabelu - ALTER TABLE

Dodavanje novog atributa, odnosno nove kolone u tabelu:

ALTER TABLE ime_tabele
ADD (atrib tip [, atrib tip]);

Primjer 1: U tabelu ODJELJENJE dodati atribute šef_od (šef odjeljenja), i br_zap (broj zaposlenih):

ALTER TABLE ODJELJENJE
ADD (*šef_od INTEGER*, *br_zap NUMBER(2)*);

Izmjena definicije postojećeg atributa, tj. postojeće kolone u tabeli:

ALTER TABLE ime_tabele
MODIFY (atrib modifikacija [, atrib modifikacija]);

Primjer 2: U tabeli ODJELJENJE povećati dužinu atributa *br_zap* na 6 cifara.

ALTER TABLE ODJELJENJE
MODIFY (*br_zap NUMBER(6)*);

6.1.2 Izbacivanje relacije iz baze podataka - DROP TABLE

DROP TABLE ime_tabele

Ovom naredbom se izbacuje ne samo definicija tabele već i svi njeni indeksi i podaci koje ona sadrži, za razliku od naredbe **DELETE** koja može obrisati sve podatke iz tabele ali sama tabela ostaje u bazi podataka. Neki sistemi za upravljanje bazama podataka podržavaju i naredbu **DROP DATABASE** *ime_baze*, kojom se izbacuje cela baza. Sistemi orijentisani na jedan fajl, kao što je Microsoft Access, ne podržavaju ovu komandu. To se radi jednostavnim brisanjem fajla (baze podataka) sa diska naredbom za brisanje (Delete), iz operativnog sistema.

6.1.3 Indeksi

Indeksi se koriste za brzi pristup po kolonama koje se indeksiraju i omogućavaju jedinstvenu vrijednost indeksiranih kolona kada te kolone imaju ulogu primarnog ključa, posebno u sistemima kod kojih se naredbom CREATE ne mogu definisati primarni ključevi. Ali, u tabelama mogu postojati jedinstveni indeksi koji nisu primarni ključevi (kada postoji više kandidata za primarni ključ).

CREATE [UNIQUE] INDEX naziv_indeksa
ON ime_tabele (atr [, atrib])

Primjer 3: U tabeli ODJELJENJE kreirati indeks nad artibutom *imeod*.

```
CREATE INDEX naziv_ind
ON ODJELJENJE (imeod);
```

Primjer 4: U tabeli ODJELJENJE kreirati primarni ključ-jedinstveni indeks nad artibutom *brod#*.

```
CREATE UNIQUE INDEX brod_ind
ON ODJELJENJE (brod#);
```

Indeksi se izbacuju naredbom DROP INDEX naziv_indeksa.

Primjer 5: U tabeli ODJELJENJE ukloniti indeks nad atributom *imeod*.

```
DROP INDEX naziv_ind
```

Podaci se iz baza podataka mogu dobiti na dva načina. Prvi, sekvencijalni metod (Sequential Access Metod), zahteva da se pročitaju svi slogovi jedne tabele prilikom pretraživanja. Dakle, čitava datoteka od prvog do zadnjeg zapisa. Ovaj metod je neefikasan, ali jedini mogući da biste bili sigurni u tačnost dobijenog odgovora. Drugi. direktni metod (Direct Access Metod), zahteva da podaci budu indeksirani po određenom atributu i u tom slučaju moguće je direktno pristupiti podatku bez pretraživanja čitave tabele. U tu svrhu se kreira jedna struktura nalik na izokrenuto drvo, takozvano binarno stablo. Na vrhu stabla (u korenu) nalazi se pokazivač na grupe podataka - *čvorove* (nodes). Svaki čvor – roditelj (parent) sadrži najviše dva pokazivača na druge čvorove – decu. Levo se nalaze čvorovi koji imaju vrijednost manju od čvora roditelja a desno su čvorovi sa vrijednošću koja je veća od čvora roditelja.

Jedna od osnovnih operacija pri manipulisanju velikim brojem podataka je pomenuto **pretraživanje,** to jest dobijanje nove informacije na osnovu određenog broja prethodno poznatih vrijednosti nekih atributa. Međutim, ako je broj podataka veliki (stotine miliona, na primjer), onda pretraživanje nije ni najmanje jednostavan i kratak postupak i za najbrže savremene računare. Na primjer, naći neki podatak o građaninu ***XY*** u bazi podataka zemlje sa nekoliko stotina miliona stanovnika mora da potraje, jer računar mora da “pročešlja“ bazu od početka do kraja. A ukoliko je upit logički složen, vrijeme za dobijanje odgovora postaje nedopustivo dugo.

Iz toga razloga se, pri pretraživanju velikih baza podataka, pribjegava njihovom **indeksiranju** po atributu (ili atributima) po kome se vrši pretraživanje. Sve tabele su po pravilu indeksirane po primarnom ključu i tada **Indeksna tabela** (uvijek je izvedena od osnovne), ima samo dva atributa koji definišu drugačije, po nekoj zakonitosti, složene slogove ili n-torke (poređane po primarnom ključu). Inače svaka indeksna tabela ima dva dijela:

- prvi dio-lista atributa, po kojima se vrši pretraživanje (i po kojima se vrši uređivanje indeksa), i
- drugi dio, tzv. **indeks**, koji služi za vezu sa osnovnom tabelom.

Pokažimo postupak kreiranja indeksa nad jednim atributom, koji nije primarni ključ, na jednom jednostavnom primjeru. Pretpostavimo da tabela GRAĐANIN, ima oblik:

GRAĐANIN < matbr#, prezime, ime, datrođ, adresa, .. >

a jedan njen dio se vidi u tabeli 6.1.

Redni broj zapisa - *Redbr* (**Record number**) vodi se u većini programskih paketa za obradu baza podataka za svaku tabelu - relaciju automatski inkrementiranjem nekog brojača (ili interno), i taj broj se najčešće upotrebljava za kreiranje indeksne tabele (nazovimo je INDGRAD) po nekom atributu koji nije ključni, na primjer **prezime**.

U indeksnoj tabeli INDGRAD su podaci (n-torke) složeni drugim redom (u ovom slučaju po abecedi, po prezimenima, ali se to može uraditi i po nekoj drugoj varijabli po veličini ili po datumu) i pored vrijednosti atributa **prezime** indeksirana tabela ima samo još i vrijednost indeksa, to jest broja zapisa u tabeli iz koje je izvedena – dakle GRAĐANIN**.** Nalaženje građanina poznatog prezimena izvodi se sada u dva koraka. Pošto su u indeksiranoj tabeli podaci (prezimena) složeni po poznatom zakonu (abeceda), to se traženi podatak prvo nalazi u njoj u nekoliko koraka (pretraživanjem binarnog stabla pošto se znaju pravila slaganja po abecedi, postupak je isti kao pri traženju rječi u rječniku ili broja u telefonskom imjeniku) bez potrebe da se "češlja" cijela tabela, a onda uz pomoć vrijednosti indeksa-rednog broja zapisa, dolazimo opet u samo jednom koraku (direktnim pristupom) i do ostalih podataka toga građanina u osnovnoj tabeli GRAĐANIN**.**

Naravno, pravljenje indeksnih tabela (od jedne osnovne tabele može se napraviti više indeksnih tabela po raznim atributima, ili po više atributa) podliježe nekim pravilima. Međutim, uvijek mora biti zadovoljen uslov da postoji kriterijum po kome se vrijednosti u indeksnoj tabeli mogu poređati.

Naravno nijesu svi atributi dobri kandidati za indeks. Nije noguće praviti indekse nad kolonama tipa bit-map, text ili slika, a kako je veličina indeksa ograničena, nijesu pogodne (ni dozvoljene) velike kolone tipa CHAR, VARCHAR, BINARY i sl. Za indeks su kandidati kolone po kojima se najčešće vrši pretraživanje, grupisanje, sortiranje i selekcija, a po pravilu su to: spoljnji (strani) ključevi i kolone koje učestvuju u klauzulama GROUP BY i ORDER BY (koje će kasnije biti detaljno objašnjene).

GRAĐANIN

Redbr	matbr#	prez	ime	datrođ	adresa
1	1324764	Antić	Ante	10.07.54	Beograd
2	9763421	Jović	Jovan	24.12.33	Valjevo
3	4513298	Marić	Maks	13.03.76	Bor
4	3344228	Babić	Miro	02.02.77	Uzice
5	3524999	Rodić	Ana	05.10.84	Zemun
6	7623087	Ljujić	Vera	23.11.49	Beograd
7	6653129	Perić	Petar	17.03.11	Trebinje

INDGRAD

index	prez
1	Antić
4	Babić
2	Jović
6	Ljujić
3	Marić
7	Perić
5	Rodić

Tabela 6.1 Tabela GRAĐANIN i njoj pridruženi indeks INDGRAD

Na kraju, pomenimo da indeksiranje ima i svojih nedostataka. Naime, za pretraživanje, zbog čega se ovakve tabele prvenstveno i prave, indeksirane datoteke su izuzetno efikasne, ali se zato prilikom ažuriranja (dodavanje i brisanje podataka) osnovne tabele gubi računarsko vrijeme, jer se indeksne datoteke, svaki put nakon izmjene osnovne tabele, moraju reindeksirati. To je i osnovni razlog zašto se tabele sa malim brojem podataka – zapisa, ne indeksiraju. Sa brzim računarom, i neindeksirana manja tabela može se u veoma kratkom vremenu pretražiti od početka do kraja.

Pored indeksiranja i ciljanim grupisanjem podataka na fizičkom medijumu (disku) efikasnost i brzina rada mogu se bitno povećati. Rješenje se sastoji u tome da se slogovi koji se najčešće uzastopno koriste smjeste na disk na iste, ili susjedne segmente. Tako, ako imamo slogove **S1** i **S2** smještene na isti segment diska **D1** onda će se pri dohvatu **S1** jednovremeno u bufferu računara naći i podaci sloga **S2,** pa ako nam i oni odmah nakon što smo iskoristili podatke sa **S1** zatrebaju, oni su već tu, "pri ruci", u operativnom dijelu memorije i pristup njima je zato brz. Ponekad se svi zapisi na disku smještaju ne po redosledu unosa već uređeni po nekom redoslijedu. To su takozvane sortirane tabele i imaju najbrži pristup podacima, ali se one posle svakog ažuriranja moraju iznova sortirati. Dakle, kao i kod upotrebe indeksa produžava se vrijeme održavanja baze.

6.2 Naredbe za rukovanje podacima

Naredba **SELECT** služi za "dohvatanje" jednog ili više podataka iz jedne, ili iz više tabela. Rezultat ove naredbe je neka informacija a ima opet najčešće strukturu relacije, pa tako ima svoje atribute i vrijednosti atributa, ali kao fizička tabela, stalno prisutna na disku, ne postoji. Dakle, rezultat ovakvog upita je virtuelna neimenovana tabela koja postoji samo u radnoj, operativnoj memoriji.

Međutim i pored toga što ne postoji fizički na disku u formi tabele, rezultat naredbe SELECT može se koristiti kao argument neke druge naredbe SELECT (koja je sastavni dio prve), prilikom pravljenja složenih upita, jer za sve vrijeme izvršavanja naredbe SELECT parcijalni rezultati postoje kao tabele u memoriji računara. Rezultat upita može biti prost neizmjenjen sadržaj jedne ili više tabela, ali isto tako može biti i neka nova vrijednost koja je izračunata na bazi podataka koji postoje u bazi. Prava snaga koncepta baza podataka i SQL-a upravo leži u tome da možemo dobijati i nove, izračunate informacije koje ne postoje kao zapis (podatak) u bazi. Analizom tih novih podataka u interaktivnom radu sa bazom na licu mesta donosimo odluke i kreiramo nove upite sa ciljem dobijanja novih informacija. Rezultat upita je najčešće tabela, relacija (složeni SELECT), a ne samo jedan podatak ili slog (prosti SELECT).

U svim daljim razmatranjima pretpostavićemo da je naredba SELECT tipa složeni SELECT**,** da se koristi u interaktivnom načinu rada, pa

će se u primjerima koji slijede naći i takvi gdje se jednovremeno pristupa podacima iz više tabela. Opšti oblik naredbe SELECT, odnosno upitnog bloka SELECT glasi:

SELECT [ALL DISTINCT] lista atributa 1
FROM lista tabela (relacija)
[**WHERE** lista uslova1]
[**GROUP BY** lista atributa 2]
[**HAVING** lista uslova 2]
[**ORDER BY** lista atributa 3]
UNION [**ALL**]

nakon koje može da slijedi i naredna SELECT naredba u slučaju složenog upita nad jednom ili više relacija. U naredbi SELECT se:

- definišu-selektuju atributi (**SELECT**) čije vrijednosti želimo dobiti (odgovara **operatoru projekcije)**, zatim se
- izdvajaju relacije u kojima se nalaze vrijednosti tih atributa (**FROM**) (odgovara **operatoru spajanja**), onda se sa
- (**WHERE, HAVING**) definišu uslove koje podaci treba da zadovoljavaju pri izdvajanju, što odgovara **operatoru restrikcije** (**selekcije**). Na kraju, mogu se postaviti i zahtjevi kojima se rezultati
- grupišu (**GRUP BY**), ili
- na neki način uređuju (**ORDER BY**).

Prema tome, upitni blok predstavlja kompoziciju operacija projekcije, restrikcije i spajanja, ali njime nije određen redosled u kojem se te operacije obavljaju. Zbog toga je upitni blok opštija, manje proceduralna konstrukcija od ovih operacija relacione algebre.

Odredbe, klauzule SELECT i FROM su obavezne, a ako se ne postavi nikakav uslov za selekciju ili uređivanje, ostale klauzule (WHERE, HAVING, ...) jednostavno se izostavljaju.

Koristeći naredbu SELECT moguće je :

- izdvojiti neke atribute,
- izmjeniti redoslijed atributa (redoslijed atributa u odgovoru isti je kao redoslijed atributa koji je naveden u naredbi SELECT a ne mora biti isti kao redoslijed atributa dat u definiciji tabele).
- spriječiti pojavu višestrukih n-torki.

Značenje opcija **ALL**, odnosno **DISTINCT** je sledeće:

- **ALL** prikazuje (vraća) sve podatke, to jest sve n-torke, koje ispunjavaju postavljeni uslov, tako da se u tom slučaju u rezultatu mogu pojaviti i identične n-torke (pa rezultat onda očito ne mora biti i relacija),
- **DISTINCT** eliminiše sve višestruke n-torke, u rezultatu se nalaze samo one koje su različite (rezultat je prema tome opet relacija).

Ako se ne navede nijedan parametar, podrazumijeva se **ALL,** ali ima verzija SQL-a koje u tom slučaju podrazumijevaju i **DISTINCT** i na to treba obratiti pažnju, jer u suprotnom dobijeni rezultat možda neće biti relacija, što može izazvati greške u daljnjoj obradi.

Na kraju, napomenimo da SQL ne "razumije" znak **#** i $ u imenima atributa kao oznake za prepoznavanje ključnih atributa, i da su ti znaci u primjerima u ovoj knjizi upotrebljeni samo zato da bi se ključni atribut razlikovao od ostalih, i time primjeri bili pregledniji.

Sve operacije i klauzule SQL-a prikazaćemo na primjerima relizovanim na već kreiranoj bazi podataka jednog preduzeća.

RADNIK <idbr#,kvalif, ime, posao, rukovodilac, dat_zap, premija, plata, brod$>
ODJELJENJE <brod#, naziv, mesto>
PROJEKAT <brproj#, imeproj, sredstva>
UČEŠĆE <idbr#, brproj#, funkcija,>

Neka se u tabelama nalaze podaci prikazani na slici 6.2.

Tabele nijesu urađene u potpunosti po pravilima normalizacije, ali to je učinjeno s namjerom da se prikažu određene negativne pojave, anomalije (koje nastaju pri upisu, brisanju i ažuriranju podataka), a takođe i da bi bilo moguće na jednoj bazi prikazati što više naredbi i mogućnosti SQL-a.

6.2.1 Upiti nad jednom tabelom

Najjednostavniju grupu upita čine oni koji prikazuju prost, neizmijenjen sadržaj jedne tabele.

Primjer 6: Prikaži celokupan sadržaj tabele ODJELJENJE.

```
SELECT * FROM ODJELJENJE;
```

Rezultat će biti čitava bazna tabela (slika 6.2. b)) jer znak * je sinonim za traženje svih atributa, a kod nekih RDBMS (na primjer Access), kao i Windows-u i DOS-u, zamenjuje bilo koji niz znakova (džoker znak).

RADNIK : Table

	IDBR#	IME	POSAO	KVALIF	RUKOVODI	DATZAP	PREMIJA	PLATA	BROD$
+	5367	Petar	vozac	KV	5780	01-jan-78	1900	1300	20
+	5497	Aco	radnik	KV	5662	17-feb-90	800	1000	10
+	5519	Vaso	prodavac	VKV	5662	07-nov-91	1300	1200	10
+	5652	Jovan	radnik	KV	5662	31-maj-80	500	1000	10
+	5662	Jovo	upravnik	VSS	5842	12-avg-83		2400	10
+	5696	Miro	radnik	KV	5662	30-sep-91	0	1000	10
+	5780	Bozo	upravnik	VSS	5842	11-avg-84		2200	20
+	5786	Pavle	upravnik	VSS	5842	22-maj-83		2800	30
+	5842	Savo	direktor	VSS		15-dec-81		3000	40
+	5867	Simo	savetnik	VSS	5842	08-avg-70		2750	40
+	5874	Tomo	radnik	KV	5662	19-apr-71	1100	1000	10
+	5898	Andro	nabavljac	KV	5786	20-jan-80	1200	1100	30
+	5900	Slobo	vozac	KV	5780	03-okt-78	1300	900	20
+	5932	Mita	savetnik	VSS	5842	25-mar-65		2600	20
+	5953	Pero	nabavljac	KV	5786	12-jan-79	0	1100	30
+	6234	Marko	analiticar	VSS	5786	17-dec-90	3000	1300	10
+	6789	Janko	rukovodila	VS		23-dec-99	10	3900	40
+	7890	Ivan	analiticar	VSS	5786	17-dec-90	3200	1600	20
	0						0	0	0

a)

ODJELJENJE : Table

	BROD#	IMEOD	MESTO
+	10	komercijala	Novi Beograd
+	20	plan	Dorcol
+	30	prodaja	Stari Grad
+	40	direkcija	Banovo Brdo
+	50	erc	Zemun

b)

PROJEKAT : Table

	BRPROJ#	IMEPROJ	SREDSTVA
+	100	uvoz	3,00E+06
+	200	izvoz	2,00E+06
+	300	plasman	6,00E+06
+	400	projektovanje	5,00E+06
	0		0,00E+00

c)

UCESCE : Table

IDBR#	BRPROJ#	BRSATI	FUNKCIJA
5497	400	2000	IZVRŠILAC
5519	300	2000	IZVRŠILAC
5652	100	1000	IZVRŠILAC
5652	300	1000	IZVRŠILAC
5662	300	2000	ŠEF
5696	200	2000	ŠEF
5696	300	2000	IZVRŠILAC
5780	200	2000	ORGANIZATOR
5786	100	2000	KONSULTANT
5842	100	2000	ŠEF
5867	200	2000	KONSULTANT
5874	300	2000	IZVRŠILAC
5898	200	2000	IZVRŠILAC
5900	100	2000	IZVRŠILAC
5932	100	500	KONSULTANT
5932	200	1000	ORGANIZATOR
5932	300	500	NADZOR
5953	100	1000	IZVRŠILAC
5953	300	1000	IZVRŠILAC
6234	100	500	NADZOR
6234	200	1200	IZVRŠILAC
6234	300	300	KONSULTANT
6789	200	2000	IZVRŠILAC
7890	300	2000	IZVRŠILAC
0	0	0	

d)

Slika 6.2 Baza podataka preduzeća

Primjer 7: Prikaži nazive svih odjeljenja u preduzeću.

```
SELECT imeod
FROM ODJELJENJE;
```

Rezultat upita dat je na slici 6.3.

imeod
komercijala
plan
prodaja
direkcija
erc

Slika 6.3 Nazivi odjeljenja

Primjer 8: Prikaži nazive svih poslova u preduzeću.

```
SELECT posao
FROM RADNIK;
```

Rezultat upita dat je na slici 6.4 a).

Primjer 9: Prikaži nazive svih raznih poslova u preduzeću.

```
SELECT DISTINCT posao
FROM RADNIK;
```

Rezultat upita dat je na slici 6.4 b).

POSAO
vozac
radnik
prodavac
radnik
upravnik
radnik
upravnik
upravnik
direktor
savetnik
radnik
nabavljac
vozac
savetnik
nabavljac
analiticar
rukovodila
analiticar

POSAO
analiticar
direktor
nabavljac
prodavac
radnik
rukovodila
savetnik
upravnik
vozac

a) *b)*

Slika 6.4 Prikaz poslova u preduzeću

Isti efekat se kod nekih DBMS postiže klauzulom jedinstven (**UNIQUE**), ali ovo ne važi kod svih. Recimo, u Accessu nije moguće:

```
SELECT UNIQUE posao
FROM RADNIK;
```

Kako u praksi, u većini slučajeva, postoji potreba za eliminacijom identičnih n-torki, to treba praktično uvijek koristiti oblik SELECT DISTINCT**,** a ne samo SELECT, a mi ćemo u primjerima koji slijede smatrati da je to ponuđena default opcija i nećemo je posebno navoditi**.**

Atributi se u naredbi SELECT mogu navoditi i tako što će im se pridodati i naziv relacije kojoj pripadaju (preporučljivo je uvijek koristiti ovakav način pisanja, a obavezno ga moramo koristiti onda ako, u dvije relacije koje pretražujemo, postoje atributi sa istim imenom).

Naziv relacije se tada može pisati i skraćeno, samo pomoću prvog slova naziva tabele, ali pod uslovom da se prvo slovo u nazivu relacije razlikuje, ako su nazivi atributa identični. Tako su slijedeće naredbe potpuno identične sa naredbom u prethodnom primjeru:

Primjer 10: Prikaži nazive svih raznih poslova u preduzeću.

```
SELECT DISTINCT radnik.posao
FROM RADNIK ;

SELECT DISTINCT r.posao
FROM RADNIK R;
```

Atributima se unutar naredbe SELECT mogu dati i druga imena koja će biti prikazana u rezultatu upita. Ovo je naročito korisno ako se unutar ove naredbe vrše i neke matematičke operacije.

Primjer 11: Prikaži šifre svih odjeljenja u preduzeću.

a) Ova informacija može se dobiti iz tabele RADNIK:

```
SELECT DISTINCT radnik.[brod$] AS "sifre odjeljenja"
FROM RADNIK;
```

Rezultat je dat na slici 6.5 a).

sifre odjeljenja
10
20
30
40

a)

b) Isti podaci mogu se dobiti iz tabele ODJELJENJE:

```
SELECT odjeljenje.[brod#] AS  sifra
FROM ODJELJENJE;
```

Rezultat je dat na slici 6.5 b).

sifre
10
20
30
40
50

b)

Slika 6.5 Šifre odjeljenja

Treba uočiti nekoliko detalja:

- u primjeru a) moramo koristiti klauzulu **DISTINCT** jer u jednom odjeljenju radi više radnika pa bi u odgovoru bilo ponovljeno svako odjeljenje više puta.
- u primjeru b) ne moramo koristiti klauzulu **DISTINCT** jer je šifra odjeljenja primarni ključ relacije i samim tim je jedinstven.
- u primjeru a) moramo iza klauzule **AS** (kao) koristiti znake navoda ili uglaste zagrade [ ] (u Accessu) jer novo ime je sastavljeno od više riječi, a u primjeru b) ne moramo.

Napomjena: U nekim RDBMS nije potrebno navoditi klauzulu AS već se samo stavi razmak, tj. prazno mjesto (space), to je ekvivalentno klauzuli AS (ovo nije slučaj u Accessu). Tako su sledeće naredbe SELECT ekvivalentne prethodnim:

```
SELECT O.[brod#] šifra
FROM ODJELJENJE O;
```

```
SELECT DISTINCT R.[brod$] [šifre odjeljenja]
FROM RADNIK R;
```

Odredba **WHERE**

U svim dosadašnjim primjerima u rezultatu su se pojavljivali svi redovi, jedne tabele. Moguće je primjeniti naredbu SELECT samo na neke n-torke koje zadovoljavaju ili ne zadovoljavaju zadate uslove. U tu svrhu koristimo klauzulu WHERE (*gde je*), i ona nam omogućuje:

- Izdvajanje (selekciju) redova koji zadovoljavaju neki uslov,
- Izdvajanje redova koji zadovoljavaju više uslova (AND),
- Izdvajanje redova koji zadovoljavaju bar jedan od uslova (OR),
- Izdvajanje redova koji zadovoljavaju složene uslove (AND i OR),

- Izdvajanje redova čija je vrijednost unutar nekih granica (BETWEEN),
- Izdvajanje redova čija vrijednost pripada nekoj listi vrijednosti (IN),
- Izdvajanje redova koji ne zadovoljavaju neke uslove (NOT, IS NOT),
- Izdvajanje redova ako neka vrijednost postoji (EXISTS) itd.

Sintaksa pisanja uslova **(WHERE),** pomoću izraza upoređivanja, u opštem slučaju je slijedeća:

argument1 ***operator upoređivanja*** **argument2**

a argumenti mogu biti:

- jedan atribut,
- skup atributa (u zagradama odvojeni zapetom), ili
- naredba **SELECT**

koja vraća najviše jednu n-torku, dok operator upoređivanja može biti bilo koji od operatora:

- **veći (>),**
- **manji (<),**
- **veći ili jednak (>=),**
- **manji ili jednak (>=),**
- **jednak (=),** i
- **različit (< > ili ≠).**

Ako se uporjeđuju argumenti koji se sastoje od više atributa, tada oba argumenta moraju imati jednak broj atributa, a uporjeđuje se prvi sa prvim, drugi sa drugim itd. Konačno, napomjenimo da atributi koji se uporjeđuju moraju biti istog, ili kompatibilnog tipa podataka.

Prilikom postavljanja uslova mogu se koristiti sljedeće opcije:

- **BETWEEN**
- **IN**
- **EXISTS.**

Ne ulazeći u sve mogućnosti navedenih opcija, pokažimo na primjerima kako se one koriste.

Primjer 12: Prikaži kvalifikaciju, platu i ime zaposlenih u odjeljenju 30.

```
SELECT R.kvalif, R.plata, R.ime, R.[brod$]
FROM RADNIK R
WHERE R.[brod$]=30;
```

kvalif	plata	ime	brod$
VSS	2800	Pavle	30
KV	1100	Andro	30
KV	1100	Pero	30

Slika 6.6 Podaci o zaposlenima u odjeljenju 30

Napomjena: Uočimo da redosled atributa u rezultatu odgovara redosledu atributa u naredbi SELECT a ne redosledu atributa u samoj fizičkoj tabeli.

Primjer 13: Prikaži ime, posao i platu zaposlenih u odjeljenju 30 čija je plata veća od 2000.

```
SELECT R.ime, R.posao, R.plata, R.[brod$]
FROM RADNIK R
WHERE R.[brod$]=30 AND plata>2000;
```

IME	POSAO	PLATA	BROD$
Pavle	upravnik	2800	30
		0	0

Slika 6.7 Podaci o radnicima u odjeljenju 30 sa platom većom od 2000

Primjer 14: Prikaži ime, posao upravnika i direktora.

```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE (R.posao="direktor") OR
        (R.posao="upravnik");
```

IME	POSAO	BROD$
Jovo	upravnik	10
Bozo	upravnik	20
Pavle	upravnik	30
Savo	direktor	40

Slika 6.8 Podaci o direktoru i upravnicima

Napomjena: Isti rezultat se dobija korišćenjem klauzule **IN** (u prevodu "u") čiji opšti oblik glasi: **atribut [NOT] IN** (lista skalarnih vrijednosti).

Primjer 15: Prikaži ime i posao upravnika i direktora.

```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
      WHERE R.posao IN ("direktor", "upravnik");
```

IME	POSAO	BROD$
Jovo	upravnik	10
Bozo	upravnik	20
Pavle	upravnik	30
Savo	direktor	40

Slika 6.9 Podaci o direktoru i upravnicima

Napomjena: Poželjno je da redoslijed ispitivanja uslova regulišete upotrebom zagrada, jer u protivnom možete dobiti potpuno neočekivane i vjerovatno netačne odgovore na upit jer uslovi nisu interpretirani na odgovarajući način.

Primjer 16: Prikaži ime i posao upravnika i analitičara iz odjeljenja 10.

a)
```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
  WHERE R.posao="upravnik" OR
  R.posao="analiticar" AND R.[brod$]=10;
```

IME	POSAO	BROD$
Jovo	upravnik	10
Bozo	upravnik	20
Pavle	upravnik	30
Marko	analiticar	10
		0

a) Netačan odgovor

b)
```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
  WHERE (R.posao="upravnik" OR
  R.posao="analiticar") AND R.[brod$]=10;
```

ime	posao	brod$
Jovo	upravnik	10
Marko	analiticar	10
		0

b) Tačan odgovor

Slika 6.10 Upravnici i analitičari iz odjeljenja 10

Primjer 17: Prikaži ime i platu zaposlenih čija je plata od 2600 do 3000.

```
SELECT R.ime, R.plata
FROM RADNIK R
WHERE R.plata >= 2600 AND R.plata <=3000;
SELECT ime, plata
FROM RADNIK
WHERE plata BETWEEN  2600 AND 3000;
```

ime	plata
Pavle	2800
Simo	2750
Savo	3000
Mita	2600

Slika 6.11 Zaposleni sa platama od 2600 do 3000

Sintaksa opcije **BETWEEN** (u prevodu "između") ima opšti oblik:

argument1 [NOT] BETWEEN argument2 AND argument3

a rezultat ovog testa je istinit ako se vrijednost za **argument2** nalazi (ili ne nalazi - **NOT**) između vrijednosti **argument1** i **argument3** uključujući i granice toga intervala.

Opcija **EXISTS** (u slobodnom prevodu – "postoji") koristi se za provjeru postojanja najmanje jedne n-torke u rezultatu pretraživanja. Sintaksa ove naredbe glasi:

EXISTS (relacijski izraz).

Odgovor je istinit (**TRUE**) ako relacijski izraz daje barem jednu n-torku kao rezultat, u suprotnom je neistinit (**FALSE**). Pored ove pomenute tri opcije šire verzije SQL-a imaju još neke mogućnosti kao na primjer:

- **LIKE,**
- **MATCH,**
- **ALL- OR – ANY i**
- **UNIQUE,**

a njihovo značenje se može naći u kompletnim priručnicima (manualima) za korišćenje programskog paketa **SQL-a**.

Odredba **ORDER BY**

Odredba **ORDER BY** slaže n-torke po nekom redoslijedu (po abecedi, po veličini itd.) u rastućem ili opadajućem poretku. Odredba ORDER BY je uvijek poslednja klauzula u SELECT bloku, jer najpre se selektuju n-torke a na kraju se uređuju. I ova opcija može imati više atributa po kojima se vrši ređanje.

Primjer 18: Prikaži ime i kvalifikaciju zaposlenih čija imena počinju slovom **p**.

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE  'p%';
```

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE  'p%'
ORDER BY R.kvalif;
```

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE  'p%'
ORDER BY R.ime;
```

IME	KVALIF
Petar	KV
Pavle	VSS
Pero	KV

a) neuređen odgovor

IME	KVALIF
Pero	KV
Petar	KV
Pavle	VSS

b) uređen po kvalifikaciji

IME	KVALIF
Pavle	VSS
Pero	KV
Petar	KV

c) uređen po imenima

*Slika 6.12 Spisak zaposlenih čija imena počinju slovom **p***

Napomjena: U raznim verzijama SQL-a postoje džoker znaci, tj. znaci koji mogu zamijeniti bilo koji niz znakova **%** (a u Access-u je to znak ***) ili jedan znak **_** (u Access-u je to znak **?**).

Navedimo još nekoliko primjera kako bi izlistali imena koja zadovoljavaju neki od uslova:

- ime se završava slovom **a** — **WHERE** ime **LIKE** '%a' ;
- treće slovo imena je **V** — **WHERE** ime **LIKE** '_ _v%' ;
- treće slovo imena je **v** — **WHERE** ime **LIKE** '??v*' ; (u Access-u)
- u imenu nema slovo **N** — **WHERE** ime **NOT LIKE** '%n%' ;
- ime je dužine 6 slova — **WHERE** ime **LIKE** '_ _ _ _ _ ' ;
- ime nije dužine 5 slova — **WHERE** ime **NOT LIKE** '_ _ _ _ _' ;
- u imenu je slovo **l** isped slova **N** — **WHERE** ime **LIKE** '%l %N %' ;
- broj ima **dve cifre** crticu **cifru** — **WHERE** tel **LIKE** '[0-9][0-9]-[0-9]' ;

Primjer 19: Prikaži ime, kvalifikaciju, platu i premiju uređenu:

a) po kvalifikaciji u opadajućem, po plati u rastućem a po premiji u opadajućem redoslijedu,

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY kvalif DESC , plata, premija DESC;
```

b) po plati u rastućem, a po kvalifikaciji i premiji u opadajućem redosledu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY  plata, kvalif DESC, premija DESC;
```

c) po premiji i kvalifikaciji u opadajućem, a po plati u rastućem redosledu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY premija DESC, kvalif DESC, plata;
```

Napomjena: Redoslijed sortiranja odgovara redoslijedu navođenja atributa u klauzuli **ORDER BY**. Najprije se sortira izvještaj po prvom navedenom atributu, zatim po drugom, itd. Sortiranje može biti izvršeno u rastućem redoslijedu – **ASC** (**Ascedent**) i to je pretpostavljena (**default**) vrijednost, pa se ne mora navoditi.

Ako želimo da odgovor bude uređen u opadajućem redoslijedu (**Descedent**) po vrijednosti nekog atributa moramo iza imena navesti reč **DESC**. Ako postoji više n-torki sa jednakim vrijednostima atributa njihov redoslijed je onda proizvoljan. Ovakav rezultat se može izbjeći sortiranjem po atributu koji je primarni ključ, gdje su dvije iste vrijednosti atributa isključene. Ako se sortiranje vrši po nekom atributu koji ima i vrijednost ***Null*** (kao što bi mogao biti atribut *premija* u posljednjem primjeru) onda se sve vrijednost ***Null*** grupišu zajedno.

IME	KVALIF	PLATA	PREMIJA
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	
Janko	VS	3900	10
Vaso	VKV	1200	1300
Slobo	KV	900	1300
Tomo	KV	1000	1100
Aco	KV	1000	800
Jovan	KV	1000	500
Miro	KV	1000	0
Andro	KV	1100	1200
Pero	KV	1100	0
Petar	KV	1300	1900

a)

ime	kvalif	plata	premija
Slobo	KV	900	1300
Tomo	KV	1000	1100
Aco	KV	1000	800
Jovan	KV	1000	500
Miro	KV	1000	0
Andro	KV	1100	1200
Pero	KV	1100	0
Vaso	VKV	1200	1300
Marko	VSS	1300	3000
Petar	KV	1300	1900
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	
Janko	VS	3900	10

b)

ime	kvalif	plata	premija
Ivan	VSS	1600	3200
Marko	VSS	1300	3000
Petar	KV	1300	1900
Vaso	VKV	1200	1300
Slobo	KV	900	1300
Andro	KV	1100	1200
Tomo	KV	1000	1100
Aco	KV	1000	800
Jovan	KV	1000	500
Janko	VS	3900	10
Miro	KV	1000	0
Pero	KV	1100	0
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	

c)

Slika 6.13 Uređivanje redoslijeda n-torki

Upotreba **NULL** vrijednosti

Jedna od najvećih novina koje su donele relacione baze podataka jeste mogućnost prikazivanja nepostojećeg podatka čija vrijednost je nedefinisana. To su **Null** vrijednosti, i one ponekad moraju postojati.

1. Kada u bazi postoje atributi za koje su Null vrijednosti “normalne” jer to svojstvo nije primjenljivo na sve primjerke nekog entiteta. Takav je na primjer atribut *premija* u tabeli RADNIK. Ako to svojstvo nije primjenljivo na većinu primjeraka entiteta onda taj atribut treba eliminisati iz tabele još u fazi projektovanja. U slučaju da je taj podatak vrlo važan za one koji to svojstvo imaju, onda se kreira nova tabela samo za one objekte koji to svojstvo posjeduju (u ovom slučaju može se napraviti tabela PREMIJA<*idbr#*, *premija*>).

2. Ako vrijednost nekog atributa za neke objekte još nije poznata ili nije dozvoljena (neki radnici još uvijek nemaju telefon, rukovodioca, ili nisu raspoređeni ni u jedno odjeljenje).

3. Kada nije nastupio momenat djelovanja nekog atributa (plata ili premija za mart poznati su tek tokom aprila).

Za testiranje vrijednosti kolona koje sadrže Null vrijednosti na raspolaganju su samo dvije opcije **IS NULL** i **IS NOT NULL**, a moguće je koristiti i operatore logičkog poređenja.

Treba imati u vidu šta će biti rezultat operacije pri upotrebi operatora logičkog poređenja = i ≠ (tabele 1 i 2) i operatora **IS NULL** i **IS NOT NULL** (tabela 3) .

=	True	False	Null
True	True	False	Null
False	False	True	Null
Null	Null	Null	Null

Tabela 1.

≠	True	False	Null
True	False	True	Null
False	True	False	Null
Null	Null	Null	Null

Tabela 2.

	Is Null	Is Not Null
<value>	False	True
True	False	True
False	False	True
Null	True	False

Tabela 3.

Primjer 20: Prikaži ime, kvalifikaciju, platu i premiju zaposlenih koji:

a) imaju premiju.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
WHERE premija IS NOT NULL;
```

ime	kvalif	plata	premija
Petar	KV	1300	1900
Aco	KV	1000	800
Slobo	KV	900	1300
Vaso	VKV	1200	1300
Miro	KV	1000	0
Tomo	KV	1000	1100
Andro	KV	1100	1200
Pero	KV	1100	0
Jovan	KV	1000	500
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Janko	VS	3900	10

a)

b) nemaju premiju.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
WHERE premija IS NULL;
```

ime	kvalif	plata	premija
Jovo	VSS	2400	
Bozo	VSS	2200	
Pavle	VSS	2800	
Simo	VSS	2750	
Savo	VSS	3000	
Mita	VSS	2600	

b)

Slika 6.14 Rad sa nedefinisanim vrijednostima

Napomjena: Ovde treba uočiti razliku između objekata koji nemaju premiju (imaju Null vrijednost) i objekata koji imaju premiju a vrijednost premije može biti i 0 (radnici Pero i Miro).

Odredba **GROUP BY**

Odredba **GROUP BY,** koju treba da slijedi lista atributa, koristi se za grupisanje n-torki na osnovu nekog kriterijuma. Naime, naredba **SELECT** kao rezultat daje opet relaciju, pa n-torke nisu složene ni po kakvom redu, jer to po definiciji relacije nije ni potrebno.

Naredbom **GROUP BY** n-torke u relaciji bivaju presložene na način da sve n-torke unutar grupe imaju jednake vrijednosti atributa po kojima se grupišu. Ako se navede više atributa po kojima treba vršiti grupisanje, prvo se ređaju n-torke sa jednakom vrijednošću prvog atributa, zatim se unutar tih grupa preslažu n-torke prema vrijednostima drugog atributa, itd.

Primjer 21: Prikaži ime, kvalifikaciju, platu i premiju:

a) grupisanu po kvalifikaciji, po plati i po premiji,

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
GROUP BY kvalif, plata, premija;
```

b) uređenu po kvalifikaciji, plati i premiji u rastućem redosledu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY kvalif, plata, premija;
```

ime	kvalif	plata	premija
Slobo	KV	900	1300
Miro	KV	1000	0
Jovan	KV	1000	500
Aco	KV	1000	800
Tomo	KV	1000	1100
Pero	KV	1100	0
Andro	KV	1100	1200
Petar	KV	1300	1900
Vaso	VKV	1200	1300
Janko	VS	3900	10
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	

a) grupisanje podataka

ime	kvalif	plata	premija
Slobo	KV	900	1300
Miro	KV	1000	0
Jovan	KV	1000	500
Aco	KV	1000	800
Tomo	KV	1000	1100
Pero	KV	1100	0
Andro	KV	1100	1200
Petar	KV	1300	1900
Vaso	VKV	1200	1300
Janko	VS	3900	10
Marko	VSS	1300	3000
Ivan	VSS	1600	3200
Bozo	VSS	2200	
Jovo	VSS	2400	
Mita	VSS	2600	
Simo	VSS	2750	
Pavle	VSS	2800	
Savo	VSS	3000	

b) sortiranje podataka

Slika 6.15 Sličnost dejstva klauzula GROUP BY i ORDER BY

Napomjena: Ova sličnost je samo prividna a prava uloga klauzule GROUP BY biće opisana pri upotrebi zbirnih (agregatnih) funkcija gde se određene funkcije (na primjer srednja vrijednost) primjenjuju na grupe podataka sa nekim zajedničkim svojstvom (po odjeljenjima).

Napomjena: Atributi po kojima se vrši grupisanje moraju biti navedeni i u SELECT naredbi, a u nekim RDBMS, kao recimo u Access-u, mora se vršiti grupisanje po svim atributima navedenim u naredbi SELECT, dakle u prethodnom primjeru grupisanje se mora izvršiti i po imenu iako to nismo željeli.

6.2.2 Upiti nad jednom tabelom sa izračunavanjem novih vrijednosti

Pretraživanje neke baze podataka u svrhu dobijanja novih i relevantnih informacija je svakako najinteresantniji oblik primjene analize baze podataka. Upravo u pretraživanju je SQL pokazao svoja preimućstva nad ostalim paketima. SQL ima ugrađen veliki broj gotovih funkcija za *dobijanje zbirnih informacija* (AVG, SUM, MIN, MAX i manje poznate COUNT), za *obavljanje aritmetičkih operacija i uobličavanje rezultata* (POWER, ROUND, TRUNC, ABS, SIGN, MOD, SQRT) kao i za *rad sa tekstom*, tj. *sa nizovima karaktera* (LENGHT, SUBSTR, INSTR,

UPPER, LOWER, TO_NUM, TO_CHAR, NVL, DECODE itd.). Pri pretraživanju baze podataka, u naredbi SELECT, mogu se kombinovati funkcije i aritmetičke operacije nad pojedinim atributima i grupisati njihove vrijednosti po nekom kriterijumu (atributu).

Primjer 22: Prikaži najmanju, najveću, srednju platu i broj zaposlenih:

a) u cjelom preduzeću,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
       AVG(plata) AS srednja, COUNT(*) AS  broj
FROM RADNIK;
```

b) u cjelom preduzeću, sa zaokrugljivanjem na dve decimale,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
       ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS  broj
FROM RADNIK;
```

c) po odjeljenjima,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
       ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS  broj
FROM RADNIK
GROUP BY brod$;
```

d) u odjeljenju 10,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca, ROUND(AVG(plata), 2) AS srednja,
       COUNT(*) AS  broj
FROM RADNIK
WHERE brod$=10;
```

najmanja	najveca	srednja	broj
900	3900	1786,1111	18

a)

najmanja	najveca	srednja	broj
900	3900	1786,11	18

b)

najmanja	najveca	srednja	broj	brod$
1000	2400	1271,43	7	10
900	2600	1720	5	20
1100	2800	1666,67	3	30
2750	3900	3216,67	3	40

c)

najmanja	najveca	srednja	broj
1000	2400	1271,43	7

d)

Slika 6.16 Sumarne funkcije i GROUP BY klauzula

Napomjena: U primjeru a) srednja vrijednost je izračunata sa velikim brojem decimala, pa je u ostalim primerima korišćena funkcija za zaokruživanje (ROUND(art, n)) na dvije decimale.

Napomjena: U primjeru c) agregatne funkcije su primjenjene na grupe podataka, po odjeljenjima. Naime, GROUP BY omogućava dobijanje sumarne informacije za svaku različitu vrijednost kolone po kojoj se vrši grupisanje.

Napomjena: U primjeru d) prikazana je mogućnost selektivnog grupisanja na bazi WHERE klauzule. Dakle, GROUP BY klauzula zamjenjuje višestruko pisanje SELECT naredbe sa različitim uslovima.

Grupisanje se može vršiti po više kolona, i tada svaka različita kombinacija kolona predstavlja jednu grupu. U okviru dobijenih grupa mogu se uvoditi dodatni uslovi za selekciju primjenom klauzule **HAVING** (*koji imaju*). SQL naredbe mogu sadržati aritmetičke izraze sastavljene od funkcija, imena kolona i konstanti povezanih aritmetičkim operatorima (+, -, * i /). Kada u izrazima treba koristiti vrijednosti kolone koja može imati Null vrijednosti onda treba koloni dodjeliti neutralnu vrijednost za željenu operaciju, na primjer pri sabiranju neutralna vrijednost je 0. Dodjela neke vrjednosti koloni koja sadrži Null vrijednosti vrši se funkcijom **NVL(atrib, vrijednost)**. Tako se vrjednost **0** u koloni **premija** dodjeljuje zaposlenima koji nemaju premiju funkcijom **NVL**(*premija*,0). U Access-u se u tu svrhu koristi funkcija NZ(*premija*).

Primjer 23: Izračunaj broj zaposlenih koji obavljaju različite poslove unutar svakog odjeljenja.

```
SELECT brod$, posao, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK
GROUP BY brod$, posao;
```

Primjer 24: Prikaži koje poslove obavlja više od 1 radnika unutar svakog odjeljenja.

```
SELECT brod$, posao, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK
GROUP BY brod$, posao
HAVING COUNT>1;
```

brod$	posao	broj zaposleni
10	analiticar	1
10	prodavac	1
10	radnik	4
10	upravnik	1
20	analiticar	1
20	savetnik	1
20	upravnik	1
20	vozac	2
30	nabavljac	2
30	upravnik	1
40	direktor	1
40	rukovodila	1
40	savetnik	1

a) razni poslovi po odjeljenjima

brod$	posao	broj zaposleni
10	radnik	4
20	vozac	2
30	nabavljac	2

b) više od jednog radnika obavlja posao

Slika 6.17 Grupisanje po više kolona i dodatna selekcija n-torki koje zadovoljavaju uslov

Klauzule GROUP BY i WHERE mogu se koristiti zajedno, pri tome *GROUP BY mora uvek biti iza WHERE klauzule*, jer najpre treba izvršiti

selekciju (smanjiti broj n-torki), a zatim se one grupišu sa GROUP BY, a onda se dodatno izabiraju grupe klauzulom HAVING. I ključnu riječ HAVING mora slijediti lista uslova, ali za razliku od WHERE sada se iz rezultata **eliminišu sve one n-torke koje ne zadovoljavaju uslove** navedene u listi. Po pravilu ova operacija izvodi se nad grupom podataka. Ako nije definisana grupa (opcijom GROUP BY) smatra se da je dobijeni rezultat jedna grupa.

Primjer 25: Odrediti srednju godišnju platu unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

```
SELECT brod$, AVG(plata)*12 AS  [prosek plata]
FROM RADNIK
WHERE posao NOT IN (‘direktor’, ‘upravnik’)
GROUP BY brod$;
```

brod$	prosek plata
10	13000
20	19200
30	13200
40	39900

Slika 6.18 Prosječne plate

Primjer 26: Odrediti srednja godišnja primanja unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

a)
```
SELECT brod$, AVG(plata +NVL(premija,0))*12 AS  [prosek primanja], COUNT(*) AS
       [broj zaposlenih], SUM(plata + NVL(premija,0))*12 AS [ukupni prihod]
FROM RADNIK
WHERE posao NOT IN (‘direktor’, ‘upravnik’)
GROUP BY brod$;
```

b)
```
SELECT brod$, AVG(plata +premija)*12 AS  [prosek primanja], COUNT(*)AS  [broj zaposlenih],
   SUM(plata + (premija)*12 AS [ukupni prihod], COUNT(premija) AS [sa premijom]
FROM RADNIK
WHERE posao NOT IN (‘direktor’, ‘upravnik’)
GROUP BY brod$;
```

brod$	prosek primanja	broj zapos	ukupni prihod
10	26400	6	158400
20	38400	4	153600
30	20400	2	40800
40	39960	2	79920

a) tačan rezultat

brod$	prosek primanja	broj zapos	ukupni prihod	sa premijom
10	26400	6	158400	6
20	40800	4	122400	3
30	20400	2	40800	2
40	46920	2	46920	1

b) netačan rezultat

Slika 6.19 Prosječna primanja po odjeljenjima

U primjeru 26. a) proizvod prosječnih primanja i broja zaposlenih jednak je ukupnom prihodu, dok to nije slučaj u primjeru b) kada je pri izračunavanju prosjeka uzet u obzir samo broj zaposlenih koji imaju premiju (posljednja kolona u izvještaju). Tačnije, pri izračunavanju prosječnih primanja ukupni prihod dijeljen je samo sa brojem onih koji imaju premiju, a ne sa ukupnim brojem zaposlenih u odjeljenju ne uzimajući u obzir direktora i upravnike.

U prethodnim primerima korišćeni specijalni relacioni operatori koji su opšteprihvaćeni: izračunavanje zbirnih podataka, proširivanje i preimjenovanje. Sem ovih neki isporučioci sistema za upravljanje bazama podataka nude i druge dopune. Tako na primjer, Microsoft je autor tri dopune: transform, rollup i cube.

Izračunavanje zbirnih podataka (engl. summarize operator) formira zapise koji sadrže zbirne podatke grupisane na osnovu zadatih polja. Po jedan zapis-red za svaku različitu vrednost grupe polja. Ako je navedeno više od jednog polja za grupisanje, grupe se ugnježduju. Polja navedena u listi polja u iskazu SELECT moraju biti takođe navedena i u listi polja za grupisanje (u iskazu GROUP BY) ili kao argument neke od zbirnih funkcija. Zbirne funkcije navedene u iskazu SELECT ne moraju biti u listi polja za grupisanje u iskazu GROUP BY.

U zbirnim funkcija Null vrednosti se uzimaju u obzir i čine grupu, ali ih zbirne funkcije zanemaruju. Kao posledica se javljaju netačni i neočekivani rezultati (primjer 26). Taj problem se javlja obično ako se jedan atribut u listi polja za grupisanje navede kao argument neke zbirne funkcije.

Zbirni podaci su vrlo korisni kada treba proučavati podatke na višem nivou apstrakcije od onog koji se čuva u bazi podataka.

Proširenje (engl. extend operator) je operator koji omogućava da se definišu i proračunaju nova, virtuelna polja koja se izračunavaju na osnovu vrednosti uskladištenih u bazi podataka. Ovi proračuni mogu biti proizvoljne složenosti a kombinuju se konstante i imena atrubuta pomoću aritmetičkih operacija i funkcija (primjeri 25 i 26).

Preimjenovanje (engl. rename operator) omogućava da se neki atributi, virtuelna polja dobijena primenom operatora proširenja ili tabele nazovu drugim imenom. Preimenovanje povećava razumljivost dobijenih rezultata (primjeri 22-26). Posebno je korisno kada treba ostvariti spajanje tabele sa samom sobom (samospoj, engl. self join, primjer 34). Ovo omogućava da se svaka upotreba iste tabele definiše kao logički zasebna.

Transform je dopuna koja je postojala samo kod MS Access-u. Transform preuzima rezultate zbirne operacije (GROUP BY) i prikazuje ih rotirane za 90^0. Ova opcija je poznata kao unakrsni upit (engl. crosstab query) i postoji u najnovijoj verziji SQL Server-a. Opšti oblik bloka TRANSFORM je:

TRANSFORM zbirna funkcija
SELECT lista atributa
FROM lista tabela
WHERE lista uslova1
GROUP BY lista atributa za grupisanje
PIVOT zaglavlja atributa 1 [**IN** lista vrednosti]

Klauzula *TRANSFORM zbirna funkcija* definiše zbirne podatke od kojih će se sastojati rezultujući skup zapisa. SELECT blok mora da sadrži klauzulu GROUP BY a ne može da sadrži klauzulu HAVING. Lista atributa u odredbi SELECT i lista atributa za grupisanje u odredbi GROUP BY po pravilu su identične.

Odredba *PIVOT* zadaje polje, odnosno atribut koji će se pojavljivati u zaglavlju kolona u tabeli rezultata. Neobavezna odredba **IN** omogućava da se u rezultujućoj tabeli zadaju imena kolona i redosled pojavljivanja kakav je zadat u listi vrednosti. Bez ove odredbe u rezultujućoj tabeli kolone će biti date abecednim redosledom sleva nadesno.

Rollup operator postoji samo u SQL Server-u a omogućava da se na bazi parcijalnih zbirnih vrednosti izračunatih na osnovu GROUP BY izračunaju ukupni zbirovi. Ovaj operator se dodaje kao proširenje odredbe GROUP BY:

GROUP BY lista atributa za grupisanje WITH ROLLUP.

Cube operator je takođe proširenje odredbe GROUP BY, i omogućava da se izračunavaju zbirni podaci pri svakoj promeni vrednosti u svakoj koloni navedenoj u listi za grupisanje. Slično kao i *rollup* i *cube* nalaže da se izračunaju zbirni podaci za dodatne grupe podgrupa iz liste za grupisanje.

6.2.3 Upiti nad jednom relacijom kao argument u upitu nad drugom relacijom – ugnježdeni upiti

Zajednička karakteristika svih do sada prezentiranih primjera bila je **postavljanje upita uvijek samo nad jednom relacijom,** jer se odgovor uvijek mogao naći unutar jedne relacije. Manipulisanje informacionim sistemom zahtijeva međutim često dobijanje odgovora za koje upit treba postaviti nad dvije ili više relacija istovremeno. U tom slučaju moraju se koristiti ugnježdeni upiti ili se mora poduzeti operacija spajanja dobijenih tabela (najčešće prirodnog spajanja), sa nekim projekcijama i/ili restrikcijama nad njima.

Pogledajmo u primjerima koji slijede tehniku postavljanja i ovakvih upita.

Pretpostavimo, na primjer, da nam je potrebna sledeća informacija: spisak imena svih zaposlenih u odjeljenju koje je locirano na Dorćolu. To se postiže ulaganjem rezultata jednog upita u WHERE klauzulu drugog upita.

Prvi, unutrašnji upit, bi trebao iz relacije ODJELJENJE dati odgovor koja je šifra odjeljenja (brod#) koje je locirano na *Dorćolu*.

Kada dobijemo da je to odjeljenje čija je šifra brod#=*20*, onda drugim, spoljnim upitom, iz relacije RADNIK tražimo spisak imena zaposlenih u odjeljenju gdje je brod$=*20*.

Primjer 27: Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

I)
```
SELECT brod#
FROM ODJELJENJE
WHERE mesto='Dorcol';
```

BROD#
20

Odgovor na I) upit

II)
```
SELECT ime, brod$
FROM RADNIK
WHERE brod$=20;
```

ime	brod$
Petar	20
Slobo	20
Bozo	20
Mita	20
Ivan	20

Odgovor na II) upit

Slika 6.20 Spisak zaposlenih na Dorćolu

Ulaganje odgovora jednog-unutrašnjeg upita kao vrednost u WHERE klauzulu drugog-spoljnjeg upita:

III)
```
SELECT ime, brod$
FROM RADNIK
WHERE RADNIK.[BROD$]  = (SELECT ODJELJENJE.[BROD#]
                         FROM ODJELJENJE
                         WHERE ODJELJENJE.mesto='Dorcol');
```

ime	brod$
Petar	20
Slobo	20
Bozo	20
Mita	20
Ivan	20

Slika 6.21 Spisak zaposlenih na Dorćolu

Primjer 28: Izlistaj ime, posao i platu zaposlenih u odjeljenju *10* koji imaju isti posao kao zaposleni u odjeljenju *plan*.

```
SELECT ime, posao, plata
FROM RADNIK
WHERE  brod$ = 10  AND  Posao  IN  (SELECT posao
                                    FROM RADNIK
                                    WHERE brod$ IN (SELECT brod#
                                                    FROM ODJELJENJE
                                                    WHERE imeod='plan'));
```

poslovi u planu i komercijali : Select ...

	ime	posao	plata
	Jovo	upravnik	2400
	Marko	analiticar	1300
▶			0

Record: 3 of 3

Slika 6.22 Zaposleni u odjeljenju plan koji rade iste poslove koji postoje u komercijali

Primjer 29: Ko su najbolje plaćeni radnici u svakom odjeljenju.

```
SELECT ime, brod$, posao, plata
```

```
FROM RADNIK
WHERE (brod$, plata) IN ( SELECT brod$, MAX(plata)
                          FROM RADNIK
                          GROUP BY brod$ );
```

Napomjena: Spoljašnji i unutrašnji upit mogu biti povezani po vrijednostima više atributa. U tom slučaju ako se uporjeđuju argumenti koji se sastoje od više atributa, oba argumenta moraju imati jednak broj atributa, a uporjeđuje se prvi sa prvim, drugi sa drugim itd. Konačno, napomjenimo da atributi koji se upoređuju moraju biti istog, ili kompatibilnog tipa podataka.

Kao što smo to već napomjenuli, *imena atributa mogu se pisati navođenjem i imena relacije kojoj taj atribut pripada, a ovo je neophodno uraditi ako se odlučimo da u raznim relacijama jedne baze podataka koristimo ista imena atributa*.

Složenija pretraživanja, kao što smo to vidjeli u prethodnim primjerima, koriste obavezno takozvana "potpretraživanja" to jest, pretraživanja rezultata prethodnog pretraživanja. Pre nego što započne izvršavanje spoljnjeg upita, unutrašnji upit je već izvršen i njegov rezultat je poznat i već su konkretne vrijednosti smještene u memoriju računara. Naredba SELECT daje, naime, kao rezultat virtuelnu relaciju (koja ne egzistira i fizički, na disku, ali je dostupna za vrijeme izvođenja te naredbe u vidu relacije u memoriji računara) koja se može dalje pretraživati. Ovakav pristup je moguć samo u slučaju da se u konačnom izvještaju pojavljuju samo podaci iz jedne relacije, kao što je u svim prethodnim primjerima bio slučaj. Dakle, povezivanje tabela dinamičkom zamjenom rezultata jednog, unutrašnjeg upita u WHERE klauzulu drugog, spoljnjeg upita, može se primijeniti samo ako su svi podaci koji se prikazuju u spoljnjem upitu iz jedne tabele.

Ali u nekim slučajevima u rezultatu spoljnjeg upita kombinuju se podaci iz više tabela. U tom slučaju mora se izvršiti spajanje (**JOIN**) dveju ili više tabela. Spajanje tabela vrši se korišćenjem zajedničkih atributa, tj. atributa koji su definisani nad istim domenima.

6.2.4 Spajanje relacija

Spajanje relacija (engl. Join) je operacija kombinovanja podataka iz više relacija koje su nastale u procesu projektovanja baze kao rezultat razdvajanja podataka iz jedne relacije u više relacija sa ciljem eliminisanja redundanse i anomalija pri upisu, brisanju i ažuriranju. Spojevi između relacija razvrstavaju se prema načinu poređenja kolona između kojih je uspostavljena veza i načinu na koji se tumače rezultati poređenja. Formalno gledano, sve vrste spajanja mogu se ostvariti pomoću odgovarajućeg uslova u odredbi WHERE. Vrlo često je to i bolje rešenje jer tada mašina

baze podataka lakše optimizuje izvršenje iskaza. Uopšteno gledano spojevi se dijele u dvije grupe:

- Spojevi koji izdvajaju samo zapise za koje je uslov spajanja ispunjen (tačan, True). To su unutrašnji spojevi (engl. Inner Joins).
- Spojevi koji izdvajaju sve zapise kao unutrašnji spojevi plus preostale iz jednog, drugog ili oba skupa. To su takozvani spoljni spojevi (eng. Outer Joins).

Unutrašnji spojevi mogu biti ostvareni po jednakosti (engl. equi join) ili nejednakosti bilo koje vrste (engl. theta join).

Spajanje po jednakosti (**EQUIJOIN**)

Najčešće se koristi spajanje po jednakosti ili jednakovredni spojevi. Pri tome izdvajaju se samo zapisi koji u zadatim poljima imaju jednake vrijednosti. Posebna vrsta spajanja po jednakosti je *prirodno spajanje* koje zadovoljava sljedeće uslove.

- Operator porjeđenja mora biti jednakost.
- U spajanju moraju učestvovati sva polja koja su zajednička za obe relacije.
- U skupu rezultata pojavljuje se samo jedan skup zajedničkih polja.

Neki SUBP, kao na primjer Access kod tabela dobijenih prirodnim spajanjem nude i određene pogodnosti. Access pri spajanju tabela kod kojih postoji veza *jedan prema više* i zajednička polja u rezultatu potiču iz tabele na strani više, automatski se vrši takozvano *preslikavanje redova* (engl. Row Fix-Up) ili automatsku zamenu (engl. AutoLookup). Kada korisnik obrasca unese neku vrijednost za kontrolu vezanu za jedno polje koje se pojavljuje u spoju, Access automatski popuni odgovarajućim vrijednostima kontrole vezane za polja na strani *više*.

Spajanje po nejednakosti, teta spoj (**THETAJOIN**)

Teta spojevi su svi spojevi koji se zasnivaju na operatoru poređenja koji nije jednakost, odnosno na operatorima. <, >, <>, >=, <=. U praksi se rijetko koriste, a najčešće pri pronalaženju zapisa koji sadrže vrijednosti koje veće ili manje od nekih prosječnih ili zbirnih vrijednosti.

Tabela se može spajati i sa samom sobom (**SELF-JOIN**), kada unutar tabele postoji relacija između pojedinih n-torki. Potreba za ovom vrstom spajanja javlja u slučajevima kada postoje unutrašnje veze između pojedinačnih zapisa u jednoj tabeli, unarne veze. To je slučaj sa tabelom RADNIK, jer su neki zaposleni rukovodioci nekim drugim radnicima.

Spoljni spojevi (**OUTER JOIN**)

Po svojoj prirodi mogu biti lijevi (engl. left outer join), desni (engl. right outer join) i potpuni (engl. full outer join). Izdvajaju i kombinuju sve zapise koji zadovoljavaju uslov spajanja plus preostale iz jednog, drugog ili oba skupa. Vrijednosti koje nedostaju (bez parnjaka) zamjenjuju se Null vrijednostima.

Lijevi spoljni spoj učitava sve zapise iz skupa na strani *jedan* u vezi tipa *jedan prema više*, dok desni spoljni spoj učitava sve zapise na strani *više*.

Kod nekih SUBP, na primjer Access i SQL Server vrstu spajanja zasniva na redoslijedu kojim su imena relacija navedena u odredbi FROM u iskazu SELECT. Prva tabela je lijeva a druga desna. Samim tim će sledeća dva iskaza dati isti rezultat: sve zapise iz relacije A i samo one zapise iz skupa B koji ispunjavaju uslov (iskaz je tačan):

SELECT * FROM A LEFT OUTER JOIN B ON uslov

SELECT * FROM B RIGHT OUTER JOIN A ON uslov.

Potpuni spoljni spoj, prikazuje sve zapise iz obe relacije, pri čemu one koji zadovoljavaju uslov kombinuje. Ako neki SUBP ne podržava potpuni spoljni spoj, on se može simulirati pomoću unije lijevog i desnog spoljnog spoja.

Primjer 30: Izlistaj ime, posao, ime odjeljenja i mjesto gdje rade svi zaposleni.

```
SELECT ime, posao, RADNIK.[brod$], ODJELJENJE.[brod#], imeod, mesto
FROM ODJELJENJE, RADNIK
WHERE ODJELJENJE.[brod#] = RADNIK.[brod$];
```

U Access-u:
```
SELECT RADNIK.IME, RADNIK.POSAO, RADNIK.[BROD$], ODJELJENJE.[BROD#],
       ODJELJENJE.IMEOD, ODJELJENJE.MESTO
FROM ODJELJENJE INNER JOIN RADNIK ON ODJELJENJE.[BROD#] = RADNIK.[BROD$];
```

IME	POSAO	BROD$	BROD#	IMEOD	MESTO
Aco	radnik	10	10	komercijala	Novi Beograd
Vaso	prodavac	10	10	komercijala	Novi Beograd
Jovo	upravnik	10	10	komercijala	Novi Beograd
Miro	radnik	10	10	komercijala	Novi Beograd
Tomo	radnik	10	10	komercijala	Novi Beograd
Jovan	radnik	10	10	komercijala	Novi Beograd
Marko	analiticar	10	10	komercijala	Novi Beograd
Petar	vozac	20	20	plan	Dorcol
Slobo	vozac	20	20	plan	Dorcol
Bozo	upravnik	20	20	plan	Dorcol
Mita	savetnik	20	20	plan	Dorcol
Ivan	analiticar	20	20	plan	Dorcol
Pavle	upravnik	30	30	prodaja	Stari Grad
Andro	nabavljac	30	30	prodaja	Stari Grad
Pero	nabavljac	30	30	prodaja	Stari Grad
Simo	savetnik	40	40	direkcija	Banovo Brdo
Savo	direktor	40	40	direkcija	Banovo Brdo
Janko	rukovodila	40	40	direkcija	Banovo Brdo

Slika 6.23 Spajanjem tabela dobija se semantički bogatija relacija

Primjer 31: Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

```
SELECT ime, brod$
FROM RADNIK, ODJELJENJE
WHERE RADNIK.[brod$]= ODJELJENJE.[brod#]
        AND mesto='Dorcol' ;
```

IME	BROD$
Petar	20
Slobo	20
Bozo	20
Mita	20
Ivan	20

Slika 6.24 Spajanjem tabela dobija se isti rezultat

Napomjena: Rezultat je naravno isti kao u primjeru 27, ali je postupak dobijanja različit, i u ovom slučaju bi vrijeme dobijanja odgovora bilo značajno duže.

U svim prethodnim primjerima u klauzuli WHERE korišćeno je spajanje po jednakosti (ODJELJENJE.[brod#] = RADNIK.[brod$]), ali moguće je spajanje po bilo kom operatoru poređenja. Ključnu riječ WHERE**,** u naredbi SELECT moraju da slijede uslovi koje treba da zadovolje podaci u n-torkama iz određene relacije a koji se žele dobiti kao rezultat. Ako se ne navede nikakav uslov, onda naredba SELECT daje Dekartov proizvod (**CARTESIAN JOIN**) relacija navedenih u listi relacija. To je poznati Dekartov proizvod dva skupa. Tako, na primjer naredba:

SELECT * FROM A, B;

daje Dekartov proizvod relacija **A** i **B**. Ako relacija A ima 1000 n-torki i relacija B 1000 n-torki onda nova relacija dobijena spajanjem ovih dveju relacija ima 1.000*1.000, tj. 1.000.000 n-torki. Dakle nova relacija ima vrlo veliki broj zapisa. Sada slijedi pretraživanje tog skupa i izdvajanje samo onih n-torki koje zadovoljavaju uslov iz klauzule WHERE. Ovo je vrlo dugotrajan postupak, za razliku od tehnike ugnježdenih upita gde se najpre izvrši selekcija n-torki koje zadovoljavaju uslov, a tek onda tako redukovan skup se proverava u sledećem upitu. Iako se ugnježdeni upiti uvek mogu realizovati preko operacije prirodnog spajanja to se ne preporučuje zbog drastičnog povećanja vremena obrade upita.

Neka je u bazu dodat još jedan zaposleni koji još nije raspoređen ni u jedno odjeljenje:

IDBR#	IME	POSAO	KVALIF	RUKOVODILAC	DATZAP	PREMIJA	PLATA	BROD$
7891	Mirko	analiticar	VSS		29-jun-02	3000	2000	

Spoljnje spajanje (**OUTER JOIN**) koristimo da se u rezultat spajanja uključe i one n-torke koje ne zadovoljavaju uslov spajanja, tj. nemaju parnjaka u obe tabele, ali zadovoljavaju uslov iz WHERE klauzule. Ako hoćemo da u odgovor uključimo i podatke iz druge tabele, koji nemaju parnjaka u prvoj

tabeli, to se zove desno spajanje (**RIGHT JOIN**), a obrnuto je levo spajanje (**LEFT JOIN**).

Primjer 32: Izlistaj sve podatke o odjeljenjima i radnicima za radnike čija je premija veća od 2000 .

```
SELECT *
FROM ODJELJENJE,  RADNIK
WHERE ODJELJENJE.[brod#] = RADNIK.[brod$](+) AND  premija >2000;
```

U Access-u:
```
SELECT *
FROM ODJELJENJE RIGHT JOIN RADNIK ON ODJELJENJE.[BROD#] = RADNIK.[BROD$]
WHERE (((RADNIK.PREMIJA)>2000));
```

BROD#	IMEOD	MESTO	IDBR#	IME	POSAO	KVALIF	RUKOV	DATZAP	PREMIJA	PLATA	BROD$
10	komercijala	Novi Beograd	6234	Marko	analiticar	VSS	5786	17-dec-90	3000	1300	10
40	direkcija	Banovo Brdo	7890	Ivan	analiticar	VSS	5786	17-dec-90	3200	1600	40
			7891	Mirko	analiticar	VSS		29-jun-02	3000	2000	

Slika 6.25 U odgovoru su i podaci o neraspoređenom radniku

Primjer 33: Izlistaj sve podatke o odjeljenjima i radnicima za odjeljenja čija imena počinju slovima **d** i **e**.

```
SELECT *
FROM ODJELJENJE,  RADNIK
WHERE  (ODJELJENJE.[brod#](+) = RADNIK.[brod$])  AND
        (imeod BETWEEN 'd%'  AND 'f%');
```

U Access-u:
```
SELECT *
FROM ODJELJENJE LEFT JOIN RADNIK ON ODJELJENJE.[BROD#] = RADNIK.[BROD$]
WHERE ODJELJENJE.[IMEOD] BETWEEN 'd*' AND 'f*';
```

BROD#	IMEOD	MESTO	IDBR#	IME	POSAO	KVALIF	RUKOV	DATZAP	PREMIJA	PLATA	BROD$
40	direkcija	Banovo Brdo	5867	Simo	savetnik	VSS	5842	08-avg-70		2750	40
40	direkcija	Banovo Brdo	5842	Savo	direktor	VSS		15-dec-81		3000	40
40	direkcija	Banovo Brdo	7890	Ivan	analiticar	VSS	5786	17-dec-90	3200	1600	40
40	direkcija	Banovo Brdo	6789	Janko	rukovodila	VS		23-dec-99	10	3900	40
50	erc	Zemun									

Slika 6.26 U odgovoru su i podaci o odjeljenju bez radnika

Primjer 34: Prikaži imena, posao i broj odjeljenja radnika kojima je rukovodilac *Savo*.

```
SELECT R.ime AS [ime radnika], R.posao, ŠEF.ime AS [ime šefa], R.brod
FROM RADNIK R,  RADNIK ŠEF
WHERE ŠEF.idbr = R.rukovodilac AND ŠEF.ime="Savo";
```

self join : Select Query

ime radnika	POSAO	ime šefa	BROD
Jovo	upravnik	Savo	10
Bozo	upravnik	Savo	20
Pavle	upravnik	Savo	30
Simo	savetnik	Savo	40
Mita	savetnik	Savo	20

Slika 6.27 U odgovoru su i podaci o odjeljenju bez radnika

Primjer 35: Izlistaj šifre radnika koji rade na dva i više projekta.

```
SELECT Učešće.Idbr#, COUNT(Idbr) AS [Broj projekata]
FROM Učešće
GROUP BY Učešće.Idbr#
HAVING (((Count(*))>=2))
ORDER BY Učešće.Idbr#;
```

Idbr	Broj projekata
5652	2
5932	3
5953	2
6234	3

Slika 6.28 Šifre radnika koji rade na više projekata

Primjer 36: Izlistaj broj sati, šifru, ime i platu radnika koji rade na dva i više projekta.

```
SELECT UČEŠĆE.idbr#, sum(UČEŠĆE.brsati) AS [broj sati], RADNIK.idbr#,
       RADNIK.ime, RADNIK.plata, Count(*) AS [broj projekata]
FROM RADNIK, UČEŠĆE
WHERE RADNIK.IDBR# = UČEŠĆE.IDBR#
GROUP BY radnik.idbr#, RADNIK.IME
HAVING (((Count(*))>1));
```

UCESCE.IDBR	broj sati	RADNIK.IDBR	IME	plata	broj projekata
5652	2000	5652	Jovan	1000	2
5932	2000	5932	Mita	2600	3
5953	2000	5953	Pero	1100	2
6234	2000	6234	Marko	1300	3

Slika 6.29 Imena i plate radnika koji rade na više projekata

Primjer 37: Izlistaj šifru radnika i broj sati na projektu za radnike koji rade na projektu *uvoz*.

```
SELECT DISTINCT UČEŠĆE.idbr#, UČEŠĆE.brsati, UČEŠĆE.brproj#, PROJEKAT.brproj#,
                PROJEKAT.imeproj
FROM PROJEKAT, UČEŠĆE
WHERE PROJEKAT.brproj# = UČEŠĆE.brproj# AND
                         (UČEŠĆE.brproj# = (SELECT PROJEKAT.brproj#
                                           FROM PROJEKAT
                                           WHERE PROJEKAT.imeproj = 'uvoz'));
```

IDBR	BRSATI	UCESCE.BRPR(	PROJEKAT.BRPR	IMEPROJ
5786	2000	100	100	uvoz
5842	2000	100	100	uvoz
5900	2000	100	100	uvoz
5953	1000	100	100	uvoz
5652	1000	100	100	uvoz
5932	500	100	100	uvoz
6234	500	100	100	uvoz

Slika 6.30 Izvještaj o radnicima i broju sati na projektu 'uvoz'

Primjer 38: Izlistaj šifru odjeljenja kao i šifru, ime, platu i posao najbolje plaćenog radnika u svakom odjeljenju.

SQL -
```
SELECT RADNIK.brod# AS BROD, first( RADNIK.idbr#) as [šifra], first( RADNIK.ime) As
[ime], Max(RADNIK.plata) AS [najveća plata], first(RADNIK.posao) as [posao]
FROM RADNIK
GROUP BY RADNIK.brod;
```

BROD	šifra	ime	najveća plata	posao
10	5497	Aco	2400	radnik
20	5367	Petar	2600	vozac
30	5786	Pavle	2800	upravnik
40	5867	Simo	3900	savetnik

Slika 6.31 Šifra, ime, plata i posao najbolje plaćenog radnika u svakom odjeljenju

Napomjena: U prethodnom primjeru predikat prvi *First* obezbjeđuje da se u odgovoru pojavi samo prva n-torka ako ih ima više sa istom (najvećom) platom. U narednom primjeru nije poštovana konvencija da se ključne riječi SQL jezika pišu velikim slovima. To naravno nije dovelo do pojave grešaka jer SQL nije osjetljiv na velika i mala slova. Dakle **as**, **As** i **AS** su ista riječ, kao što je **COUNT** isto što i **Count**, a **Max**, **max** i **MAX** označavaju funkciju **najveći**. Isto važi i za imena tabela i atributa: **radnik**, **Radnik** i **RADNIK** označavaju tabelu u kojoj se skladište podaci o zaposlenima, a **brod**, **Brod** i **BROD** označavaju ime jednog te istog atributa u tabeli RADNIK.

Primjer 39: Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.

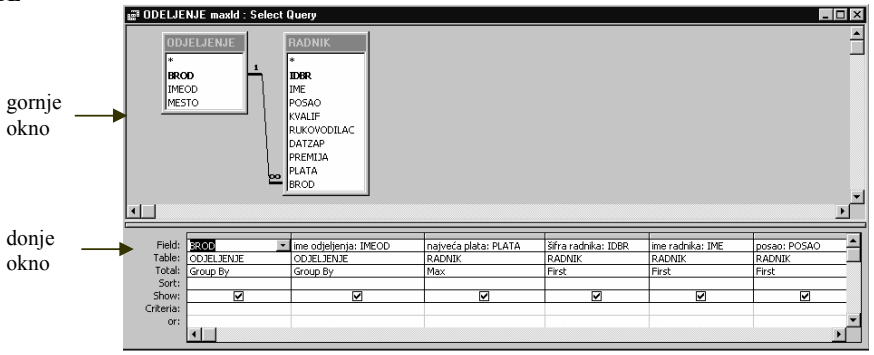
SQL

```
SELECT ODJELJENJE.brod#, imeod AS [ime odjeljenja], Max(plata) AS [najveća plata],
       First(RADNIK.idbr) AS [šifra radnika], First(ime) AS [ime radnika], First(posao)
FROM ODJELJENJE, RADNIK
WHERE ODJELJENJE.brod# = RADNIK.brod$
GROUP BY ODJELJENJE.brod#, imeod;
```

Access

```
SELECT ODJELJENJE.brod, ODJELJENJE.imeod AS [ime odjeljenja],
       Max(RADNIK.plata) AS [najveća plata], First(RADNIK.idbr) AS [šifra radnika],
       First(RADNIK.ime) AS [ime radnika], First(RADNIK.posao) AS posao
FROM ODJELJENJE INNER JOIN RADNIK ON ODJELJENJE.brod = RADNIK.brod$
GROUP BY ODJELJENJE.brod, ODJELJENJE.imeod;
```

QBE –



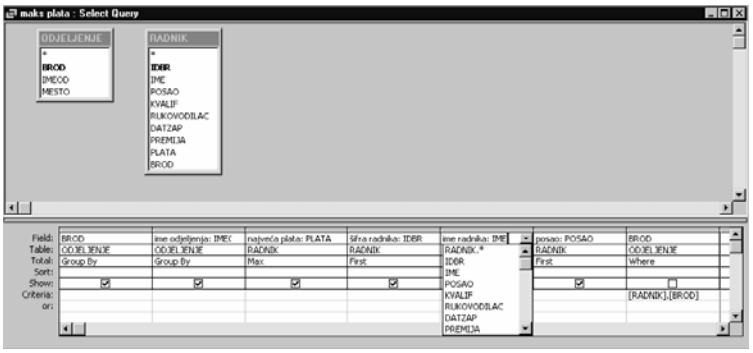
Dvodimenzionalni upit po primjeru

BROD	ime odjeljenja	najveća plata	šifra radnika	ime radnika	posao
10	komercijala	2400	5497	Aco	radnik
20	plan	2600	5367	Petar	vozac
30	prodaja	2800	5786	Pavle	upravnik
40	direkcija	3900	5867	Simo	savetnik

Slika 6.32 Rezultat sva tri upita je isti

Napomjena: Na slici 6.32. a) prikazan je jedan alat u Access-u koji se zasniva na izradi ***upita po primjeru*** (engl. **Q**uery **B**y **E**xample, **QBE**). Za razliku od programskih i drugih upitnih jezika ovaj jezik ima dvodimenzionalnu sintaksu (tabela sa kolonama i vrstama). Druga važna osobina jeste da je sam upit izražen ''primjerom'' (**''by example''**). Umjesto da je data procedura za dobijanje željenog odgovora, korisnik daje primjer kako hoće da bude izračunat željeni odgovor, korisnik daje primjer tabele koju želi da dobije. Upit se u QBE-u izražava pomoću kostura tabele koja daje relacionu shemu. Tačnije, korisnik izabira ovaj kostur iz postojećih tabela. Dakle, najprije izabere tabele (gornje okno), a onda iz pojedinih tabela atribute (donje okno). Kada izabere atribute onda za svaki od njih po potrebi definiše skup željenih vrijednosti – domena, tj. uslova (polje **Criteria**). Zbog toga se ovaj metod smatra relacionim računom domena. Unutar jednog domena može se tražiti zadovoljenje jednog od više uslova (**OR**), ili istovremeno zadovoljenje uslova u raznim domenima (**AND**). Između tabela mogu se praviti relacije – prirodno spajanje (kao na slici 6.32.), ili se te relacije prave preko uslova za neki atribut (kao na slici 6.33. preko atributa brod# u tabeli ODJELJENJE i atributa brod$ u tabeli RADNIK). Ovi jezici nemaju mogućnost kreiranja ugnježdenih upita već je samo moguće izvršiti spajanje tabela.

Primjer 40: Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.



Slika 6.33 Izbor atributa i postavljanje ograničenja za skup dozvoljenih vrijednosti

Napomjena: Rezultat ovog *upita po primjeru* je isti kao na slici 6.32 b).

Napomjena: Standardne, ugrađene funkcije izabiraju se u polju **Total**, u polju **Sort** se zadaju uslovi za uređivanje (sortiranje) odgovora. Tabele se definišu u polju **Table**, a polja se izabiraju iz padajućih listi u polju **Field**. Neka polja su neophodna

radi zadavanja uslova za grupisanje ili spajanje tabela ali nisu neophodna u izvještaju pa se mogu učiniti nevidljivim. To se radi preko polja **Show**, a u primjeru na slici 6.33. poslednje polje preko kojeg je ostvareno spajanje tabela nije vidljivo u izvještaju.

6.2.5 Ažuriranje baze podataka

Dodavanje, izmjena (ažuriranje u užem smislu) i brisanje podataka vrši se naredbama **INSERT**, **UPDATE** i **DELETE**. Kod primjene ovih naredbi ne garantuje se očuvanje integriteta baze podataka, pa se zato njihovo direktno korišćenje ne preporučuje, jer tada o integritetu mora da brine sam korisnik. Ove naredbe koristi neposredno samo administrator baze podataka. Normalno ažuriranje podataka vrši se preko aplikacija za interaktivno ažuriranje u koje su ugrađene procedure za zaštitu integriteta, a sastavni deo ovih procedura su naredbe **INSERT**, **UPDATE** i **DELETE**.

Dodavanje novih n-torki u postojeće relacije je veoma jednostavno a izvodi se naredbom **INSERT**. Opšti oblik glasi:

INSERT INTO ime relacije **[**lista atributa**] VALUES [**lista *vrijednosti* **]**

uz napomenu da se **ovom naredbom odjednom u relaciju mogu unijeti vrijednosti atributa samo jedne n-torke**. Podrazumijeva se dodavanje na kraju tabele, jer redoslijed navođenja n-torki u relaciji nije bitan.

Postoje tri tipa ovih naredbi:

1. za ubacivanje vrijednosti SVIH atributa jedne n-torke,
2. za ubacivanje vrijednosti NEKIH atributa jedne n-torke,
3. za ubacivanje podataka iz jedne tabele u drugu.

1. Za ubacivanje vrijednosti SVIH atributa jedne n-torke, nije potrebno specificirati nazive atributa.

Primjer 41: U relaciju **RADNIK** dodaj podatke o novom zaposlenom koji ima šifru **7891,** ime mu je "**Mirko"**, radi na poslovima **analitičara**, ima visoku stručnu spremu (**VSS**), još nema šefa(Null), počeo je da radi 29-jun-02, ima premiju 3000, platu 2000 i još nije raspoređen ni u jedno odjeljenje (Null).

```
INSERT INTO RADNIK VALUES
( 7891, Mirko, analiticar, VSS, Null, 29-jun-02, 3000, 2000, Null);
```

Napomjena: Ukoliko podatak za neki od atributa nije poznat, kao podatak se unosi vrijednost ***Null***. Redosled podataka u listi VALUES mora biti isti kao u definiciji tabele.

2. *Za ubacivanje vrijednosti NEKIH atributa jedne n-torke, potrebno je specificirati nazive atributa i njihove vrijednosti i to u identičnom poretku.*

Primjer 42: U relaciju RADNIK dodaj podatke o novom zaposlenom, ime mu je "**Mirko"**, koji ima šifru **7891,** ima visoku stručnu spremu (**VSS**), počeo je da radi na dan 29-jun-02.

```
INSERT INTO RADNIK (ime, idbr#, kvalif, datzap)
VALUES (Mirko, 7891, VSS, 29-jun-02);
```

3. *Za ubacivanje vrijednosti NEKIH ili SVIH atributa n-torki, iz jedne tabele u drugu potrebno je da su tabele identično definisane ili se moraju specificirati nazivi atributa i njihove vrijednosti (pomoću naredbe SELECT) i to u identičnom poretku.*

Primjer 43: U relaciju *POVIŠICA<idbr#, povišica, datzap>* dodati podatke o analitičarima i vozačima i dati im povećanje plate od 15%.

```
INSERT INTO POVIŠICA (idbr#, povišica, datzap)
SELECT idbr#, plata*1.15, datzap
FROM RADNIK
WHERE posao IN ('analiticar', 'vozac');
```

Naredba INSERT ubacuje u tabelu POVIŠICA podatke o svim analitičarima i vozačima iz tabele RADNIK pri čemu platu povećava za 15%. Naravno, originalni podaci o platama radnika u tabeli RADNIK ostali su nepromijenjeni.

Brisanje podataka u relaciji može se izvesti pojedinačno, ili grupno. Komandom **DELETE** uvijek se briše cijela n-torka, a ne samo pojedina vrijednost nekog atributa. Sintaksa naredbe u opštem slučaju glasi:

```
DELETE FROM ime relacije [ WHERE lista uslova ]
```

Primjer 44: U relaciji RADNIK obriši sve podatke o radniku **idbr# =** *5953* koji je otišao u penziju.

```
DELETE FROM RADNIK WHERE idbr# = 5953;
```

Ako nije naveden uslov (WHERE) brišu se svi podaci. Iz relacije RADNIK mogu se obrisati svi oni koji su se zaposlili prije *17-feb-90*.

Primjer 45: Izbrisati podatke o radnicima koji su se zaposlili posle 17-feb-90.

```
DELETE FROM RADNIK
WHERE (RADNIK.datzap) < ' 17-feb-90 ';
```

Iz relacije RADNIK mogu se obrisati svi oni koji su radili u odjeljenju *priprema*.

Primjer 46: Izbrisati podatke o svim radnicima koji rade u odjeljenju *priprema*.

```
DELETE FROM RADNIK
WHERE (RADNIK.brod$) = (SELECT brod#
                        FROM ODJELJENJE
                        WHERE imeod= 'priprema');
```

Modifikacija postojećih podataka (**ažuriranje** u užem smislu) izvodi se komandom **UPDATE - SET**. Opšti oblik naredbe glasi:

```
UPDATE ime relacije
SET atribut1=vrijednost1, atribut2=vrijednost2,
WHERE  [ lista uslova ]
```

tako da se ovom naredbom može mijenjati i vrijednost samo jednog podatka unutar jedne n-torke.

Primjer 47: Šef zaposlenog čija je šifra **idbr# = *5932*** je rukovodilac čija šifra je *5842,* a naredba glasi:

```
UPDATE RADNIK SET rukovodilac= 5842 WHERE  idbr# = 5932;
```

Ali ovom se naredbom može mjenjati i vrijednost jednog podatka unutar više n-torki.

Primjer 48: Prebaci sve zaposlene iz odjeljenja 30 u odjeljenje 20, a naredba glasi:

```
UPDATE RADNIK SET brod$= 20 WHERE  brod$ = 30;
```

Napomjena: Pri upotrebi naredbi za ažuriranje treba biti vrlo oprezan jer one mjenjaju stanje baze, tj. vrijednosti podataka. Zbog toga se preporučuje da se najprje pomoću obične **SELECT** naredbe izdvoje n-torke na koje bi se primjenile naredbe za ažuriranje, pa kada nakon analize rezultata nepobitno utvrdimo da se radi o željenoj grupi n-torki tek onda kreirati odgovarajuću naredbu **UPDATE** ili **DELETE**. Access u tu svrhu ima takozvane akcione upite pomoću kojih se mogu ne samo ažurirati podaci već i kreirati pogledi i tabele.

6.2.6 Operatori za rad sa skupovima

Operatori unije (engl. relational union), presjeka (engl. relational intersection), razlike (engr. relational difference) i Dekartov proizvod se zasnivaju na teoriji skupova, ali su donekle izmijenjeni jer se primjenjuju nad relacijama. Da bi se primjenile prve tri operacije relacije moraju biti union-kompatibilne, odnosno moraju imati isti broj atributa i odgovarajući atributi moraju imati iste korespondentne domene. Čak je dovoljno da su atributi istog tipa.

Unija je u suštini relaciona verzija sabiranja. Rezultat je relacija koja sadrži sve zapise jedne na koju su dodati zapisi druge relacije, ali su duplikati eleminisani. Unija omogućava dodavanje novih zapisa u relaciju.

Presjek je operacija koja vraća zapise koji su zajednički u obije relacije i kojom se u stvari pronalaze duplikati. Presjek se realizuje pomoću spoljnjeg spajanja. Neka postoji tabela RADNIK20 koja sadrži zapise o zaposlenima u odjeljenju 20, sa izmjenjenim podacima o radniku čiji *idbr* je 7890 (slika 6.34).

radnik20

IDBR	IME	POSAO	KVALIF	RUKOVODILAC	DATZAP	PREMIJA	PLATA	BROD
5367	Petar	vozac	KV	5780	1.1.1978	1900	1300	20
5780	Bozo	upravnik	VSS	5842	11.8.1984		2200	20
5900	Slobo	vozac	KV	5780	3.10.1978	1300	900	20
5932	Mita	savetnik	VSS	5842	25.3.1965		2600	20
7890	Ivanka	analiticar	VS	5786	17.12.1990	3200	1600	

Slika 6.34 Sa pogledom ODJELJENJE20

Presjek relacija RADNIK i RADNIK20 se realizuje kao lijevo spajanje iz kojeg su isključeni oni zapisi kod kojih je vrijednost *radnik20.idbr* jednaka Null.

Primjer 49: Kreirati presjek relacija RADNIK i RADNIK20.

```
SELECT R.IDBR#, R.IME, R.BROD$, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON RADNIK.IDBR# = radnik20.idbr
WHERE radnik20.idbr IS NOT NULL;
```

IDBR#	IME	BROD$	IDBR	IME	BROD
5367	Petar	20	5367	Petar	20
5900	Slobo	20	5900	Slobo	20
5780	Bozo	20	5780	Bozo	20
5932	Mita	20	5932	Mita	20
7890	Ivan	20	7890	Ivanka	

Slika 6.35 Presjek dveju relacija

Odgovor na ovaj upit prikazan na slici 6.35 sadrži sve zapise iz relacije RADNIK20, bez obzira što vrijednosti neprimarnih atributa nijesu iste u obje relacije.

Razlika relacija sadrži one zapise iz prve relacije koji ne postoje u drugoj relaciji. Razlika relacija RADNIK i RADNIK20 se realizuje kao lijevo spajanje iz kojeg su isključeni oni zapisi kod kojih je vrijednost *radnik20.idbr* nije jednaka Null. Operator spoljnjeg spajanja učitava sve zapise iz obe tabele i postavlja Null vrijednosti u polja koja nemaju odgovarajuće parnjake u drugoj (ovog puta lijevoj) relaciji (slika 6.36). Operator IS NULL u odredbi WHERE izdvaja iz ovog skupa samo one zapise u kojima su Null vrijednosti (odnosno one bez parnjaka, slika 6.37).

Primjer 50: Kreirati razliku relacija RADNIK i RADNIK20.

```
SELECT R.IDBR#, R.IME, R.BROD$, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON RADNIK.IDBR# = radnik20.idbr
WHERE radnik20.idbr IS NULL;
```

IDBR#	IME	BROD$	IDBR	IME	BROD
5367	Petar	20	5367	Petar	20
5497	Aco	10			
5900	Slobo	20	5900	Slobo	20
5519	Vaso	10			
5662	Jovo	10			
5696	Miro	10			
5780	Bozo	20	5780	Bozo	20
5786	Pavle	30			
5867	Simo	40			
5842	Savo	40			
5874	Tomo	10			
5898	Andro	30			
5932	Mita	20	5932	Mita	20
5953	Pero	30			
5652	Jovan	10			
6234	Marko	10			
7890	Ivan	20	7890	Ivanka	
6789	Janko	40			

Slika 6.36 Lijevo spajanje relacija

IDBR#	IME	BROD$	IDBR	IME	BROD
5497	Aco	10			
5519	Vaso	10			
5662	Jovo	10			
5696	Miro	10			
5786	Pavle	30			
5867	Simo	40			
5842	Savo	40			
5874	Tomo	10			
5898	Andro	30			
5953	Pero	30			
5652	Jovan	10			
6234	Marko	10			
6789	Janko	40			

Slika 6.37 Presjek dveju relacija

Napomjena: Slika 6.35 i slika 6.37 pokazuju da su dva zapisa ista ako su im jednaki primarni ključevi, a vrijednosti ostalih atributa mogu biti različite. Imena odgovarajućih atributa ne moraju biti ista, ali su njihovi tipovi isti ili kompatibilni.

6.3 Kreiranje i korišćenje pogleda (VIEW)

Koncept baza podataka daje mogućnost da više raznih aplikacija, pa samim tim i korisnika obrađuje iste podatke i da iz njih dobija informacije koje su relevantne za onu vrstu posla koja je svakom od njih bitna. Korisnik ne mora da zna sve detalje projekta čitave baze podataka jer njega zanimaju samo neki objekti (tabele), i samo neka njihova svojstva (atributi).

Codd je u svome pionirskom radu o relacionim bazama podataka definisao više tipova relacija a među njima i takozvani pogled, prikaz ili **VIEW.** Osnovna razlika između bazne relacije i relacije VIEW je u tome što bazne relacije postoje na memorijskom medijumu računara (disku), dok je VIEW fiktivna, virtuelna relacija koja ne postoji u bazi, ali se može formirati uvijek ako nam zatreba, ako se pozovemo na nju. Za korisnika je onda, bar što se pretraživanja tiče, otvorena ista mogućnost korišćenja ovakve relacije kao i u slučaju pretraživanja neke bazne relacije.

Pogled je prema tome virtuelna imenovana relacija koja se takođe kreira naredbom SELECT**,** ali se u bazi podataka memoriše na specifičan način. Definicija pogleda se čuva u bazi podataka u prevedenom obliku, što obezbeđuje veliku brzinu rada sa pogledima. Svaki put kada korisniku zatreba taj “pogled“ na informacioni sistem, on poziva, i tim kreira, tu virtuelnu relaciju. Kako ažuriranje (dakle izmjena podataka) ovakve relacije

nije moguće (što je i logično), to se rad sa VIEW relacijama koristi često i kao način zaštite integriteta podataka. Treba odmah napomenuti da mnogi RDBMS nemaju mogućnost rada sa pogledima (na primjer Access). Naredba za kreiranje pogleda - VIEW relacije glasi:

CREATE VIEW ime_pogleda **[** atribut1, atribut2, ….. **]** **AS**
SELECT ……..

gdje je **ime_pogleda** naziv novoformirane virtuelne relacije **VIEW**, a navedeni atributi njeni atributi. Naravno, nije potrebno ni naglašavati da u sistemu ne smije da postoji još neka bazna (ili virtuelna) relacija sa istim imenom. Na primjer, za rukovodioca odjeljenja 20 nisu značajni podaci o svim radnicima već samo o onima iz njegovog odjeljenja. U tu svrhu iz relacije ***RADNIK*** < ***idbr#, ime, ….*** > treba izdvojiti samo one zaposlene koji su od interesa:

Primjer 51: Kreirati pogled ODJELJENJE20 koje sadrži podatke o odjeljenju i ime, posao, kvalifikaciju i platu zaposlenih u odjeljenju 20:

CREATE VIEW ODJELJENJE20 **AS**
SELECT O.imeod, O.imeod, R. ime, R.posao, R.kavalif, R.plata
FROM RADNIK R, ODJELJENJE O
WHERE O.brod#=R.brod$ **AND** O.brod#=*20*;

Napomjena: U ovome primjeru nisu uvedeni novi atributi, iako ih **VIEW** relacija može imati. Imena atributa u relaciji **VIEW,** i osnovnim relacijama iz kojih je izvedena, su u ovome primjeru ostala ista.

Svaki put kada korisnik hoće da obrađuje podatke o zaposlenima u svom odjeljenju on jednostavno koristi ovaj **VIEW**.

Primjer 52: Prikaži ime, posao, kvalifikaciju i mjesto gde rade zaposleni sa visokom stručnom spremom u odjeljenju 20:

a)
SELECT O20.mesto, O20.ime, O20.posao, O20.kvalif
FROM ODJELJENJE20 O20
WHERE O20.kvalif="*VSS*"**;**

b)
SELECT mesto, ime, posao, kvalif
FROM ODJELJENJE O **INNER JOIN** RADNIK R **ON** O.brod = R.brod
WHERE (((RADNIK.kvalif)="*vss*") **AND** ((RADNIK.brod)=*20*))**;**

MESTO	IME	POSAO	KVALIF
Dorcol	Bozo	upravnik	VSS
Dorcol	Ivan	analiticar	VSS
Dorcol	Mita	savetnik	VSS

Slika 6.38 Sa pogledom ODJELJENJE20 radimo kao sa fizičkom tabelom

Napomjena: Isti rezultat (slika 6.38) dobija se i upitom a) i upitom b) u ovom primjeru.

Napomjena: Ažuriranje baze podataka preko pogleda ima brojna ograničenja posebno ako je pogled definisan nad više tabela. Najkraće rečeno, da bi mogli da ažuriramo podatke (u fizičkoj tabeli) preko pogleda, pogledi moraju biti definisani nad jednom tabelom i u definiciju moraju biti uključene sve NOT NULL kolone te tabele.

Napomjena: Pogledi objezbjeđuju:

- *jednostavnost korišćenja*, uprošćavaju se upiti, upit a) je znatno jednostavniji nego b),
- *tajnost*, mehanizam za kontrolu pristupa podacima (korisnik vidi samo neke podatke),
- *performanse*, definicija pogleda se čuva u kompajliranom, prevedenom obliku,
- *nezavisnost podataka*, mijenjaju se definicije pogleda a ne aplikacioni programi koji koriste podatke iz baze podataka preko pogleda.

Imena kolona u pogledu ne moraju biti ista kao imena kolona u tabelama iz kojih se pogled izvodi. Pogled se može izbaciti iz baze podataka naredbom DROP VIEW. Ova komanda uklanja pogled i iz relacione šeme i iz rečnika podataka baze podataka.

6.4 Upravljačke naredbe

Upotreba podataka od strane više korisnika (posebno u mrežnom okruženju) unosi dodatne opasnosti po sigurnost podataka i njihov integritet. Zbog toga su razvijeni brojni i moćni postupci zaštite podataka od slučajnog, ali i od neovlašćenog i zlonamjernog korišćenja, izmjene ili uništenja. Zaštita baze podataka se posmatra sa dva aspekta:

- integritet - zaštita od slučajnog i/ili pogrešnog ažuriranja i
- sigurnost - zaštita od neovlašćenog ažuriranja i korišćenja podataka.

Termin *integritet* ovde se koristi da označi tačnost, korektnost i zaštitu od nepravilnog unosa podataka. Narušavanje integriteta baze podataka može da nastane prilikom izvršavanja paralelnih transakcija, međutim savremeni softveri za upravljanje bazama podataka veoma dobro rješavaju ovaj problem, tako da će se ovde razmatrati samo problem zaštite integriteta prilikom izvođenja jedne transakcije.

Prilikom izrade modela podataka potrebno je definisati koje uslove podaci u bazi podataka treba da zadovolje, kada se vrši provjera i koje akcije treba preduzeti kada definisani uslovi nisu ispunjeni. O ovome se vodi računa pri definisanju tipova podataka i njihovih domena pri kreiranju tabela i relacija između njih. Pravila integriteta se definišu za operacije ažuriranja baze podataka: INSERT, UPDATE i DELETE. Kako se ova pravila mjenjaju od slučaja do slučaja, ona moraju biti podržana i u samim aplikacijama.

Pravila integriteta se djele na dvije klase:

- pravila integriteta domena (integritet entiteta) i
- pravila integriteta relacija (referencijalni integritet).

Prilikom izrade modela podataka za implementaciju baze, moraju se za svaki atribut definisati domeni, a takođe i uslovi koji moraju biti zadovoljeni prilikom izvođenja operacija nad objektima.

Termin sigurnost podataka odnosi se na mehanizme zaštite baze podataka od neovlašćenog korišćenja. Sigurnost podataka ima mnogo aspekata (kriptografija, zaštita pri prenosu, fizičko obezbjeđenje, itd.) od kojih će biti opisana samo zaštita od neovlašćenog korišćenja koju pružaju softveri za upravljanje bazama podataka.

Najrasprostranjeniji princip sigurnosti podataka je dodjela, tj., ograničavanje prava pristupa i korišćenja. Obično se svakom korisniku dodjeljuju odgovarajuće privilegije koje određuju koje operacije korisnik može da izvrši nad bazom podataka i njenim objektima (tabelama).

Tabela koju kreira neki korisnik je njegova tabela, on je njen vlasnik. Drugi korisnik je načelno ne može koristiti ukoliko mu vlasnik eksplicitno ne dodjeli prava korišćenja pomoću naredbe GRANT. Opšti oblik naredbe GRANT je:

GRANT {ALL | [ALTER, DELETE, INDEX, SELECT, UPDATE(atr)**]}**
ON [kreator**.]{**tabela|pogled**}**
TO {PUBLIC | korisnik1**[,** korisnik2 **...]}**
[WITH GRANT OPTION].

Drugim korisnicima vlasnik može dodjeliti sva prava (**ALL**) ili samo ona iz liste. Ako dozvoljavamo korisniku samo da gleda podatke treba mu dodjeliti pravo **SELECT**, a ako može i da ih briše onda i pravo **DELETE**. Promjena podataka (ažuriranje) može se ograničiti samo na neke atribute izabranih tabela ili pogleda.

Ako je drugi korisnik dobio od vlasnika i opciju [**WITH GRANT OPTION**] onda on može davati drugim korisnicima prava korišćenja tabele, ali samo ista ili manja od onih koja je on dobio od vlasnika. Oduzimanje prava vrši se naredbom REVOKE, čiji je opšti oblik:

REVOKE {ALL | [ALTER, DELETE, INDEX, SELECT, UPDATE(atr)**]}**
ON [kreator**.]{**tabela|pogled**}**
FROM {PUBLIC | korisnik1**[,** korisnik2 **...]}.**

Koncept baze podataka daje prave efekte kada se radi u mrežnom okruženju, kada veliki broj korisnika istovremeno pristupa podacima iz jedne baze. U tom slučaju postoji realna opasnost da dva ili više korisnika –

klijenata pristupi istom ili istim podacima sa ciljem čitanja ali i izmjene podataka. U tom slučaju može doći do pojave pogrešnih rezultata i ažurnost i integritet podataka mogu biti ugroženi. Da bi se spriječile štetne posledice do kojih može doći kada više korisnika istovremeno pristupa istim podacima većina sistema za upravljanje bazama podataka koristi razne tehnike zaključavanja podataka (**Data Locks**). Dakle kada jedan korisnik pokuša da izvede neku operaciju sa podacima, DBMS te podatke automatski zaključava, naravno samo ako je u pitanju operacija ažuriranja (**UPDATE** ili **DELETE**). Nema potrebe zaključavati podatke kada ih neki upit samo čita, odnosno upit ne smije da blokira ažuriranje.

Postoji više strategija zaključavanja, od vrlo pesimističkih gde se zaključava čitava tabela (**table-level locking**) ili blokovi-stranice podataka (**page-level locking**), preko zaključavanja samo onih n-torki koje se ažuriraju (**row-level locking**), do optimističkih da do izmjene istih polja neće ni doći (bez zaključavanja). Naravno zbog mogućih problema sistemi za upravljanje bazama podataka moraju imati ugrađene mehanizme čuvanja prethodnih verzija podataka, pravljenje rezervnih kopija (**BACKUP**), a takođe vode se i dnevnici transakcija.

Sve promjene nad bazom podataka izazvane SQL naredbama najčešće se odražavaju samo na stanje podataka u operativnoj memoriji korisnikovog računara ili servera. Ako želimo da se izmjene odraze na stvarne podatke na disku (server) potrebno je potvrditi ove promjene, transakcije, naredbom **COMMIT WORK** ili odustati od njih naredbom **ROLLBACK WORK**. Naime, transakcija baze podataka je logička jedinica posla koja se izvršava do kraja ili se poništava u celini. Neke transakcije mogu trajati vrlo dugo (izmjena velikog broja podataka).

Svaka izmjena podataka u bazi podataka evidentira se u dnevniku transakcija. Ako recimo nestanak struje prekine tekuću transakciju, samo dio izmjena biće zapisan na disku, a neki podaci će zadržati staru vrijednost. Naravno da ovo nikako ne smije da se desi, pa se zato čitava transakcija mora poništiti (na osnovu dnevnika transakcija) prilikom prvog narednog pokretanja sistema za upravljanje bazama podataka.

Oporavak baze podataka (**RECOVERY**) predstavlja proces vraćanja baze podataka u stanje za koje se zna da je korektno, posle nekog softverskog ili hardverskog otkaza sistema. Uzroci otkaza mogu da budu različiti: greške u programiranju, greške u operativnom sistemu i samom softveru za upravljanje bazama podataka, padanje glava diska, nestanak napajanja, sabotaža, itd.

Princip na kojem se zasniva oporavak baze podataka je *redundanca podataka*, odnosno postojanje više primjeraka-kopija jednog te istog

podatka na nekom od memorijskih uređaja (disku ili traci). Proces oporavka može se opisati na sledeći način:

- periodično se cijela baza podataka **kopira** (*dump, backup*) na neku arhivsku memoriju,
- za svaku promjenu u bazi u takozvani ***log file*** (*žurnal*) zapisuje se stara (*before image*) i nova vrednost (*after image*) sloga baze podataka,
- posle otkaza sistema, ukoliko je baza podataka oštećena, rekonstruiše se ispravno stanje baze na osnovu poslednje arhivske kopije, a ukoliko je baza samo dovedena u nekonzistentno stanje poništavaju se sve nekorektne promjene, a same transakcije se ponove.

Na kraju možemo reći da SQL ima znatno veće mogućnosti od onih koje pruža relaciona algebra i ima snagu koju obezbeđuje relacioni račun. SQL podržava pet osnovnih operatora relacione algebre: uniju, razliku, Dekartov odnosno Kartezijev proizvod (koji je predstavljen klauzulom FROM), projekcija se izvršava u SELECT klauzuli a uslovi (predikatski račun) za selekciju, tj. izdvajanje n-torki sadržani su u WHERE klauzuli.

SQL dopušta da se međurezultati smeštaju u privremene relacije, i da se zadaju, odnosno kodiraju proizvoljni izrazi relacione algebre.

SQL pruža znatno veće mogućnosti od relacione algebre, a neke od njih su: agregatne funkcije, sortiranje, itd.

Mnoge SQL implementacije omogućavaju da SQL upiti budu deo programa napisanih u jezicima opšte namjene kao što su C, C++, Java, PL/1, Pascal, Cobol ili noviji vizuelni jezici Visual Basic ili Visual C++. Ovo proširuje mogućnosti programera da manipulišu bazama podataka.

6.5 Aplikativni programi

Aplikativni programi služe isključivo i prvenstveno za “približavanje” informacionog sistema korisniku. Svaki potencijalni korisnik (blagajnik ili magacioner u nekom preduzeću, na primjer) ne mora, konačno i ne može, poznavati do u detalje cijeli informacioni sistem, jer koristi samo jedan njegov dio i to na standardan, uvijek isti način. Magacioner u skladištu jedne velike firme (podrazumijeva se da ima još magacionera i još skladišta) evidentira samo podatke o pristigloj i isporučenoj robi iz svoga magacina. Njega prema tome ne interesuje ni koncepcija baze podataka, ni

ostale mogućnosti informacionog sistema, za njega je važno da jednostavno, brzo i pouzdano može da unese podatke za koje je odgovoran i dobije samo one informacije koje on treba.

Srž svakog aplikativnog programa je prema tome neki, ka manipulisanju podacima okrenut programski paket, oko koga se onda pravi programsko okruženje koje omogućava korisniku da radi samo određene operacije bez poznavanja **SQL**-a ili nekog drugog "višeg" jezika, i bez poznavanja detaljne organizacije informacionog sistema.

Softver koji omogućava izradu takvih programa (koji sa stanovišta projektanta informacionog sistema predstavlja samo radno okruženje kojim on informacioni sistem "približava" neukom korisniku ne povećavajući njegove mogućnosti u bilo kom smislu) nalazi se danas na tržištu u više varijanti, u zavisnosti od toga za koje računarske sisteme je predviđen.

Kako su mogućnosti PC-a postale izuzetno velike (prije samo petnaestak godina i najveći računski centri bili su opremljeni lošije od nekog danas bolje opremljenog PC-a posljednje generacije), to je i većina tih programa prilagođena za korišćenje na PC-u.

Jedan od prvih programskih paketa, prilagođen i skromnim konfiguracijama personalnih računara koji rade pod operacionim sistemom DOS, bio je dBase (verzija III, III+, IV), nakon njega stekao je popularnost Clipper i FoxPro (opet u nekoliko razvojnih varijanti), koji je predviđen i za WINDOWS okruženje, čime su mogućnosti prezentacije rezultata (posebno grafičkih) umnogome poboljšane, slijede, takozvane vizuelne varijante (Visual BASIC, Visual FoxPro, Access) kod kojih su mogućnosti prezentacije rezultata dovedene skoro do savršensva.

Kod većih računarskih sistema u početku je dominirao programski jezik COBOL (COBOL je bio namijenjen za hijerarhiske sisteme), koji je razvojem relacionih baza **podataka** i naročito uvođenjem distribuiranih baza podataka, danas skoro potpuno istisnut iz primjene novim softverskim paketima od kojih su najpoznatiji: ORACLE, SQL Server, INFORMIX, SYBASE, INGRES itd.

S obzirom na to da su personalni računari danas po svojim mogućnostima potpuno uporedivi sa velikim računskim centrima od prije 10 do 15 godina, to se pomenuti sistemi za upravljanje relacionim bazama podataka mogu danas implementirati i na PC.

Isto tako, prvobitne verzije paketa, rađene isključivo za PC (dBase III na primjer), u svojim novijim izdanjima (dBase for Windows na primjer) imaju pored sopstvenih naredbi za manipulisanje podacima ugrađen i SQL kao standard u ovoj oblasti.

Konačno, i programski jezici opšte namjene, kao što su na primjer PASCAL ili C++, mogu da prihvataju i tzv. **DBF** fajlove, to jest tabelarno sređene podatke (dakle relacije), pa uz implementaciju nekog upitnog jezika (SQL-a, na primjer) manipulacija podacima i pisanje aplikativnih programa postaju mogući i na taj način.

Uobičajeni postupak razvoja i pisanja aplikacije sastoji se od:

- pisanja programa u višem programskom jeziku (koji ima "ugrađene" naredbe upitnog jezika),
- "prevođenja i povezivanja" (kompilacije i linkovanja) napisanih programa,
- testiranja programa i ispravljanja grešaka, te
- pravljenja verzije koju će korisnik moći da koristi.

Prema tome, pisanje aplikacija, ima sledeće dobre strane:

- korisnik ne mora da poznaje upitni jezik,
- korisnik ne mora da poznaje konfuguraciju baze podataka,
- mogućnost slučajnih grešaka je svedena na minimum,
- bezbjednost podataka je povećana i
- olakšan je prenos programa (jer korisnik ne mora da vodi računa o kompatibilnosti)

ali, nažalost i jednu vrlo lošu:

- korisnik, naime, može da uradi samo ono što je unapred predviđeno, tj. što mu aplikacija dozvoli**.**

6.6 Primjer razvoja aplikacije

Pokažimo na primjeru razvoja jednostavne aplikacije šta se dobija, a šta gubi. Pri tome se nećemo ograničiti ni na jedan određeni programski jezik nego pokušati objasniti sintezu aplikacije na nivou logike programiranja. Pretpostavimo da je potrebno za službenika na šalteru sekretarijata za saobraćaj napraviti aplikaciju koja će mu omogućiti da može:

- uneti novog vozača u registar vozača,
- izbrisati nekog vozača iz registra vozača,
- uneti novo vozilo u registar vozila,
- izbrisati neko vozilo iz registra vozila,

- naći ime i prezime vlasnika poznatog vozila.

Pretpostavimo dalje:

- da nam na raspolaganju stoji neki viši programski jezik,
- da su nam poznata osnovna pravila programiranja,
- da možemo koristiti programski paket za analizu i sintezu informacionih sistema,
- da su formirane dvije tabele (relacije) VOZAC i VOZILO (vidi primjer 8.3) i da
- da je omogućen pristup tim tabelama.

Kostur programa i dijalog sa korisnikom piše se uvijek u odabranom višem programskom jeziku, a onda kada i gdje je to potrebno ugrađuju se elementi upitnog jezika.

U uvodnom dijelu programa treba formirati MENU-listu mogućnosti koja će se ponuditi korisniku, i koja treba da se pojavi na ekranu u obliku neke "forme". Ta lista sadrži 5 procedura, a mogla bi da izgleda ovako:

Odaberite opciju ukucavanjem odgovarajućeg broja:

1. uvođenje novog vozača u registar vozača

2. brisanje vozača iz registra

3. uvođenje novog vozila u registar vozila

4. brisanje nekog vozila iz registra

5. nalaženje imena i prezimena vlasnika vozila na osnovu registarskih tablica njegovog vozila

Postupak u programiranju ove aplikacije može da ima ovakav tok:

PROGRAM

- Izabranu opciju (broj iz ponuđene liste) memorisati u jednu programsku varijablu, na primjer **xxx.**
  Provjeriti da li je ukucan broj veći od nule ili manji od 6.

- *Ako jeste:*
  - vratiti program na početnu listu sa napomenom o grešci.

- *Ako nije:*
  - pozvati proceduru broj **xxx,**

- vratiti program na početni MENU.

PROCEDURE

PROCEDURA 1 za poznatu vrijednost **xxx=1** omogućava unošenje novog vozača u registar vozača preko ekrana monitora.

- matični broj vozača, koji treba da bude uveden u evidenciju vozača, upisati na ekran (u za to pripremljen formular), i smjestiti ga u memoriju računara u, na primjer, varijablu yyy,

- provjeriti da li u tabeli VOZAČ već postoji osoba sa tim matičnim brojem V.matbr=′yyy′. Provjera se može izvesti SQL - upitom:

```
SELECT V.matbr
FROM VOZAČ  V
WHERE V.matbr=′yyy′;
```

- *Ako u odgovoru na upit postoji jedna n-torka:*
  - dati odgovarajuću informaciju na ekran (na primjer: “Vozač pod tim imenom i prezimenom već se nalazi u evidenciji”),
  - zatvoriti tabelu VOZAČ,
  - vratiti na početni izbor.
- *Ako u odgovoru na upit ne postoji nijedna n-torka onda:*
  - formirati masku na ekranu za unošenje podataka o vozacu i unijeti podatke o njemu. Oblik te maske kao i način unošenja podataka zavisi od programskog paketa koji se koristi. **SQL** naredba bi mogla da glasi:

    ```
    INSERT INTO VOZAČ (sv#,prezime,ime,……..)
    VALUES (28079351173513, Petrovic, Petar, ……..);
    ```

  - zatvoriti tabelu VOZAČ,
  - vratiti se na početni MENU,
  - kraj procedure.

PROCEDURA 2 (**xxx=2**) omogućava brisanje vozača iz evidencije.

- Matični broj vozača, koji treba da bude brisan iz evidencije vozača, upisati na ekran (u za to pripremljen formular), i smjestiti ga u memoriju računara u, na primjer, varijablu yyy**.**
- provjeriti (na isti način kao i u **PROCEDURI 1**) da li u tabeli VOZAČ već postoji takva osoba.

- *Ako ne postoji n-torka sa unesenim matbr=´yyy´:*
  - vratiti program na početni izbor.

- *Ako postoji:*
  - prikazati podatke o njoj na ekranu,
  - postaviti pitanje ”Da li želite brisanje vozača iz evidencije”?

- *Ako je odgovor potvrdan:*
  - Izvršiti brisanje (na primjer):

```
DELETE FROM VOZAC  V
WHERE  V.matbr# = ´yyy´;
```

  - zatvoriti tabelu VOZAČ.
- *Ako je odgovor na pitanje bio negativan:*
  - zatvoriti tabelu VOZAČ,
  - vratiti na početni izbor,
  - kraj procedure.

Procedure 3 i **4** su logički analogne sa **Procedurama 1** i **2** samo se umjesto tabele VOZAČ aktivira tabela VOZILA pa nema potrebe da ih navodimo.

PROCEDURA 5 (**xxx=5**), omogućava nalaženje vlasnika vozila na osnovu registarskog broja njegovog vozila:

- registarski broj vozila, čijeg vlasnika hoćemo da pronađemo, upisati na ekran (u za to pripremljen formular), i smjestiti ga u memoriju računara u, na primjer, varijablu yyy
- pronaći vozilo na osnovu registarskog broja regbr#=yyy i izdvojiti matični broj vlasnika.

SQL – program bi mogao da glasi:

```
SELECT  V.matbr
FROM VOZILA   V
WHERE regbr# = ´VA-132-345´;
```

- *Ako postoji tražena n-torka u tabeli VOZILA:*
  - memorisati vrijednost V.matbr (zzz=V.matbr)
    otvoriti tabelu VOZAČ
  - pronaći vozača sa matičnim brojem V.matbr#=zzz.

Upit bi mogao da ima oblik:

```
SELECT *
FROM VOZAČ  V
WHERE  V.matbr# =´zzz´;
```

- *Ako postoji tražena n-torka u tabeli VOZAČ prikazati sve podatke na ekranu i zatim:*
  - zatvoriti tabelu VOZILA,
  - zatvoriti tabelu VOZAČ,
  - vratiti program na glavni izbor.

- *Ako ne postoji nijedna n-torka sa tim matičnim brojem dati odgovarajuću informaciju na ekran i zatim:*
  - zatvoriti tabelu VOZILA,
  - zatvoriti tabelu VOZAČ,
  - vratiti program na glavni izbor.

- Ako ne postoji nijedna n-torka u tabeli VOZILA sa traženim registarskim brojem vozila:
  - dati odgovarajuću informaciju na ekran,
  - zatvoriti tabelu VOZILA,
  - vratiti program na glavni izbor,
  - kraj procedure.

Službenik sekretarijata za saobraćaj u nekoj opštini, za koga je ovaj program pripremljen, ne zna, i ne treba da zna, ništa o strukturi baze podataka, a i pored toga može efikasno da je koristi. Nažalost, samo u

jednom njenom aspektu, onom koji je programiran aplikacionim programom.

Drugim rječima, korisniku su na rasploganju samo pet operacija i ništa van toga. Ako je, recimo, neki automobil promjenio vlasnika (izvršena kupoprodaja), službenik sekretarijata za saobraćaj nema na raspolaganju mogućnost promjene vlasnika automobila.

Da bi obavio tu aktivnost, službenik mora najprije da uvede novog vlasnika u registar vozača (ako već nije registrovan), zatim da izbriše vozilo iz registra i da to isto vozilo uvede u taj isti registar kao novo vozilo i pridruži ga prethodno uvedenom vlasniku.

Neke informacije i aktivnosti u ovakvom informacionom sistemu nijesu moguće ni na koji način. Tako, recimo, nije moguće saznati koliko u Beogradu ima automobila marke Mercedes novijih od dvije godine, ili ko je vlasnik nekog ukradenog automobila sa kojeg je skinuta tablica.

Treba reći da u bazi podataka postoje svi potrebni podaci za dobijanje ovakvih informacija (na primjer, u bazi postoji podatak o broju motora vozila), ali to prilikom projektovanja aplikacije nije uzeto u obzir, i ne može se ostvariti bez izmjena aplikacije, tj. bez izrade nove aplikacije.

Glava 7

Primjeri

7.1 Informacioni sistem sekretarijata za saobraćaj

U sekretarijatu za saobraćaj vode se slijedeći podaci o vozačima:

- *matični broj, ime i prezime vozača, datum rođenja, adresa, broj telefona, broj vozačke dozvole, eventualne kazne,.....*

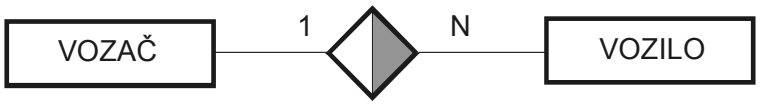
i vozilima

- *registarski broj vozila, marka i tip vozila, broj motora, broj šasije, snaga motora, boja, vrsta vozila, godina proizvodnje.....*

Objekti u sistemu su prema tome VOZAČ i VOZILO. Veza među objektima je tipa 1 : N (jer je dozvoljeno, to dozvoljavaju propisi, da jedan vozač posjeduje više vozila, ali jedno vozilo može imati samo jednog vlasnika). E-R model je krajnje jednostavan (slika 7.1) a relacioni model, izveden na osnovu ER-modela, ima oblik:

VOZAČ *<matbr#, prez, ime, adresa, tel, brdoz, kazneni bodovi >*

VOZILO*<regbr#, marka, tip, brmot, bršas, snaga, god, matbr >.*



Slika 7.1 E-R model informacionog sistema sekretarijata za saobraćaj

Upit koji, na primjer, daje spisak svih vlasnika vozila ***"Opel"*** tip ***"Rekord"*** godina proizvodnje ***"1989"***, glasi:

```
SELECT V.ime, V.prez
FROM VOZAČ  V,  VOZILO  VL
```

```
WHERE  V.matbr# = VL.matbr
     AND VL.marka = ´opel´
     AND VL.tip =  ´rekord´
     AND VL.god  = ´1989´ ;
```

Upit kojim saznajemo ime i prezime vlasnika kola čiji registarski broj počinje sa *"REG123"* mogao bi da glasi:

```
SELECT  V.prez, V.ime
FROM  VOZAČ  V,  VOZILO  VL
WHERE  V.matbr#  = VL.matbr
       AND  VL.regbr#  = ´REG123*´;
```

Spisak vozača vozila marke *"Mercedes"* sa motorom jačim od *"100"* kW dobili bismo upitom:

```
SELECT  V.prez, V.ime
FROM  VOZAČ  V
WHERE  V.matbr# =
        (SELECT  VL. matbr
         FROM  VOZILO  VL
        WHERE  VL.marka  =  ´Mercedes ´
        AND  VL.snaga  > 100);
```

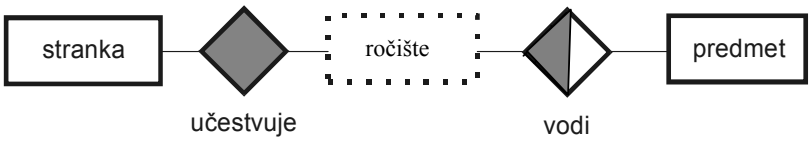
7.2 Dio informacionog sistema advokatske kancelarije

Poslovanje advokatske kancelarije treba prebaciti na računar i formirati odgovarajući informacioni sistem. U tu svrhu, za početak, projektant je nakon prvog intervjua sa advokatom, kada je saznao osnovna pravila vođenja postupka pred sudom, saznao da:

- *u jednoj parnici može da učestvuje više stranki,*
- *jedna stranka može da vodi više različitih parnica, a*
- *jedan spor može biti predmet više ročišta,*

Objekti u sistemu su prema tome STRANKA, PREDMET i ROČIŠTE pri čemu je PARNICA egzistencijalno i identifikaciono zavisan objekat od STRANKE, i PREDMETA.

ER-model (slika 7.2) E-R prikazan na slici 7.2.



Slika 7.2 E-R model dijela informacionog sistema advokatske kancelarije

S obzirom na vezu N:M među objektima STRANKA i ROČIŠTE relacionom modelu:

STRANKA< *mbrs#, ime, prezime, zanimanje, adresa, tel., …*>
PREDMET < *šifpar#, vrsta_spora, text, …*>
ROČIŠTE < *šifpar#, datroč#, vrijeme, rezultat, …*>.

potrebno je dodati i jednu veznu relaciju:

UČESTVUJE < *mbrs#, šifpar#, datroč*>.

Ovako koncipiran sistem omogućava praćenje aktivnosti u vođenju sporova pred sudom.

7.3 Informacioni sistem sportskog saveza

U sportskom savezu grada vodi se evidencija o sportskim društvima - klubovima na njegovoj teritoriji. U cilju formiranja informacionog sistema sportskog saveza, u prvoj fazi, od sekretara sportskog saveza u razgovoru su dobijene prve informacije o organizaciji i načinu poslovanja saveza i klubova u njemu. To su:

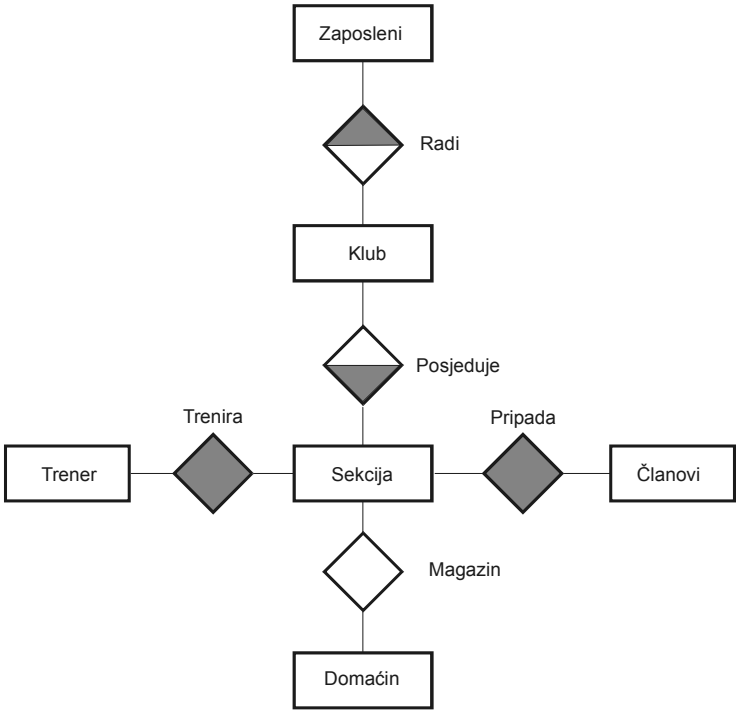
- *Klubovi u gradu imaju svoje prostorije sa telefonima i imaju određen broj zaposlenih službenika,*
- *Klubovi su interno organizovani po sekcijama (u klubu može da ih bude više), a neke od sekcija imaju i svoje posebne prostorije, eventualno i po jednog službenika –domaćina,*
- *Član kluba može da bude takmičar ili simpatizer. Prilikom upisa od svakog člana se uzimaju osnovni podaci (ime, prezime, adresa, telefon, status, itd). Pojedinac može da bude učlanjen u više sekcija jednog kluba,*
- *Klubovi angažuju trenere, klub može imati više trenera za jednu sekciju (koja ima više članova), a dozvoljava se mogućnost da jedan trener trenira više sekcija unutar istog kluba.*

Entiteti - objekti određeni su šifrom (identifikatorom) i nizom atributa koji su od značaja. Na osnovu dobijenih informacija projektant je uveo sljedeće objekte:

- KLUB, sa podacima o svim klubovima u savezu,
- ZAPOSLENI, sa podacima o stalno zaposlenim,
- SEKCIJA, sa podacima o sekcijama po klubovima,
- ČLANOVI, sa podacima o članovima svih klubova,
- TRENER, sa osnovnim podacima o trenerima, i
- DOMAĆIN, sa osnovnim podacima o domaćinu sekcije (ukoliko ga sekcija ima).

Između entiteta postoje *veze* definisane uslovima poslovanja.

- RADI - povezuje *zaposlene* (službenike) u nekom klubu sa odgovarajućim *klubom*. Način veze je obavezan, a tip je 1:N jer broj službenika u klubu može da bude i veći od jedan.



Slika 7.3 E-R model informacionog sistema sportskog saveza

- POSJEDUJE - povezuje *klub* sa njegovim *sekcijama*. Način veze je obavezan, (klub mora da posjeduje bar jednu sekciju da bi mogao da egzistira), a tip je 1 : N, jer u klubu može da postoji više sekcija.
- TRENIRA - povezuje *trenera* sa *sekcijama* koje trenira. Način veze je obavezan, a tip je N:M jer je dozvoljeno da jedna sekcija ima više trenera, i da jedan trener priprema više raznih sekcija,
- PRIPADA - povezuje *članove* sa odgovarajućim *sekcijama*. Veza je obavezna i takođe tipa N:M jer jedan član može biti u više sekcija, a sekcija može da ima više članova, i
- MAGACIN - povezuje *sekciju* sa njenim *domaćinom*, veza je opcionalna (sekcija ne mora da ima magacionera), tipa je 1:1 obzirom da jedna sekcija može da ima samo jednog magacionera.

Relacioni model dobijen je na osnovu E-R modela prikazanog na slici 7.3. Broj atributa u stvarnosti može biti i veći, a zbog jednostavnosti je ograničen na max. četiri. Relacioni model glasi:

KLUB < *šifkluba#, naziv, adresa, telefon* >

SEKCIJA < *šifsekcije#, šifd, naziv, adresa, telefon, šifkluba* >

TRENER < *šiftren#, ime, prezime, adresa, telefon* >

ČLAN < *šifčl#, ime, prezime, adresa, telefon* >

ZAPOSLENI < *šifsekcije#, ime, prez., adresa, tel., funkcija, šifkluba*>

DOMAĆIN < *šifd#, ime, prezime, adresa, telefon* >

sa dvije vezne relacije kojima se eliminišu veze M:N

TRENIRA < *šiftren#, šifsekcije#* >

PRIPADA < *šifčl#, šifsekcije#, status* >

Postavimo nekoliko SQL upita:

1. Spisak svih klubova koji imaju sekciju stonog tenisa?

```
SELECT K.naziv
FROM KLUB  K, SEKCIJA  S
WHERE K.šifkluba# = S.šifkluba#
     AND  S.naziv = 'stoni tenis';
```

2. Imena i prezimena trenera *odbojke* dobijamo upitom:

```
SELECT  T.ime, T.prezime, K.naziv
FROM  TRENER  T, TRENIRA  TR,  SEKCIJA  S, KLUB  K
WHERE  S.šifsekcije# = TR.šifsekcije#
   AND  TR.šiftren# = T.šiftren#
   AND  S.šifkluba# = K.šifkluba#
   AND  S.naziv = 'odbojka' ;
```

ili, drugačije:

```
SELECT TRENER.ime, TRENER.prezime
FROM TRENER
WHERE TRENER.šiftren# IN
      (SELECT  TRENIRA.šiftren#
       FROM  TRENIRA
       WHERE  TRENIRA.šifsekcije#  IN
              (SELECT  SEKCIJA.šifsekcije#
              FROM  KLUB
              WHERE  KLUB.šifkluba# =
                           (SELECT SEKCIJA. šifkluba
                            FROM  SEKCIJA
                           WHERE  SEKCIJA.naziv ='odbojka')));
```

3. Naziv kluba u kome kao blagajnik radi "*Markovic"*

```
SELECT K.naziv
FROM  KLUB  K
WHERE  K.šifkluba#  IN
                 (SELECT Z.šifkluba#
                   FROM ZAPOSLENI  Z
                   WHERE  Z.funkcija = 'blagajnik'
                   AND  Z.prezime = 'Markovic');
```

4. Spisak sekcija sa nazivom kluba koje nemaju svoga domaćina?

```
SELECT  S.naziv,   K.naziv
FROM  DOMAĆIN D, SEKCIJA  S,  KLUB  K
WHERE  D.ime AND D.prezime = Null
     AND  S.šifkluba# = K.šifkluba#;
```

5. Spisak svih *takmičar*a u košarkaškoj sekciji kluba "*Partizan*" ?

```
SELECT C.prezime, C.ime
FROM  CLAN   C
WHERE C.sc#  IN
        (SELECT P.sc#
         FROM  PRIPADA   P
         WHERE  P.status = 'takmicar'
         AND P.šifsekcije#  IN
                (SELECT S.šifsekcije#
                 FROM SEKCIJA  S
                 WHERE S.naziv ='Košarka'
                 AND  S.šifkluba#  IN
                        (SELECT K.šifkluba#
                         FROM  KLUB  K
                          WHERE  K.naziv ='Partizan')));
```

7.4 Informacioni sistem školske biblioteke

U razgovoru sa bibliotekarkom školske biblioteke, za koju je rukovodstvo škole odlučilo da poslovanje i evidenciju prebaci na računar, projektant informacionog sistema dobio je sljedeće podatke:

- *svaka knjiga u biblioteci dobija, pored broja odgovarajuće Narodne biblioteke (UDK klasifikacija), i interni registarski broj,*
- *o svakoj knjizi vodi se evidencija o imenu autora, naslovu knjige, broju stranica, nazivu izdavača, godini izdanja,*
- *svaki korisnik (čitalac) biblioteke mora biti registrovan. Prilikom registracije uzimaju se podaci o njegovom matičnom broju, imenu i prezimenu, broju lične karte, danu, mjesecu i godini rođenja, mjestu rođenja, adresi i telefonu (ako ga posjeduje),*
- *prilikom uzimanja knjige registruje se korisnik, datum uzimanja i vraćanja knjige, i datum do kada mora biti vraćena.*

Daljnim uvidom u poslovanje biblioteke naknadno je utvrđeno da poslovnik biblioteke određuje da se:

- *jedna knjiga može nalaziti samo kod jednog čitaoca,*
- *od istog autora u biblioteci mogu postojati knjige različitih naslova*
- *biblioteka može posjedovati više istih knjiga jednoga autora,*

- *jedan čitalac može istovremeno da bude zadužen sa više knjiga (maksimalno tri).*

Na osnovu ovih podataka koncipiran je ER-model slijedeće strukture*:*

Entiteti - objekti u sistemu su:

- KNJIGA sa svim podacima o knjizi kao što su ime autora, eventualni koautori, naziv knjige, godina izdanja, itd. ,
- PRIMJERAK sa registracionim podacima biblioteke,
- ČITALAC sa svim podacima o čitaocu,
- NA_ČITANJU, egzistencijalno i identifikaciono zavisan objekat od objekata PRIMJERAK i ČITALAC**,** u kome su podaci o datumu uzimanja i vraćanja knjige. Dakle, ovaj objekat spada u klasu slabih objekata.

Veze definišu uslovi poslovanja.

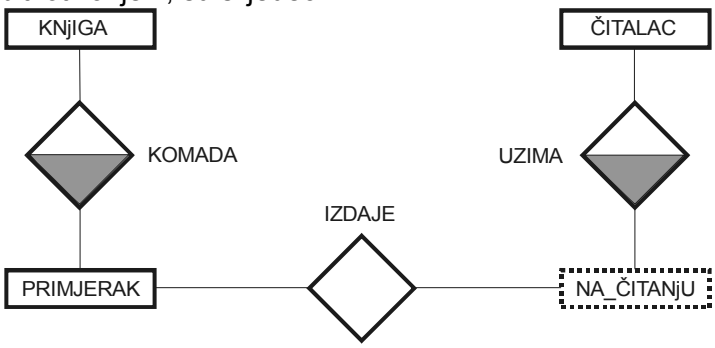
- Veza IZDAJE, između objekata PRIMJERAK i NA_ČITANJU je opcionalna (knjiga može, ali ne mora biti na čitanju) i tipa je 1:1.
- Veza UZIMA, između ČITALAC i NA_ČITANJU je obavezna tipa 1:N, jer jedan čitalac može imati do tri knjige na čitanju**.**
- Veza KOMADA, između KNJIGA i PRIMJERAK je obavezna i tipa 1:N, jer biblioteka može posjedovati više primjeraka iste knjige.

Grafički prikaz E-R modela vidi se na slici 7.4 a.

Relacioni model slijedi direktno iz predloženog E-R modela i glasi :

KNJIGA < *udk#, naz, autor, izd., god, mjesto*>
PRIMJERAK < *šifknjige#, udk* >
ČITALAC< *šifčit#, matbr, ime, prezime, datrod, mjesto, adresa, tel* >
NA_ČITANJU < *šifknjige#, šifčit#, datuzima, datvraća* >.

Međutim, pri realizaciji sistema, tokom unosa podataka o knjigama i testiranja prototipa aplikacije, uočeni su neki nedostatci. Ti nedostatci, koji su morali biti otklonjeni, su slijedeći:



7.4 a) E-R model informacionog sistema školske biblioteke

1. *mnoge knjige imaju više (a ne jednog) autora,*
2. *za efikasno pronalaženje relevantne literature potrebno je svrstati knjige u određene oblasti upotrebom ključnih reči,*
3. *mogu postojati iste knjige koje su izdali različiti izdavači, i*

Predloženi E-R model mora se prema tome modifikovati, jer novi, strožiji uslovi koje informacioni sistem školske biblioteke treba da zadovolji glase:

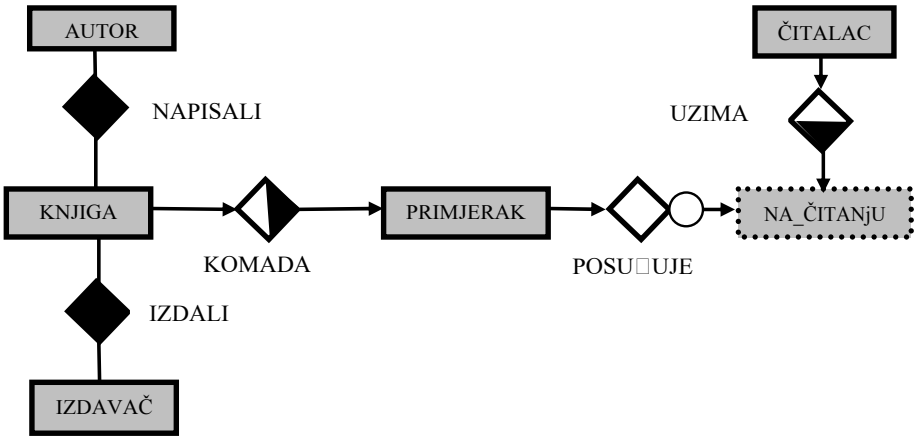
- *svaka knjiga koja postane vlasništvo biblioteke mora pored evidencionog broja Narodne biblioteke (UDK klasifikacija) dobiti i interni registarski broj,*
- *o svakoj knjizi vodi se evidencija o imenima svih autora (ako ih ima više), naslovu knjige, broju stranica, nazivu izdavača, godini izdanja i ključnim riječima (do maksimalno pet),*
- *svaki korisnik (čitalac) biblioteke mora biti registrovan u biblioteci. Prilikom registracije uzimaju se podaci o matičnom broju, imenu i prezimenu, broju lične karte, adresi i telefonu,*
- *prilikom uzimanja knjige registruje se knjiga i korisnik, datum uzimanja i vraćanja knjige te datum do kada mora biti vraćena,*
- *jedna knjiga može se nalaziti samo kod jednog čitaoca,*
- *u biblioteci postoje iste knjige koje su izdali različiti izdavači,*
- *od istog autora mogu postojati knjige različitih naslova, kao i više istih knjiga jednoga autora, i*
- *jedan čitalac može da bude zadužen sa više knjiga*.

E-R model informacionog sistema baziran na ovim uslovima dobija se kao proširenje postojeće prve verzije (prednost relacionih modela – mogu se dorađivati) tako da trud prilikom sinteze prvog modela nije bio uzaludan. Struktura bi mogla da bude sljedeća – slika 7.4b. *Entiteti* u sistemu su sada:

  - KNJIGA**,** sa podacima o knjizi,
  - AUTOR**,** sa podacima o autorima,
  - IZDAVAČ**,** sa podacima o izdavaču,
  - PRIMJERAK**,** sa registracionim podacima biblioteke,
- ČITALAC**,** sa podacima o čitaocu, i
- NA_ČITANJU**,** egzistencijalno i identifikaciono zavisan objekat od objekata PRIMJERAK i ČITALAC**,** sa podacima o datumu uzimanja i vraćanja knjige - slabi objekat.

Veze su:

- NAPISALI –povezuje KNJIGU sa AUTORIMA. Tip veze je N : M,
- IZDALI –povezuje KNJIGU sa IZDAVAČIMA. Tip veze je N : M, jer knjiga može biti izdana od više izdavača, i
- veza IZDAJE preimenuje se u POSUĐUJE **a** definiše vezu između objekata PRIMJERAK i NACITANJU**.** Tip veze je 1:1.



Slika 7.4 b) E-R model informacionog sistema školske biblioteke

Relacioni model slijedi direktno iz E-R modela:

Objekti su:

AUTOR < *šifautora#, ime, prezime, adresa, zvanje*, …>
KNJIGA < *udk#, naz, kr1, kr2, kr3, kr4, kr5*, …>
IZDAVAČ < *šifizd#, naziv, adresa*, …>
PRIMJERAK < *šifprim #, udk*, …>
ČITALAC< *šifčit#, ime, prezime, adresa, tel,* …>
NA_ČITANJU < *šifprim#, šifčit#, datuzimanja, datvracanja,* …>.

te vezni objekti

IZDALI < *šifizd#, udk#, godina*, …>
NAPISALI < *šifautora#, udk#, redni_broj_autora*, …>

1. Upitom:

```
SELECT K.autor
FROM KNJIGA  K
WHERE K.autor = 'Marko Marković';
```

dobijamo knjige čiji je autor "*Marko Marković*".

2. Ime i prezime čitaoca kod kojega je na čitanju knjiga *Robinson*:

```
SELECT Č.prezime, Č.ime
FROM  ČITALAC  Č, PRIMJERAK P, NA_ČITANJU  N
WHERE Č.šifčit# = N. šifčit#
        AND N.šifknjige# = P.šifknjige#
        AND P.šifknjige# = K.šifknjige#
        AND  K.naziv = 'Robinson';
```

3. Posljednji upit se može postaviti i na drugi način:

```
SELECT Č.prezime, Č.ime
FROM  ČITALAC  Č
WHERE Č. šifčit#  IN
        (SELECT N. šifčit#
        FROM NA_ČITANJU  N
        WHERE  N.šifknjige#  IN
                (SELECT  P.šifknjige#
                FROM PRIMJERAK P,
                WHERE  P.šifknjige#  = (SELECT K.šifknjige#
                                        FROM  KNJIGA K
                                        WHERE K.naziv =‘Robinson ‘ )));
```

Napomena: *Brzina kojom će sistem dati odgovore nije ista. U slučaju (2) zbog tri operacije prirodnog spajanja četiri tabele, ako svaka ima samo po 1000 zapisa kreira se tabela od 10^{12} redova. Sada se izabiraju oni koji zadovoljavaju uslove iz instrukcije WHERE.*

Vreme odgovora sistema biće značajno duže nego u slučaju (3) gdje se sukcesivno izdvajaju (selektuju) n-torke koje zadovoljavaju postavljene uslove.

7.5 Dio informacionog sistema studentske službe

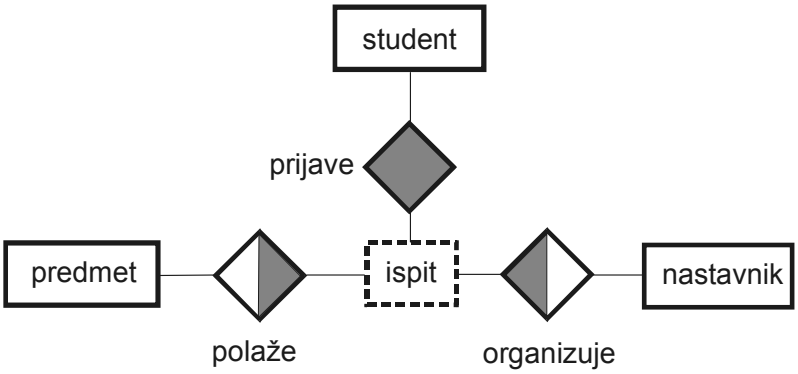
U svakoj studentskoj službi vode se, između ostaloga, i podaci o:

- *studentima (brind, prezime, ime, adresa, …),*
- *nastavnicima (šifranas, ime, prezime, adresa, tel, …),*
- *predmetima (šifrapred, naziv, semestar, …), i*
- *ispitima (datum, ocjena, ime kand., ime prof., naziv pred.).*

Pretpostavimo da statut fakulteta omogućava da:

- *jedan predmet može se polagati više puta,*
- *na jednom ispitu polaže se jedan predmet,*
- *jedan student može prijaviti više ispita,*
- *jedan ispit može polagati više studenata,*
- *jedan nastavnik može da predaje više predmeta,*
- *jedan predmet može da predaje samo jedan nastavnik,*
- *jedan nastavnik organizuje više ispita, i*
- *jedan ispit organizuje samo jedan nastavnik.*

Na osnovu ovih podataka ER-model sistema može imati strukturu koja se vidi na slici 7.5



Slika 7.5 E-R model dijela informacionog sistema studentske službe

Relacioni model, izveden sa slike 7.5 ima sljedeću strukturu: Objekti su:

NASTAVNIK <*šifranast#, prezime, ime, adresa, tel,* … .>
PREDMET < *šifrapred#, šifranast, naziv, semestar,* … >
STUDENT < *brind#, prezime, ime, adresa, šifrapred*, … >
ISPIT < *datispita#, šifrapred#, šifranast#, vrijeme, sala, tip,* … >.

i vezni objekat:

PRIJAVE < *brind#, datispita#, šifrapred#, šifranast#, ocjena*, … >

Relacija ISPIT ima složen ključ (*šifrapred#, datispita#*), jer se ispit iz nekog predmeta može ponoviti, nekog drugog datuma, pa se samo tako može jednoznačno identifikovati.

Vezni objekat PRIJAVE ima takođe složen ključ sastavljen od tri atributa (da bi se znalo koji student polaže toga datuma taj ispit) i jedan svoj atribut – rezultat ispita, to jest *ocjenu*.

Ako, na primjer, hoćemo da dobijemo spisak imena i prezimena svih studenata sa brojem njihovog indeksa a koji su u ispitnom roku *"april 1990"* polagali predmet *"matematika"* i dobili ocjenu *"8"*, rezultat se može se dobiti upitom:

```
SELECT STUDENT.brind#, STUDENT.prezime, STUDENT.ime
 FROM STUDENT, PRIJAVE, PREDMET
 WHERE  STUDENT.brind# = PRIJAVE.brind#
        AND PREDMET.šifrapred# = PRIJAVE.šifrapred#
        AND PREDMET.naziv = ´matematika´
        AND PRIJAVE.datispita = ´april 1990´
        AND PRIJAVE.ocjena = 8 ;
```

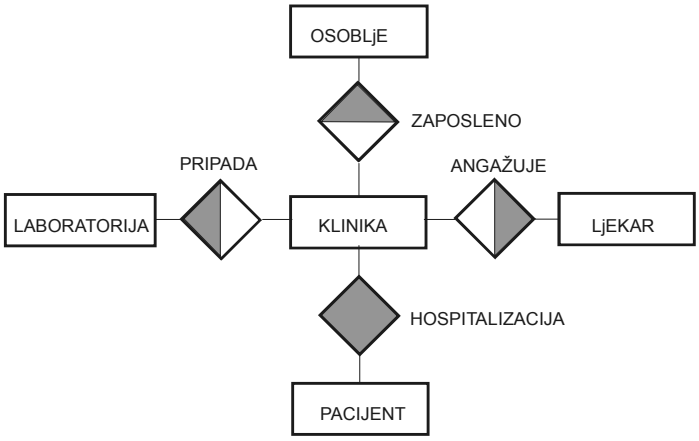
7.6 Dio informacionog sistema kliničkog centra

U administraciji kliničkog centra vodi se evidencija o:

- KLINIKAMA (*naziv, adresa, telefon, broj kreveta,* …),
- LJEKARIMA (*ime, prezime, specijalnost, klinika, adresa, tel,* …),
- OSOBLJU (*ime, prezime, klinika, ljekar, adresa, tel,*…),
- PACIJENTIMA (*mat.br., ime, prezime, adresa, tel,* …), i
- LABORATORIJAMA (*šifra, naziv, specijalnost, tel,* …).

Sistem poslovanja omogućava da:

- *jedna klinika angažuje više ljekara, ali jedan ljekar može biti angažovan samo na jednoj klinici,*
- *na jednoj klinici radi više osoblja, ali jedan član osoblja može da radi samo na jednoj klinici,*
- *jednoj klinici može da pripada više laboratorija, ali jedna laboratorija pripada samo jednoj klinici,*
- *jedna klinika hospitalizuje više pacijenata, ali jedan pacijent može biti hospitalozovan više puta, na raznim klinikama.*



Slika 7.6 E-R model dijela informacionog sistema kliničkog centra

ER-model koji prati opisani sistem poslovanja klinike vidi se na slici 7.6 a relacioni model ovog dijela informacionog sistema kliničkog centra ima prema tome slijedeću strukturu:

Objekti su:
KLINIKA < *šifklinike#, naziv, adresa, tel, broj_kreveta*, …>
LJEKAR <*lmbr#, ime, prezime, spec., adresa, tel, šifklinike,* …>
PACIJENT < *pmbr#, ime, prezime, adresa, tel,….. >*
LABORATORIJA < *šiflab#, naziv, specijalnost, adresa, tel, šifklinike,*

OSOBLJE <*ombr#, ime, prezime, funkcija, adresa, tel, šifklinike, …*>

te vezna relacija

HOSPITALIZACIJA < *brhosp#, pmbr, šifklinike, lmbr, datprijema, anamneza, dijagnoza, terapija, ishod, …*>.

U tri tabele koristi se isti atribut: *mbr – matični broj* pa je napravljena razlika dodavanjem još jednog slova i (za ljekare), p (za pacijente) i o (za osoblje).

Vezna relacija, HOSPITALIZACIJA ima složeni ključ koji omogućava jednoznačan pristup svakom pacijentu prilikom svakog prijema u bolnicu pošto isti pacijent može više puta biti hospitalizovan.

Ako, na primjer, tražimo pacijente (ime i prezime, telefon i naziv klinike na kojoj su bili liječeni), koji su bili primljeni u kliničkom centru u periodu od 01.01.1995. do 01.01.1997., pod dijagnozom "*mišija groznica*", a otpušteni sa dijagnozom ”*izliječen*” SQL upit bi mogao da ima formu:

```
SELECT  P.ime, P.prezime, P.tel, K.naziv
FROM PACIJENT P, HOSPITALIZACIJA H, KLINIKA K
WHERE K.šifklinike# = H.šifklinike
    AND H.pmbr= P.pmbr#
    AND H.datpr  BETWEEN  01.01.1995
    AND   01.01.1997
    AND H.dijagnoza = “mišija groznica”
    AND H.ishod = “izliječen”;
```

7.7 Informacioni sistem video kluba

Vlasnik video kluba odlučio je, u cilju efikasnijeg rada, da svoje poslovanje vodi pomoću računara i tako ubrza proces izdavanja i vraćanja kaseta. U video klubu posjeduje veliki broj video kaseta, raznoga žanra, a neke filmove ima i u više primjeraka – kaseta (ili CD-ova). Rok vraćanja je tri dana. Sistem poslovanja video-kluba dozvoljava da:

- *jedan član posudi do tri kasete jednovremeno, ali jedna određena kaseta može biti iznajmljena samo jednom članu,*
- *od istog režisera ima u video klubu više različitih filmova, ali jedan film režirao je uvijek samo jedan režiser,*
- *od jedne producentske kuće u video klubu ima više filmova, ali jedan film je napravljen samo u jednoj producentskoj kući,*
- *jedan glumac-glumica igra u više različitih filmova, ali i u filmu može da igra više od jednoga glumca-glumice,*
- *u video klubu ima više kaseta sa filmovima od istoga scenariste, ali jedan film ima samo jednoga scenaristu.*

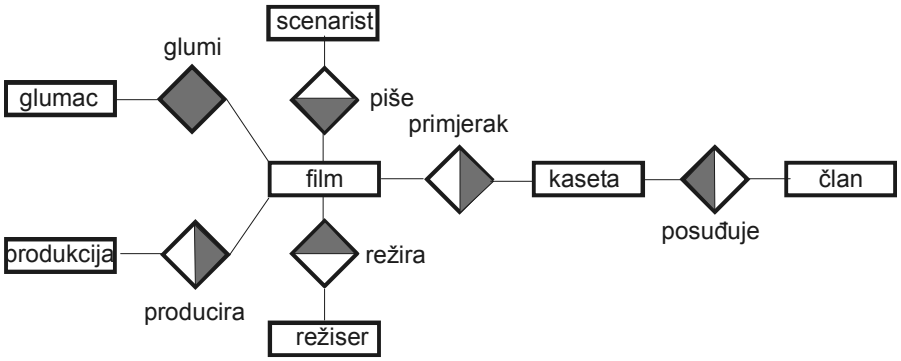
Da bi se mušterijama u što kraćem roku mogao dati odgovor da li u video-klubu postoji film - kaseta koji oni traže, informacioni sistem je koncipiran ovako: Objekti – entiteti u sistemu su:

- FILM
- KASETA
- PRODUKCIJA
- GLUMAC
- REŽISER
- SCENARISTA
- ČLAN

a pobrojani uslovi definišu dodatne veze u sistemu. U ovome slučaju samo jedna veza N:M rezultira veznom relacijom i to:

- GLUMI

E-R model ovoga sistema se vidi na slici 7.7:



Slika 7.7 E-R model poslovanja video-kluba

Relacioni model izveden iz ER-modela sa slike 7.7 i glasi:

FILM <*šiffil#, naziv, šifprod, šifrež, šifscen, žanr,* ...>
KASETA <*šifkasete#, šiffil, mbr, datizd, datvrać, cijena*, ...>
PRODUKCIJA < *šifprod#, naziv, drzava*, ...>
GLUMAC <*šifgl#, ime_i_ prezime,* …>
REŽISER <*šifrež#, ime_i_ prezime, šiffil,* …>
SCENARISTA <*šifscen#, ime_i_ prezime, šiffil,* …>
ČLAN <*mbr#, ime_i_ prezime, adresa, telefon,* …>

sa veznim objektom

GLUMI < šiffil #, šifgl# >,

1. Odgovor koje sve filmove posjeduje video klub koje je producirao "*Metro Goldvin Mayer*", a u njima igraju *Džon Vejn* ili *Din Martin* dobijamo:

```
SELECT F.naziv
FROM FILM F,
WHERE F.šifprod# = ( SELET P. šifprod #
FROM PRODUKCIJA P
WHERE P.naziv = ”Metro Goldvin Mayer” )
AND F. šiffil#  = ( SELET GLUMI. šiffil#
FROM GLUMI
WHERE GLUMI. šifgl# =( SELECT G. šifgl#
FROM GLUMAC G
WHERE G.ime_i_prezime IN  (”Din Martin” , ”Džon Vejn”));
```

7.8 Dio informacionog sistema lanca robnih kuća

Direkcija lanca robnih kuća odlučila je da formira informacioni sistem. Inženjer projektant, u razgovoru sa rukovodiocima i radnicima došao je do prvog modela informacionog sistema. Objekti u njemu bili bi:

- PRODAVNICE
- RADNIK
- ŠEF
- LOKACIJA
- KUPAC
- ROBA
- DOBAVLJAČ

Sistem poslovanja postavlja dodatne uslove:

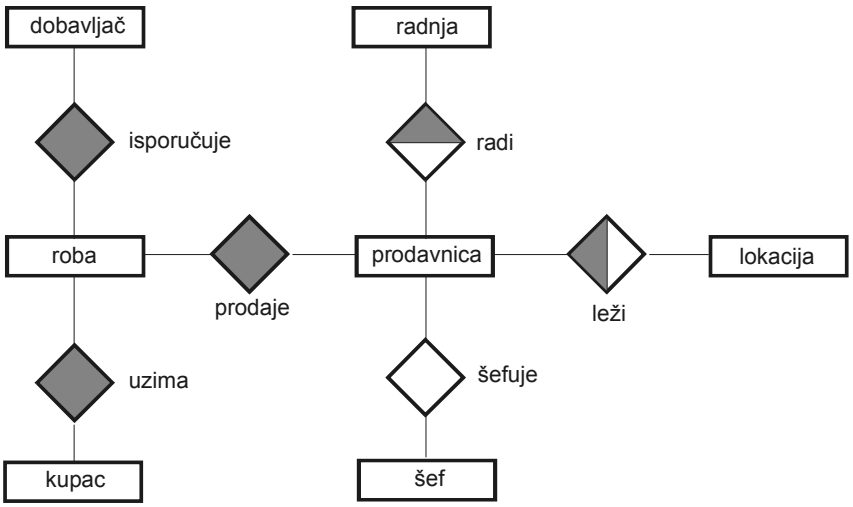
- *jedan isporučilac može da isporučuje raznu robu, ali se jedan artikal može nabavljati i od više isporučilaca,*
- *jedan kupac kupuje više artikala, a isti artikal može biti kupljen od više kupaca,*
- *u prodavnici ima na prodaju više raznih artikala, a svaki artikal se može nalaziti u više prodavnica,*
- *u jednoj prodavnici radi više radnika, ali svaki radnik može da radi samo u jednoj prodavnici,*
- *jedna prodavnica ima jednog šefa, i jedan šef može biti “gazda” samo u jednoj prodavnici, i*
- *na jednoj lokaciji može biti više prodavnica, ali jedna prodavnica može biti samo na jednoj lokaciji.*

Uslovi poslovanja rezultiraju prema tome sljedećim dodatnim veznim relacijama:

- ISPORUKA

- UZIMA
- PRODAJE

E-R model sistema vidi se na slici 7.8.



Slika 7.8 E-R model lanca prodavnica neke robne kuće

Relacioni model izveden iz ER-modela sa slike 7.8 glasi:

PRODAVNICE < *šifprod#, naziv, šiflok, mbrš*, …..>
RADNIK < *mbrr#, ime, prezime, adresa, tel.,* ….>
ŠEF < *mbrš#, ime, prezime, adresa, telefon*,…..>
LOKACIJA < *šiflok#, naziv, brojstanov*, …>
KUPAC < *mbrk#, ime, prezime, adresa, tel,* …>
ROBA < *šifrob#, naziv, cijena*, …>
DOBAVLJAČ < *šifisp#, naziv, adresa, tel.,* …>.

uz vezne reacije:

ISPORUKA < *šifisp#, šifrob#, brojkomada,* …>
UZIMA < *mbrk#, šifrob#, brojkomada, datum,* …>
PRODAJE < *šifrob#, šifprod#, brojkomada,* …>.

Upit: " *Kojeg datuma, u kojoj prodavnici je kupac xxx kupio proizvod yyy "*? mogao bi da izgleda ovako:

```
SELECT P.naziv, U.datum,
FROM KUPAC  K, ROBA  R, UZIMA  U, PRODAVNICA  P, PRODAJE  PD
WHERE   P.šifprod#=PD. šifpord#
            AND PD. šifrob#= R. šifrob#
```

AND (ROBA R.naziv = “*yyy*”
AND R. šifrob#= U. šifrob#)
AND (U.mbrk#= K.mbrk#
AND K.prezime = “*xxx*”);

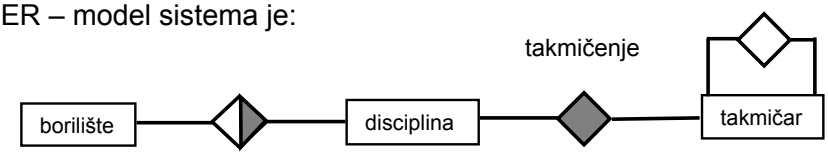
7.9 Informacioni sistem sportskog takmičenja

Takmičenje u zimskim sportovima treba da se održi na raznim borilištima u muškim i ženskim disciplinama. Organizacioni komitet formirao je bazu podataka koja sadrži potrebne podatke o:

- TAKMIČARIMA,
- BORILIŠTIMA i
- DISCIPLINAMA.

Propozicije takmičenja dopuštaju da jedan takmičar može da učestvuje u više raznih disciplina (na primjer slalom, veleslalom), da u pojedinim disciplinama nastupa više takmičara, dok u nekima (umjetničko klizanje, na primjer) nastupaju i takmičarski parovi.

1. ER – model sistema je:



2. Odgovarajući relacioni model glasi:

Objekti u sistemu su:

TAKMIČAR <*st#, ime, prezime, dat_roď.,.....sp*>
DISCIPLINA < *sd#, naziv, sb*>
BORILIŠTE < *sb#, naziv, lokacija*>

sa veznom relacijom

TAKMIČENJE < *st#, sd#, datum_takmičenja, sb* >

Zadatak: Prikazati ime i prezime partnera koji nastupa u disciplini "klizanje" ako je jedan od partnera AAA.

SELECT *ime, prezime*
FROM *TAKMIČAR T1, DISCIPLINA D*
WHERE *naziv.D = "klizanje"*
AND *T1.ime = "AAA",*
AND T1.sp IS NOT Null
AND T1.st =T1.sp

Postaviti dodatno SQL upite koji daju odgovore na sljedeća pitanja:

1. *spisak takmičarskih disciplina sa datumima kada se takmičenje održava i naznakom na kome borilištu?*

2. *saznati za disciplinu „veleslalom - muškarci“ kada (datum) i gdje je (lokacija-naziv borilišta), kada je održano to takmičenje kao i ime i prezime takmičara koji je postigao najbolji rezultat.*
3. *spisak svih takmičara (ako ih ima) koji na dan takmičenja (neka je to bio datum dd.mm.gggg.) nisu imali punih 18 godina pa treba da budu diskvalifikovani.*
4. *na osnovu imena i prezimena takmičarke u disciplini „klizanje - parovi“, saznati ime i prezime njenog partnera?*
5. *spisak svih takmičara koji nastupaju u više od jedne discipline?*

7.10 Primjer razvoja aplikacije u dBase IV

Danas se u upotrebi nalazi niz programskih paketa koji su specijalno pripremljeni za instalaciju na personalnim računarima, a za korišćenje u oblasti eksploatacije informacionih sistema.

Programski paket dBase (verzija III, III+ ili IV) pojavio se među prvima na tržištu, nije zahtijevan u pogledu konfiguracije računara a mogao se nabaviti u formi interpretera i compilera (Clipper) i zato je dugo vremena bio u upotrebi, pa su se neke aplikacije zadržale i do danas.

Ovaj programski paket spada u takozvane kvazi-relacione jezike (što znači da se veze između tabela moraju uspostaviti programskom instrukcijom) tako da je program “čitljiviji“ za nekoga ko pred sobom ima izvorni kod, ali je programiranje istovremeno komplikovanije. To je razlog zbog kojega je, kao jedan od primjera, prikazana upotreba programskog paketa dBase.

Da bi čitaocu, koji do sada nije imao kontakta sa paketom dBase**,** procedura programiranja bila razumljivija, svaka instrukcija će biti komentarisana, što konačna verzija takvog programa ne zahtijeva.

Kao primjer uzmimo već analizirani informacioni sistem sekretarijata za saobraćaj. Pretpostavimo da su tabele VOZAČ i VOZILA kreirane i popunjene sa odgovarajućim podacima.

Na početku svakog dBase programa instrukcijom SET mogu se izabrati pogodnosti koje će biti prisutne u tom programu. Program pod nazivom EVIDENCA ima svoje ime sa ekstenzijom PRO**.** Na primjer:

```
PROGRAM EVIDENCA.PRO
SET TALK OFF
SET ECHO OFF
SET DATE GERMAIN
```

Komanda SET TALK OFF *isključuje poruke obavještenja za vrijeme izvođenja programa. Uključivanje te mogućnosti (*SET TALK ON*) preporučljivo je u fazi pisanja programa, ili traženja grešaka u već napisanom programu.*

Komanda SET ECHO ON *uključuje prikazivanje instrukcija prilikom izvođenja programa. I ovde važi isto pravilo, suprotnu mogućnost (OFF) treba koristiti u fazi testiranja programa.*

*Komanda (*GERMAIN*)određuje načni pisanja datuma u programu. Nama je najbliži način pisanja, kao u Njemačkoj to jest pisanje datuma u formi dan.mjesec.godina.*

CLEAR - *Instrukcija CLEAR briše ekran*

```
@ 0,20 SAY  " 1. Moguci poslovi "
@ 5,20 SAY  " 2. Uvodenje novog vozaca u registar "
@ 6,20 SAY  " 3. Brisanje vozaca iz registra "
@ 7,20 SAY  " 4. Uvodenje vozila u registar "
@ 8,20 SAY  " 5. Brisanje vozila iz registra "
@ 9,20 SAY  " 6. Nalazenje vlasnika na osnovu reg.br."
@ 11,20 SAY  " 7.  Kraj posla "
```

Sve navedene instrukcije služe za ispisivanje poruka (teksta koji je naveden pod navodnim znacima) na ekranu monitora. Brojevi, odvojeni zapetom, su koordinate na ekranu na kojima hoćemo da se željeni tekst pojavi. Prvi broj označava red (0 do 24), drugi broj kolonu (0 do 79).

```
ACCEPT " Upisite odgovarajuci broj " TO  xxx
CLEAR
```

Instrukcija ACCEPT *služi za prihvatanje podatka sa tastature računara i smještanje u memoriju računara na adresu xxx (xxx je memorijska varijabla). Ova instrukcija omogućava jednovremeno i ispisivanje poruka na ekranu monitora.*

```
SET PROCEDURE TO EVIDENCA
DO CASE
        CASE xxx = "1"
                DO alfa
        CASE xxx = "2"
                DO beta
        CASE xxx = "3"
                DO gama
        CASE xxx = "4"
                DO delta
        CASE xxx = "5"
                DO eta
        CASE xxx = "6"
                DO theta
        CASE xxx = "7"
                DO kraj
        OTHERWISE
```

```
        WAIT “Pogresno ukucano-pritisnite neki taster”
ENDCASE

CLEAR
```

SET PROCEDURE TO EVIDENCA *naznačava da glavni program ima i podprograme - procedure.*

Instrukcijom DO CASE *korisnik dobija mogućnost da izborom broja opcije aktivira željenu proceduru* (ALFA, BETA, GAMA...). *U slučaju da korisnik otkuca nepostojeći broj (7<xxx<1) na ekranu se javlja odgovarajuća poruka.*

Instrukcija WAIT *zaustavlja program i ispisuje poruku*

Procedure ALFA, BETA *itd. pišu se kao odvojeni programi. Napišimo u ovom slučaju samo proceduru* THETA *jer koristi dvije datoteke.*

```
PROCEDURE THETA
CLEAR
SELECT 1
USE VOZILA
```

Instrukcija SELECT USE *“otvara” pozvanu tabelu i daje joj broj. U konkretnom slučaju broj tabele* VOZILA *će biti* 1.

```
LOCATE FOR regbr = “VA-132-345”
IF EOF()
        ? “ Vozilo sa tim reg. br. Nije u evidenciji “
        CLOSE VOZILA
        RETURN
ELSE
        yyy = matbr
        SELECT 2
        USE VOZAC
        LOCATE FOR matbr = yyy
        CLEAR
        IF EOF()
                ? “ Trazeni vozac nije u evidenciji”
                CLOSE ALL
        ELSE
                @SAY 3,15 “Podaci o vlasniku vozila”
                @SAY 5,15 “Prezime      : “ prez
                @SAY 6,15 “Ime          : “ ime
                @SAY 7,15 “Telefon      : “ tel
                @SAY 8,15 “Adresa       : “ adresa
                @SAY 9,15 “ Dozvola br : “ brdoz
        ENDIF
ENDIF
CLOSE ALL
```

RETURN

Instrukcija LOCATE *odgovara operaciji restrikcije i iz tabele* VOZILA *izdvaja samo onu n-torku koja ima traženu vrijednost atributa regbr#. Kako je atribut regb# ključni, to će postojati samo jedna, ili nijedna, takva n-torka.*

Ukoliko ne postoji nijedna takva n-torka (nema vozila sa tim registarskim brojem) instrukcija LOCATE će se pozicionirati na kraj - EOF(End Of File) tabele VOZILA.

U tom slučaju treba dati odgovarajuću poruku na ekran (postiže se naredbom ? koja odgovara naredbi PRINT), zatvoriti tabelu VOZILA (CLOSE), i vratiti program u glavni program instrukcijom RETURN.

Ukoliko u tabeli VOZILA postoji takva n-torka (ELSE):

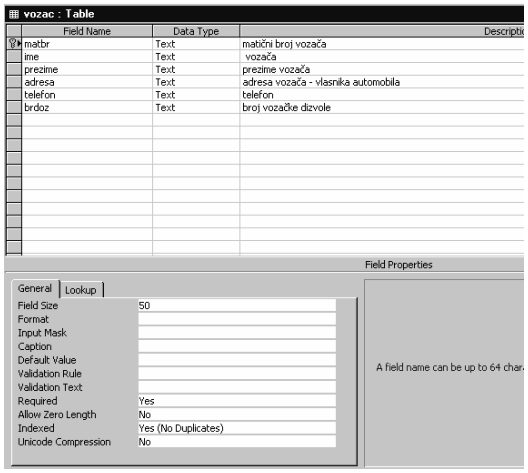
- *promjenljivoj yyy treba pripisati vrijednost podatka atributa matbr u toj n-torki,*
- *"očistiti" ekran kako bi na ekranu mogla biti ispisana poruka,*
- *"otvoriti" tabelu* VOZAČ *(njen broj će biti 2),*
- *pronaći u njoj n-torku za koju je zadovoljen uslov: matbr = yyy,*
- *ako je nema dati odgovarajuću poruku na ekranu,*
- *ako je prisutna odštampati sve podatke o vozaču iz te n-torke tabele* VOZAČ,
- *zatvoriti obje otvorene tabele* (CLOSE ALL*), i*
- *vratiti se u glavni program.*

I preostale procedure se mogu programirati na sličan način. Naravno, ovaj primjer prije svega treba da pokaže kako se nekada radilo, kada nisu postojali moćni vizuelni softverski alati poput onih koji su ugrađeni u Access. Zbog toga ćemo ovaj isti primjer uraditi i upotrebom Accessovih "čarobnjaka".

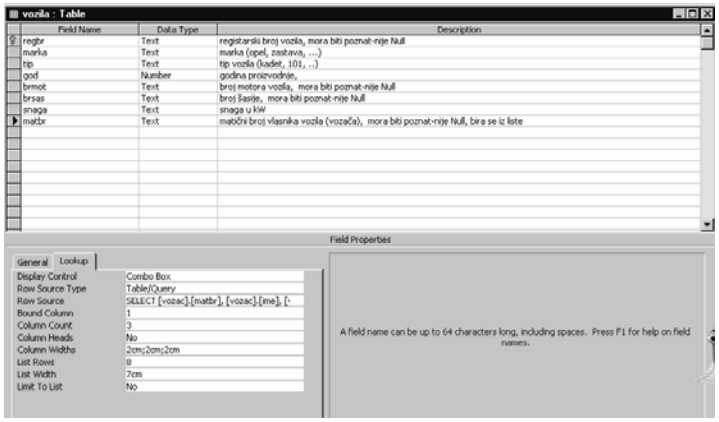
7.11 Primjer razvoja aplikacije u Access 2003

Ovaj postupak ćemo pokazati na primjeru informacionog sistema sekretarijata za saobraćaj (7.1). Najpre kreiramo tabele koje odgovaraju modelu sa slike 7.1. saglasno pravilima za prevođenje modela objekti-veze u relacionu formu: ime entiteta postaje ime tabele, identifikator entiteta postaje primarni ključ tabele, a atributi entiteta postaju kolone u tabeli. Definicija tabela data je na slikama 7.9. i 7.10.

Veza tipa 1:N nameće spoljnji ključ *matbr* tabeli na strani više (u tabeli VOZILA), koji je ujedno primarni ključ u tabeli na strani 1 (u tabeli VOZAČ). *Jedno vozilo **pripada** samo jednom vozaču, a jedan vozač može da **poseduje** više vozila*. Dakle relacioni model odgovara onom sa slike 7.11. Da smo nametnuli referencijalni integritet sa kaskadnim brisanjem i ažuriranjem, onda bi relacioni model izgledao kao na slici 7.12., pa onda sam Access brine o ispravnosti i ažurnosti podataka pri unosu i brisanju.



Slika 7.9 Definicija tabele VOZAČ



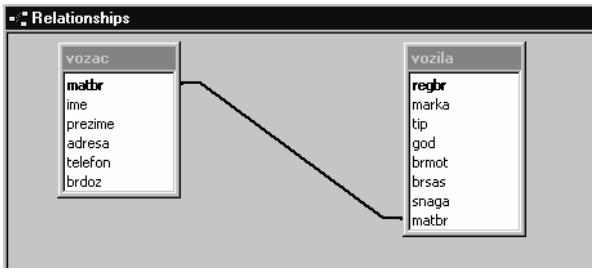
*Slika 7.10 Definicija tabele VOZILA sa nametnutim integritetom za polje **matbr***

Pri tome možemo u polje **Look Up** podatka *matbr* u tabeli VOZILA nametnuti dodatna pravila integriteta: ovo polje je padajuća lista (Combo Box) koja može uzimati samo vrijednosti koje postoje u istoimenom polju u

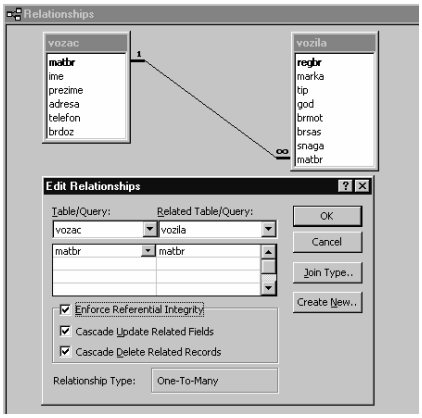
tabeli VOZAC, slika 7.12. I u ovom slučaju imamo ne samo proveru ispravnosti podataka već pri unosu ne moramo da ukucavamo podatak (i rizikujemo pojavu greške), već podatak izabiramo iz liste u kojoj može biti prikazano i ime i prezime vozača kao dodatna mera provere. To se postiže upitom:

SELECT vozac.matbr, vozac.ime, vozac.prezime
FROM vozac**;**

U polju granična kolona (Bound Column) upišemo vrednost 1 (prva kolona u upitu je matbr), broj kolona je 3 (Column Count), širina svake kolone je 2 cm, a širina liste 8cm.



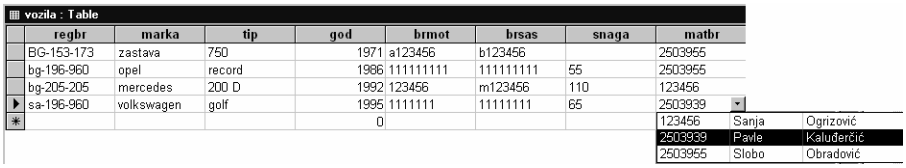
Slika 7.11 Relacioni model sekretarijata za saobraćaj



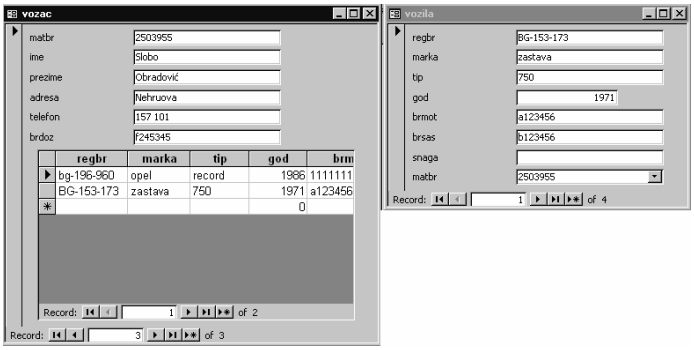
Slika 7.12 Relacioni model sa nametnutim referencijalnim integritetom

Zahvaljujući nametnutom integritetu unos u tabelarnom prikazu (Datasheet) je vrlo bezbedan i lak, slika 7.13.

Za ovako kreiran model koristeći Access-ove čarobnjake, na primjer Autoform bez ikakvog dodatnog truda dobijamo forme za unos podataka u osnovne tabele, prikazane na slici 7.14.



Slika 7.13 Pri unosu neki podaci se prosto biraju iz ponuđene liste

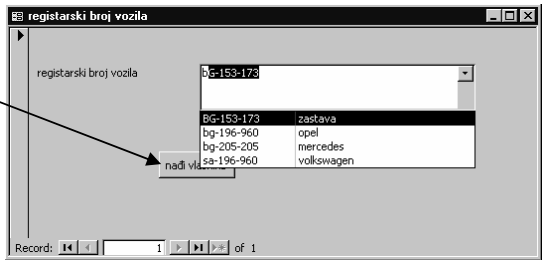


Slika 7.14 Sam Access generiše obrasce za pregled i unos zapisa

Uočavamo da je zbog veze 1:N između tabela VOZAČ i VOZILA u obrascu za upis novog vozača u registar odmah data mogućnost da upišemo i podatke o vozilu ako su nam ti podaci dostupni. Obrazac vozac ima podobrazac za vozila. Ovi obrasci se dodatno mogu doraditi u Design modu tako da im se dodaju još neke osobine koji sam Access nije mogao da doda jer mu naravno nije poznata logika aplikacije.

Napravimo obrazac ***regbr*** za zadavanje registarskog broja vozila preko padajuće liste koja uzima podatke iz tabele VOZILA, slika 7.15.

dugme za pokretanje upita **parametar**
na osnovu kojeg se generiše obrazac **vlasnik vozila**

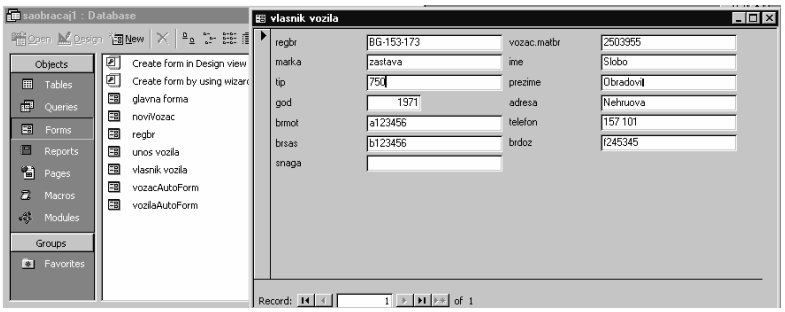


Slika 7.15 Registarski broj se upiše ili izabere iz liste

Sada kreiramo jedan parametarski upit koji omogućava da na osnovu unetog registarskog broja vozila, iz prethodnog obrasca, dobijemo podatke i o vlasniku i o vozilu. To je upit *parametar*:

```
SELECT vozila.*, vozac.*
FROM vozac INNER JOIN vozila ON [vozac].[matbr]=[vozila].[matbr]
WHERE ((([vozila].[regbr])=[forms]![regbr]!combo0));
```

Klikom na dugme *nađi vlasnika* na obrascu *regbr* pokreće se obrazac *vlasnik vozila* kreiran direktno čarobnjakom na osnovu upita *parametar*. Na ekranu će se pojaviti svi pomenuti podaci, slika 7.16.



*Slika 7.16 U obrascu **vlasnik vozila** prikazani su svi podaci o vozilu i vlasniku*

Na kraju napravimo *glavnu formu*, početni obrazac (slika 6.17.) koji se pojavljuje pri pokretanju aplikacije (postavimo je u Tools / Start Up) na kojoj se nalaze samo komandna dugmad za izbor opcija: nađi vlasnika vozila, unos novog vozača i unos novog vozila.

*Slika 7.17 Pri pokretanju aplikacije pojavljuje se **glavna forma**, a ne radni prozor*

Kada kliknemo na dugme nađi *vlasnika vozila*, pokreće se obrazac *regbr*, a preko njega obrazac *vlasnik vozila*. Klikom na preostala dugmad pokreću se drugi obrasci (aplikacije).

To se postiže na osnovi Visual Basic koda koji generiše sam Access ali te procedure za obradu događaja (na primjer za obradu događaja klik()) možemo i sami napisati:

Dugme nađi vlasnika vozila

```
Private Sub Command0_Click()
On Error GoTo Err_Command0_Click

   Dim stDocName As String
   Dim stLinkCriteria As String

   stDocName = "regbr"
   DoCmd.OpenForm stDocName, , , stLinkCriteria

Exit_Command0_Click:
   Exit Sub

Err_Command0_Click:
   MsgBox Err.Description
   Resume Exit_Command0_Click
End Sub
```

Dugme *unos novog* vozača

```
Private Sub Command1_Click()
On Error GoTo Err_Command1_Click

   Dim stDocName As String
   Dim stLinkCriteria As String

   stDocName = "noviVozac"
   DoCmd.OpenForm stDocName, , , stLinkCriteria

Exit_Command1_Click:
   Exit Sub

Err_Command1_Click:
   MsgBox Err.Description
   Resume Exit_Command1_Click

End Sub
```

Dugme *unos novog* vozila

```
Private Sub Command2_Click()
On Error GoTo Err_Command2_Click

   Dim stDocName As String
   Dim stLinkCriteria As String

   stDocName = "unos vozila"
   DoCmd.OpenForm stDocName, , , stLinkCriteria

Exit_Command2_Click:
   Exit Sub

Err_Command2_Click:
   MsgBox Err.Description
   Resume Exit_Command2_Click

End Sub
```

Obrazac (forma) **regbr:**

```
Private Sub nadji_vlasnika_Click()
On Error GoTo Err_nadi_vlasnika_Click

Dim stDocName As String
   stDocName = "vlasnik vozila"
   DoCmd.OpenQuery stDocName, acNormal, acEdit
Exit_nadji_vlasnika_Click:
   Exit Sub

Err_nadji_vlasnika_Click:
   MsgBox Err.Description
   Resume Exit_nadi_vlasnika_Click
   End Sub
```

```
Private Sub Command3_Click()
On Error GoTo Err_Command3_Click
   Dim stDocName As String
   Dim stLinkCriteria As String
   stDocName = "vlasnik vozila"
   DoCmd.OpenForm stDocName, , , stLinkCriteria
Exit_Command3_Click:
   Exit Sub

Err_Command3_Click:
   MsgBox Err.Description
   Resume Exit_Command3_Click

End Sub
```

Alati koje nudi Access kao relacioni sistem za upravljanje bazama podataka olakšavaju rad programera i projektanata, omogućavaju brzo i lako generisanje obrazaca koji imaju vrlo lep izgled, pregledni su i što je vrlo važno, uvek se mogu doraditi ukoliko smo nešto ispustili. Pri tome, najčešće ne moramo menjati ni logički projekat informacionog sistema (E-R ili relacioni model), ni fizičku implementaciju (bazu podataka).

Spisak primjera u SQL-u

Postavka zadataka

Primjer 1: U tabelu ODJELJENJE dodati atribute šef_od (šef odjeljenja), i br_zap (broj zaposlenih).

Primjer 2: U tabeli ODJELJENJE povećati dužinu atributa *br_zap* na 6 cifara.

Primjer 3: U tabeli ODJELJENJE kreirati indeks nad artibutom *imeod*.

Primjer 4: U tabeli ODJELJENJE kreirati primarni ključ - jedinstveni indeks nad artibutom *brod#*.

Primjer 5: U tabeli ODJELJENJE ukloniti indeks nad atributom *imeod*.

Primjer 6: Prikaži cjelokupni sadržaj tabele ODJELJENJE.

Primjer 7: Prikaži nazive svih odjeljenja u preduzeću.

Primjer 8: Prikaži nazive svih poslova u preduzeću.

Primjer 9: Prikaži nazive svih raznih poslova u preduzeću.

Primjer 10: Prikaži nazive svih raznih poslova u preduzeću.

Primjer 11: Prikaži šifre svih odjeljenja u preduzeću.

Primjer 12: Prikaži kvalifikaciju, platu i ime zaposlenih u odjeljenju 30.

Primjer 13: Prikaži ime, posao i platu zaposlenih u odjeljenju 30 čija je plata veća od 2000.

Primjer 14: Prikaži ime, posao upravnika i direktora.

Primjer 15: Prikaži ime i posao upravnika i direktora.

Primjer 16: Prikaži ime i posao upravnika i analitičara iz odjeljenja 10.

Primjer 17: Prikaži ime i platu zaposlenih čija je plata od 2600 do 3000.

Primjer 18: Prikaži ime i kvalifikaciju zaposlenih čija imena počinju slovom **p**.

Primjer 19: Prikaži ime, kvalifikaciju, platu i premiju uređenu:

a) po kvalifikaciji u opadajućem, po plati u rastućem, a po premiji u opadajućem redoslijedu,

b) po plati u rastućem, a po kvalifikaciji i premiji u opadajućem redoslijedu.

c) po premiji i kvalifikaciji u opadajućem, a po plati u rastućem redoslijedu.

Primjer 20: Prikaži ime, kvalifikaciju, platu i premiju zaposlenih koji:

a) imaju premiju.

b) nemaju premiju.

Primjer 21: Prikaži ime, kvalifikaciju, platu i premiju:

a) grupisanu po kvalifikaciji, po plati i po premiji,

b) uređenu po kvalifikaciji, plati i premiji u rastućem redosledu.

Primjer 22: Prikaži najmanju, najveću, srednju platu i broj zaposlenih:

a) u cjelom preduzeću,

b) u cjelom preduzeću, sa zaokrugljivanjem na dvije decimale,

c) po odjeljenjima,

Primjer 23: Izračunaj broj zaposlenih koji obavljaju različite poslove unutar svakog odjeljenja.

Primjer 24: Prikaži koje poslove obavljaja više od 1 radnika unutar svakog odjeljenja.

Primjer 25: Odrediti srednju godišnju platu unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

Primjer 26: Odrediti srednja godišnja primanja unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

Primjer 27: Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

Primjer 28: Izlistaj ime, posao i platu zaposlenih u odjeljenju *10* koji imaju isti posao kao zaposleni u odjeljenju *plan*.

Primjer 29: Ko su najbolje plaćeni radnici u svakom odjeljenju.

Primjer 30: Izlistaj ime, posao, ime odjeljenja i mjesto gdje rade svi zaposleni.

Primjer 31: Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

Primjer 32: Izlistaj sve podatke o odjeljenjima i radnicima za radnike čija je premija veća od 2000 .

Primjer 33: Izlistaj sve podatke o odjeljenjima i radnicima za odjeljenja čija imena počinju slovima **d** i **e**.

Primjer 34: Prikaži imena, posao i broj odjeljenja radnika kojima je rukovodilac *Savo*.

Primjer 35: Izlistaj šifre radnika koji rade na dva i više projekta.

Primjer 36: Izlistaj broj sati, šifru, ime i platu radnika koji rade na dva i više projekta.

Primjer 37: Izlistaj šifru radnika i broj sati na projektu radnika koji rade na projektu *uvoz*.

Primjer 38: Izlistaj šifru odjeljenja kao i šifru, ime, platu i posao najbolje plaćenog radnika u svakom odjeljenju.

Primjer 39: Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.

Primjer 40: Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.

Primjer 41: U relaciju **RADNIK** dodaj podatke o novom zaposlenom koji ima šifru **7891,** ime mu je "**Mirko"**, radi na poslovima **analitičara**, ima visoku stručnu spremu (**VSS**), još nema šefa(Null), počeo je da radi 29-jun-02, ima premiju 3000, platu 2000 i još nije raspoređen ni u jedno odjeljenje (Null).

Primjer 42: U relaciju RADNIK dodaj podatke o novom zaposlenom, ime mu je "**Mirko"**, koji ima šifru **7891,** ima visoku stručnu spremu (**VSS**), počeo je da radi na dan 29-jun-02.

Primjer 43: U relaciju *POVISICA***<***idbr#, povišica, datzap***>** dodati podatke o analitičarima i vozačima i dati im povećanje plate od 15%.

Primjer 44: U relaciji RADNIK obriši sve podatke o radniku **idbr# =** *5953* koji je otišao u penziju.

Primjer 45: Izbrisati podatke o radnicima koji su se zaposlili posle 17-feb-90.

Primjer 46: Izbrisati podatke o svim radnicima koji rade u odjeljenju *priprema*.

Primjer 47: Premesti zaposlenog čija je šifra **idbr# = *5932*** kod novog rukovodioca čija šifra je *5842*,.

Primjer 48: Prebaci sve zaposlene iz odjeljenja *30* u odjeljenje *20*, a naredba glasi:

Primjer 49: Kreirati presjek relacija RADNIK i RADNIK20 (tabela sa istim atributima kao radnik ali sadrži samo zapise o radnicima iz odjeljenja 20).

Primjer 50: Kreirati razliku relacija RADNIK i RADNIK20 (tabela sa istim atributima kao radnik ali sadrži samo zapise o radnicima iz odjeljenja 20).

Primjer 51: Kreirati pogled ODJELJENJE20 koje sadrži podatke o odjeljenju i ime, posao, kvalifikaciju i platu zaposlenih u odjeljenju *20*:

Primjer 52: Prikaži ime, posao, kvalifikaciju i mjesto gdje rade zaposleni sa visokom stručnom spremom u odjeljenju *20*.

Rješenja zadataka

Primjer 1 (str. 88): U tabelu ODJELJENJE dodati atribute šef_od (šef odjeljenja), i br_zap (broj zaposlenih).

```
ALTER TABLE ODJELJENJE
ADD (šef_od  INTEGER, br_zap NUMBER(2));
```

Primjer 2 (str. 88): U tabeli ODJELJENJE povećati dužinu atributa *br_zap* na 6 cifara.

```
ALTER TABLE ODJELJENJE
MODIFY (br_zap NUMBER(6));
```

Primjer 3 (str. 88): U tabeli ODJELJENJE kreirati indeks nad artibutom *imeod*.

```
CREATE INDEX naziv_ind
ON ODJELJENJE (imeod);
```

Primjer 4 (str. 89): U tabeli ODJELJENJE kreirati primarni ključ - jedinstveni indeks nad artibutom *brod#*.

```
CREATE UNIQUE INDEX brod_ind
ON ODJELJENJE (brod#);
```

Primjer 5 (str. 89): U tabeli ODJELJENJE ukloniti indeks nad atributom *imeod*.

```
DROP INDEX naziv_ind
```

Primjer 6 (str. 93): Prikaži cjelokupni sadržaj tabele ODJELJENJE.

```
SELECT * FROM ODJELJENJE;
```

Primjer 7 (str. 94): Prikaži nazive svih odjeljenja u preduzeću.

```
SELECT imeod  FROM ODJELJENJE;
```

Primjer 8 (str. 94): Prikaži nazive svih poslova u preduzeću.

```
SELECT posao  FROM RADNIK;
```

Primjer 9 (str. 95): Prikaži nazive svih raznih poslova u preduzeću.

```
SELECT  DISTINCT  posao   FROM RADNIK;

SELECT  UNIQUE   posao   FROM RADNIK;
```

Primjer 10 (str. 95): Prikaži nazive svih raznih poslova u preduzeću.

```
SELECT DISTINCT radnik.posao  FROM RADNIK ;
SELECT DISTINCT r.posao   FROM RADNIK R;
```

Primjer 11 (str. 96): Prikaži šifre svih odjeljenja u preduzeću.

a) Ova informacija može se dobiti iz tabele RADNIK:

```
SELECT DISTINCT radnik.[brod$] AS "sifre odjeljenja"  FROM RADNIK;
```

b) Isti podaci mogu se dobiti iz tabele ODJELJENJE:

```
SELECT odjeljenje.[brod#] AS  sifra  FROM ODJELJENJE ;
```

Primjer 12 (str. 97): Prikaži kvalifikaciju, platu i ime zaposlenih u odjeljenju 30.

```
SELECT R.kvalif, R.plata, R.ime, R.[brod$]
FROM RADNIK R
WHERE R.[brod$]=30;
```

Primjer 13 (str. 98): Prikaži ime, posao i platu zaposlenih u odjeljenju 30 čija je plata veća od 2000.

```
SELECT R.ime, R.posao, R.plata, R.[brod$]
FROM RADNIK R
WHERE R.[brod$]=30 AND plata>2000;
```

Primjer 14 (str. 98): Prikaži ime, posao upravnika i direktora.

```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE (R.posao="direktor") OR  (R.posao="upravnik");
```

Primjer 15 (str. 98): Prikaži ime i posao upravnika i direktora.

```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE R.posao IN ("direktor", "upravnik");
```

Primjer 16 (str. 98): Prikaži ime i posao upravnika i analitičara iz odjeljenja 10.

a)
```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
  WHERE R.posao="upravnik" OR  R.posao="analiticar" AND R.[brod$]=10;
```

b)
```
SELECT R.ime, R.posao, R.[brod$]
FROM RADNIK R
WHERE (R.posao="upravnik" OR  R.posao="analiticar") AND R.[brod$]=10;
```

Primjer 17 (str. 99): Prikaži ime i platu zaposlenih čija je plata od 2600 do 3000.

```
SELECT R.ime, R.plata
FROM RADNIK R
WHERE R.plata >= 2600 AND R.plata <=3000;

SELECT ime, plata
FROM RADNIK
WHERE plata BETWEEN 2600 AND 3000;
```

Primjer 18 (str. 99): Prikaži ime i kvalifikaciju zaposlenih čija imena počinju slovom **p**.

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE 'p%';
```

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE 'p%'
ORDER BY R.kvalif;
```

```
SELECT R.ime, R.kvalif
FROM Radnik R
WHERE R.ime LIKE 'p%'
ORDER BY R.ime;
```

Primjer 19 (str. 100): Prikaži ime, kvalifikaciju, platu i premiju uređenu:

d) po kvalifikaciji u opadajućem, po plati u rastućem, a po premiji u opadajućem redoslijedu,

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY kvalif DESC , plata, premija DESC;
```

e) po plati u rastućem, a po kvalifikaciji i premiji u opadajućem redoslijedu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY plata, kvalif DESC, premija DESC;
```

f) po premiji i kvalifikaciji u opadajućem, a po plati u rastućem redoslijedu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY premija DESC, kvalif DESC, plata;
```

Primjer 20 (str. 102): Prikaži ime, kvalifikaciju, platu i premiju zaposlenih koji:

a) imaju premiju.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
WHERE premija IS NOT NULL;
```

b) nemaju premiju.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
WHERE premija IS NULL;
```

Primjer 21 (str. 103): Prikaži ime, kvalifikaciju, platu i premiju:

c) grupisanu po kvalifikaciji, po plati i po premiji,

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
GROUP BY kvalif, plata, premija;
```

d) uređenu po kvalifikaciji, plati i premiji u rastućem redosledu.

```
SELECT ime, kvalif, plata, premija
FROM RADNIK
ORDER BY kvalif, plata, premija;
```

Primjer 22 (str. 104): Prikaži najmanju, najveću, srednju platu i broj zaposlenih:

a) u cjelom preduzeću,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
       AVG(plata) AS srednja, COUNT(*) AS broj
FROM RADNIK;
```

b) u cjelom preduzeću, sa zaokrugljivanjem na dvije decimale,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
                ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS  broj
FROM RADNIK;
```

c) po odjeljenjima,

```
SELECT MIN(plata) AS najmanja, MAX(plata) AS najveca,
          ROUND(AVG(plata), 2) AS srednja, COUNT(*) AS  broj
FROM RADNIK
GROUP BY brod$;
```

Primjer 23 (str. 105): Izračunaj broj zaposlenih koji obavljaju različite poslove unutar svakog odjeljenja.

```
SELECT brod$, posao, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK
GROUP BY brod$, posao;
```

Primjer 24 (str. 105): Prikaži koje poslove obavljaja više od 1 radnika unutar svakog odjeljenja.

```
SELECT brod$, posao, COUNT(*) AS  [broj zaposlenih]
FROM RADNIK
GROUP BY brod$, posao
HAVING COUNT>1;
```

Primjer 25 (str. 106): Odrediti srednju godišnju platu unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

```
SELECT brod$, AVG(plata)*12 AS  [prosek plata]
FROM RADNIK
WHERE posao NOT IN ('direktor', 'upravnik')
GROUP BY brod$;
```

Primjer 26 (str. 106): Odrediti srednja godišnja primanja unutar svakog odjeljenja ne uzimajući u obzir plate direktora i upravnika.

a)
```
SELECT brod$, AVG(plata +NVL(premija,0))*12 AS  [prosek primanja], COUNT(*) AS
        [broj zaposlenih], SUM(plata + NVL(premija,0))*12 AS [ukupni prihod]
FROM RADNIK
WHERE posao NOT IN ('direktor', 'upravnik')
GROUP BY brod$;
```

b)
```
SELECT brod$, AVG(plata +premija)*12 AS  [prosek primanja], COUNT(*)AS  [broj zaposlenih],
   SUM(plata + (premija)*12 AS [ukupni prihod], COUNT(premija) AS [sa premijom]
FROM RADNIK
WHERE posao NOT IN ('direktor', 'upravnik')
GROUP BY brod$;
```

brod$	prosek primanja	broj zapos	ukupni prihod
10	26400	6	158400
20	38400	4	153600
30	20400	2	40800
40	39960	2	79920

brod$	prosek primanja	broj zapos	ukupni prihod	sa premijom
10	26400	6	158400	6
20	40800	4	122400	3
30	20400	2	40800	2
40	46920	2	46920	1

Primjer 27 (str. 109): Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

I)
```
SELECT brod#
FROM ODJELJENJE
WHERE mesto='Dorcol';
```

II)
```
SELECT ime, brod$
FROM RADNIK
WHERE brod$=20;
```

III)
```
SELECT ime, brod$
FROM  RADNIK
WHERE  RADNIK.[BROD$]  =  (SELECT ODJELJENJE.[BROD#]
                           FROM ODJELJENJE
                           WHERE ODJELJENJE.mesto='Dorcol');
```

Primjer 28 (str. 109): Izlistaj ime, posao i platu zaposlenih u odjeljenju *10* koji imaju isti posao kao zaposleni u odjeljenju *plan*.

```
SELECT ime, posao, plata
FROM RADNIK
WHERE  brod$=10  AND  Posao  IN  (SELECT posao
                                  FROM RADNIK
                                  WHERE brod$ IN (SELECT brod#
                                                  FROM ODJELJENJE
                                                  WHERE imeod='plan'));
```

Primjer 29 (str. 110): Ko su najbolje plaćeni radnici u svakom odjeljenju.

```
SELECT ime, brod$, posao, plata
FROM RADNIK
WHERE  (brod$, plata)  IN  (SELECT brod$, MAX(plata)
                            FROM RADNIK
                            GROUP BY brod$ );
```

Primjer 30 (str. 112): Izlistaj ime, posao, ime odjeljenja i mjesto gdje rade svi zaposleni.

```
SELECT ime,  posao, RADNIK.[brod$], ODJELJENJE.[brod#], imeod, mesto
FROM ODJELJENJE,  RADNIK
WHERE ODJELJENJE.[brod#] = RADNIK.[brod$];
```

Primjer 31 (str. 113): Izlistaj spisak imena zaposlenih koji rade na Dorćolu.

```
SELECT ime, brod$
FROM RADNIK, ODJELJENJE
WHERE RADNIK.[brod$]= ODJELJENJE.[brod#]  AND mesto='Dorcol' ;
```

Primjer 32 (str. 114): Izlistaj sve podatke o odjeljenjima i radnicima za radnike čija je premija >2000 .

```
SELECT *
FROM ODJELJENJE,  RADNIK
WHERE ODJELJENJE.[brod#] = RADNIK.[brod$](+) AND  premija >2000;
```

Primjer 33 (str. 114): Izlistaj sve podatke o odjeljenjima i radnicima za odjeljenja čija imena počinju slovima **d** i **e**.

```
SELECT *
FROM ODJELJENJE,  RADNIK
WHERE  (ODJELJENJE.[brod#](+) = RADNIK.[brod$])  AND
          (imeod BETWEEN 'd%'  AND 'f%');
```

Primjer 34 (str. 114): Prikaži imena, posao i broj odjeljenja radnika kojima je rukovodilac *Savo*.

```
SELECT R.ime AS [ime radnika], R.posao, ŠEF.ime AS [ime šefa], R.brod
      FROM RADNIK R,  RADNIK ŠEF
      WHERE ŠEF.idbr = R.rukovodilac AND ŠEF.ime="Savo";
```

Primjer 35 (str. 114): Izlistaj šifre radnika koji rade na dva i više projekta.

```
SELECT Ucesce.Idbr#, COUNT(Idbr) AS [Broj projekata]
FROM Ucesce
GROUP BY Ucesce.Idbr#
HAVING (((Count(*))>=2))
ORDER BY Ucesce.Idbr#;
```

Primjer 36 (str. 115): Izlistaj broj sati, šifru, ime i platu radnika koji rade na dva i više projekta.

```
SELECT UCESCE.idbr#, sum(UCESCE.brsati) AS [broj sati],
       RADNIK.idbr#, RADNIK.ime, RADNIK.plata, Count(*) AS [broj projekata]
FROM RADNIK, UCESCE
WHERE RADNIK.IDBR# = UCESCE.IDBR#
GROUP BY radnik.idbr#, RADNIK.IME
HAVING (((Count(*))>1));
```

Primjer 37 (str. 115): Izlistaj šifru radnika i broj sati na projektu radnika koji rade na projektu *uvoz*.

```
SELECT DISTINCT UCESCE.idbr#, UCESCE.brproj#, UCESCE.brsati, PROJEKAT.brproj#,
                PROJEKAT.imeproj
FROM PROJEKAT, UCESCE
WHERE PROJEKAT.brproj# = UCESCE.brproj# AND
            (UCESCE.brproj# = (SELECT PROJEKAT.brproj#
                              FROM PROJEKAT
                              WHERE PROJEKAT.imeproj = 'uvoz'));
```

Primjer 38 (str. 115): Izlistaj šifru odjeljenja kao i šifru, ime, platu i posao najbolje plaćenog radnika u svakom odjeljenju.

```
SQL - SELECT RADNIK.brod# AS BROD, first( RADNIK.idbr#) as [šifra],
          first( RADNIK.ime) as [ime], Max(RADNIK.plata) AS [najveća plata],
          first(RADNIK.posao) as [posao]
      FROM RADNIK
      GROUP BY RADNIK.brod;
```

Primjer 39 (str. 116): Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.

```
QBE - SELECT ODJELJENJE.brod, imeod AS [ime odjeljenja], Max(plata) AS [najveća plata],
          First(RADNIK.idbr) AS [šifra radnika], First(ime) AS [ime radnika], First(posao) AS
      posao
      FROM ODJELJENJE INNER JOIN RADNIK ON
          ODJELJENJE.BROD = RADNIK.BROD
      GROUP BY ODJELJENJE.brod, imeod;
```

Primjer 40 (str. 117): Izlistaj šifru i ime odjeljenja kao i platu, šifru, ime i posao najbolje plaćenog radnika u svakom odjeljenju.

```
SQL   SELECT ODJELJENJE.brod#, imeod AS [ime odjeljenja], Max(plata) AS [najveća plata],
          First(RADNIK.idbr) AS [šifra radnika], First(ime) AS [ime radnika], posao
      FROM ODJELJENJE, RADNIK
      WHERE ODJELJENJE.brod# = RADNIK.brod$
      GROUP BY ODJELJENJE.brod#, imeod;
```

Primjer 41 (str. 118): U relaciju **RADNIK** dodaj podatke o novom zaposlenom koji ima šifru **7891,** ime mu je "**Mirko"**, radi na poslovima **analitičara**, ima visoku stručnu spremu (**VSS**), još nema šefa(Null), počeo je da radi 29-jun-02, ima premiju 3000, platu 2000 i još nije raspoređen ni u jedno odjeljenje (Null).

```
INSERT INTO RADNIK VALUES
( 7891, Mirko, analiticar, VSS, Null, 29-jun-02, 3000, 2000, Null);
```

Primjer 42 (str. 119): U relaciju RADNIK dodaj podatke o novom zaposlenom, ime mu je "**Mirko"**, koji ima šifru **7891,** ima visoku stručnu spremu (**VSS**), počeo je da radi na dan 29-jun-02.

```
INSERT INTO RADNIK (ime, idbr#, kvalif, datzap)
VALUES (Mirko, 7891, VSS, 29-jun-02);
```

Primjer 43 (str. 119): U relaciju *POVISICA<idbr#, povišica, datzap>* dodati podatke o analitičarima i vozačima i dati im povećanje plate od 15%.

```
INSERT INTO POVISICA (idbr#, povišica, datzap)
SELECT idbr#, plata*1.15, datzap
FROM RADNIK
WHERE posao IN ('analiticar', 'vozac');
```

Primjer 44 (str. 119): U relaciji RADNIK obriši sve podatke o radniku **idbr# =** *5953* koji je otišao u penziju.

```
DELETE FROM RADNIK WHERE idbr# = 5953;
```

Primjer 45 (str. 119): Izbrisati podatke o radnicima koji su se zaposlili posle 17-feb-90.

```
DELETE FROM RADNIK  WHERE (RADNIK.datzap) <  ' 17-feb-90 ';
```

Primjer 46 (str. 119): Izbrisati podatke o svim radnicima koji rade u odjeljenju *priprema*.

```
DELETE  FROM   RADNIK
WHERE  (RADNIK.brod$)  =   ( SELECT brod#
                             FROM ODJELJENJE
                             WHERE imeod= 'priprema');
```

Primjer 47 (str. 120): Premesti zaposlenog čija je šifra **idbr# = *5932*** kod novog rukovodioca čija šifra je *5842*,.

```
UPDATE RADNIK SET rukovodilac= 5842 WHERE  idbr# = 5932;
```

Primjer 48 (str. 120): Prebaci sve zaposlene iz odjeljenja *30* u odjeljenje *20*, a naredba glasi:

```
UPDATE RADNIK SET brod$= 20  WHERE  brod$ = 30;
```

Primjer 49 (str. 121): Kreirati presjek relacija RADNIK i RADNIK20.

```
SELECT R.IDBR#, R.IME, R.BROD$, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON RADNIK.IDBR# = radnik20.idbr
WHERE radnik20.idbr IS NOT NULL;
```

Primjer 50 (str. 121): Kreirati razliku relacija RADNIK i RADNIK20 (tabela sa istim atributima kao radnik ali sadrži samo zapise o radnicima iz odjeljenja 20).

```
SELECT R.IDBR#, R.IME, R.BROD$, radnik20.idbr, radnik20.ime, radnik20.brod
FROM RADNIK R LEFT JOIN radnik20 ON RADNIK.IDBR# = radnik20.idbr
WHERE radnik20.idbr IS NULL;
```

Primjer 51 (str. 123): Kreirati pogled ODJELJENJE20 koje sadrži podatke o odjeljenju i ime, posao, kvalifikaciju i platu zaposlenih u odjeljenju *20*:

```
CREATE VIEW ODJELJENJE20 AS
SELECT O.imeod, O.imeod, R. ime, R.posao, R.kavalif, R.plata
FROM RADNIK R, ODJELJENJE O
WHERE  O.brod#=R.brod$ AND O.brod#=20;
```

Primjer 52 (str. 123): Prikaži ime, posao, kvalifikaciju i mjesto gdje rade zaposleni sa visokom stručnom spremom u odjeljenju *20*.

a)

```
SELECT O20.mesto, O20.ime, O20.posao, O20.kvalif
FROM ODJELJENJE20 O20
WHERE O20.kvalif="VSS";
```

b)

```
SELECT mesto, ime, posao, kvalif
FROM ODJELJENJE O  INNER JOIN RADNIK R   ON
        ODJELJENJE.brod = RADNIK.brod
WHERE (((RADNIK.kvalif)="vss") AND ((RADNIK.brod)=20));
```

Literatura:

1. E. F. Codd *"A Relation Model of Data for Large Shared Data Banks" Communications of the ACM,* Volume 13, Number 6, (June 1970), pages 377-387.

   U ovome radu Codd je prvi definisao principe i strukturu relacionog modela.

2. Chen P. *"The Entity-Relationship Model: Toward a Unified View of Data" ACM Transactions on Database Systems,* Volume 1, Number 1, (January 1976), pages 9-36.

   U radu su prikazani principi i struktura E-R modela

3. Alagić S. *Relacione baze podataka*, "Svjetlost" , Sarajevo, 1984.
   Jedna od prvih knjiga iz ove oblasti na našem jeziku. Nedovoljna za dobijanje potpune slike o današnjem stanju.

4. Radovan M. *Projektiranje informacijskih sistema* "Informator", Zagreb 1989.

   Sveobuhvatan prilaz projektovanju informacionih sistema bez ulaženja u način obrade podataka.

5. Marjanović Z. *ORACLE relacioni sistem za upravljanje bazom podataka* "Breza", Ljig 1990.

   Vrijednost knjige je u tome što se u njoj mogu naći i elementi programskog jezika SQL, mnoštvo upita i elementi projektovanja baze podataka, kao i niz primjera – aplikativnih programa.

6. Simić R. *Organizacija podataka* "Naučna knjiga", Beograd,1990.
   Pregledan uvod u tehniku prikupljanja i organizacije podataka.

7. Mišić V. *Relaciona baza podataka Rdb/VMS* "Tehnička knjiga", Beograd, 1990
   Autor se bavi problemima programiranja i razlikama između verzija SQL-a.

8. Filipović M. *dBASE III+ priručnik* "Boris Kidrič" Vinča, Beograd. 1991
   U knjizi se mogu naći elementi programskog jezika, elementi projektovanja baze podataka, kao i niz praktičnih aplikativnih primjera – programa.

9. Bobrowski S. *Oracle 7 i obrada podataka po modelu klijent/server* "Mikro knjiga", Beograd, 1995.

Autor se bavi problemima obrade projektovanja i administriranje baza podataka primenom SQL-a.

10. Vujnović R. *SQL i relacijski model podataka* "Znak", Zagreb, 1995.

Paralelno se analizira modeliranje podataka i njihova obrada, pomoću SQL-a.

11. Wynkoop S. *Vodič kroz SQL Server 7.0*, "CET", Beograd, 1999.

Veoma koristan priručnik kako za početnike tako i za iskusne projektante, naročito pogodan za administratore Microsoft RDBMS SQL Servera.

12. Obradović S. i grupa autora *MS-Access Projektovanje baza podataka i aplikacija* "Viša elektrotehnička škola", Beograd, 2006.

Knjiga posvećena programiranju u Accessu 2003 sa osvrtom na principe projektovanja relacionih baza podataka

13. Grupa autora *Access 2003 korak po korak* "CET", Beograd, 2004.

Knjiga posvećena programiranju u Accessu 2003 bez ulaženja u principe projektovanja relacionih baza podataka

14. Дончев А., Обрадовић С. и група аутора *База от данни* "Технически университет - Габрово", Габрово, Бугарска, 2004.

Knjiga posvećena programiranju u Accessu 2003 sa osvrtom na principe projektovanja relacionih baza podataka

15. Riordan R. *Projektovanje baza podataka* "Mikro knjiga", Beograd, 2006.

Knjiga posvećena projektovanju baza podataka bez ulaženja u detalje implementacije na nekom RDBMS ali sa osvrtom na Microsoft-ove sisteme MS Access i SQL Server

Sadržaj

I DIO - SINTEZA LOGIČKOG MODELA

1. Uvodna razmatranja 1
1.1 Pojam podatka 3
1.2 Objekat posmatranja - entitet 4
1.3 Model podataka 8
1.3.1 Hijerarhijski model 9
1.3.2 Mrežni model 14
1.3.3 Relacioni model 15

2. Relacioni model **19**
2.1 Pojam relacije 20
2.2 Entitet 22
2.3 Atribut 22
2.4 Domen 23
2.5 Primarni ključ 23
2.6 Spoljnji ključ 24
2.7 Coddova pravila 25
2.8 Distribuirane baze podataka 28

3. Osnove relacione algebre **31**
3.1 Tradicionalni operatori pogodni za ažuriranje 31
3.1.1 Unija 32
3.1.2 Presjek 33
3.1.3 Razlika 33
3.1.4 Proizvod 33
3.2 Specijalni operatori pogodni za izvještavanje 34
3.2.1 Selekcija 34
3.2.2 Projekcija 35
3.2.3 Spajanje (Join) 35
3.2.4 Operacija dijeljenja 36
3.3 Dodatni operatori 37

4. Sinteza relacionog modela **39**
4.1 E-R model 40
4.1.1 Osnovne definicije i pojmovi E-R modela 40
4.1.2 Objekti 40

4.1.3 Veze 42
4.1.4 Vezni objekti 42
4.1.5 Osobine veza 44
4.2 Prevođenje ER-modela na relacioni oblik 50
4.2.1 Prevođenje binarnih veza na relacioni oblik 50
4.2.2 Prevođenje unarnih veza na relacioni oblik 52
4.2.3 Prevođenje veza reda većeg od dva 53
4.3 Normalizacija 55
4.3.1 Prva, druga i treća normalna forma 57
4.3.2 Problem gubitka informacija 60
4.3.3 Ostale normalne forme 60
4.3.4 Primjeri normalizacije 61

5. Osnove projektovanja 65
5.0 Osnove projektovanja 66
5.1 Izvori podataka - informacija 66
5.2 Izbor metoda 69
5.3 Izrada modela 69
5.4 Analiza postojećeg sistema 69
5.5 Projektovanje novog sistema 71
5.6 Realizacija sistema 74
5.6 Vođenje i vrednovanje projekta 75

II DIO - SINTEZA FIZIČKOG MODELA

6. SQL 79
6.1 Naredbe za definisanje podataka 83
6.1.1 Izmjena vrijednosti atributa, dodavanje novog atributa u postojeću tabelu - ALTER TABLE 88
6.1.2 Izbacivanje relacije iz baze podataka - DROP TABLE 88
6.1.3 Indeksi 88
6.2 Naredbe za rukovanje podacima 91
6.2.1 Upiti nad jednom tabelom 93
Odredba WHERE 96
Odredba ORDER BY 99
Upotreba NULL vrijednosti 101
Odredba GROUP BY 102
6.2.2 Upiti nad jednom tabelom sa izračunavanjem novih vrijednosti 103
6.2.3 Upiti nad jednom relacijom kao argument u upitu nad drugom relacijom – ugnježđeni upiti 108

6.2.4 Spajanje relacija 110
6.2.5 Ažuriranje baze podataka 118
6.2.6 Operatori za rad sa skupovima 120
6.3 Kreiranje i korišćenje pogleda (VIEW) 122
6.4 Upravljačke naredbe 124
6.5 Aplikativni programi 127
6.6 Primjer razvoja aplikacije 129

7. Primjeri 135
7.1 Informacioni sistem sekretarijata za saobraćaj 135
7.2 Dio informacionog sistema advokatske kancelarije 136
7.3 Informacioni sistem sportskog saveza 137
7.4 Informacioni sistem školske biblioteke 140
7.5 Dio informacionog sistema studentske službe 144
7.6 Dio informacionog sistema kliničkog centra 146
7.7 Informacioni sistem video kluba 147
7.8 Dio informacionog sistema lanca robnih kuća 149
7.9 Informacioni sistem sportskog takmičenja 151
7.10 Primjer razvoja aplikacije u dBase IV 152
7.11 Primjer razvoja aplikacije Accessu 155

Zadaci 163

Literatura 173