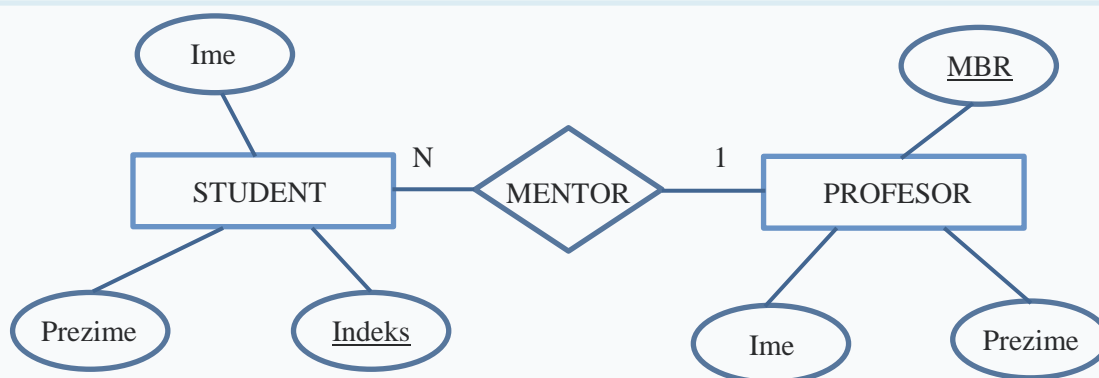


Univerzitet u Nišu  
Elektronski fakultet

Leonid Stoimenov

# UVOD U BAZE PODATAKA



Ime	Prezime	MBR
Milan	Petrović	123456
Milan	Ilić	654321
Aleksandar	Jovanović	345612

Ime	Prezime	Indeks	MentorMBR
Petar	Petrović	1111	123456
Milan	Milanović	2222	654321
Jovan	Jovanović	3333	654321

```
SELECT STUDENT.Ime, STUDENT.Prezime
FROM STUDENT, PROFESOR
WHERE MentorMBR=MBR AND
PROFESOR.Ime="Petar";
```

Edicija: Udžbenici



**Univerzitet u Nišu**  
**Elektronski fakultet**

---

**Leonid Stoimenov**

# **Uvod u Baze podataka**

---

**Edicija: Udžbenici**

---

**Niš, 2013**

Prof. dr Leonid Stoimenov  
UVOD U BAZE PODATAKA

*Izdavač:* Elektronski fakultet u Nišu  
P. fah 73, 18000 Niš  
<http://www.elfak.ni.ac.yu>

*Recenzenti:* Prof. dr Ivan Luković  
Prof. dr Milena Stanković

*Glavni i odgovorni urednik:* Prof. dr Dragan Tasić

Odlukom Nastavno-naučnog veća Elektronskog fakulteta u Nišu, br. 07/05-004/13-003 od 31.10.2013., rukopis je odobren za štampu kao univerzitetski udžbenik.

**ISBN 978-86-6125-099-6**

CIP - Каталогизација у публикацији  
Народна библиотека Србије, Београд

Preštampavanje ili umnožavanje ove knjige nije dozvoljeno bez pismene dozvole izdavača.

Tiraž: 300 primeraka

---

Štampa: Elektronski fakultet, Niš

# Predgovor

Baze podataka su, po mišljenju autora, jedna od značajnijih oblasti računarstva i informatike. Posledica toga je postojanje odgovarajućih kurseva na tehničkim ali i ne-tehničkim fakultetima koji se bave različitim aspektima baza podataka – od osnovnih kurseva za početnike, do naprednih kurseva koji se bave specijalizovanim temama iz ove široke oblasti.

## Namena udžbenika

Navedeni materijal je u skladu sa planom i programom predmeta *Baze podataka* na II godini osnovnih akademskih studija na Elektronskom fakultetu u Nišu i u potpunosti pokriva sve teme sa predavanja i vežbi na modulu Računarstvo i informatika. Udžbenik je namenjen studentima koji se prvi put susreću sa bazama podataka i ne zahteva nikakvo posebno predznanje osim opšteg znanja o računarstvu i informatici sa I godine studija. Udžbenik se može koristiti i na drugim modulima ovih studija. Udžbenik sadrži koncizan prikaz teorijske osnove baza podataka, brojne primere i zadatke za samostalni rad studenata. Kao takav, on predstavlja materijal za spremanje kompletnog ispita iz predmeta *Baze podataka*.

## Struktura udžbenika

Udžbenik sadrži osam poglavlja i literaturu. Struktura i redosled poglavlja je takav da prati teme koje se obrađuju na predavanjima i vežbama iz uvodnog kursa Baze podataka. Svako poglavlje sadrži veliki broj primera koji treba da pomognu studentima u savladavanju gradiva.

U Poglavlju 1, pod naslovom *Baze podataka – osnovni pojmovi*, dat je kratak pregled osnovnih pojmova iz oblasti baza podataka, pregled i poređenje dva pristupa: konvencionalne obrade zasnovane na datotekama i korišćenje baza podataka.

U Poglavlju 2, pod naslovom *Modeli podataka i proces projektovanja baze podataka*, opisani su nivoi apstrakcije kod DBMSa, opisani modeli podataka na različitim nivoima apstrakcije i njihovi osnovni koncepti, i opisan generalni postupak projektovanja baza podataka.

U Poglavlju 3, pod naslovom *Konceptualno projektovanje baze podataka*, prikazani su osnovni koncepti ER i EER, opisan je proces projektovanja odnosno korišćenja ovih koncepata. Dat je pregled grafičke notacije za ER i EER dijagrame, i date su preporuke za izvođenje ER i EER modeliranja. U ovom poglavlju su dati primeri projektovanja baza podataka.

U Poglavlju 4, pod naslovom *Relacioni model podataka*, dat je prikaz koncepata relacionog modela i opis odgovarajuće terminologije. Dat je osvrt i na integritetnu komponentu ovog modela, kao i algoritam za preslikavanje ER i EER modela podataka u relacioni.

U Poglavlju 5, pod naslovom *Relaciona algebra*, dat je pregled osnovnih operacija relacione algebre i veliki broj primera korišćenja kao i upita relacione algebre.

U Poglavlju 6, pod naslovom *Normalizacija i normalne forme*, prikazane su funkcionalne zavisnosti i njihovo izvođenje, pojam ključa relacije, analiza šeme relacije i

## Uvod u baze podataka: Sadržaj

---

normalne forme. Osim definicije normalnih formi dati su primeri testiranja, opisan je i ilustrovan primerima postupak normalizacije.

U Poglavlju 7, pod naslovom *Upitni jezik SQL*, uz veliki broj primera i zadataka za vežbu prikazane su mogućnosti SQL jezika od naredbi za kreiranje podataka i modifikaciju šeme do naredbi za pretraživanje i manipulaciju podacima.

U Poglavlju 8, pod naslovom *Kreiranje korisničkih aplikacija: ADO.NET*, prikazane su osnove ADO.NETa, odgovarajućih provajdera podataka i opisan je način za direktan pristup podacima. Prikazani su i primeri za kreiranje konekcije ka bazi podataka, kreiranje SQL komande i korišćenje *datareaderi* objekta.

### Online materijal

Dodatni materijal kao podrška udžbeniku je dostupan studentima i može se naći na Moodle stranicama predmeta: <http://cs.elfak.ni.ac.rs/nastava>.

Plan i program predmeta *Baze podataka*, koji se sluša na II godini studija u u letnjem semestru, može se naći na Web-u, na adresi: <http://www.elfak.ni.ac.rs>, na stranici koja se odnosi na Akreditaciju 2013.godine.

### Zahvalnica

Autor se zahvaljuje recenzentima ovog praktikuma, prof.dr Ivanu Lukoviću i prof.dr Mileni Stanković, na trudu koji su uložili čitajući ovaj praktikum, zapažanjima, primedbama i korisnim sugestijama, koje su svakako doprinele kvalitetu ovog praktikuma.

Takođe, autor se zahvaljuje svojim saradnicima dr Aleksandru Stanimiroviću i Milošu Bogdanoviću koji su svojim dugogodišnjim radom uradili mnogo toga da ovaj kurs bude popularan među studentima, a posebno na trudu koji su uložili u pisanju poslednja dva poglavlja ovog materijala.

Na kraju, ali sa najviše emocija, autor se zahvaljuje prof.dr Slobodanki Đorđević-Kajan, profesorki Elektronskog fakulteta u penziji, koja je najzaslužnija za uvođenje ovog i drugih srodnih kurseva, i koja je bitno uticala na oblikovanje tema koje su prezentirane i u ovom udžbeniku.

U Nišu, avgust 2013. godine

Autor

# Sadržaj

<b>1</b>	<b>BAZE PODATAKA – OSNOVNI POJMOVI .....</b>	<b>1</b>
1.1	PODACI I INFORMACIJE .....	1
1.2	KONVENCIONALNI SISTEMI U ODNOSU NA BAZE PODATAKA .....	2
1.3	SISTEM BAZA PODATAKA .....	4
1.4	ŠTA JE BAZA PODATAKA? .....	5
1.5	ŠTA JE SISTEM ZA UPRAVLJANJE BAZAMA PODATAKA? .....	8
1.6	TIPOVI SISTEMA BAZA PODATAKA .....	11
1.7	ZANIMLJIVOSTI: ISTORIJA DBMSA .....	13
<b>2</b>	<b>MODELI PODATAKA I PROCES PROJEKTOVANJA BAZE PODATAKA ....</b>	<b>15</b>
2.1	NIVOI APSTRAKCIJE U DBMS-U .....	15
2.2	MODELI PODATAKA .....	17
2.3	PROJEKTOVANJE BAZE PODATAKA .....	18
<b>3</b>	<b>KONCEPTUALNO PROJEKTOVANJE BAZE PODATAKA .....</b>	<b>23</b>
3.1	ENTITETI, TIP ENTITETA I SKUP ENTITETA .....	24
3.1.1	<i>Atributi</i> .....	26
3.1.2	<i>Ključ tipa entiteta</i> .....	32
3.2	POVEZNICI, SKUP POVEZNIKA I TIP POVEZNIKA .....	34
3.3	OGRANIČENJA NAD TIPOM POVEZNIKA .....	40
3.4	SLABI TIP ENTITETA .....	46
3.5	PREGLED GRAFIČKE NOTACIJE ER DIJAGRAMA .....	47
3.6	PRIMERI KONCEPTUALNOG MODELIRANJA .....	49
3.6.1	<i>Preporuke kod modeliranja</i> .....	56
3.7	EER MODEL .....	57
3.7.1	<i>Koncepti EER modela podataka</i> .....	57
3.7.2	<i>Notacija i primeri korišćenja koncepata EER modela</i> .....	59
<b>4</b>	<b>RELACIONI MODEL PODATAKA .....</b>	<b>65</b>
4.1	OSNOVE RELACIONOG MODELA PODATAKA .....	65
4.2	KONCEPTI RELACIONOG MODELA PODATAKA .....	66
4.3	DOMENI, ATRIBUTI, TORKE I RELACIJE .....	66
4.3.1	<i>Svojstva šeme relacije i relacije</i> .....	70
4.4	KLJUČ ŠEME RELACIJE .....	70
4.4.1	<i>Pregled terminologije za ključeve</i> .....	72
4.5	ŠEMA I POJAVA RELACIONE BAZE PODATKA .....	73
4.5.1	<i>Predstavljanje šema relacione baze podataka</i> .....	75
4.6	OGRANIČENJA U RELACIONOM MODELU .....	76
4.6.1	<i>Specifikacija ograničenja u DBMSu</i> .....	78
4.7	OPERACIJSKA KOMPONENTA .....	83
4.8	PREGLED TERMINOLOGIJE OSNOVNIH KONCEPATA .....	83
4.9	PRESLIKAVANJE ER/EER MODELA U RELACIONI MODEL .....	84
<b>5</b>	<b>RELACIONA ALGEBRA .....</b>	<b>91</b>

5.1	SELEKCIJA.....	92
5.2	PROJEKCIJA .....	94
5.3	PREIMENOVANJE .....	96
5.4	OPERACIJE RELACIONE ALGEBRE IZ TEORIJE SKUPOVA .....	96
5.5	$\Theta$ -SPOJ .....	100
5.6	FUNKCIJE AGREGACIJE.....	103
5.7	PRIMERI UPITA U RELACIONOJ ALGEBRI .....	103
<b>6</b>	<b>NORMALIZACIJA I NORMALNE FORME.....</b>	<b>109</b>
6.1	FUNKCIONALNE ZAVISNOSTI.....	109
6.2	IZVOĐENJE FUNKCIONALNIH ZAVISNOSTI.....	112
6.2.1	<i>Zatvarač skupa funkcionalnih zavisnosti.....</i>	<i>112</i>
6.2.2	<i>Zatvarač skupa atributa.....</i>	<i>114</i>
6.3	KLJUČ ŠEME RELACIJE.....	115
6.4	ANALIZA ŠEME RELACIONE BAZE PODATAKA.....	120
6.5	NORMALNE FORME I NORMALIZACIJA .....	124
6.5.1	<i>Prva normalna forma .....</i>	<i>125</i>
6.5.2	<i>Druga normalna forma (2NF).....</i>	<i>127</i>
6.5.3	<i>Treća normalna forma (3NF) .....</i>	<i>131</i>
6.5.4	<i>Boyce-Codd-ova normalna forma (BCNF).....</i>	<i>134</i>
6.5.5	<i>Denormalizacija .....</i>	<i>138</i>
<b>7</b>	<b>UPITNI JEZIK SQL .....</b>	<b>141</b>
7.1	BAZA PODATAKA PREDUZEĆE.....	142
7.2	SQL NAREDBE ZA KREIRANJE PODATAKA .....	144
7.2.1	<i>Tipovi podataka .....</i>	<i>144</i>
7.2.2	<i>Ograničenja kolone .....</i>	<i>148</i>
7.2.3	<i>Ograničenja tabele .....</i>	<i>150</i>
7.3	MODIFIKACIJA ŠEME RELACIONE BAZE PODATAKA .....	154
7.3.1	<i>Brisanje tabele.....</i>	<i>154</i>
7.4	PRETRAŽIVANJE PODATAKA.....	156
7.4.1	<i>Naredba SELECT.....</i>	<i>156</i>
7.4.2	<i>Kaluzule SELECT i FROM.....</i>	<i>157</i>
7.4.3	<i>Klauzula WHERE .....</i>	<i>159</i>
7.4.4	<i>Klauzula ORDER BY.....</i>	<i>165</i>
7.4.5	<i>Aritmetičke funkcije .....</i>	<i>166</i>
7.4.6	<i>Funkcije za rad sa stringovima.....</i>	<i>168</i>
7.4.7	<i>Funkcije agregacije .....</i>	<i>169</i>
7.5	NAPREDNE KLAUZULE SELECT NAREDBE I SPAJANJE TABELA.....	170
7.5.1	<i>Spajanje tabela .....</i>	<i>170</i>
7.5.2	<i>Dekartov proizvod (cross-join).....</i>	<i>172</i>
7.5.3	<i>Unutrašnji spoj (inner-join).....</i>	<i>172</i>
7.5.4	<i>Levi spoljašnji spoj (left-outer join).....</i>	<i>173</i>
7.5.5	<i>Desni spoljašnji spoj (right-outer join) .....</i>	<i>174</i>
7.5.6	<i>Potpuni spoljašnji spoj (full-outer join).....</i>	<i>175</i>
7.5.7	<i>Klauzule GROUP BY i HAVING .....</i>	<i>178</i>
7.5.8	<i>Kombinovanje rezultata više SQL upita .....</i>	<i>181</i>
7.5.9	<i>Ugnježdeni upiti.....</i>	<i>185</i>



7.5.10	<i>Pseudo kolona ROWNUM</i> .....	189
7.5.11	<i>Klauzula CONNECT BY...START WITH</i> .....	191
7.6	SQL NAREDBE ZA MANIPULACIJU PODACIMA .....	192
7.6.1	<i>Dodavanje novih podataka</i> .....	193
7.6.2	<i>Ažuriranje podataka</i> .....	196
7.6.3	<i>Brisanje podataka</i> .....	198
7.7	NAREDBE ZA RAD SA POGLEDIMA .....	200
7.8	NAREDBE ZA RAD SA INDEKSIMA .....	203
<b>8</b>	<b>KREIRANJE KORISNIČKIH APLIKACIJA: ADO.NET</b> .....	<b>205</b>
8.1	OSNOVE ADO.NET-A.....	205
8.2	ADO.NET DATA PROVIDER-I.....	206
8.3	DIREKTAN PRISTUP PODACIMA KORIŠĆENJEM ADO.NET-A .....	209
8.4	KREIRANJE KONEKCIJE KA BAZI PODATAKA.....	209
8.5	KREIRANJE SQL KOMANDE.....	210
8.6	KORIŠĆENJE DATAREADER OBJEKTA.....	211
8.7	KORIŠĆENJE BEZKONEKCIONOG SLOJA ADO.NET-A ZA PRISTUP PODACIMA 216	
<b>9</b>	<b>LITERATURA</b> .....	<b>223</b>

## Spisak slika

Slika 1-1. Konvencionalni sistem zasnovan na datotekama .....	2
Slika 1-2. Obrada zasnovana na bazama podataka .....	4
Slika 1-3. Komponente sistema baza podataka.....	5
Slika 1-4. Tabela u bazi podataka sa označenom vrstom i kolonom .....	7
Slika 1-5. Međusobna povezanost podataka u bazi podataka .....	7
Slika 1-6. Personalni sistem baza podataka .....	11
Slika 1-7. <i>Enterprise</i> sistem baza podataka .....	12
Slika 2-1. Arhitektura tri šeme DBMSa.....	16
Slika 2-2. Faze projektovanja baza podataka.....	20
Slika 3-1. Konceptualno projektovanje baze podataka .....	23
Slika 3-2. Grafička notacija za tip entiteta.....	26
Slika 3-3. Predstavljanje atributa u ER dijagramu .....	28
Slika 3-4. Složeni (kompozitni) atributi u ER dijagramu .....	30
Slika 3-5. Grafička notacija za izvedeni atribut.....	30
Slika 3-6. Grafička notacija za viševrednosni atribut .....	31
Slika 3-7. Grafička notacija za prikaz tipa entiteta i njegovih atributa.....	33
Slika 3-8. Grafička notacija za tip poveznika .....	34
Slika 3-9. Ternarni tip poveznika SNABDEVA.....	36
Slika 3-10. Neke pojave tipa poveznika RADI-NA.....	36
Slika 3-11. Neke pojave tipa poveznika RUKOVODI .....	37
Slika 3-12. Neke pojave tipa poveznika JE_RUKOVODILAC .....	37
Slika 3-13. Neke pojave rekurzivnog tipa poveznika U-BRAKU .....	38
Slika 3-14. Neke pojave ternarnog tipa poveznika SNABDEVA .....	38
Slika 3-15. Prikaz uloge u ER modelu.....	39
Slika 3-16. Označavanje kardinaliteta .....	41
Slika 3-17. Veza 1:N između tipova entiteta RADNO-MESTO i RADNIK....	42
Slika 3-18. ER dijagram sa označenim kardinalitetima tipa poveznika.....	42
Slika 3-19. Označavanje participacije za tip veze RASPOREĐEN .....	43
Slika 3-20. Označavanje participacije za tip veze RADI-U.....	43
Slika 3-21. Označavanje strukturalnih ograničenja .....	45

Slika 3-22. Primer označavanja strukturalnih ograničenja .....	46
Slika 3-23. Slabi tip entiteta u ER dijagramu .....	46
Slika 3-24. Primer ER dijagrama: vlasnici i njihova vozila.....	51
Slika 3-25. ER dijagram baze podataka PREDUZEĆE .....	52
Slika 3-26. EER model baze podataka VIDEO_KLUB .....	54
Slika 3-27. ER dijagram baze podataka o autorima i njihovim delima .....	55
Slika 3-28. ER dijagram baze podataka o publikacijama .....	56
Slika 3-29. Primer specijalizacije .....	60
Slika 3-30. Primer generalizacije.....	61
Slika 3-31. Hijerarhija klasa .....	61
Slika 3-32. Primer deljive klase.....	62
Slika 3-33. Primer kategorije .....	62
Slika 3-34. EER dijagram za zadatak za vežbu .....	64
Slika 4-1. Tabela sa podacima koja odgovara relaciji <i>radnik</i> .....	70
Slika 4-2. Ostvarivanje veza preko stranih ključeva.....	72
Slika 4-3. Jedna pojava baze podataka PREDUZEĆE .....	74
Slika 4-4. Relaciona šema baze podataka PREDUZEĆE .....	76
Slika 4-5. Ekvivalentni skup pojmova kod baza podataka .....	83
Slika 4-6. Pregled osnovnih koncepata relacionog modela .....	84
Slika 4-7. ER dijagram baze podataka PREDUZEĆE .....	87
Slika 4-8. Relacioni model baze podataka PREDUZEĆE .....	89
Slika 4-9. Deo šeme koji sadrži ternarnu vezu SNABDEVA.....	89
Slika 4-10. Relacioni model baze podataka VIDEO_KLUB.....	90
Slika 5-1. Rezultat selekcije su izdvojene vrste relacije .....	92
Slika 5-2. Primer rezultata primene selekcije .....	94
Slika 5-3. Primer primene operacije projekcije .....	95
Slika 5-4. Rezultat projekcije nad relacijom <i>radnik</i> .....	96
Slika 5-5. Primer primene skupovnih operacija.....	98
Slika 5-6. Primer primene Dekartovog proizvoda .....	99
Slika 5-7. Primer primene operacije deljenja.....	99
Slika 5-8. Primer primene operacije spoja.....	101
Slika 5-9. Primer primene prirodnog spoja.....	102
Slika 6-1. Primer loše projektovane relacije .....	121
Slika 6-2. Nenormalizovana relacija.....	126
Slika 6-3. Relacije nakon normalizacije do 2NF .....	131

Slika 6-4. Ilustracija parcijalnih i tranzitivnih zavisnosti .....	132
Slika 6-5. Relacije nakon normalizacije do 3NF .....	134
Slika 6-6. Relacija KlijentIntervju .....	138
Slika 6-7. Relacija KlijentIntervju nakon normalizacije.....	138
Slika 7-1. Relacija KlijentIntervju nakon normalizacije.....	143
Slika 8-1. Struktura Data Provider-a .NET platforme .....	207
Slika 8-2. Bezkonekcioni pristup podacima .....	217

# 1 BAZE PODATAKA – OSNOVNI POJMOVI

Moderne kompanije i institucije poseduju različite elektronske (računarske, informacione) sisteme koje koriste kao podršku u procesu obrade informacija, koje nastaju kako unutar samog sistema tako i onih koji dolaze spolja. Takvi informacioni sistemi obezbeđuju kako osoblju tako i spoljnim korisnicima (kupci, dobavljači, agencije i sl) da pristupe informacijama kompanije sa različitim nivoima prioriteta i prava pristupa. To mogu da budu sistemi za upravljanje dokumenata, sistemi za upravljanje projektima, e-mailing sistemi, intranet, internet stranice i sl. Svi ovi sistemi imaju jedan neizostavan deo – **bazu podataka**, koja čuva sve informacije koje se obrađuju i obezbeđuje pristup tim informacijama. Baze podataka su ključna komponenta kod standardnih informacionih sistema, ali i sistema e-poslovanja i drugih Web zasnovanih aplikacija. Koriste ih oragnizacije i preduzeća od onih najmanjih do globalnih korporacija i milioni korisnika.

## 1.1 Podaci i informacije

Pojam interakcije čoveka sa svojom okolinom zasniva se na tri osnovna koncepta:

- 1) podatak,
- 2) informacija i
- 3) znanje (uz koje se danas sve više vezuje pojam mudrost).

Dve osnovne operacije nad elementima koji su definisani ovim konceptima su predstavljanje i obrada.

**Podatak** (lat. *datum* – deo informacije, eng. *Data*) je jednostavna neobrađena izolovana činjenica koja ima neko značenje i koja se obrađuje i čuva na računaru. Podaci su osnova baza podataka i predstavljaju činjenice, pojmove ili događaje, vezani za objekat posmatranja, predstavljeni na unapred dogovoreni, formalizovan način. Podaci su znakovni prikaz činjenica i pojmova koji opisuju svojstva objekata.

**Informacija** (lat. *Informare*, eng. *Information*) predstavlja rezultat analize i organizacije podataka na način da daje/generiše novo znanje. Obrada podataka

podrazumeva proces prevođenja podataka u informacije, tako da *Informacija* (u obradi podataka) predstavlja smisao dodeljen podacima na osnovu dogovora za njihovo predstavljanje.

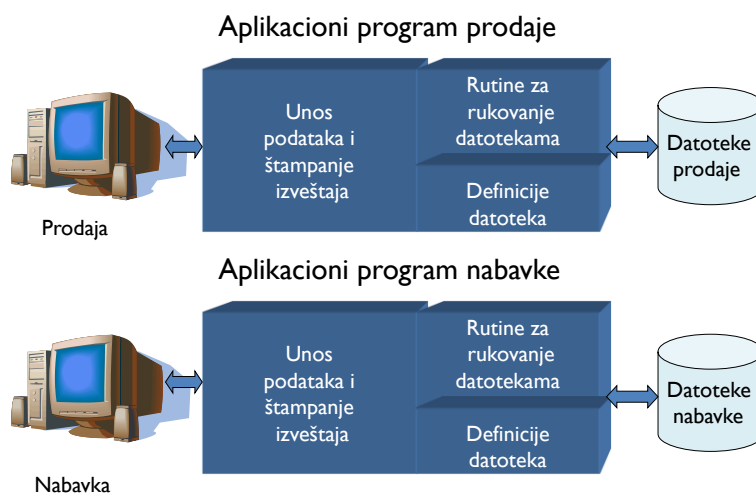
Pojam **Znanje** se u oblasti računarstva odnosi na sisteme veštačke inteligencije. Objašnjenje ovog pojma i način korišćenja znanja nije predmet ovog uvodnog kursa (dodatne informacije: kurs Veštačka inteligencija, VII semestar).

### 1.2 Konvencionalni sistemi u odnosu na baze podataka

Tradicionalni pristup obrade podataka podrazumeva korišćenje sistema zasnovanog na datotekama (engl. *File-based system*). To je tzv. *ad hoc* pristup kod koga:

- postoji datoteka ili skup datoteka podataka potrebnih za definisanu aplikaciju,
- razvijaju se programi koji obrađuju podatke iz datoteka i
- svaki program definiše sopstvene podatke i upravlja tim podacima.

Često se koristi pojam Automatska obrada podataka (AOP) kod koje postoje međusobno nezavisne aplikacije jednog informacionog sistema (IS), i za svaku aplikaciju se kreiraju i održavaju posebne datoteke sa svim potrebnim podacima.



**Slika 1-1. Konvencionalni sistem zasnovan na datotekama**

Na slici 1-1 prikazan je konvencionalni sistem sa dve aplikacije jednog preduzeća, za sektor Prodaja i sektor Nabavka. Podaci se čuvaju u nezavisnim datotekama. Obe aplikacije sadrže rutine za unos podataka i štampanje izveštaja, kao i rutine za rukovanje datotekama. Definicija datoteka je ugrađena u programe. Datoteke su nezavisne i mogu da sadrže i iste podatke, pa je održavanje tih

podataka složenije i prepušteno programu odnosno programeru u toku razvoja aplikacije.

Osnovne karakteristike ovakvog sistema zasnovanog na datotekama su:

- Ugrađeni su opisi fizičkih karakteristika podataka (ili fizičkih struktura podataka) u aplikacioni program.
- Ne postoji kontrola pristupa podacima, osim one koju vrše aplikacioni programi.

Prednosti konvencionalne obrade zasnovane na datotekama su jednostavnost projektovanja i realizacije. Međutim, obe osnovne karakteristike su i potencijalni problemi kod korišćenja. Poželjno je da definicija podataka bude odvojena i nezavisna od aplikacija, a takođe, poželjna je potpuna kontrola pristupa podacima i to nezavisno od konkretnog aplikacionog programa. Kao posledica, nedostaci ovih sistema su:

- nepovezanost aplikacija,
- ponavljanje podataka,
- neusaglašenost (nekonzistentnost) podataka,
- čvrsta povezanost programa i podataka:
  - programi za obradu podataka zavise od načina fizičkog struktuiranja podataka,
  - promena strukture podataka zahteva promenu programa.
- ograničena mogućnost zajedničkog korišćenja podataka,
- ograničena raspoloživost podataka i
- neadekvatna realizacija oporavka od pada sistema.

Posledice navednih nedostataka su da je obrada podataka skupa i da se zahteva neko bolje rešenje. Rešenje problema su **baze podataka** koje treba da obezbede:

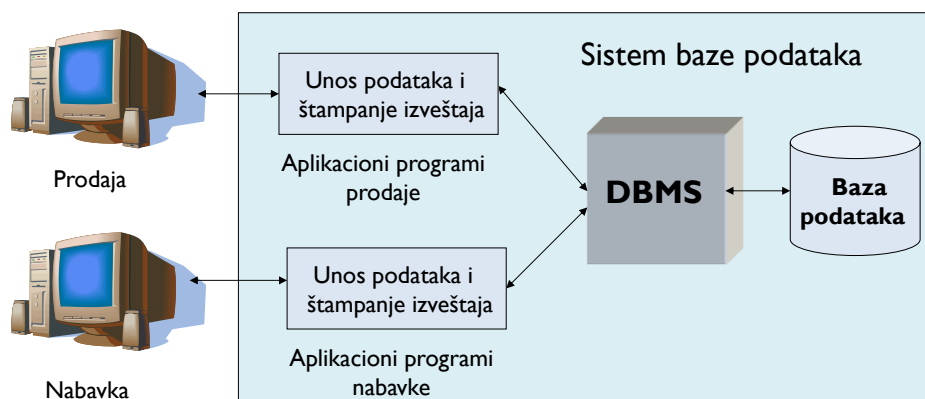
- Nezavisnost struktura podataka od programa koji ih obrađuju.
- Nezavisnost programa od struktura podataka.
- Minimalnost ponavljanja podataka.
- Da obrada podataka nije vezana za programski jezik opšte namene, već za viši "upitni" jezik.
- Korišćenje baze podataka od strane većeg broja korisnika.

Obrada zasnovana na bazama podataka prikazana je na slici 1-2. Baze podataka podržavaju rad većeg broja aplikacija, koje preko Sistema za upravljanje bazama podataka omogućava pristup podacima u bazi podataka. Sistem za upravljanje bazama podataka je zadužen za ažurnost podataka. Aplikacije, Sistem za

upravljanje bazama podataka i baza podataka čine Sistem baza podataka (objašnje ovih pojmova je dato u odeljcima 1.3, 1.4 i 1.5).

Važne odrednice pristupa zasnovanog na korišćenju baza podataka su:

- Podaci su deljivi.
- Baza podataka sadrži i podatke i njihove definicije (opisi, šeme).
- Definicije podataka se nalaze u katalogu sistema (rečnik sistema) i nazivaju se meta podaci.
- Definicije podataka su odvojene od aplikacionih programa.
- Podaci su logički povezani (entiteti, atributi, veze).



Slika 1-2. Obrada zasnovana na bazama podataka

### 1.3 Sistem baza podataka

Sistem baza podataka sadrži tri osnovne komponente (slika 1-3):

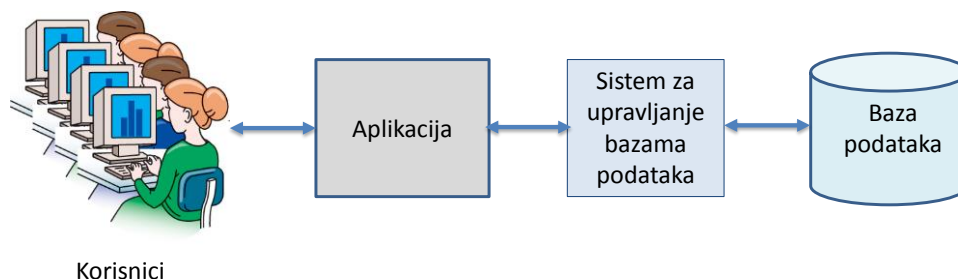
- (1) aplikacija nad bazom podataka,
- (2) sistem za upravljanje bazama podataka (Database Management System – DBMS), i
- (3) baza podataka.

**Baza podataka** predstavlja kolekciju povezanih podataka organizovanih u logičke celine predstavljene tabelama. U sistemu baza podataka ona je zadužena za čuvanje podataka. Naredne sekcije ove lekcije su posvećene objašnjenju pojma baza podataka.

**Sistem za upravljanje bazama podataka** ili DBMS je računarski program (u praksi obično skup računarskih programa odnosno aplikacija) koji obezbeđuje kreiranje, obradu i administraciju nad bazom podataka. DBMS dobija zahteve u obliku upita i prevodi te zahteve u odgovarajuće akcije nad bazom podataka.



DBMS je obično jako velika, složena aplikacija koja obezbeđuje veliki broj funkcionalnosti za korisnika. U poglavlju o SQL upitnom jeziku (poglavlje 7), koji je glavna karika u procesu obrade podataka iz baza podataka, dato je više detalja o zadavanju upita.



Slika 1-3. Komponente sistema baza podataka

**Aplikacija nad bazom podataka** predstavlja jednu ili više aplikacija (računarskih programa) koje predstavljaju posrednike između korisnika i DBMSa. Aplikacije mogu čitaju ili modifikuju podatke iz baze podataka tako što šalju SQL zahteve DBMSu koji vraća podatke (obično u obliku tabela). S druge strane, aplikacija na pogodan način prikazuje korisniku dobijene podatke. Osnovne informacije o razvoju aplikacija nad bazama podataka su date u poglavlju 8.

**Korisnici** su četvrta komponenta sistema baza podataka. Oni preko formi aplikacije pregledaju, unose i modifikuju podatke iz baze podataka, a preko izveštaja mogu da prikažu ili štampaju rezultate pretraga i analiza podataka.

## 1.4 Šta je baza podataka?

Baza podataka, koja predstavlja kolekcija povezanih podataka, se projektuje za neku organizaciju kao što su: kompanija, banka, fakultet, grad, biblioteka, supermarket i sl. Baza podataka, kako se vidi iz definicije date u prethodnom odeljku, predstavlja kolekciju podataka koji su organizovani u tabele. Podaci su međusobno povezani, a mogu da se koriste za jednu ili više aplikacija.

Slučajni skup podataka nije baza podataka. Baza podataka se projektuje i gradi za specifičnu namenu i predstavlja neki aspekt realnog sveta organizacije za koju se ona projektuje. To je tzv. **minisvet** – deo realnog sveta za koji je neophodno čuvati i obrađivati podatke. Na primer, za bazu podataka Fakultet mini svet može da se odnosi na podatke o studentima, ispitima, nastavnicima, službama i sl. Promene u minisvetu utiču na bazu podataka tako da mogu da utiču na promene u podacima (napr. za bazu podataka Fakultet upis novih studenata) ili na promene u opisu baze podataka (napr. promena organizacije fakulteta).

**Primer:** Baza podataka Fakultet

Baza podataka Fakultet se odnosi na mini-svet fakultet i može da obuhvata podatke nekim aspektima rada fakulteta odnosno ne mora da pokrije kompletno poslovanje i rad fakulteta. Ova baza podataka može da sadrži podatke o:

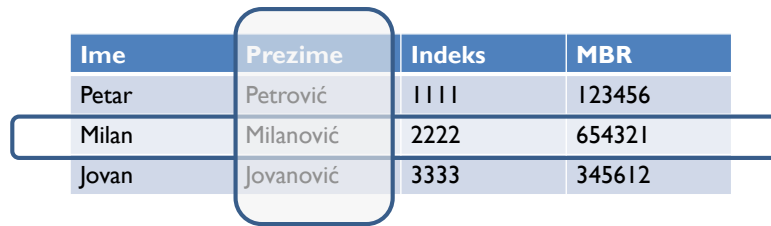
- **Entitetima** – objektima realnog sveta, kao što su:
  - studijski programi,
  - studenti,
  - predmeti,
  - učionice
- **Vezama** među ovim entitetima:
  - student sluša predmet,
  - predmet se izodi u učionici,
  - predmet pripada studijskom programu

Osnovna namena baze podataka je da bude **repozitorijum** (skladište) za podatke. Podaci mogu biti različitog tipa, tekstualni, numerički, slike, audio i video zapisi i sl. Skup podataka pripremljen tako da se mogu jednostavno koristiti, tj. pregledati, pretraživati, sortirati, upoređivati, itd., ali i menjati. Podaci u bazi podataka se čuvaju tako da je unos novih podataka, kao i čitanje i pretraživanje postojećih, je jednostavno, efikasno i ako je moguće, bez grešaka.

Iz „definicije“ baze podataka vidi se da je ona **kolekcija međusobno povezanih podataka**, koji su **organizovani u tabele**. U ovoj „definiciji“ dve su činjenice od značaja – organizacija podataka u tabele i njihova međusobna povezanost.

Podaci u bazama podataka su organizovani u **dvodimenzionalne tabele**. Tabela može da ima više **kolona**, gde svaka kolona predstavlja neku **osobinu ili atribut** objekta realnog sveta. **Vrste** tabele čine konkretni podaci, odnosno konkretne vrednosti osobina/atributa nekog objekta. Pri tome redosled podataka u tabeli nije relevantan i tabela nema duplikata ili je njihov broj minimalan.

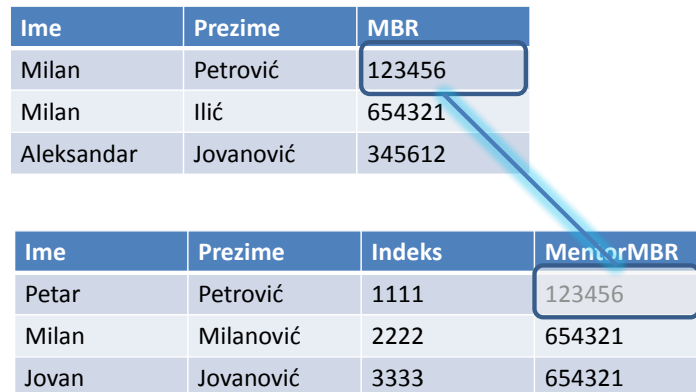
Na primer, jedna tabela može da sadrži informacije o studentima (slika 1-4). Kolone tabele definišu attribute odnosno čuvaju vrednosti izabranih osobina studenata: ime, prezime, indeks, matični broj, i sl. Na slici 1-4. prikazana je kolona Prezime koja čuva prezimena studenata. Vrste u takvoj tabeli su studenti, tako da se svaka vrsta odnosi na jednog studenta. Na slici 1-4 označena je vrsta koja se odnosi na studenta Milana Milanovića sa indeksom 2222 i matičnim brojem 654321. Presek jedne vrste i jedne kolone predstavlja ćeliju koja sadrži vrednost odgovarajuće osobine odnosno atributa. U ovom slučaju, to je vrednost prezimena studenta Milanovića.



Ime	Prezime	Indeks	MBR
Petar	Petrović	1111	123456
Milan	Milanović	2222	654321
Jovan	Jovanović	3333	345612

Slika 1-4. Tabela u bazi podataka sa označenom vrstom i kolonom

Koje će tabele da sadrži baza podataka zavisi od problema za koji treba realizovati bazu podataka. Na primer, baza podataka se može odnositi na školu, pa će u tom slučaju tabele biti o učenicima, nastavnicima, odeljenjima, i sl. Postupak izbora i definisanja tabela za bazu podataka je deo procesa modeliranja odnosno izgradnje **modela podataka**. Model podataka je detaljno objašnjen u sekciji nakon sekcije o DBMSu.



Ime	Prezime	MBR
Milan	Petrović	123456
Milan	Ilić	654321
Aleksandar	Jovanović	345612

Ime	Prezime	Indeks	MentorMBR
Petar	Petrović	1111	123456
Milan	Milanović	2222	654321
Jovan	Jovanović	3333	654321

Slika 1-5. Međusobna povezanost podataka u bazi podataka

**Međusobna povezanost podataka** je ono po čemu se baza podataka razlikuje u odnosu na tradicionalni pristup zasnovan na datotekama i programe za unakrsna izračunavanja ko što je Excel. Povezanost podataka obezbeđuje značajne prednosti kod pretraživanja kada korisnik može da na osnovu veza izvuče mnogo više podataka. Na primer, ako u bazi podataka postoje dve tabele: tabela koja čuva podatke o studentima i tabela sa podacima o njihovim mentorima (slika 1-5), veza između studenta i mentora je ostvarena preko matičnog broja mentora. Uparivanjem vrednosti MBR i MentorMBR može da se ostvari veza i da se za studenta dobiju i svi podaci o mentoru ili obrnuto – da dobijete podatke o svim studentima čiji je mentor zadati profesor. Ovi podaci mogu da se obezbede odgovarajućim zahtevom (SQL upitom) DBMSu (objašnjenje u poglavlju o upitnom jeziku SQL).

Opisi podataka i ograničenja nad podacima (tzv. **metapodaci**) su sastavni deo sistema baze podataka. Baza podataka pored podataka sadrži i **metapodatke**, odnosno podatke o samoj strukturi baze podataka. Metapodaci mogu da se odnose na imena tabela, imena kolona u svakoj tabeli, na podatke o korisnicima podataka, kao i raznim pomoćnim strukturama koje obezbeđuju brz pristup podacima (indeksi).

Ovi opisi se nalaze u sistemskom **katalogu** koji je dostupan i DBMS-u i korisnicima kojima je potrebno poznavanje strukture baze podataka. **Nezavisnost programa i podataka** je obezbeđena tako što su opisi podataka u DBMS katalogu i nezavisni su od programa za pristup podacima.

## 1.5 Šta je Sistem za upravljanje bazama podataka?

Softverski sistem koji omogućava korisnicima definisanje, kreiranje i manipulisanje bazom podataka naziva se **sistem za upravljanje bazama podataka** (eng. *Database Management System - DBMS*).

- **Definisanje** se odnosi na specificiranje tipova podataka, struktura i ograničenja za podatke koje treba memorisati u bazi podataka.
- **Kreiranje** podrazumeva proces memorisanja podataka na nekom medijumu koji kontroliše DBMS.
- **Manipulisanje** podrazumeva postavljanje upita bazi podataka da bi se našli specifični podaci, ažuriranje baze podataka da bi se unele promene nastale u mini svetu i generisanje izveštaja na osnovu podataka memorisanih u bazi podataka.

Za kontrolisani pristup podacima u bazi podataka DBMS treba da obezbedi:

- **Sigurnosni sistem**, koji omogućava pristup bazi podataka neautorizovanim korisnicima (sigurnosni servisi), odnosno samo autorizovani korisnici mogu da koriste podatke u skladu sa definisanim privilegijama (autorizacioni servisi).
- **Integritetni sistem**, koji održava konzistentnost podataka u bazi podataka, odnosno da se sve promene dešavaju u skladu sa definisanim pravilima.
- **Sistem za kontrolu i upravljanje konkurentskim pristupom**, koji dopušta deljivi pristup podacima iz baze podataka, tj da se obezbedi korektno ažuriranje podataka kada više korisnika pokušava istovremeno da vrši ažuriranja.

- **Sistem za kontrolu i upravljanje oporavkom baze podataka**, koji omogućava rekonstrukciju prethodnog konzistentnog stanja u slučaju neke hardverske ili softverske neispravnosti.
- **Katalog** kome korisnici mogu pristupati, koji sadrži opis podataka koji su memorisani u bazi podataka.
- **Podršku za transakcije**, koja obezbeđuje korektno izvršavanje niza transakcija koje mogu biti međusobno zavisne; transakcija je skup operacija upisa i čitanja iz baze podataka koji se tretira kao celina tj ima svoj početak i kraj.
- **Razne korisničke funkcije**, kao što su import, eksport podataka, statističke analize i funkcije za nadgledanje.

DBMS obično nudi jezike koji omogućavaju kontrolisani pristup i rad sa podacima:

- **Jezik za opis podataka** (*eng. Data Definition Language - DDL*), koji omogućava korisnicima definisanje tipa i strukture podataka, kao i ograničenja nad podacima memorisanim u bazi podataka (naredne lekcije – CREATE TABLE naredba).
- **Jezik za manipulaciju podacima** (*eng. Data Manipulation Language - DML*), koji omogućava korisnicima umetanje, ažuriranje, brisanje i pretraživanje podataka iz baze podataka (naredne lekcije – SELECT, INSERT INTO, UPDATE naredbe).
- **Jezik za definisanje načina memorisanja podataka** (*eng. Storage Definition Language - SDL*), koji se koristi za specificiranje interne šeme baze podataka.
- **Kontrolisani pristup bazi podataka**, što uključuje različite funkcije i mehanizme za pristup podacima u bazi podataka.

Prednosti koje obezbeđuje DBMS u odnosu na tradicionalni pristup zasnovan na datotekama su brojne:

- **Nezavisnost podataka** - Aplikacioni programi su nezavisni od reprezentacije i načina memorisanja podataka. DBMS nudi aplikaciji apstraktni pogled na podatke.
- **Efikasan pristup podacima** - DBMS koristi posebne tehnike za efikasno memorisanje i pretraživanje podataka.
- **Integritet i bezbednost podataka** - DBMS kontroliše integritet podataka koristeći ograničenja koja je definisao projektant baze podataka. DBMS

nadgleda pristupne privilegije korisnika, tako da svaki korisnik može videti samo podatke za koje poseduje pristupne privilegije

- Administracija podataka - Kada su podaci deljivi centralizovana administracija podataka donosi značajno poboljšanje (zaduženje administratora baze podataka)
- Konkurentni pristup i oporavak - DBMS upravlja konkurentnim pristupom podacima od strane više korisnika, i ima ugrađene mehanizme oporavka baze podataka od različitih neispravnosti.
- Kraće vreme razvoja aplikacija - DBMS ima ugrađene različite funkcije za rad sa podacima što aplikacionim programerima značajno olakšava zadatak razvoja aplikacije i nudi interfejs iz jezika visokog nivoa ka bazi podataka.

Postoje i nedostaci, koji ne ograničavaju značajno prednosti koje DBMS pruža. Ipak, za rešavanje problema i aplikacije koje rade sa malom količinom podataka, i tradicionalna obrada zasnovana na datotekama može da bude bolji izbor od korišćenja baza podataka. Neki nedostaci korišćenja baza podataka i DBMSa su:

- Složenost – DBMS zahteva poznavanje rada svih njegovih komponenti, pa je posao administratora jako zahtevan.
- Veličina – za instalaciju DBMSa i čuvanje podataka obično treba daleko više memorijskog prostora u odnosu na tradicionalni pristup.
- Troškovi za DBMS – obuhvataju dodatne troškove za instalaciju i održavanje DBMSa, kao i dodatne troškove za nabavku hardvera odnosno servera na kojima se DBMS i baza podataka instaliraju.
- Troškovi konverzije – ako postoji potreba da se postojeći podaci konvertuju u format definisan bazom podataka; kod novih sistema ovi troškovi mogu da se odnose na unos postojećih podataka iz drugih izvora informacija.
- Performanse – zavise od samog DBMSa, podešavanja, iskustva administratora i sl.
- Veći uticaj neispravnosti – neispravnosti mogu direktno da utiču na rad DBMSa, što podrazumeva da DBMS mora da bude u stanju da rešava takve probleme.

Izbor DBMSa može takođe da bude jedan od problema u procesu razvoja sistema baze podataka. Danas na tržištu postoji veliki broj proizvođača DBMSa koji nude sistema različitih performansi i koji su namenjeni različitim segmentima tržišta. Koji DBMS ćete izabrati zavisi od tipa i veličine problema koji treba da

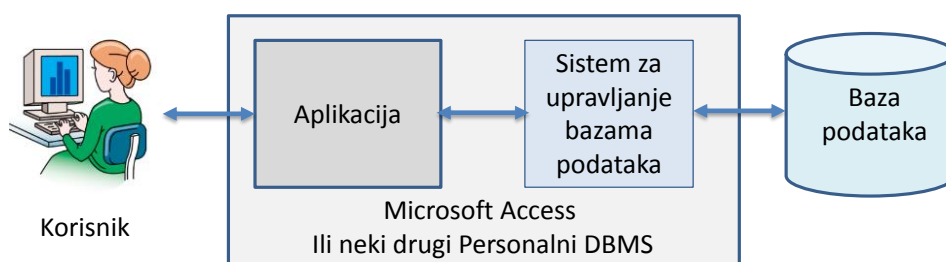
rešite realizacijom aplikacije. U narednoj sekciji dat je kratak prikaz tipova sistema baza podataka.

## 1.6 Tipovi sistema baza podataka

Tehnologija baza podataka se može koristiti za veliki broj aplikacija. Praktično danas skoro i da ne može da se realizuje aplikacija koja ne koristi neki sistem baza podataka za čuvanje podataka – bez obzira da li se radi o standardnim desktop aplikacijama, kao što su knjigovodstvene aplikacije, sistemi za upravljanje dokumentima, sistemi za banke, i sl, ili se radi o modernim Web aplikacijama koje obezbeđuju složenu funkcionalnost u distribuiranom okruženju, od on-line kupovine do raznih socijalnih mreža i sl. U zavisnosti od aplikacije, zahtevi prema bazi podataka mogu značajno da se razlikuju.

Jedan granični slučaj se odnosi na potrebu za aplikacijom za evidenciju kućnih troškova. U tom slučaju ona obično sadrži samo nekoliko tabela, gde svaka tabela može da ima samo nekoliko stotina vrsti. Aplikaciju, a samim tim i bazu podataka, koristi samo jedan korisnik. Za takve sisteme se obično koristi naziv **personalni sistemi baza podataka** (slika 1-6). Naravno, ovakvi sistemi mogu da se primene i na mnogo složenije aplikacije od evidencije kućnog budžeta. Na primer, mogu da pokriju i poslovanje manjeg preduzeća, ili da podrže rad nekog Web sajta.

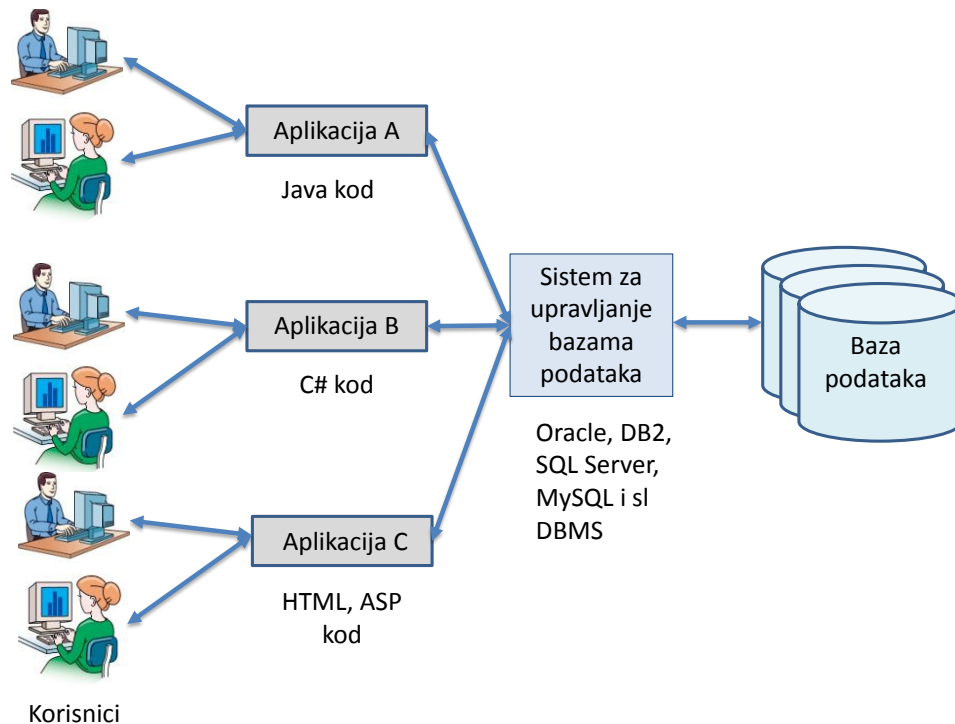
S druge strane, ako se baza podataka koristi u velikoj kompaniji koja ima više organizacionih jedinica, gde svaka od njih ima sopstvene poslovne procese, neophodna vam je podrška sistema baza podataka koji može da obezbedi čuvanje i pretragu velike količine informacija na više distribuiranih lokacija. Takvi sistemi, poznati kao **enterprise sistemi baza podataka** (slika 1-7), sadrže veliki broj tabela, a neke od njih mogu da imaju i nekoliko stotina hiljada vrsta i više, podacima može konkurentno da pristupa veliki broj korisnika i moraju da rade 24 časa dnevno, 7 dana u nedelji.



Slika 1-6. Personalni sistem baza podataka

Veličina problema, odnosno tip sistema baza podataka koji treba da se primeni, određuje izbor sistema za upravljanje bazama podataka. Naravno, između personalnih i *enterprise* sistema postoji širok opseg drugih sistema različitog nivoa

složenosti u odnosu na količinu podataka koju treba obrađivati. Za takve sisteme može da se izabere DBMS u zavisnosti od zahteva koji se postavljaju pred sistemom baze podataka, ali i u zavisnosti od cene i raspoloživih sredstava za razvoj.



**Slika 1-7. Enterprise sistem baza podataka**

Komponente personalnog sistema baza podataka su prikazane na slici 6. Microsoft Access ili neki drugi personalni DBMS, ima ulogu aplikacije i DBMSa. Personalni sistem baza podataka je razvijen tako da se korisniku obezbedi jednostavan razvoj aplikacija. Na primer, korišćenjem Access-a, korisnik može da razvija aplikaciju ili da koristi funkcije DBMSa, često ne primećujući razliku između ova dva dela. Praktično, Access u sebi sadrži razvojno okruženje za aplikacije i DBMS. Na taj način skriveni su mnogi aspekti koji se odnose na „dešavanja iza scene“ u procesu obrade podataka. Iako u pozadini i ovaj tip DBMSa radi sa SQLom, korisniku je obezbeđen veliki broj čarobnjaka (wizard-a), čijom upotrebom može da se pojednostavi posao realizacije aplikacije. Na prvi pogled to predstavlja olakšanje, ali ako nešto od onoga što treba uraditi nije pokriveno čarobnjakom, kao i za bilo koji ozbiljni zahvat, potrebno je da zna šta je „iza scene“. Za razvoj bilo kog velikog sistema, korisnik mora da nauči i te skrivene tehnologije.



Na slici 1-7. prikazan je *enterprise* sistem baza podataka. Na slici su prikazane aplikacije razvijane u različitim jezicima: Java, C#, HTML i ASP.NET. Takve aplikacije koriste velike DBMS sisteme za upravljanje bazom podataka. Čarobnjaka kao kod personalnih sistema uglavnom nema, ali postoje slični alati koji mogu da pomognu u razvoju takvih sistema. Programeri moraju da napišu kod koji će da obezbedi pristup i pretraživanje podataka korišćenjem funkcija DBMSa.

Iako su početne prednosti koje pruža skrivanje tehnologija i složenosti DBMSa kod personalnih sistema jako zgodne, očigledno je da za razvoj velikih sistema neophodno mnogo šire znanje. Tehnologije prezentovane u narednim lekcijama daju mogućnost razvoja velikih sistema, ali i mogućnost da se personalni sistemi obogatite elementima koji nisu deo čarobnjaka.

## 1.7 Zanimljivosti: istorijat DBMSa

U ovom odeljku dat je kratak pregled istorije razvoja DBMSa. Ovaj pregled je samo pogled autora na istorijski razvoj i ne mora da predstavlja pravo stanje. Moguće je da su u toku više od 50 godina razvoja ove oblasti, neke faze nisu navedene a imale su istorijski značaj. Ono što je značajno je da se oblast baza podataka intenzivno razvija i da je njen značaj za razvoj informacionih sistema ogroman. Istorijski, razvoj DBMSa počinje 60-tih godina prošlog veka sa povremenim drastičnim promenama i traje sve do današnjih dana:

- 60-tih godina 20-tog veka
  - projekat Apollo, na inicijativu predsednika Kenedija. Rezultat ovog projekta je GUAM (Generalized Update Access Method)
- Sredinom 60-tih
  - IBM je prihvatio GUAM ideju i razvio IMS (Information Management System) – to je **hijerarhijski DBMS**
- Sredinom 60-tih godina
  - General Electric je razvio IDS (Integrated Data Store) – to je **mrežni DBMS**
  - Prvi DBMS opšte namene koga je projektovao Charles Bachman, prvi dobitnik Turing-ove nagrade koju dodeljuje ACM (ekvivalent Nobelove nagrade za Računarske nauke)
  - Bachman je nagradu dobio za rad u oblasti baza podataka
- 1967. je formirana, u okviru CODASYL-a, Database Task Group (DBTG) koja je uradila standard za DBMS;
  - 1969 draft report
  - 1971 final report

- 1970. E.F.Codd iz IBM Research Lab je predložio **relacioni model**
- Prvi komercijalni relacioni DBMS su proizvedeni krajem 70-tih i početkom 80-tih godina 20-tog veka:
  - Projekat System R (IBM) krajem 70-tih (SQL)
  - DB2 i SQL/DS (IBM) 80-tih
  - Oracle (Oracle Corporation) 80-tih
- Danas postoji na hiljade relacionih DBMS-ova za mainframe i PC okruženja:
  - INGRES (Computer Associates)
  - INFORMIX, DB2 (IBM)
  - Office Access, Visual FoxPro (Microsoft)
  - InterBase, JDataStore (Borland)
  - R:Base (R:Base Technologies)
- 1976. Chen je predložio ER model
- 1979. Codd je predložio proširenu verziju relacionog modela RM/T i 1990. RM/V2 koji se ubraja u semantičke modele podataka
- Druga polovina 90-tih godina 20-tog veka: Objektno-orijentisani DBMS (OODBMS) i Objektno-relacioni DBMS (ORDBMS)
- Početak ovog veka
  - Postrelacione BP
  - XML *native* BP i podrška za XML u RDBMS
- Poslednjih desetak godina
  - Relacione baze podataka + nove tehnologije + skalabilnost
  - NoSQL baze podataka
- U ovom trenutku: Trendovi vraćanja na stare, dobre vrednosti RDBMS

## 2 MODELI PODATAKA I PROCES PROJEKTOVANJA BAZE PODATAKA

Baza podataka kao kolekcija neredundantnih povezanih podataka koji se koriste za jednu ili više aplikacija čuva podatke koji se odnose na deo realnog sveta, tzv **mini svet**, za koji se ona projektuje i razvija. U procesu razvoja baze podataka najpre se formira model realnog sistema, a zatim se na osnovu tog modela gradi formalni sistem. Razlog za formiranje modela je da se istaknu značajne karakteristike sistema i da se privremeno stave u drugi plan sve one karakteristike koje nisu bitne za sistem koji se razvija.

Postoji mnogo različitih mogućnosti da se modelira sistem. U fazi modeliranja, zadatak sistem analitičara, odnosno inženjera zahteva, je da otkrije funkcije koje sistem mora izvršavati, podatke koje mora pamtit i obrađivati, informacije koje mora obezbeđivati za potrebe korisnika, sekvence u kojima se funkcije moraju izvršavati i u kojima se može pristupiti podacima. Model sistema koji se odnosi na podatke predstavlja šemu (ili opis) baze podataka. Model podataka predstavlja vrstu meta-modela, koji omogućava projektovanje šeme baze podataka.

U ovom poglavlju dat je najpre opis nivoa apstrakcije u DBMSu, a nakon toga i opis odgovarajućih modela podataka. Značajan deo procesa projektovanja baza podataka se odnosi na korišćenje konceptualnih modela, pa je naredno poglavlje posvećeno modelu entiteta i veza.

### 2.1 Nivoi apstrakcije u DBMS-u

U DBMS-u podaci su opisani u **tri nivoa apstrakcije** (slika 2-1.):

- **Eksterna šema** (često se naziva i **podšema**): ima ih više, po jedna za svaku grupu korisnika
- **Konceptualna šema**: postoji jedna za celu bazu podataka
- **Interna šema**: postoji jedna za celu bazu podataka

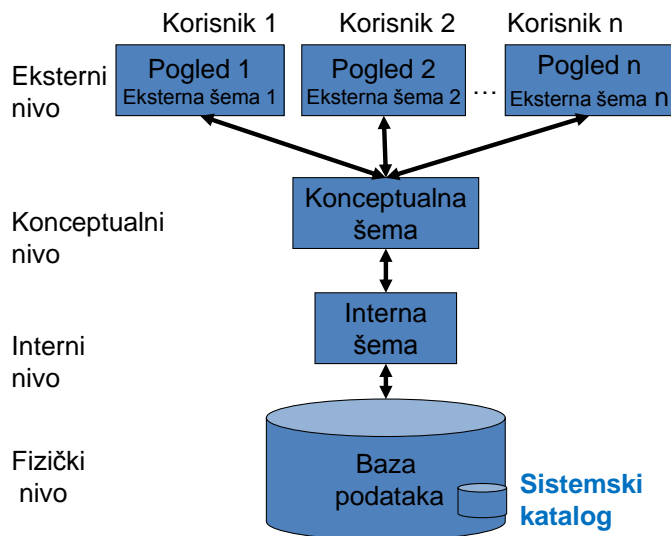
Šema baze podataka je opis baze podataka. DBMS je odgovoran za **preslikavanje** između ovih šema, odnosno za preslikavanje konceptualne u internu šemu i obrnuto, kao i za preslikavanje eksterne u konceptualnu šemu i obrnuto.

Šeme, odnosno opisi se memorišu u sistemskom katalogu DBMS-a. Ovakva arhitektura je poznata kao Arhitektura tri šeme DBMS-a (ANSI-SPARC arhitektura, slika 8). Svi DBMS-i ne razlikuju sva tri nivoa. Cilj arhitekture 3-šeme je odvojiti pogled korisnika baze podataka od načina njene fizičke reprezentacije.

**Instancu baze podataka** čine **podaci** koji se trenutno nalaze u bazi podataka. Jednoj šemi baze podataka može odgovarati više instanci. Šema baze podataka se naziva i **intenzija** baze podataka, a instanca **ekstenzija** ili **stanje** baze podataka.

Eksterni nivo (nivo pogleda) je korisnički pogled na bazu podataka. Svaki pogled na bazu podataka opisuje samo onaj deo baze podataka koji je od interesa za konkretnu grupu korisniku i prikriva ostali deo baze podataka od ove grupe korisnika. Sadrži veliki broj **eksternih šema** ili **pogleda korisnika**. Različiti pogledi mogu imati različite reprezentacije istih podataka.

Konceptualni (logički) nivo sadrži konceptualnu šemu koja predstavlja globalni opis baze podataka. Konceptualni nivo predstavlja pogled na bazu podataka **svih korisnika** baze podataka. Opisuje koji se podaci čuvaju u bazi podataka, kako su ti podaci strukturirani i kako su međusobno povezani.



Slika 2-1. Arhitektura tri šeme DBMSa

Konceptualna šema opisuje entitete, attribute i njihove veze, ograničenja nad podacima, semantičke informacije o podacima, sigurnost i integritet informacija i ne sadrži nikakve detalje o načinu memorisanja podataka!

Interni nivo predstavlja fizičku reprezentaciju baze podataka na računaru. Sadrži **internu (fizičku) šemu** koja opisuje kako su podaci memorisani u bazi podataka. Interna šema opisuje sve detalje o memorisanju podataka, definiše strukturu i organizaciju fajlova koji se koriste za smeštanje baze podataka. Ovaj nivo koristi

usluge fajl sistema operativnog sistema (OS) za smeštanje podataka na memorijski medijum, za formiranje indeksa, za pretraživanje podataka itd. Na internom nivou se definiše dodela prostora na disku za podatke i indekse, opis slogova, smeštanje slogova, kompresija podataka i tehnike enkripcije podataka.

Ispod internog nivoa je **fizički nivo** kojim može da upravlja operativni sistem po direktivama DBMS-a. Funkcije OS-a i DBMS-a na fizičkom nivou nisu strogo razgraničene, pa variraju od jednog do drugog DBMS-a. Neki DBMS-i koriste metode pristupa OS-a, dok drugi imaju sopstvene fajl sisteme.

Opisana arhitektura tri šeme omogućava opis i memorisanje podataka preko DBMS-a. Korisnici baze podataka su u nekom realnom svetu, podaci memorisani u bazi podataka predstavljaju neke aspekte tog realnog sveta (mini svet). DBMS omogućava korisnicima da **opišu** (definišu) podatke koje žele memorisati u bazi podataka. Za kreiranje modela baze podataka (tj. šeme baze podataka) koristi se model podataka. Šema baze podataka predstavlja opis logičke strukture podataka, odnosno same baze podataka.

## 2.2 Modeli podataka

Proces razvoja baze podataka podrazumeva formiranje modela realnog sistema, na osnovu izabranih značajnih karakteristika sistema. Modeliranje strukture podataka je zadatak projektanta baze podataka i obuhvata aktivnosti koje se odnose na otkrivanje i izbor podataka koji baza podataka mora pamtit i obrađivati. Ovako dobijene informacije su osnova za kreiranje modela baze podataka izabranim **modelom podataka**.

**Model podataka** je integrisana kolekcija koncepata za opis i manipulaciju podacima, vezama između podataka i ograničenjima nad podacima i vezama. To je matematička apstrakcija koja se koristi za projektovanje modela realnog sistema.

Model podataka sadrži skup koncepata i konstrukcija koji se koristi za opis sadržaja baze podataka i veza između podataka u bazi podataka. Taj opis, koji se zove i **šema baze podataka**, se odnosi na tip činjenica uključenih u bazu podataka i pravila ili ograničenja koja postoje.

Model podataka ima tri komponente:

- **Strukturnu** koja se sastoji od skupa pravila prema kojima se baza podataka može konstruisati,
- **Integritetnu** koja definiše tipove ograničenja nad vrednostima atributa, veze između podataka i međusobnu uslovljenost podataka,
- **Operacijsku** koja definiše tipove operacije nad strukturama podataka i koja modelira dinamičke osobine realnog sistema.

U odnosu na ANSI-SPARC arhitekturu DBMS-a modeli podataka mogu biti:

- Eksterni modeli podataka – služe za predstavljanje pogleda korisnika na realni svet,
- Konceptualni modeli podataka – služe za predstavljanje logičkog pogleda ili pogleda zajednice na realni sistem nezavisno od DBMS-a,
- Interni modeli podataka – služe za predstavljanje konceptualne šeme na takav način da je razumljiva za DBMS.

Da bi model podataka bio upotrebljiv mora obezbeđivati prostu korespodenciju između koncepata koje nudi i koncepata realnog sveta. Prema tipu koncepata koji nude za opis baze podataka, modeli podataka se razvrstavaju u tri kategorije:

- **Konceptualni modeli podataka** nude koncepte visokog nivoa, kao što su entiteti, atributi i veze, koji su razumljivi i za krajnjeg korisnika i za projektanta baze podataka, i koji se mogu lako preslikati u implementacioni model podataka. Najpoznatiji konceptualni model podataka je **model entitet-veza** (prikazan u poglavlju 3).
- **Implementacioni modeli podataka** nude koncepte koji su prihvatljivi za projektanta baze podataka i koji su pogodni za direktnu implementaciju na računaru. Najpoznatiji implementacioni modeli podataka su **relacioni** (poglavlje 4), objektno-relacioni i objektno-orijentisani. Za predstavljanje informacije u relacionom modelu podataka koristi se kolekcija tabela, kod objektno-relacionog modela podataka koristi se kolekcija objekata klasa i tabela, a kod objektno-orijentisanog modela podataka koristi se kolekcija klasa i njihovih međusobnih veza.
- **Fizički modeli podataka** nude koncepte za opis načina memorisanja baze podataka u računaru. To su koncepti niskog nivoa, kao što su tipovi podataka, formati slogova, kriterijumi uređenja slogova, memorijske strukture podataka i pristupni putevi.

Konceptualni modeli podataka ne zavise od izbora DBMSa i ne utiču izbor. S druge strane izbor implementacionog modela određuje izbor tipa DBMSa. Najpoznatiji implementacioni model je relacioni model. Ogromna većina komercijalni sistema koristi relacioni model, pa se za takve baze podataka koristi naziv **relacione baze podataka**, a odgovarajući DBMS je **relacioni DBMS** (RDBMS). U ovom materijalu se pod pojmom „baza podataka“ podrazumeva „relaciona baza podataka“.

### 2.3 Projektovanje baze podataka

Problem koji treba rešiti u fazi projektovanja baze podataka se odnosi na projektovanje **logičke** i **fizičke** strukture jedne ili više baza podataka da bi se zadovoljile informacione potrebe korisnika u organizaciji za definisani skup

operacija. Ciljevi su da se zadovolje informacioni zahtevi specificiranih korisnika i aplikacija i da se obezbedi struktuiranje informacija koje je prirodno i lako za razumevanje. Pri tome je potrebno podržati zahteve obrade i zahteve performansi (vreme odgovora, vreme obrade i memorijski prostor).

*Realni svet* (koji treba predstaviti modelom podataka), odnosno njegov deo koji je od značaja za proces projektovanja baze podataka (**mini-svet**) se sastoji od **objekata** koji mogu biti stvarni ili apstraktni i koje nazivamo **entitetima**.

Svaki objekat, odnosno entitet, poseduje neka **svojstva**. Na primer, entitet "vozilo" ima vlasnika, registarski broj, datum registracije, godinu proizvodnje, proizvođača, marku, boju, tip motora, i dodatnu opremu. Svojstva ili atributi objekta će biti **predstavljena kolonama** u odgovarajućoj tabeli baze podataka.

Objekti međusobno mogu biti **povezani** različitim odnosima odnosno **relacijama**. Svaka takva relacija može da poseduje posebna svojstva. Relacije se mogu iskoristiti kod pretraživanja međusobno povezanih podataka, na primer, kod pretraživanja podataka o registrovanim vozilima i njihovim vlasnicima. Više detalja o načinu povezivanja podataka iz tabela se nalazi u sekciji 4.5 koja se odnosi na ključeve relacija.

Važno: Izborom objekata, definisanjem njihovih svojstava i prepoznavanjem veza između objekata, izvršeno je modeliranje mini-sveta odnosno dela realnog sveta.

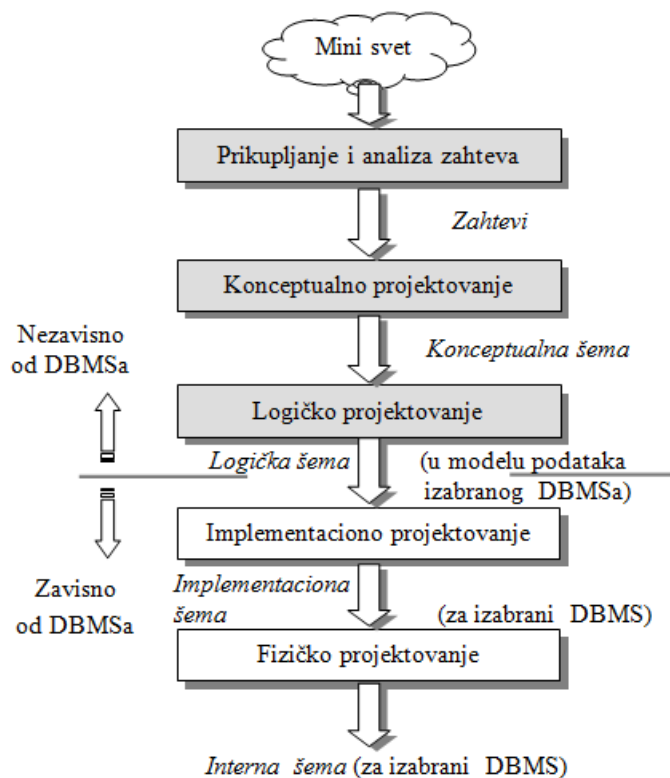
Faze procesa projektovanja baze podataka, prikazane na slici 2-2., su:

**Prikupljanje i analiza zahteva** je prva faza projektovanja u kojoj se prikupljaju informacije o podacima i transakcijama imajući u vidu sadašnje i buduće potrebe korisnika. Prikupljaju se imena i opisi entiteta, imena i opisi atributa, izvori, načini upotrebe, tajnost, značaj i vrednost svih podataka. Prikupljeni podaci se analiziraju i formiraju se jasni i konzistentni *zahtevi za podacima i transakcijama*.

U fazi **konceptualno projektovanje** baze podataka projektuje se *konceptualni model baze podataka*, nezavistan od bilo kakvih implementacionih detalja, kao što su izabrani DBMS, aplikacioni programi, programski jezici, hardverska platforma, tražene performanse i slično.

Konceptualni model baze podataka sadrži detaljan opis entiteta, veza i ograničenja. Koriste se dva prilaza. Po prvom se nezavisno projektuju *lokalni konceptualni modeli podataka* za svaki pogled korisnika na bazu podataka, a zatim se ovi lokalni konceptualni modeli integrišu u jedinstven *globalni konceptualni model baze podataka*. Po drugom se prvo projektuje globalni konceptualni model baze podataka, a zatim se na osnovu ovog globalnog modela projektuju lokalni modeli za pojedine poglede korisnika na bazu podataka. U ovom materijalu koristiće se drugi prilaz. Izlaz iz ove faze projektovanja su *konceptalna i eksterne*

šeme baze podataka. Za konceptualno projektovanje šeme baze podataka korišćen je ER i/ili EER model podataka.



Slika 2-2. Faze projektovanja baza podataka

U fazi **logičko projektovanje** projektuje se *logički model baze podataka*. On se dobija preslikavanjem konceptualnog modela u model podataka DBMSa koji je odabran za implementaciju baze podataka, pri čemu se ne uzimaju u razmatranje konkretne osobine odabranog DBMSa, niti bilo kakve fizičke karakteristike sistema.

Kao implementacioni model podataka može se koristiti relacioni, objektno-relacioni ili objektno-orijentisani model podataka. Rezultati ove faze projektovanja su *logička šema baze podataka* i *skup logičkih podšema*. Nakon izbora DBMSa za realizaciju baze podataka, logička šema se prevodi u *implementacionu šemu*, a logičke podšeme u *implementacione podšeme*. Ova faza logičkog projektovanja često se naziva *implementaciono projektovanje*.

Na osnovu implementacione šeme vrši se opis baze podataka u jeziku za opis podataka (DDL) i njena realizacija, dok se na osnovu implementacionih podšema grade transakcioni programi (aplikacije nad bazom podataka). U ovom materijalu



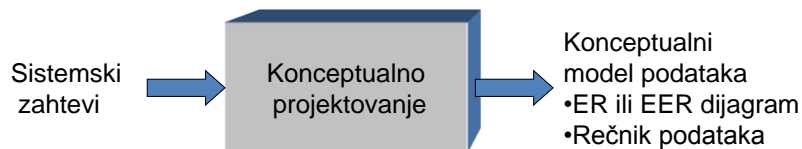
kao implementacioni model podataka koristiće se relacioni model, a za opis implementacionih šema koristiće se DDL podskup SQLa. U mnogim slučajevima SQL naredbe uključuju i neke implementacione detalje, tako da se kompletna SQL specifikacija baze podataka može dobiti tek nakon faze fizičkog projektovanja.

U fazi **fizičko projektovanje** baze podataka biraju se interne memorijske strukture podataka i pristupni putevi za datoteke baza podataka da bi se dobile tražene performanse za planirane aplikacije. Izlaz iz ove faze projektovanja je *interna šema baze podataka*. U ovom kursu za opis interne šeme takođe će se koristiti SQL naredbe.



### 3 KONCEPTUALNO PROJEKTOVANJE BAZE PODATAKA

Cilj konceptualnog projektovanja je kreiranje takvog projekta konceptualnog modela baze podataka da što vernije opisuje realni sistem za koji se projektuje baza podataka (slika 3-1.). Na osnovu prikupljenih zahteva, korišćenjem nekog modela podataka visokog nivoa, gradi se konceptualni model baze podataka. U okviru ovog materijala za fazu konceptualnog projektovanja izabran je model entiteta i veza i njegova proširena verzija za podršku objektno orijentisanih koncepata, tzv unapređeni model entiteta i veza.



Slika 3-1. Konceptualno projektovanje baze podataka

**Model entitet-veza** (engl. *Entity Relationship Model*), i njegovo proširenje, **unapređeni model entiteta i veza** (engl. *Enhanced Entity Relationship Model*) je jedan od najpopularnijih semantičkih modela podataka koji se koristi na konceptualnom nivou. Nadalje u tekstu korišće se skraćenice **ER model** za model entiteta i veza, odnosno **EER model** za unapređeni ER model.

Model entiteta i veza koristi ER dijagram za grafičko predstavljanje. Autor ovog modela je Chen 1976 godine, a danas postoji više verzija (E)ER modela i više grafičkih notacija. Često se u literaturi može naći i sa alternativnim nazivima: model entiteta i poveznika ili model objekti-veze.

Osnovna ideja kod ER modela je da se za realni sistem, odnosno za deo realnog sveta (tzv *mini svet*) za koji se projektuje baza podataka, može identifikovati postojanje entiteta (objekata realnog sveta) i veza između njih. Svaki entitet predstavlja “nešto” što postoji kao posebna celina, odnosno razlikuje se u odnosu na druge entitete koji egzistiraju u mini svetu i može se odnositi na realni subjekat, objekat, događaj, pojavu ili apstraktni pojam iz mini sveta. ER model se zasniva na

ovakvoj percepciji realnog sveta koja podrazumeva identifikaciju klase sličnih entiteta i specifikaciji veza između njih putem odgovarajućih grafičkih koncepata.

Strukturna komponenta ER modela podataka sadrži osnovne koncepte kao što su: entitet, tip entiteta i skup entiteta, atributi, poveznik (veza), tip poveznika i skup poveznika. EER model sadrži sve koncepte osnovnog ER modela i dodaje nove koncepte iz objektno-orijentisane paradigme, kao što su: generalizacija i specijalizacija (potklase, natklase i nasleđivanje atributa), kategorije, deljive klase. ER model ima dobro definisana ograničenja u okviru integritetne komponente koja se uglavnom eksplicitno definišu i odnose se na integritet domena, kardinalnost tipa poveznika, participaciju tipa poveznika, slabi tip entiteta i ključ tipa entiteta.

### 3.1 Entiteti, tip entiteta i skup entiteta

**Entitet** je **subjekat, objekat, događaj, pojava ili apstraktni pojam** o kome se prikupljaju, memorišu, obrađuju i prezentiraju podaci u automatizovanim informacionim sistemima i koji se može **jednoznačno identifikovati** i na taj način izdvojiti u skupu sličnih entiteta.

Entitet može biti objekat koji **fizički** egzistira (stvarni objekat, npr. osoba, kola, kuća ili radnik) ili objekat koji **konceptualno** egzistira (apstraktni objekat, npr. kompanija, posao ili predmet na univerzitetu).

*Stvarni objekti* mogu biti prirodni (na primer, reke, planine, mora, šume) ili veštački (na primer, telefonske ili vodovodne instalacije, putevi, vozila, zgrade). *Apstraktni objekti* mogu biti događaji (na primer, osiguranje vozila, popis stanovništva, prodaja robe, uplate na žiro račun, polaganje ispita, overa semestra, upis na fakultet), pojmovi (na primer, nauka, znanje, učenje, obuka, profit, republika, opština, kompanija, vremenska prognoza), stanja (na primer, radi na projektu, sluša predavanje, boravi u bolnici) i uloge (na primer, službenik, stranka, doktor, pacijent, radnik, profesor, student, stanovnik).

Kandidati za entitete u realnom sistemu mogu da budu:

- Organizacione jedinice: fakultet, katedra, biblioteka,...
- Lokacije: učionica, raskrsnica, radarska pozicija, laboratorija,...
- Uloge: radnik, službenik, student, profesor, stanovnik, referent,...
- Događaji koji se pamte: ispit, utakmica, predavanje, ispit,...
- Uređaji: proizvodna traka, računar, skener, radar, antena, ...
- Drugi sistemi sa kojima sistem interaguje

**Primer:** Potencijalni entiteti za mini svet Fakultet i mini svet Preduzeće

- Mini svet FAKULTET može da se odnosi na sledeće entitete:
  - student

- profesor
- predmet
- laboratorija
- udžbenik
- Za mini svet PREDUZEĆE, potencijalni entiteti mogu biti:
  - radnik
  - sektor (organizaciona jedinica)
  - radno mesto
  - projekat
  - plan

**Skup entiteta** (*engl. Entity Set*) je kolekcija svih entiteta određenog tipa entiteta u bazi podataka u nekom trenutku. Praktično, entiteti koji egzistiraju u ljudskom intelektu kao predstave realnih subjekata, objekata ili događaja mogu se klasifikovati u skupove sličnih entiteta za koji se koristi termin **skup entiteta**.

**Primer:** Skupovi entiteta

- studenti jednog fakulteta
- radnici jednog preduzeća
- proizvodi jednog preduzeća
- zgrade jednog preduzeća
- uplate na žiro račun

**Tip entiteta** (*egl. Entity Type*) je **model** skupa entiteta i koristi se za opis skupa entiteta. Tip entiteta opisuje **šemu** ili **intenziju** skupa entiteta koji dele istu strukturu. Kolekcija entiteta određenog tipa entiteta grupisana u skup entiteta se takođe naziva **ekstenzija** tipa entiteta.

Svaki tip entiteta u ER modelu podataka ima **ime** (imenica u jednini), koje je jedinstveno. Obeležavanje tipa entiteta se vrši na sledeći način:

$$E(A_1, A_2, \dots, A_n)$$

gde je E ime tipa entiteta, a  $A_i \mid i = 1, n$  atributi odabrani za modeliranje entiteta E.

Za skup entiteta se koristi ime tipa entiteta.

Za obeležavanje tipa entiteta koriste se dva stila:

- Stil A: Obeležavaju se velikim slovima latinične azbuke.
- Stil B (UML notacija) Obeležavaju se malim slovima latinične azbuke osim prvog slova.

**Primer:** Tip entiteta

- Ime RADNIK se koristi i za tip entiteta i za trenutni skup svih entiteta radnik u bazi podataka kojoj taj tip entiteta pripada
- Tipovi entiteta RADNIK1, RADNIK2, RADNIK3 modeliraju na razne načine entitet radnik iz realnog sistema:
  - **RADNIK1** (IME, DATUM-ROĐENJA, ADRESA, TELEFON, SEKTOR, LD, ČLANOVI-PORODICE)
  - **RADNIK2** (IME, SEKTOR, LD)
  - **RADNIK3** (ID-RADNIKA, MATIČNI-BROJ-STANOVNIKA, IME-RADNIKA)

**Pojava (instanca) tipa entiteta** se odnosi na skup podataka kojima je opisan jedan konkretni entitet zadatog tipa.

**Primer:** Pojava tipa entiteta

Dve pojave tipa entiteta RADNIK2(IME, SEKTOR, LD) mogu da budu:

- SAVA KRSTIĆ, INSTITUT, 85000
- JOVAN RISTIĆ, TEST, 57000

Predstavljanje tipa entiteta u ER dijagramu se vrši jednostavnim pravougaonikom koji sadrži ime tipa entiteta (slika 3-2.).



**Slika 3-2. Grafička notacija za tip entiteta**

### 3.1.1 Atributi

Atribut je osobina, obeležje odnosno svojstvo koje opisuje neke aspekte objekta koji treba da se čuvaju. Svaki entitet ima obeležja odnosno svojstva koja ga opisuju. Svi entiteti jednog skupa entiteta imaju bar jedno zajedničko svojstvo na osnovu koga su svrstani u isti skup.

U prethodnom odeljku je navedeno da tip entiteta, kao model skupa entiteta opisuje šemu skupa entiteta koji dele istu strukturu. Tip entiteta sadrži attribute koji su odabrani za modeliranje skupa entiteta.

**Primer:** Atributi entiteta

- Entitet radnik može imati svojstva, odnosno attribute: matični broj radnika, ime, prezime, datum rođenja,...
- Entitet student može imati attribute: Broj indeksa, ime, prezime, datum rođenja, godina studija, ...

Za obeležavanje atributa koriste se dva stila:

- Stil A: Obeležavaju se velikim slovom sa crticom ili znakom za podvlačenje kao poveznikom između reči.
- Stil B (UML notacija): Obeležavaju se nizom reči koje se pišu malim slovima, osim prvog slova svake reči. Nema delimitera između reči.

**Primer:** Obeležavanje atributa

Stil A:

IME  
DATUM\_UPISA ili DATUM-UPISA  
BOJA\_KOLA ili BOJA-KOLA

Stil B:

Ime  
DatumUpisa  
BojaKola

Svaki konkretan entitet ima vrednosti za svaki od atributa koji ga opisuje. Na primer, jedan od studenta ima broj indeksa 1231, njegovo ime je Petar (odnosno vrednost atributa ime je Petar), prezime Petrović, godina studija je III.

**Domen atributa** specificira skup mogućih vrednosti atributa. Svaki konkretan entitet za svaki od atributa ima određenu vrednost iz odgovarajućeg domena.

Obeležavanje domena: *dom(ime\_atributa)*

**Primer:** Domen atributa

BOJA\_KOLA, OCENA\_STUDENTA i PRELAZNA\_OCENA:  
dom(BOJA\_KOLA) = (bela, crvena, zelena)  
dom(OCENA\_STUDENTA) = (5, 6, 7, 8, 9, 10)  
dom(PRELAZNA\_OCENA) = (6, 7, 8, 9, 10)

Ograničenje domena imaju gotovo svi modeli podataka. Opšti oblik ograničenja domena je: (*tip podatka, dužina podatka, uslov*)

Tip podatka i dužina podatka su standardna ograničenja domena. Tip podatka definiše vrstu znakova putem kojih se izražava vrednost atributa. Dužina podatka definiše maksimalni broj znakova koji mogu biti upotrebljeni za izražavanje vrednosti atributa. Često ova ograničenja nisu dovoljno precizna, pa se dodaje treća komponenta – uslov, koja preciznije definiše ograničenje domena.

Standardna ograničenja domena su tipovi podataka sa dužinama tih podataka: INTEGER(broj cifara), REAL(broj cifara ispred zapete, broj cifara iza zapete), CHARACTER(broj znakova), DATE, LOGICAL, itd.

**Primer:** Ograničenje domena

- Ako je dom(OCENA) pridruženo standardno ograničenje INTEGER(2), tada važi  $\text{dom(OCENA)} = \{0, 1, 2, \dots, 99\}$
- Ako je dom(NAZIV\_PREDMETA) pridruženo standardno ograničenje CHARACTER(15), tada važi da naziv predmeta može biti bilo koji niz dužine 15 znakova

**Pojava (instanca) atributa** je konkretna vrednost koju atribut uzima iz svog domena da predstavi podatak o određenom entitetu. Samo instanca atributa može predstavljati podatak.

**Primer:** Pojava atributa

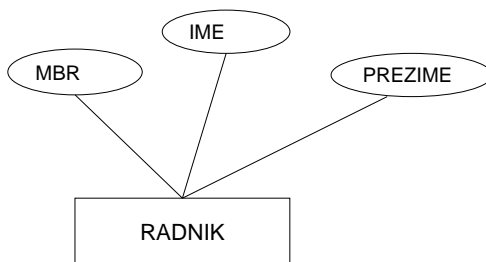
Student PERA ILIĆ ima broj indeksa RE 5034/88. Tada je podatak PERA ILIĆ instanca atributa IME\_STUDENTA, a RE5034/88 instanca atributa BROJ\_INDEKSA

**Primer:** Pojava svih atributa

Dve vrednosti svih atributa tipa entiteta **RADNIK**(IME, ADRESA, DATUM\_ROĐENJA, KUCNI\_TELEFON, ZANIMANJE, SEKTOR, LD):

- *Pera Ilić, Niška 16 18000 Niš, 05-FEB-68, 018-315-072, inženjer elektronike, Razvoj, 35000*
- *Mina Stojanović, Majakovskog 29 18000 NIŠ, 25-JUL-85, 018-515-799, diplomirani pravnik, Zajedničke službe, 25000*

Predstavljanje atributa u ER dijagramu je preko elipsi koje sadrže ime atributa i povezani su linijama sa odgovarajućim tipom entiteta. Na slici 3-3. prikazan je tip entiteta RADNIK sa atributima Matični broj (MBR), Ime i Prezime.



**Slika 3-3. Predstavljanje atributa u ER dijagramu**

Atributi mogu da budu **prosti i složeni**. Atribut je **prost (elementaran)** ako se dalje ne može dekomponovati ili ako se u konkretnoj situaciji ne dekomponuje na



komponente koje čine atribut. Vrednost elementarnog atributa je **elementarni (prost) podatak**. Atribut je **složen (kompozitan)** ako je sastavljen od više elementarnih atributa. Vrednost složenog atributa je **složeni (strukturni) podatak**.

**Primer:** Prosti i složeni atributi

Prosti atributi:

OCENA-STUDENATA, BOJA-AUTOMOBILA, NAZIV-PROIZVODA

Složeni atributi:

ADRESA-STUDENTA, IME-STUDENTA, DATUM-UPISA

Složeni atribut se može podeliti (dekomponovati) na manje delove koji predstavljaju više osnovnih atributa sa nezavisnim značenjem:

**Primer:** Dekomponovanje složenih atributa

- **ADRESA\_STUDENTA** se može dekomponovati na
  - ULICA, BROJ\_ZGRADE, BROJ\_STANA, GRAD
- **IME\_STUDENTA** se može dekomponovati na
  - LIME, SREDNJE\_SLOVO, PREZIME
- **DATUM\_UPISA** se može dekomponovati na
  - DAN, MESEC, GODINA

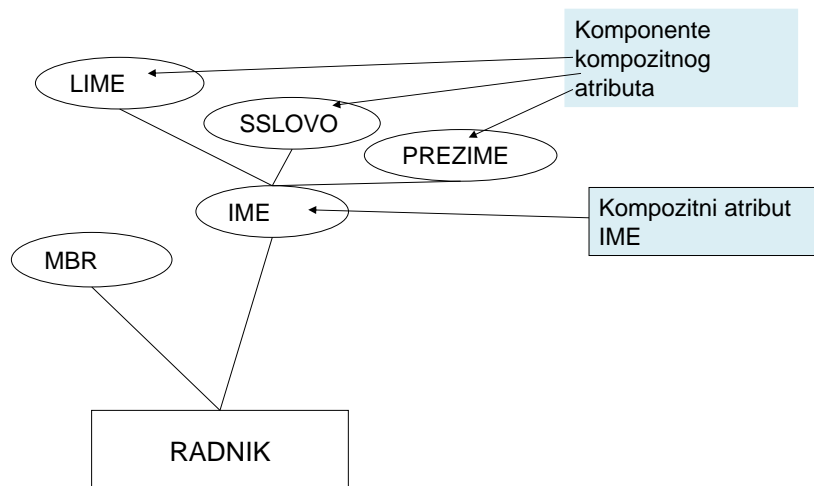
U ovom materijalu dekomponovan složeni atribut će se označavati na sledeći način:

- **ADRESA\_STUDENTA**( ULICA, BROJ\_ZGRADE, BROJ\_STANA, GRAD)
- **IME\_STUDENTA** (LIME, SREDNJE\_SLOVO, PREZIME)
- **DATUM\_UPISA** (DAN, MESEC, GODINA)

Postavlja se pitanje kada i zašto koristiti složene attribute? Složeni atributi su korisni za modeliranje situacije gde se korisnici ponekad obraćaju atributu kao celini, a ponekad pojedinim komponentama tog atributa. Ako se složeni atribut uvek referencira kao celina treba ga modelirati kao prost atribut. Na primer, ako kod adrese studenta nema potrebe za nezavisnim referenciranjem pojedinih komponenti adrese, adresu studenta treba modelirati kao prost atribut.

Predstavljanje složenih (kompozitnih) atributa u ER dijagramu je na sličan način kao predstavljanje prostih atributa – elipsom sa upisanim imenom

kompozitnog atributa koja je povezana sa odgovarajućim tipom entiteta (slika 3-4.). Komponente se predstavljaju na isti način, ali su povezane sa odgovarajućim kompozitnim atributom čiji su delovi.



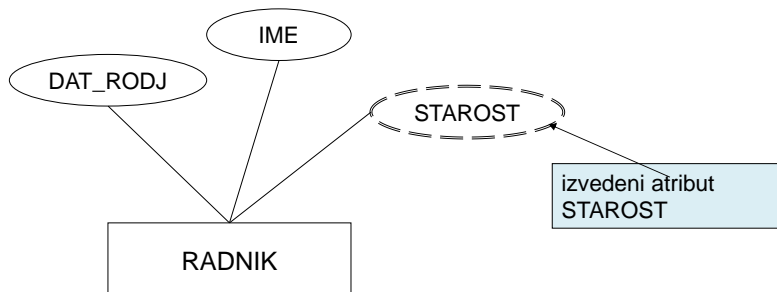
Slika 3-4. Složeni (kompozitni) atributi u ER dijagramu

**Izvedeni atribut** je atribut čije se vrednosti dobijaju primenom nekog algoritma na vrednosti drugih atributa (elementarnih, složenih, izvedenih). Vrednost izvedenog atributa je **izvedeni podatak**.

**Primer:** Izvedeni atribut

- Atribut SREDNJA\_OCENA studenta se može dobiti primenom algoritma za izračunavanje srednje vrednosti na atribut OCENA u entitetu STUDENT
- Atribut BROJ\_ZAPOSLENIH u preduzeću se izvodi primenom algoritma prebrojavanja pojava entiteta RADNIK

Predstavljanje izvedenih atributa u ER dijagramu se vrši na isti način kao i kod prostih atributa, s tim da je linija isprekidana (slika 3-5.).



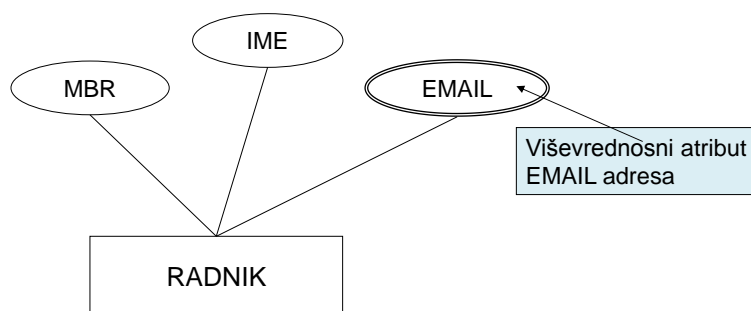
Slika 3-5. Grafička notacija za izvedeni atribut

### Jednovrednosni i viševrednosni atributi

Za određeni entitet jedan atribut može imati jednu ili više vrednosti. U prvom slučaju atribut je **jednovrednosni**, a u drugom **viševrednosni**.

#### Primer: Jednovrednosni i viševrednosni atributi

- Atribut DATUM\_ROĐENJA entiteta RADNIK je jednovrednosni atribut jer radnik može imati samo jedan datum rođenja, npr.05-FEB-68
- Atribut KUĆNI\_TELEFON entiteta RADNIK je viševrednosni atribut jer radnik može imati više telefona, npr.520-505 i 224-061



Slika 3-6. Grafička notacija za viševrednosni atribut

Predstavljanje viševrednosnih atributa u ER dijagramu se vrši na isti način kao i kod prostih atributa, s tim da je linija elipse dvostruka (slika 3-6).

#### Null vrednosti atributa

U nekim slučajevima, može da se desi da za konkretan entitet neki od atributa nema definisanu vrednost. U takvim situacijama se koristi specijalna vrednost - **Null** vrednost atributa. Praktično, atribut nema definisanu vrednost u dva slučaja: ako se radi o neprimerenom svojstvu za konkretan entitet, ili ako vrednost atributa postoji, ali je nepoznata.

Null vrednost može da ima dva značenja:

- Postojeća ali nepoznata vrednost – Vrednost atributa za posmatrani entitet ne postoji ili još uvek nije poznata. Na primer, za radnika koji je tek treba ili je tek počeo da radi vrednost atributa prethodni radni staž nije poznata.
- Neprimerena vrednost atributa – Vrednost atributa za posmatrani nije primenjiva. Na primer, ako za relaciju RADNIK imamo atribut FAKULTET u kome se čuva naziv fakulteta koji je radnik završio, svi radnici sa srednjom školskom spremom će imati NULL vrednost za taj atribut.

**Primer:** Semantika Null vrednosti

Semantika Null vrednosti, neprimerena vrednost atributa:

Ako radnik IVANA GOCIĆ nema telefon u stanu, atribut TELEFON za ovog radnika će imati NULL vrednost.

Ako radnik IVANA GOCIĆ nije trenutno raspoređena ni u jednom sektoru, atribut SEKTOR će imati NULL vrednost

Semantika NULL vrednosti, postojeća ali nepoznata vrednost:

Ako je adresa radnika IVANE GOCIĆ nepoznata, atribut ADRESA će imati NULL vrednost

### 3.1.2 Ključ tipa entiteta

Da bismo jedan entitet jednoznačno identifikovali u posmatranom skupu entiteta on mora posedovati neko svojstvo, ili kombinaciju od nekoliko svojstava, takvu da vrednost tog ili tih svojstava jednoznačno određuju svaku pojavu tog tipa entiteta. Takva svojstva nazivamo karakterističnim, a njihove vrednosti koristimo kao **identifikator entiteta** unutar skupa.

Na primer, skup entiteta RADNIK predstavljamo relacijom/tabelom gde svaka vrsta odgovara jednom entitetu, odnosno predstavlja jednog konkretnog radnika. U tom slučaju neophodno je prepoznati svojstva (atribute) koje možemo da koristimo za identifikaciju radnika unutar skupa radnika.

Minimalni skup atributa koji ima jedinstvenu vrednost za sve pojave određenog tipa entiteta je **ključ** tog tipa entiteta. Svaki tip entiteta poseduje bar jedan ključ. Tip entiteta može imati više ključeva - to su **ključevi kandidati**. Jedan od ključeva kandidata se bira za **primarni ključ** tipa entiteta.

**Primer:** Ključ tipa entiteta

Zadat je tip entiteta **RADNIK3** (ID\_RADNIKA, MBR, IME\_RADNIKA) gde su ID\_RADNIKA jedinstveni identifikator radnika unutar preduzeća, a MBR jedinstveni matični broj radnika.

Ključ može biti ID\_RADNIKA ali i MBR. ID\_RADNIKA ima jedinstvene vrednosti za radnike tog preduzeća, a MBR je jedinstven za sve stanovnike pa i za radnike tog preduzeća. Oba atributa su kandidati za ključeve.

**Primer:** Ključ tipa entiteta

Identifikovati primarne ključeve tipova entiteta RADNIK(IME, SSLOVO, PREZIME, DATUM\_RODJENJA, MBR) i SEKTOR(SBROJ, NAZIV).

U relaciji RADNIK primarni ključ je očigledno matični broj radnika (MBR). Kod radnika se može identifikovati i potencijalni kompozitni ključ kandidat, na primer od kombinacije atributa ime (ime, srednje slovo i prezime zajedno) i

datuma rođenja. Naravno, ovakav ključ se može izabrati ako ne postoji neki očigledniji i jednostavniji kao što je u ovom slučaju matični broj.

Primarni ključ u relaciji SEKTOR je broj sektora, odnosno atribut SBROJ, zato što na jedinstven način identifikuje svaki sektor u preduzeću (ne mogu da postoje dva sektora sa istim brojem). Ključ kandidat (i potencijalni primarni ključ) u ovoj relaciji može biti i naziv sektora, uz pretpostavku da sektori ne mogu da imaju ista imena.

Za obeležavanje primarnog ključa se koriste dva načina:

- Stil A: primarni ključ u tipu entiteta se podvlači kontinualnom linijom
- Stil B: iza imena primarnog ključa se stavlja povelica (#)

**Primer:** Označavanje ključa

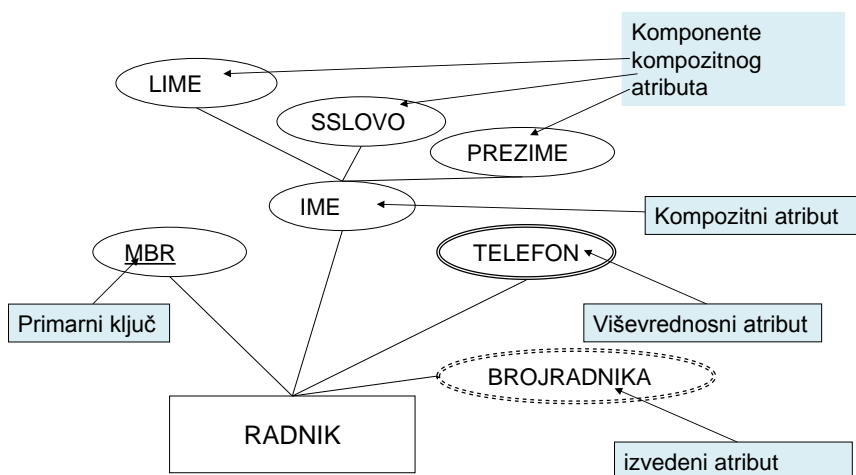
U tipu entiteta RADNIK3 označen je ključ MATIČNI\_BROJ\_RADNIKA korišćenjem stila A:

RADNIK3(ID\_RADNIKA, MBR, IME)

korišćenjem stila B:

RADNIK3(ID\_RADNIKA#, MBR, IME)

Kompletna grafička notacija za predstavljanje atributa u ER dijagramu je data na slici 3-7. Prikazan je tip entiteta RADNIK čiji je primarni ključ MBR, i koji ima složeni atribut IME, viševrednosni TELEFON i izvedeni atribut BROJRADNIKA.



Slika 3-7. Grafička notacija za prikaz tipa entiteta i njegovih atributa

### 3.2 Poveznici, skup poveznika i tip poveznika

U realnom sistemu između entiteta postoje određeni odnosi (relacije). Među entitetima jednog skupa, kao i među entitetima više skupova, mogu postojati različite relacije, što se u ER modelu podataka modelira kao **tip poveznika** odnosno **tip veze**. Tip veze  $R$  među tipovima entiteta  $E_1, E_2, \dots, E_n$ , definiše skup asocijacija ili skup veza među entitetima ovih tipova. Obično tip veze označavamo kao  $R(E_1, E_2, \dots, E_n)$ .

**Skup poveznika**  $R = \{e_1, e_2, \dots, e_n | e_i \in E_i, i=1, \dots, n\}$  predstavlja relaciju (u matematičkom smislu) između  $n$  ( $n \geq 2$ ) skupova entiteta. Skupovi  $E_i$  ne moraju biti različiti, a svaka  $n$ -torka  $(e_1, e_2, \dots, e_n)$  u  $R$  predstavlja jedan poveznik odnosno jednu vezu. Svaki entitet u  $n$ -torci ima svoju ulogu koja se u ER dijagramu može navesti. Ako se uloge entiteta u  $n$ -torci eksplicitno navedu tada redosled navođenja entiteta nije bitan. Između dva ista skupa entiteta može postojati više različitih skupova poveznika. Ako poveznik povezuje entitete istog skupa, naziva se **rekurzivnim**.

Tip poveznika je model skupa poveznika. Obično se za naziv tipa poveznika koristi naziv skupa poveznika, tj.  $R(E_1, E_2, \dots, E_n; B_1, B_2, \dots, B_m)$ . Za svaki tip entiteta  $E_i | i=1, n$  se kaže da participira u tipu poveznika  $R$ .

**Pojava** odnosno **instance tipa poveznika** se odnosi na konkretne veze između pojedinih entiteta odgovarajućih skupova entiteta koji učestvuju u vezi.

**Primer:** Tip poveznika

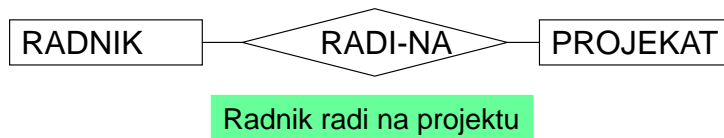
Mogući tipovi poveznika između tipova entiteta RADNIK i PROJEKAT:

**RADI-NA**(RADNIK,PROJEKAT) – definiše vezu radnika sa projektima

**RUKOVODI**(RADNIK,PROJEKAT) – definiše radnika rukovodioca projekta

Rekurzivni tip poveznika za tip entiteta RADNIK:

**JE-RUKOVODILAC**(RADNIK,RADNIK) – definiše odnos radnika i nadređenog/šefa (u ovom slučaju važna je uloga!)



Slika 3-8. Grafička notacija za tip poveznika

Predstavljanje tipa poveznika u ER dijagramu je dato na slici 3-8. Ime tipa poveznika je navedeno velikim slovima, a prikazane su veze ka tipovima entiteta

koji učestvuju u tom tipu poveznika. Na slici je prikazan tip poveznika RADI-NA koji definiše vezu radnika sa projektom.

**Primer:** Skup poveznika

Posmatračemo entitete RADNIK i PROJEKAT. Neka u realnom sistemu između ova dva entiteta postoje odnosi

- “radnik radi na projektu”
- “radnik rukovodi projektom”
- “radnik je rukovodilac radniku”

Neka su Mina i Pera elementi skupa RADNIK, a Web server element skupa PROJEKAT

Poveznik (Mina, Web server) može imati značenje “Radnik Mina radi na projektu Web server”.

Poveznik (Pera, Web server) može imati značenje “Radnik Pera rukovodi projektom Web sever”.

Poveznik (Pera, Mina) može imati značenje “Radnik Pera je rukovodilac radnika Mina”.

U ovim primerima nisu eksplicitno navedene uloge entiteta u poveznicama.

**Stepen tipa poveznika** je broj tipova entiteta koji participiraju u tipu poveznika. Tip poveznika stepena 1 naziva se rekurzivni, tip poveznika stepena 2 se naziva binarni, tip poveznika stepena 3 se naziva ternarni, itd. U praksi se najčešće koriste **binarni tipovi poveznika**.

**Primer:** Stepentipapoveznika

Dva tipična primera rekurzivnog tipa poveznika:

- Pomenuta veza radnika i njegovog nadređenog (rukovodioca), koji je takođe radnik (podaci o radnicima, bez obzira da li je rukovodilac ili ne se čuvaju u istoj tabeli):

JE-RUKOVODILAC(RADNIK,RADNIK)

- veza dve osobe koje su u braku:

U-BRAKU(OSOBA,OSOBA)

Binarni tipovi poveznika (radnici rade na projektima, projekti imaju rukovodioce, radnici imaju rukovodioce):

RADI-NA(RADNIK,PROJEKAT)

RUKOVODI(RADNIK,PROJEKAT)

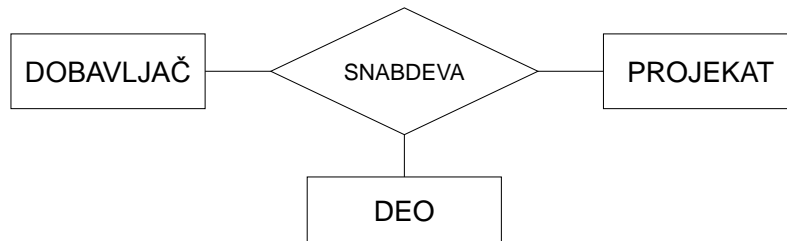
JE-RUKOVODILAC(RADNIK,RADNIK)

Ternarni tip poveznika (za potrebe projekta preko dobavljača se nabavlja deo):

SNABDEVA(DOBAVLJAČ,DEO,PROJEKAT)

Na slici 3-9. prikazan je ternarni tip poveznika koji povezuje dobavljače, projekat i deo. Tip poveznika SNABDEVA modelira vezu koja postoji u mini-svetu kada se za potrebe projekta nabavljaju delovi od nekog konkretnog dobavljača. Ternarna veza se u opštem slučaju ne može uvek predstaviti sa tri binarne veze.

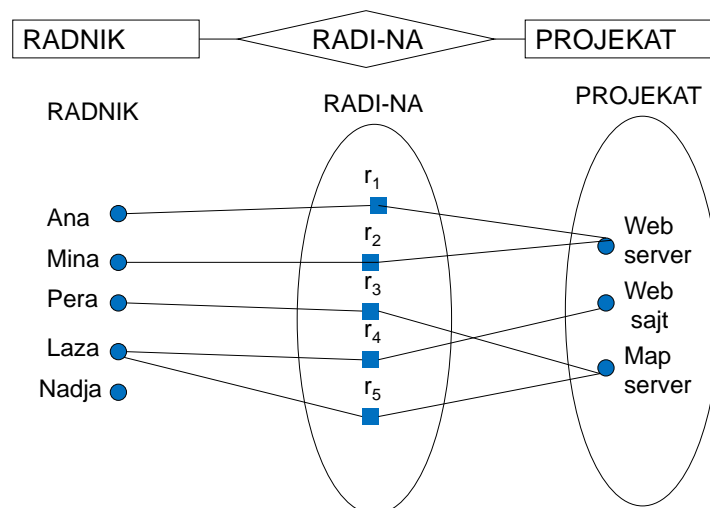
**Primer:** Ternarni tip poveznika



Slika 3-9. Ternarni tip poveznika SNABDEVA

**Primer:** Pojava tipa poveznika

Neke pojave tipa poveznika RAD-NA između radnika i projekata su date na slici 3-10.



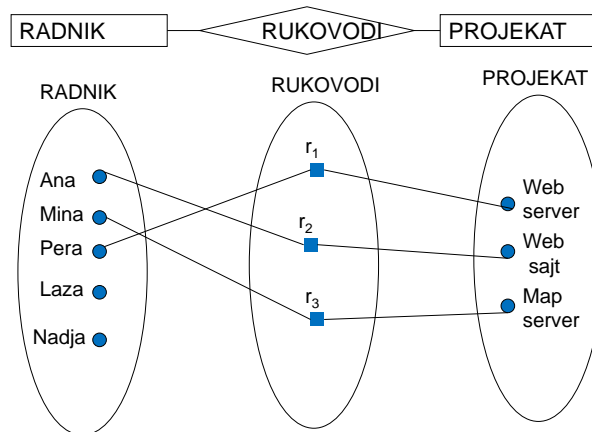
Slika 3-10. Neke pojave tipa poveznika RAD-NA



U prethodnom primeru se može uočiti da svaka pojava  $r_i$  se odnosi na jednu konkretnu vezu između jednog radnika, napr. Ane i jednog projekta, napr. Web server. Svaki od radnika i svaki od projekata može da učestvuje u više veza, ali i ne mora da učestvuje ni u jednoj vezi – što je definisano ograničenjima opisanim u narednoj sekciji.

**Primer:** Pojava tipa poveznika

Neke pojave tipa poveznika RUKOVODI (slika 3-11).

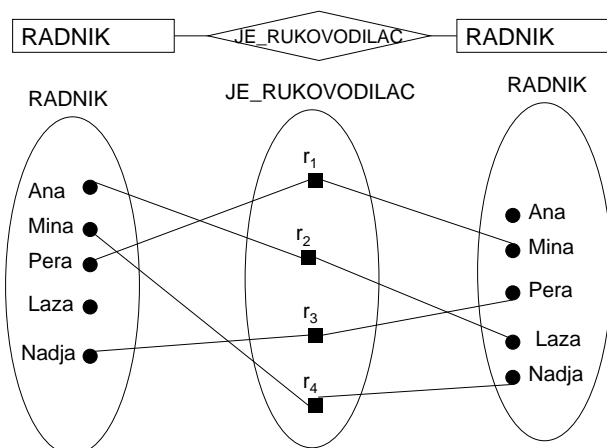


Slika 3-11. Neke pojave tipa poveznika RUKOVODI

U navedenom primeru može se uočiti da ne učestvuju svi radnici u ovoj vezi (odnosno postoje radnici koji nisu rukovodioci projekata), ali svi projekti imaju rukovodioce. Navedene tvrdnje su deo ograničenja koja treba definisati za ovaj tip veze.

**Primer:** Pojava tipa poveznika

Neke pojave tipa poveznika JE\_RUKOVODILAC (slika 3-12).

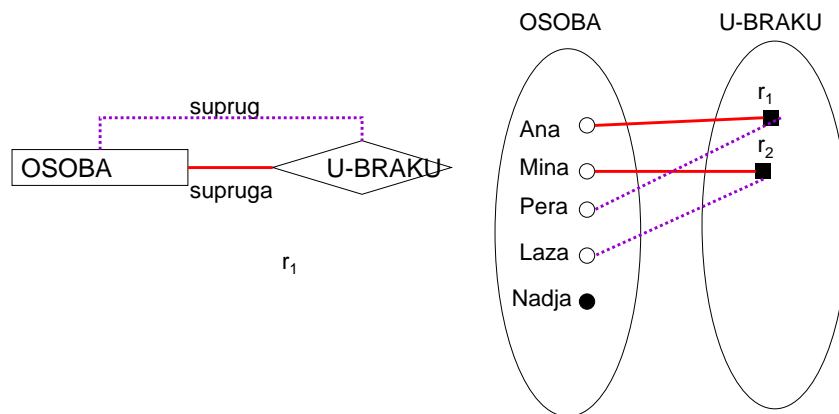


Slika 3-12. Neke pojave tipa poveznika JE\_RUKOVODILAC

U ovom slučaju je očigledno da se sa obe strane veze nalaze radnici, ali i da ne moraju svi radnici da budu rukovodioci, a praktično i nemaju svi radnici rukovodioca (najmanje jedan koji je na vrhu hijerarhije).

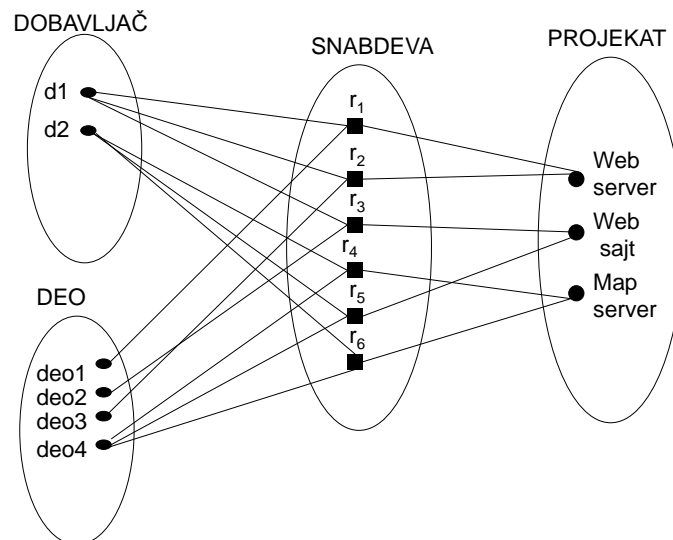
**Primer:** Pojava rekurzivnog tipa poveznika

Neke pojave rekurzivnog tipa poveznika U-BRAKU (slika 3-13).



**Slika 3-13. Neke pojave rekurzivnog tipa poveznika U-BRAKU**

Na slici 3-13 prikazan je i deo ER dijagrama koji definiše rekurzivni tip poveznika sa navednim ulogama. Pri tome su korišćene različite linije (iako to nije u skladu sa notacijom ER modela) kako bi se uočile pojave ovog rekurzivnog tipa poveznika.



**Slika 3-14. Neke pojave ternarnog tipa poveznika SNABDEVA**

**Primer:** Pojava ternarnog tipa veze

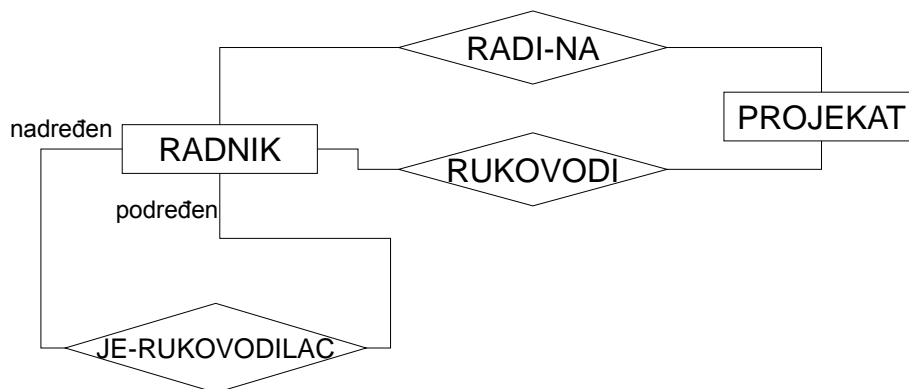
Neke pojave ternarnog tipa poveznika SNABDEVA, čiji je ER model dat u nekom od prethodnih primera (slika 3-9), su prikazane na slici 3-14.

Svaki tip entiteta koji participira u tipu poveznika ima posebnu **ulogu** u tom odnosu. **Ime uloge** označava ulogu koju igra entitet koji participira u povezniku u svakoj instanci poveznika. Ime uloge pomaže u razumevanju odnosa koji poveznik modelira. Ime uloge je posebno važno kod rekursivnih poveznika!

**Primer:** Uloga kod tipova poveznika

- U tipu poveznika RAD-NA(RADNIK,PROJEKAT) tip entiteta RADNIK je u ulozi *zaposlen*, a PROJEKAT u ulozi *poslodavac*
- U tipu poveznika JE-RUKOVODILAC(RADNIK,RADNIK) tip entiteta RADNIK igra dvojaku ulogu, kao nadređeni i podređeni radnik u hijerarhiji rukovođenja

Uloga se u ER dijagramu navodi pored linije koja povezuje simbol tipa poveznika i tip entiteta i odnosi se na ulogu tog tipa entiteta u tom tipu poveznika. Slika 3-15. prikazuje uloge za rekursivni tip veze JE\_RUKOVODILAC.



Slika 3-15. Prikaz uloge u ER modelu

**Atributi tipa poveznika**

Tip poveznika može da ima svoje atribute, na način kako ih imaju tipovi entiteta. Na primer, za tip veze sa slike 3-10, može da se definiše broj radnih sati angažovanja radnika na konkretnom projektu. Na sličan način može da se definiše i datum postavljanja rukovodioca za tip veze sa slike 3-12. U oba slučaja navedeni

atribut može da bude atribut bilo kog od dva tipa entiteta koji učestvuju u tipu veze. Izbor da li neki tip veze ima atribut, ili se taj atribut pridružuje nekom od tipova entiteta koji učestvuju u vezi može se izvršiti na osnovu činjenice da li se njegova vrednost razlikuje za svaku vezu u skupu veza.

### 3.3 Ograničenja nad tipom poveznika

Tipovi poveznika imaju izvesna ograničenja koja limitiraju moguće kombinacije entiteta koji mogu participirati u skupu poveznika. Ova ograničenja omogućavaju modeliranje prirode odnosa koji postoje među entitetima u realnom svetu. U prethodnim primerima koji su se odnosili na pojave tipa poveznika mogli ste da uočite neka od ovih ograničenja koja su se odnosila na brojnost učešća pojedinih entiteta ili na mogućnost da ne učestvuju u nekom skupu veza.

#### Primer: Ograničenja nad tipom poveznika

Neka od ograničenja koja važe za mini-svet koji se odnosi na preduzeće:

- Radnik može raditi na više projekata
- Radnik može rukovoditi samo jednim projektom
- Svaki radnik mora raditi bar na jednom projektu
- Svaki radnik, osim glavnog menadžera preduzeća, mora imati rukovodioca

Dva glavna tipa ograničenja nad tipom poveznika su:

- Odnos kardinaliteta, ili kardinalnost (brojnost)
- Participacija (odnosno učešće)

#### Ograničenje kardinalnosti

Za modeliranje realnog sveta važna je klasifikacija svih veza s obzirom na broj entiteta određene vrste koji su u određenoj vezi sa entitetima druge vrste, odnosno s obzirom na **kardinalnost** odgovarajuće relacije među povezanim tipovima entiteta. **Kardinalnost** binarnog tipa poveznika  $R(E1,E2)$  specificira **maksimalni broj** instanci poveznika u kojima entiteti  $E1$  i  $E2$  mogu participirati.

Mogući odnosi kardinaliteta tipa poveznika  $R(E1,E2)$  su:

- jedan-prema-jedan (1 : 1)
- više-prema-jedan (N : 1)
- jedan-prema-više (1 : N)
- više-prema-više (N : M)  
(gde je  $N \geq 0$  i  $M \geq 0$ )

**Veza 1 : 1.** Kod ove vrste veza svaki element prvog skupa može biti povezan sa najviše jednim elementom drugog skupa. Istovremeno svaki element drugog skupa može biti povezan sa najviše jednim elementom prvog skupa.

**Veza N : 1.** Kod ove vrste veza svaki element prvog skupa može biti povezan sa najviše jednim elementom drugog skupa. Istovremeno svaki element drugog skupa može biti povezan sa više elemenata prvog skupa.

**Veza 1 : N.** Kod ove vrste veza svaki element prvog skupa može biti povezan sa više elemenata drugog skupa. Istovremeno svaki element drugog skupa može biti povezan samo sa jednim elementom prvog skupa.

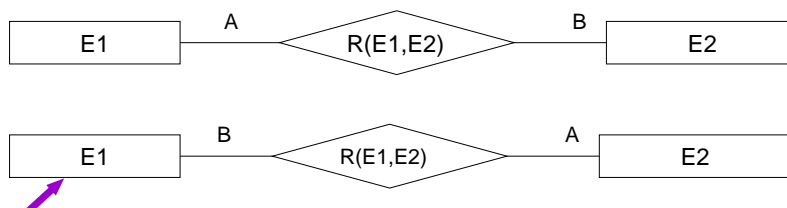
**Veza N : M.** Svaki element prvog skupa entiteta može biti povezan sa više elemenata drugog skupa, i obrnuto, svaki element drugog skupa entiteta može biti povezan sa više elemenata prvog skupa.

**Primer:** Ograničenje kardinalnosti

Tip poveznika **RADI-U(RADNIK,SEKTOR)** ima odnos kardinaliteta **N:1** što znači da jedan radnik može biti u vezi (u ulozi zaposlenog) samo sa jednim sektorom, ali svaki sektor (u ulozi poslodavca) može biti u vezi sa proizvoljnim brojem radnika (u ulozi zaposlenih).

Za označavanje kardinaliteta **A:B** tipa poveznika **R(E1,E2)**, kod ER dijagrama postoje dva načina (slika 3-16):

- **A** se navodi uz tip poveznika E1, a **B** uz tip poveznika E2, ili
- **A** se navodi uz tip poveznika E2, a **B** uz tip poveznika E1 (u ovom materijalu koristiće se ovaj način)



**Slika 3-16. Označavanje kardinaliteta**

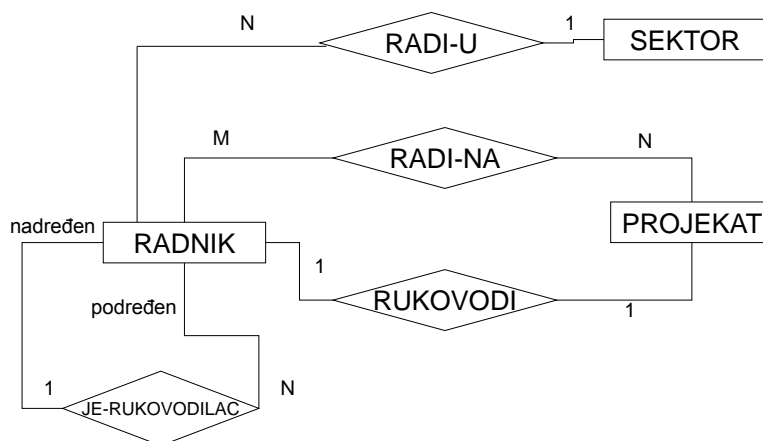
**Primer:** Označavanje kardinaliteta

Na slici 3-17. prikazan je način označavanja kardinaliteta za tip poveznika **RASPOREĐEN(RADNIK,RADNO\_MESTO)** za realni sistem u kojem:

- Radnik može biti raspoređen samo na jedno radno mesto, a
- Na jedno radno mesto može biti raspoređeno više radnika



Slika 3-17. Veza 1:N između tipova entiteta RADNO-MESTO i RADNIK



Slika 3-18. ER dijagram sa označenim kardinalitetima tipa poveznika

Na slici 3-18. prikazan je jedan primer ER dijagrama sa označenim kardinalitetima tipa poveznika. Sektori imaju više radnika, radnici mogu da rade na više projekata, a jedan projekat realizuje više radnika, a jedan od njih je rukovodilac projekta. Prikazana je i rekurzivna veza koja definiše rukovodioce.

### Ograničenje participacije

Ograničenje participacije specificira **minimalni broj** instanci poveznika u kojima entitet može participirati. Ponekad se naziva minimalno ograničenje kardinaliteta. Ograničenje participacije specificira da li postojanje (egzistencija) entiteta zavisi od toga da li je u vezi sa drugim entitetom preko tipa poveznika.

Postoje dva tipa ograničenja **participacije** entiteta u vezi. To su:

- totalna (egzistencijalna) participacija i
- parcijalna participacija

**Totalna participacija** tipa entiteta E1 u tipu veze R(E1, E2) znači da svaki entitet iz skupa entiteta E1 **mora biti povezan** sa nekim entitetom iz skupa entiteta E2 preko veze iz skupa veza modeliranih tipom veze R. To dalje znači da egzistencija nekog entiteta u skupu E1 zavisi od toga da li je povezan sa nekim entitetom iz skupa E2 preko tipa veze R. Otuda se ovaj tip participacije takođe naziva **egzistencijalna participacija**.

**Parcijalna participacija** entiteta E1 u tipu veze R(E1, E2) znači da svaki entitet iz skupa E1 **ne mora biti povezan** sa nekim entitetom iz skupa entiteta E2 preko tipa veze R. To takođe znači da u skupu entiteta E1 mogu postojati entiteti koji nisu povezani ni sa jednim entitetom iz skupa entiteta E2 preko tipa veze R.

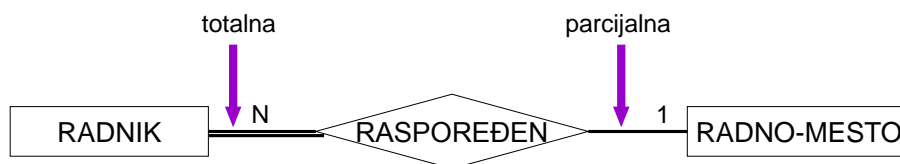
U ER dijagramu, za označavanje participacije tipa poveznika R(E1,E2) koristićemo dva različita tipa linija koje spajaju tip entiteta sa tipom poveznika:

- **Jednostruku liniju** za parcijalnu participaciju i
- **Dvostruku liniju** za totalnu participaciju

**Primer:** Označavanje participacije u ER dijagramu

Na slici 3-19. prikazana su dva prethodno navedena načina prikaza participacije za realni sistem u kojem:

- **Radnik mora biti raspoređen** na tačno jedno radno mesto, a
- Na jedno radno mesto **može biti raspoređeno** više radnika, ali mogu postojati radna mesta na koja niko nije raspoređen

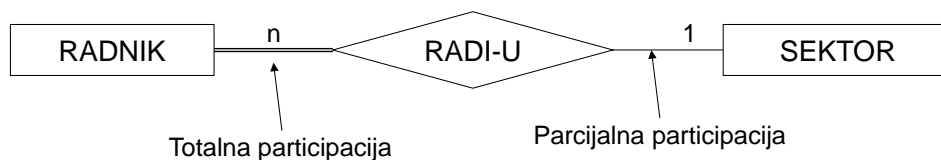


Slika 3-19. Označavanje participacije za tip veze RASPOREĐEN

**Primer:** Označavanje participacije u ER dijagramu

Ograničenje participacije u tipu poveznika RADI-U(RADNIK,SEKTOR) prikazano je na slici 3-20:

- tip entiteta **RADNIK ima totalnu participaciju** ako u preduzeću vlada pravilo da svaki radnik mora biti zaposlen u nekom sektoru
- Tip entiteta **SEKTOR ima parcijalnu participaciju** ako u preduzeću vlada pravilo da mogu postojati sektori bez zaposlenih
- 



Slika 3-20. Označavanje participacije za tip veze RADI-U

Definisanje ograničenja je jako važno u procesu projektovanja baze podataka zato što utiče kasnije na opis baze podataka i tabele koje treba kreirati. Postavlja se pitanje kako se mogu u procesu projektovanja na pravi način definisati navedena ograničenja kardinalnosti i pricipacije a da ne dođe do greške? Pristup koji autori ovog materijala koriste je jednostavan: ako posmatrate vezu između dva tipa entiteta, gledate najpre odnos između jednog entiteta iz prvog tipa entiteta i svih ostalih entiteta iz druge. Na taj način odredite ograničenje sa strane drugog tipa entiteta gde posmatrate sve entitete. Nakon toga isti postupak ponovite za drugi smer, odnosno odnos jednog entiteta iz drugog tipa entiteta sa svim entitetima iz prve.

Za pomenutu vezu **RADI-U**, između relacija **RADNIK** i **SEKTOR** (odnosi se na radnike koji rade u nekom sektoru), ograničenje kardinaliteta se može odrediti na sledeći način:

Označimo odnos kardinaliteta sa **X** i **Y** dok ne odredimo odgovarajuće vrednosti:

**RADNIK : SEKTOR**

**X : Y**

Najpre posmatramo jednog radnika i određujemo sa koliko sektora je on u vezi. Pošto jedan radnik može da radi samo u jednom sektoru, kardinalnost sa strane sektora je 1:

**RADNIK : SEKTOR**

**X : 1**

Nakon toga posmatramo jedan sektor i određujemo sa koliko radnika je u pomenutoj vezi. Jedan sektor može da ima više radnika, pa je kardinalnost sa strane radnika u ovoj vezi **N**:

**RADNIK : SEKTOR**

**N : 1**

### **Strukturalna ograničenja tipa poveznika**

Drugi način zadavanja ograničenja tipa poveznika kod ER modela su tzv. strukturalna ograničenja. Ako se posmatra binarna relacija **R** između skupova pojava dva tipa entiteta **E1** i **E2**, ona se može predstaviti putem dva preslikavanja:

**R1:  $E1 \rightarrow P(E2)$**

**R2:  $E2 \rightarrow P(E1)$**

gde je **P(E)** partitivni skup skupa **E**.

Za svako od ovih preslikavanja se definiše minimalni i maksimalni kardinalitet preslikavanja. Preslikavanjima **R1** i **R2** se može dati sledeća semantička interpretacija: **R1** je uloga entiteta iz skupa **E1**, a **R2** uloga entiteta iz skupa **E2** u njihovoj vezi opisanoj relacijom **R(E1,E2)**.



Praktično, radi se o jednostavnom načinu da se predstave oba navedena ograničenja, kardinalitet i participacija - pored svakog od tipova entiteta koji učestvuju u vezi navede se par vrednosti (min,max), što ima sledeće značenje:

- min = 0,      parcijalna participacija
- min > 0,      totalna participacija
- max = maksimalni broj entiteta određenog tipa koji učestvuju u vezi

U ovom slučaju, tip poveznika  $R(E1, E2)$  se predstavlja na sledeći način:

$$E1 : E2 = R(E1(\min1, \max1) : E2(\min2, \max2))$$

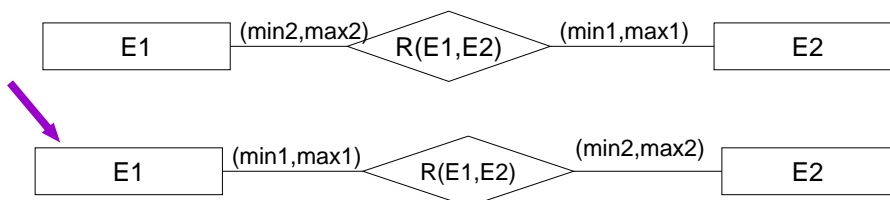
Svaki element skupa  $E1$  je u vezi sa  $\min2$  do  $\max2$  elemenata skupa  $E2$  i obrnuto, svaki element skupa  $E2$  je u vezi sa  $\min1$  do  $\max1$  elemenata skupa  $E1$ .

Minimalna vrednost definiše participaciju entiteta u tipu poveznika  $R(E1,E2)$ : ako je  $\min=1$  preslikavanje je totalno, a ako je  $\min=0$  preslikavanje je parcijalno.

Maksimalne vrednosti definišu odnos kardinaliteta tipa poveznika  $R(E1,E2)$ , tako da vrednosti  $\max1:\max2$  definišu veze: 1:1, 1:N ili N:M.

Za označavanje (min,max) ograničenja tipa poveznika  $R(E1,E2)$  u ER dijagramu postoje dva načina (slika 3-21):

- Brojnost preslikavanja se upisuje uz tip entiteta u koji se on preslikava. Par (min1,max1) se navodi na potegu uz tip poveznika  $E2$ , a par (min2,max2) na potegu uz tip poveznika  $E1$ , ili
- Brojnost preslikavanja se upisuje uz sam tip entiteta. Par (min1,max1) se navodi na potegu uz tip poveznika  $E1$ , a par (min2,max2) na potegu uz tip poveznika  $E2$ .



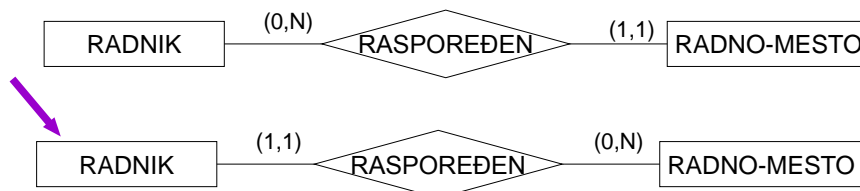
Slika 3-21. Označavanje strukturalnih ograničenja

#### Primer: Predstavljanje strukturalnih ograničenja

Predstavljanje (min,max) ograničenja tipa poveznika u ER dijagramima. Na primeru sa slike 3-22. su prikazana ova dva načina za isti realni sistem u kojem:

- Radnik može biti raspoređen samo na jedno radno mesto,

- Na jedno radno mesto može biti raspoređeno više radnika, a može radno mesto biti slobodno.



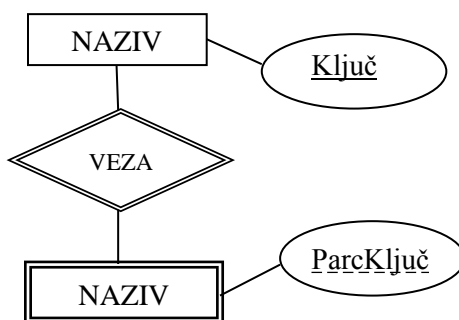
Slika 3-22. Primer označavanja strukturalnih ograničenja

### 3.4 Slabi tip entiteta

Tip entiteta može biti **regularni** (tip entiteta koji smo do sada definisali i koristili) ili **slabi tip entiteta**. Slabi tip entiteta nema sopstvene ključne attribute na način kako su definisani u odeljku 3.1.2. Entiteti slabog tipa entiteta se identifikuju preko veze sa nekim drugim tipom entiteta (regularnim ili slabim) u kombinaciji sa nekim svojim atributom. Taj drugi tip entiteta je **identifikujući tip entiteta** (vlasnički, roditeljski ili dominantni).

Poveznik koji povezuje slabi tip entiteta sa vlasnikom se naziva **identifikujući poveznik**. Slabi tip entiteta uvek totalno participira u identifikujućem tipu veze.

Slabi tip entiteta ima **parcijalni ključ** (diskriminator) koji predstavlja podskup skupa atributa slabog tipa entiteta koji na jedinstven način identifikuje slabe entitete koji su u vezi sa istim vlasničkim tipom entiteta. U najgorem slučaju, kada ne može da se izdvoji takav podskup atributa, parcijalni ključ čini skup svih atributa slabog tipa entiteta.



Slika 3-23. Slabi tip entiteta u ER dijagramu

Slabi tip entiteta se u ER dijagramu prikazuje kao regularni, s tim da se koristi dvostruka linija (slika 3-23). Postoji identifikujući tip veze prema vlasničkom tipu

entiteta prikazana takođe dvostrukom linijom. Parcijalni ključ slabog tipa entiteta se podvlači isprekidanom linijom.

### 3.5 Pregled grafičke notacije ER dijagrama

Pretodno je navedeno da se šema konceptualnog ER i EER modela podataka predstavlja se dijagramom. U prethodnim odeljcima je dat pregled osnovnih koncepata, kao i prikaz i objašnjenje grafičke notacije, a ovaj odeljak daje pregled usvojene grafičke notacije ER dijagrama prema notaciji iz knjige [Elmasri & Navathe, 2010].

Pravila koja su korišćena za obeležavanje entiteta, atributa, veza i ključeva su:

- Entitete obeležavamo velikim slovima. Na primer: u sistemu za registraciju vozila imena entiteta su: VOZILO, VLASNIK, SLUŽBENIK, REGISTRACIJA i ORGANIZACIJA.
- Atributi i veze se mogu obeležavati na dva načina: (1) nizom reči koje se pišu velikim slovima sa znakom za podvlačenje ili crticom između reči; Na primer: IME, BOJA\_KOLA, REG\_BROJ ili BOJA-KOLA, REG-BROJ; (2) nizom reči koje se pišu malim slovima, osim prvog slova svake reči, bez delimitera između reči; Na primer: Ime, BojaKola, RegBroj.
- Domen atributa A obeležavamo sa  $dom(A)$ .
- Tip entiteta E sa atributima  $A_1, A_2, \dots, A_n$  označavamo kao  $E(A_1, A_2, \dots, A_n)$
- Atributi koji čine ključ tipa entiteta biće podvučeni.

Koncepti osnovnog ER modela se po usvojenoj notaciji grafički predstavljaju na sledeći način:

- **Tip entiteta** odnosno se crta kao pravougaonik sa upisanim imenom. Ime se upisuje velikim slovima:



- **Slabi tip entiteta** se crta kao dvostruki pravougaonik sa upisanim imenom kao kod regularnog tipa entiteta:



- **Tip veze** se crta kao romb sa upisanim imenom:

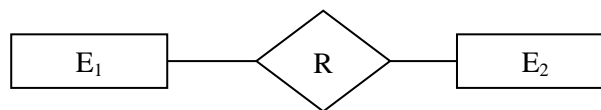


- **Identifikacioni tip veze** se crta kao dvostruki romb sa upisanim imenom:

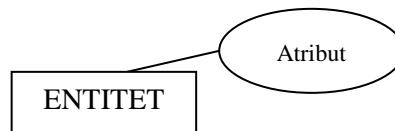


- **Tip veze R** između tipova entiteta  $E_1$  i  $E_2$  crta se kao neusmeren poteg od romba R do grafičkog simbola entiteta  $E_1$  i  $E_2$ :

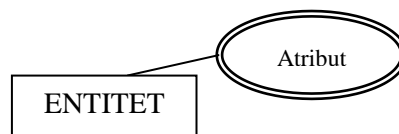
U opštem slučaju, za n-arni tip veze, crta se neusmereni poteg od romba do svakog tipa entiteta koji učestvuje u tom tipu veze.



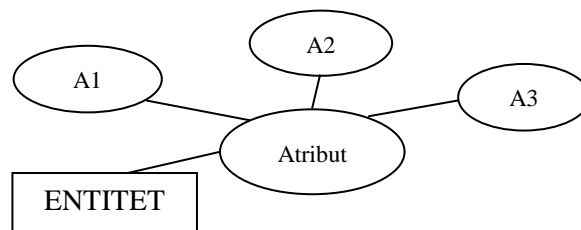
- **Prosti atributi** se prikazuju kao elipse sa upisanim nazivom i povezuju se linijom sa geometrijskom slikom tipa entiteta ili tipa poveznika na koji se odnose:



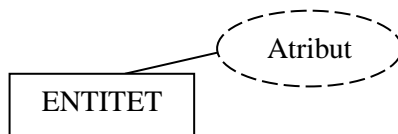
- **Viševrednosni atributi** se predstavljaju dvostrukom elipsom:



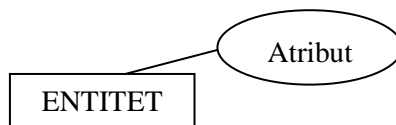
- **Složeni atributi** se predstavljaju preko svojih komponenti:



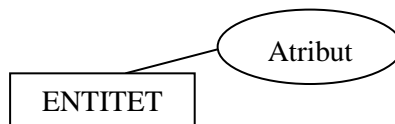
- **Izvedeni atributi** se prikazuju isprekidanom linijom:



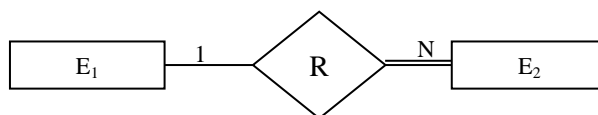
- **Ključni atributi** se prikazuju podvlačenjem imena:



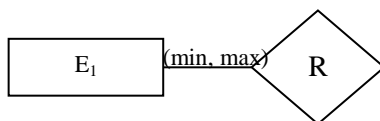
- **Parcijalni ključevi** se podvlače isprekidanom linijom:



- Ograničenja tipa veze, **kardinalnost i participacija**, prikazuju se pored potega od oznake tipa entiteta (pravougaonik) do oznake tipa veze (romb): Kardinalnost je označena brojevima, a participacija linijama. Na slici je prikazana veza tipa 1:N između tipova entiteta  $E_1$  i  $E_2$ . Jednostruka linija odgovara parcijalnom učešću tipa entiteta  $E_1$ , a dvostruka totalnom učešću tipa entiteta  $E_2$ .



- **Strukturno ograničenje** se navodi pored tipa entiteta koji učestvuje u nekoj vezi:



### 3.6 Primeri konceptualnog modeliranja

U ovom odeljku dato je nekoliko primera konceptualnog modeliranja i prikazani su ER dijagrami koji su dobijeni na osnovu zadatih zahteva, kao i primeri za vežbu.

**Zadaci za vežbu:** Osnovni elementi konceptualnog projektovanja

Nacrtati ER dijagrame za sledeće situacije:

### **Entiteti**

- (a) Brod: brod ima ime, registracioni kod, bruto nosivost, i godina igradnje; brodovi se dele na teretne i putničke.
- (b) Restoran: restorani imaju naziv, adresu, broj mesta, telefon, i vrstu hrane (roštilj, riba, pice).
- (c) Krava: krava muzara ima ime, datum rođenja, rasu i numerisnu plastičnu oznaku na uvetu.

### **Veze**

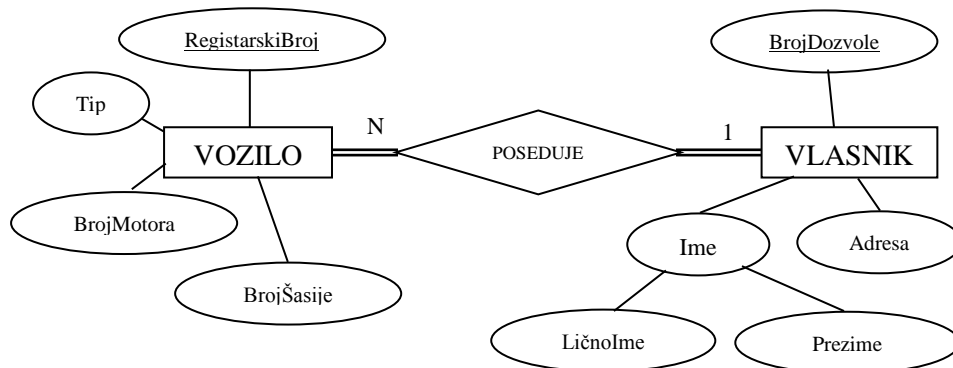
- (d) Seljak ima više krava, ali krava pripada samo jednom seljaku.
- (e) Seljak može da ima svoje krave ili da ih deli (zajedničke krave) sa drugim seljacima.
- (f) Fakultet ima više studenata, ali jedan student može da studira samo na jednom fakultetu.
- (g) Avion može da ima više putnika, ali jedan putnik može da bude na samo jednom letu u određeno vreme.
- (h) Država ima više regiona, svaki region ima više gradova.
- (i) Čevapdžinica prodaje više vrsta pljeskavica. Isti dodaci (salate) se mogu koristiti uz različite tipove pljeskavica.
- (j) Pacijent može da ima više doktora, a jedan doktor ima više pacijenata.

### **Primer:** Vlasnici i njihova vozila

Potrebno je projektovati bazu podataka koja sadrži podatke o vozilima i njihovim vlasnicima. Za vozila se čuva registarski broj, tip, broj šasije i broj motora. Za vlasnike broj dozvole, ime (lično ime i prezime) i adresa. Vlasnici mogu da imaju više vozila.

Na slici 3-24. prikazan je ER dijagram, dobijen na osnovu navedenih zahteva, koji sadrži dva tipa entiteta:

- VOZILO, sa atributima Tip, RegistarskiBroj (ključni atribut), BrojMotora, BrojŠasije, i
- VLASNIK sa atributima Ime (složeni atribut, sastoji se iz dve komponente: LičnoIme, Prezime), Adresa, BrojDozvole (ključni atribut).



**Slika 3-24. Primer ER dijagrama: vlasnici i njihova vozila**

Registarski broj je izabran kao ključ za vozila pošto na jedinstven način identifikuje svako vozilo. Za vlasnike ključ je očigledno broj dozvole. Između ova dva tipa entiteta postoji veza POSEDUJE, koja definiše odnos vlasnika i vozila. Veza je tipa 1:N, zato što vlasnik može da poseduje više vozila, a jedno vozilo ima samo jednog vlasnika. Participacija je na strani vlasnika i vozila totalna, uz pretpostavku da se u bazi podataka čuvaju podaci samo o vlasnicima koji poseduju bar neko vozilo, kao i podaci o vozilima koji imaju vlasnika.

**Zadaci za vežbu:** Projektovanje konceptualne šeme baze podataka

- Projektovati konceptualnu šemu baze podataka koja treba da čuva podatke o slikarima i muzejima u kojima se nalaze njihove slike. Za svaku sliku, treba pamtiti informacije o veličini (dimenzijama), godini kada je ona nastala, naslov i stil. Za slikare pamtiti nacionalnost, datum rođenja i datum smrti (ako je poznat). Za svaki muzej, pamtiti lokaciju, kao i specijalnost, ako postoji.
- Projektovati konceptualnu šemu baze podataka o takmičenju u odbojci. U bazi podataka treba čuvati informacije o ekipama koje učestvuju (naziv, zemlja, trener, najbolji plasman na svetskim i evropskim prvenstvima) i igračima za svaku ekipu. Za igrače pamtiti se ime i prezime, mesto u timu i broj. Brojevi igrača su jedinstveni u okviru ekipe. Treba pamtiti i podatke o utakmicama (jedinstveni identifikator, datum, vreme, sudije, dve ekipe koje igraju i konacni rezultat u setovima). Utakmice se igraju na tri dobijena seta, a za svaki set na utakmici treba pamtiti njegov redni broj i rezultat. Za utakmice pamtiti se i statistika za igrače i ekipu. Za svakog igrača, za svaku utakmicu, pamtiti se broj osvojenih poena i broj promena. Za svaku ekipu na utakmici vodi se statistika o broju as servisa, broju direktnih poena i broju poena na greške protivnika.

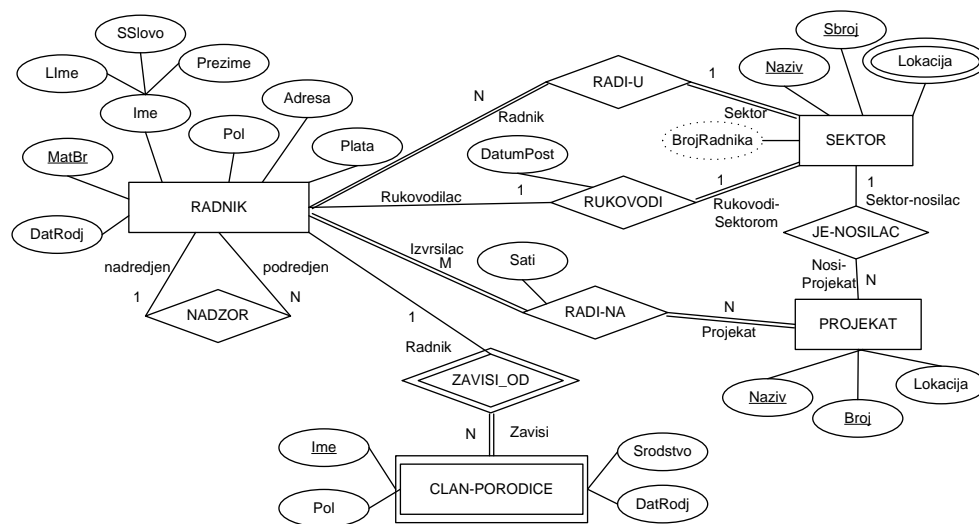
### Primer: Baza podataka PREDUZEĆE

Na osnovu navedenih zahteva projektovati ER dijagram baze podataka PREDUZEĆE.

Zahtevi:

- Preduzeće ima više sektora. Svaki sektor ima ime, broj i rukovodioca. Sektor ima bar četiri radnika. Vodi se evidencija o datumu kada je rukovodilac postavljen na tu funkciju. Sektor može imati više lokacija.
- U sektoru se radi na više projekata. Svaki projekat ima ime, broj i jedinstvenu lokaciju.
- Za svakog radnika se pamti ime, matični broj, adresa, plata, pol i datum rođenja. Svaki radnik radi u jednom sektoru, a može biti angažovan na više projekata, koje ne vodi isti sektor. Pri tome se vodi evidencija o broju radnih časova koje radnik provede na nekom od projekata. Takođe se vodi evidencija o hijerarhiji odgovornosti, odnosno evidentira se za svakog radnika ko mu je neoposredni rukovodilac.
- Vodi se i evidencija o članovima porodice. Za svakog člana evindetira se ime, pol, datum rođenja i srodstvo.

ER dijagram za bazu podataka PREDUZEĆE dat 3-25. Zahtevi su navedni tako da se lako prepoznaju tipovi enititeta i odgovarajuće veze.



Slika 3-25. ER dijagram baze podataka PREDUZEĆE



U procesu projektovanja baze podataka najpre treba prepoznati tipove entiteta, a to su redom: sektori, projekti, radnici i članovi porodice, sa atributima koji su navedeni u svakom od pasusa. Ime radnika može da bude složeni atribut, dok je lokacija sektora očigledno viševrednosni atribut. Broj radnika po sektoru se može izračunati pa predstavlja izvedeni atribut.

Član porodice je slabi tip entiteta pošto se ne može identifikovati preko navedenih atributa (ime, koji je parcijalni ključ), ali može preko vrednosti atributa roditelja odnosno radnika.

Nakon toga treba definisati tipove veza koji se mogu prepoznati čitanjem zahteva ali i na osnovu atributa koje ste definisali u prvom prolazu za prepoznate tipove entiteta. Ako postoji vrednost koju treba da čuvati u nekom tipu entiteta, a predstavlja vrednost atributa drugog tipa entiteta, sigurno se radi o vezi ova dva tipa entiteta. Na primer, u zahtevima stoji da se za sektore čuva i rukovodilac – pošto su rukovodioci radnici, to predstavlja vezu radnika i sektora. Kod radnika se takođe pamte podaci o rukovodiocu (rekurzivni tip veze radnik-radnik), a takođe i sektor u kome radi (veza sa sektorom). Na sličan način u zahtevima je definisana veza sektora i projekata, kao i radnika i projekata.

Kod veza radnika i sektora, gde se definiše rukovodilac sektora, treba čuvati informaciju o datumu postavljanja rukovodioca. Generalno, njegova vrednost se razlikuje za svaku pojedinačnu vezu u tom skupu veza i može se posmatrati kao atribut tipa veze. Ako to nije slučaj, verovatno takav atribut možete da pridružite nekom od tipova entiteta koji učestvuju u vezi. U ovim zahtevima atribut veze je i broj radnih sati koje radnik provede na nekom od projekata zato što se ta vrednost razlikuje za svakog radnika i odnosi se na konkretan projekat na kome on radi.

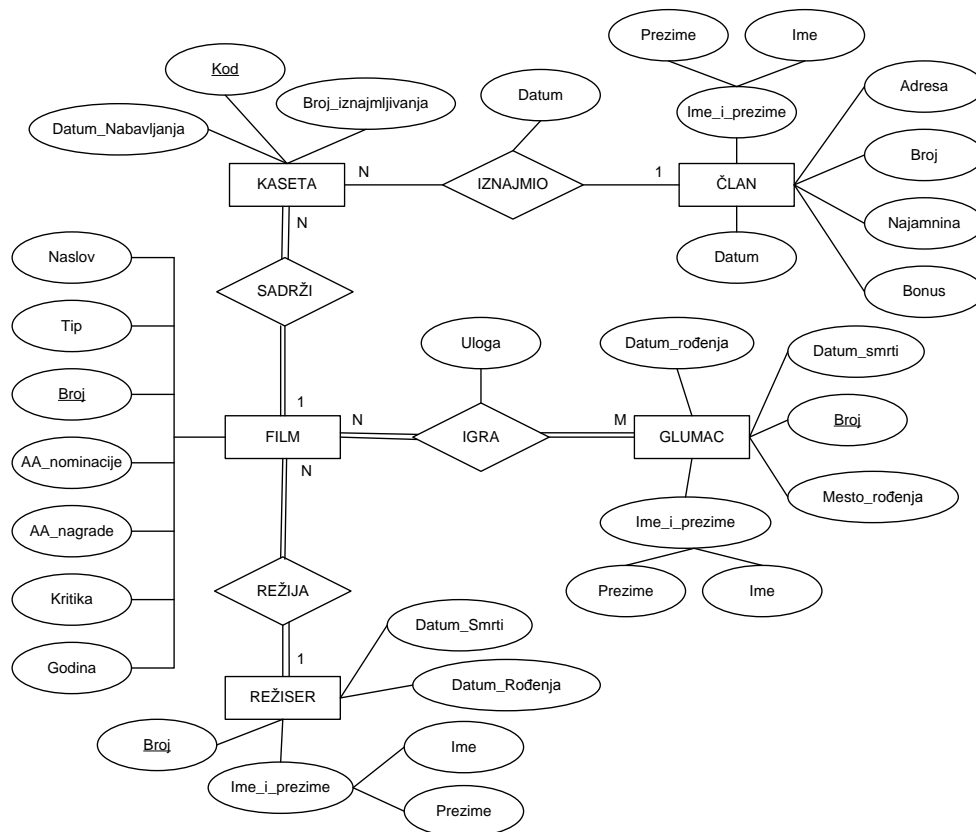
Na kraju, treba odrediti i ograničenja za svaki tip veze. Veza sektora i rukovodioca je očigledno 1:N (sektor ima jednog rukovodioca, a jedan radnik je rukovodilac jednom sektoru), gde ne moraju svi radnici da budu rukovodioci sektora (parcijalno učešće radnika), a svaki sektor mora da ima rukovodioca (totalno učešće sektora). Odnos sektora i radnika je 1:N, gde i radnici i sektori participiraju totalno. Odnos radnika i projekata na osnovu zahteva definiše kardinalitet M:N, zato što projekte realizuje više radnika, a radnici mogu da rade na većem broju projekata. Participacija je totalna zato što se podrazumeva da projekti moraju da imaju radnike a da radnici moraju da imaju neko angažovanje na nekom od projekata. Odnos sektora i projekata definiše kardinalitet 1:N, ali sa parcijalnom participacijom.

#### **Primer:** Baza podataka VIDEO\_KLUB

Projektovati ER dijagram baze podataka VIDEO\_KLUB na osnovu navedenih zahteva.

Zahtevi:

- Potrebno je pratiti sledeće informacije o filmovima: jedinstveni broj, naslov filma, režiser, tip (akcioni, komedija, drama,...), rejting filma („kritika“ u novinama označena brojem zvezdica), godina, nominacije za nagrade Akademije (tzv. „Oskar“, Academy Award), dobijene nagrade Akademije.
- Za svaki film treba pamtit i imena glumaca i tip uloge (glavna, sporedna,...). O glumcima se pamte imena (ime i prezime), podaci o datumu rođenja i mestu rođenja, datumu smrti (ako postoji).
- Za svakog glumca postoji jedinstveni identifikator. Pamti podatke o režiserima filmova: za svakog režisera postoji jedinstveni broj, pamte se imena, datum rođenja i datum smrti (ako postoji).



Slika 3-26. EER model baze podataka VIDEO\_KLUB

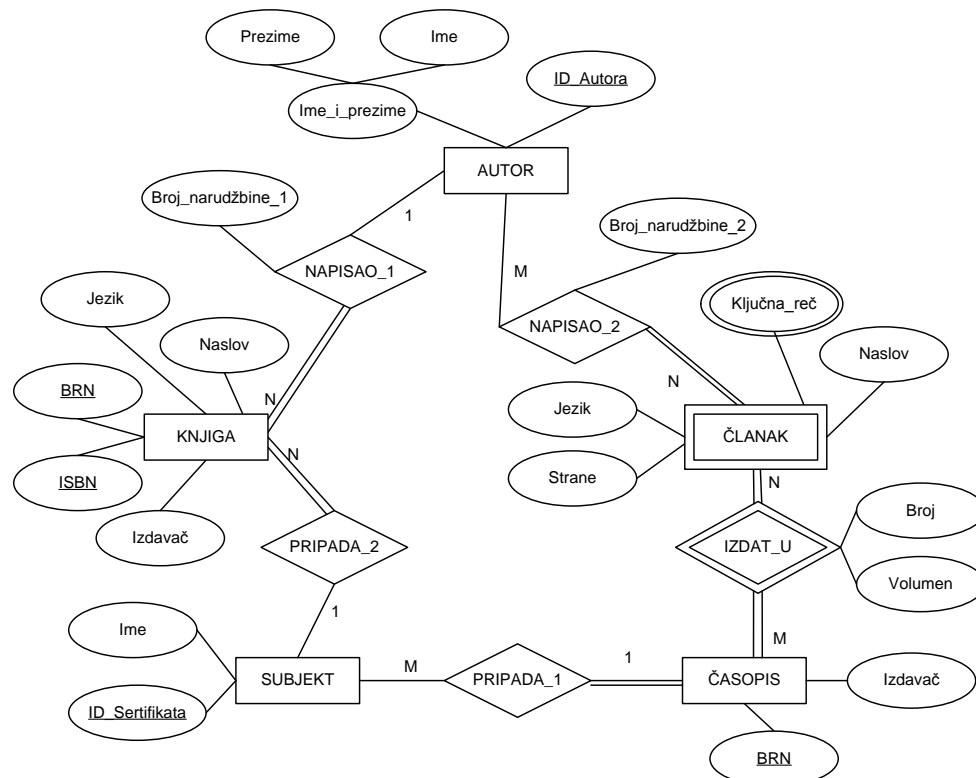
- Treba pamtit i informacije o članovima kluba: broj članske karte, ime i prezime, adresa, jedinstveni broj, datum učlanjenja, ukupan iznos najamnine (od svih iznajmljenih kaset) i vrednost ostvarenog bonusa (određuje se na osnovu pet iznajmljivanja).
- Pamtit podatke o kasetama: jedinstveni kod kasete, datum dobijanja, film koji se na njoj nalazi i broj iznajmljivanja kasete. Više kasete može biti sa istim filmom. Za svakog člana kluba treba pamtit kasete koje je uzeo i datum izdavanja.

ER dijagram baze podataka VIDEO\_KLUB je dat na slici 3-26.

**Zadaci za vežbu:** Konceptualno projektovanje – dodatni primeri

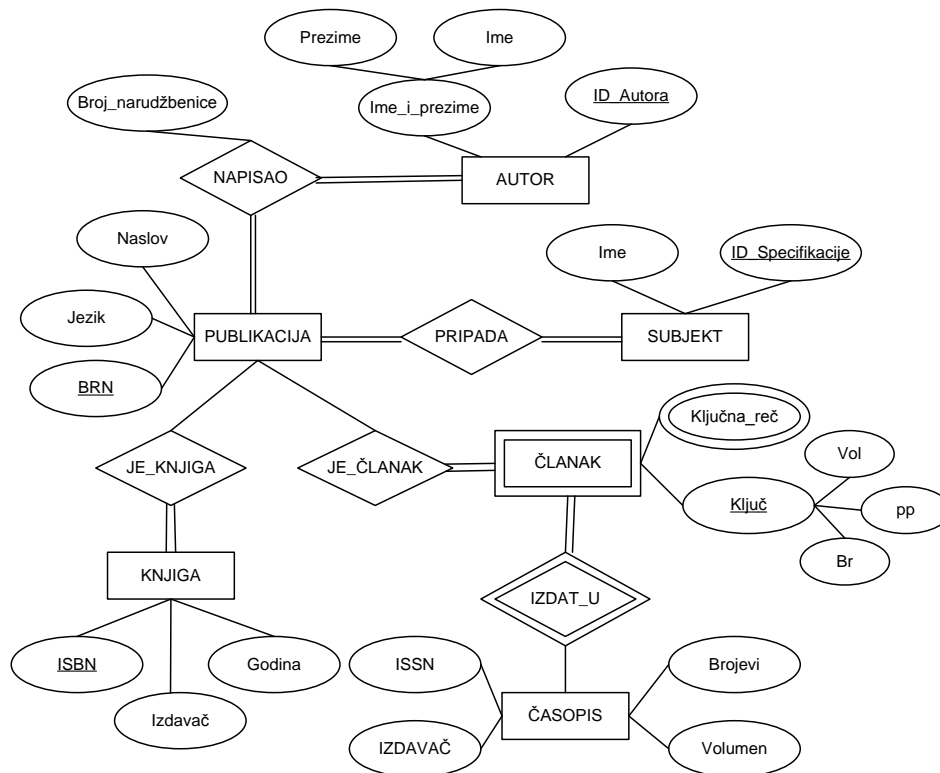
(a) Zadat je ER dijagram na slici 3-27.

1. Proučite ER dijagram i na osnovu datog modela podataka rekonstruišite zahteve baze podataka.
2. Identifikujte i označite sve nedostatke datog modela.



**Slika 3-27. ER dijagram baze podataka o autorima i njihovim delima**

(b) Kompletirati ER dijagram sa slike 3-28. dodavanjem ograničenja za veze (kardinalnost).



Slika 3-28. ER dijagram baze podataka o publikacijama

### 3.6.1 Preporuke kod modeliranja

Projektovanje ER i EER modela baze podataka je složen zadatak koji zahteva poznavanje koncepata ovih modela i dosta iskustva u radu. Osnovne preporuke kojih se treba pridržavati i koje vam mogu pomoći kod modeliranja baze podataka su:

- Identifikujte tipove entiteta detaljnim čitanjem i analizom zahteva. Tipove entiteta možete obično prepoznati traženjem imenica u rečenicama. Kao pomoć možete da koristite spisak objekata realnog sveta koji vam je dat u ovom poglavlju.
- U prvom prolazu pretpostavite da su svi entiteti koje prepoznate regularni (jaki) tipovi entiteta; proveru da li je neki tip entiteta slabi tip vršite kasnije, u narednim prolazima.

- Analizirajte attribute kod svih prepoznatnih entiteta. Ako neki od potencijalnih tipova entiteta nemaju attribute ili ih ima malo, razmotrite njihovu egzistenciju u obliku tipa entiteta. Možda se ti atributi mogu pridružiti nekom drugom tipu entiteta!
- Pronađite identifikatore za svaki jaki tip entiteta (kandidati za ključeve).
- Odredite domene atributa. Proverite da li se neki atributi mogu dekomponovati na prostije, naravno ako za to ima potrebe. Proverite da li neki atributi imaju više različitih vrednosti za konkretan entitet iz skupa. Ako ima više takvih atributa koji se odnose na isti objekat realnog sveta odnosno tip entiteta, umesto viševrednosnih atributa razmotrite uvođenje novog tipa entiteta.
- Atributi mogu pomoći i kod prepoznavanja veza. U slučaju da imate atribut koji se referencira na drugi tip entiteta koji postoji u vašem modelu, sigurno postoji veza između njih.
- U prvom prolazu definisanja veza pretpostavite da su sve veze parcijalne, a proveru da li su totalne vršite kasnije.
- Ne zaboravite da na kraju proverite ograničenja za sve prepoznate tipove veza.
- Proverite da li model sadrži redundantne komponente.
- Vodite računa o nivou detaljnosti i konzistentnosti; na primer, u početnim fazama možete izostaviti attribute iz modela i sl.
- Možete da razvijate dijagram iz više delova (celina koje prepoznate) i da ih kasnije spajate prepoznavanjem veza između delova.

### 3.7 EER Model

Enhanced Entity Relationship (EER) je prošireni ER model konceptima objektno-orijentisane (OO) paradigme. EER omogućava modeliranje kompleksnih veza između objekata u bazi podataka (multimedija, geo- i sl). Koristi dodatne koncepte koji mogu da modeliraju takve veze, kao što su koncepti potklase i natklase, specijalizacija i generalizacija i sl. EER modelu pripadaju i koncepti: IS-A hijerarhija, kategorizacija, agregacija pa čak i koncept slabog tipa entiteta, koji se još često zove i identifikaciono zavisni tip entiteta.

#### 3.7.1 Koncepti EER modela podataka

Prošireni ER model koristi koncept klase koji odgovara pojmu klase iz OO paradigme i posmatra se kao skup ili kolekcija entiteta. Praktično se tipovi entiteta mogu tretirati kao klase i obrnuto. Za klase se mogu definisati **natklase** i/ili **potklase**. Kod potklasa se podrazumeva nasleđivanje atributa i nasleđivanje veza iz

natklase. Veza između natklase i njenih potklasa se naziva **veza natklasa/potklasa** ili jednostavno **veza klasa/potklasa**.

Kod definisanja potklasa može se krenuti od postojećih klasa, kod kojih se traže moguće specijalizacije koje, pored zajedničkih osobina sadržanih u klasi, imaju svoje specifične osobine. Postupak definisanja skupa takvih potklasa je poznat kao **specijalizacija**. Obrnuti postupak, kada se od više postojećih klasa uočavanjem zajedničkih osobina definiše generalnija klasa (natklasa) je **generalizacija**.

Potklase mogu biti predikatom definisane ili korisnički definisane. Predikatom, odnosno uslovom, definisane potklase imaju to svojstvo da sve instance potklasa zadovoljavaju uslov zadat predikatom, odnosno uslovom. Korisnički definisane potklase su one kod kojih se takav uslov ne može definisati.

Za isti tip entiteta se može definisati više specijalizacija. Generalizacija i specijalizacija obezbeđuju formiranje hijerarhije klasa.

Za specijalizaciju i generalizaciju definišu se dve vrste ograničenja:

- Participacija potklasa u vezi:
  - disjunktna (eng. *disjoint*) – svaki entitet iz natklase pripada samo jednoj potklasi; i
  - preklapajuća (eng. *overlap*) – entitet iz natklase može da pripada većem broju potklasa.
- Kompletnost:
  - potpuna (eng. *total*) – svaki entitet iz natklase mora da pripada nekoj potklasi;
  - parcijalna (eng. *partial*) – svi entiteti iz natklase ne moraju da pripadaju nekoj od potklasa.

Sva navedena ograničenja su nezavisna i mogu se kombinovati, tako da se mogu prepoznati četiri tipa veze generalizacija/specijalizacija:

- *disjunktna, potpuna*
- *disjunktna, parcijalna*
- *preklapajuća, potpuna*
- *preklapajuća, parcijalna*

Navedena ograničenja definišu pravila kod dodavanja i brisanja entiteta. Kod brisanja entiteta iz superklase, brišu se i odgovarajući entiteti iz potklasa (oni koji zadovoljavaju zadati predikat, ako postoji). Kod dodavanja entiteta superklasi, može se dodati i entitet u potklasama, u zavisnosti od toga da li je kompletnost parcijalna ili totalna. Dodavanje entiteta superklase potklasi (za *disjoint*), odnosno potklasama (za *overlap*), za totalnu specijalizaciju je obavezno.

**Deljive potklase** su klase koje imaju više od jedne natklase. Deljive potklase obezbeđuju višestruko nasleđivanje kod EER modela.

Potreba za potklasama sa više od jedne superklase je dovela do uvođenja koncepta kategorija. Ako su natklase takve da se razlikuju, odnosno da su različitog tipa (obično nemaju iste ključeve), onda je takva potklasa **kategorija**.

Kod kategorija postoji selektivno nasleđivanje. Za razliku od deljivih klasa, koje nasleđuju dve ili više klasa istog tipa (obično sa istim ključevima), i kod kojih je nasleđivanje unija natklasa, kod kategorija se može reći da se radi o preseku natklasa.

Za kategorije važe ista ograničenja kao kod generalizacije/specijalizacije, tako da postoji **totalna** i **parcijalna kategorizacija**.


### 3.7.2 Notacija i primeri korišćenja koncepta EER modela

Za ove koncepte postoji odgovarajuća grafička notacija koja se koristi kao proširenje ER notacije.


EER dijagram koristi grafičku notaciju za ER dijagram i dodatke koje se odnose na proširene koncepte, koji se predstavljaju na sledeći način:

- Klasa se predstavlja na isti način kao i tip entiteta, pravouganikom sa upisanim imenom klase.
- Ograničenja:
  - Oznake za participaciju potklasa:



Disjoint

- oznaka: 

Overlap

- oznaka: 

Oznake za kompletnost:

- totalna: 
- parcijalna: 

Oznaka za kategorije:



Kao što je navedeno, klasa se odnosi se na pojam Tip entiteta iz ER modela podataka. **Specijalizacija** podrazumeva *top-down* pristup, odnosno podelu klase na **potklase**.

**Primer:** Specijalizacija

Specijalizacije klase RADNIK mogu da budu: SEKRETARICA, POSLOVODJA, INŽENJER, itd.

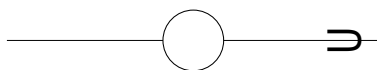
Posledica specijalizacije je **nasleđivanje** – potklasa nasleđuje atribute i veze iz natklase (napr. kod radnika: Ime, Datum rođenja,...). Potklase s druge strane mogu da imaju **jedinstvene** atribute.

**Primer:** Atributi specijalizacije

SEKRETARICA ima atribut BrzinaKucanja,

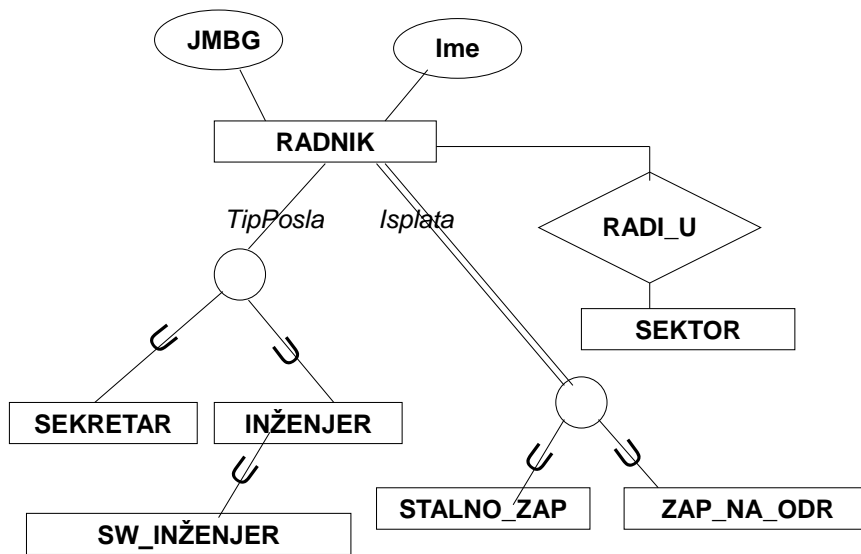
POSLOVODJA ima atribut StepenStručneSpreme, itd.

Kada postoji jedna ili više potklasa, definisanih na osnovu istog atributa (*TipPosla*), koristite sledeću grafičku notaciju za prikaz veze klasa/potklase:



Iako se po terminologiji ne vidi, kod veze klasa/potklasa se radi o vezi između tipova entiteta! Za prikaz veze (crtanje tipa poveznika) između dva različita tipa entiteta koristite uobičajenu notaciju ER dijagrama.

Na slici 3-29. prikazan je EER dijagram koji prikazuje dve specijalizacije klase RADNIK, po tipu posla i po načinu isplate. Prikazana je i klasična veza između tipova entiteta RADNIK i SEKTOR.



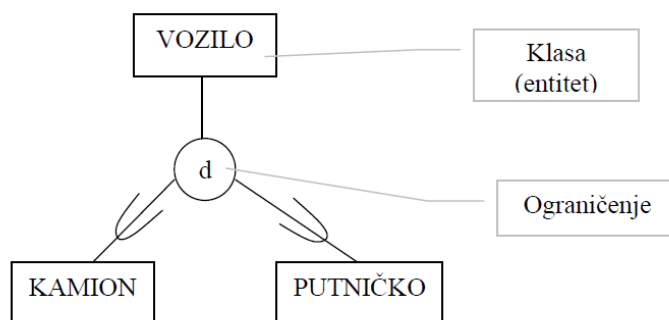
Slika 3-29. Primer specijalizacije



**Generalizacija** podrazumeva obrnuti postupak (bottom-up approach), definisanje **natklase** na osnovu zajedničkih osobina nekih klasa. Prvi korak kod generalizacije je pronalaženje zajedničkih atributa kod entiteta (odnosno klasa).

**Primer:**

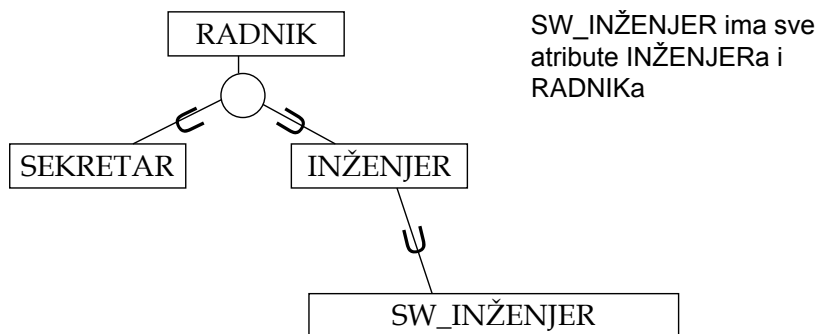
AUTOMOBIL (sa atributima boja, cena, maks.brzina) i KAMION (sa atributima boja, cena, nosivost) se mogu generalizovati u VOZILO (sa atributima boja i cena) (slika 3-30).



Slika 3-30. Primer generalizacije

Nasleđivanje, odnosno generalizacija/specijalizacija može da ide po dubini, tako da se definiše **hijerarhija klasa**. Hijerarhija podrazumeva da definisana potklasa učestvuje u nekoj klasa/potklasa relaciji.

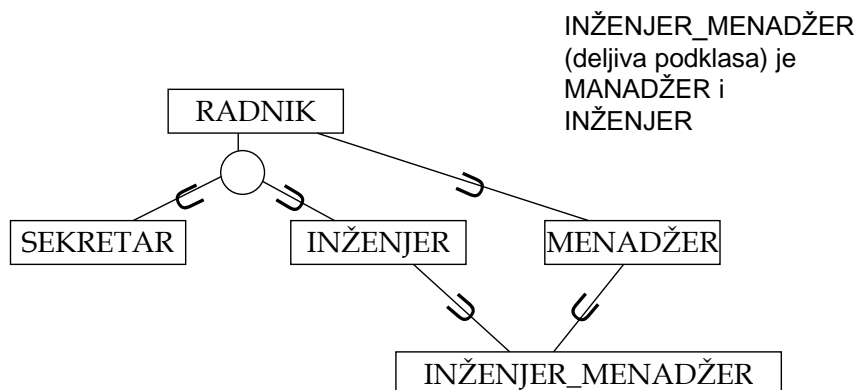
Na slici 3-31. je prikazana hijerarhija u kojoj je RADNIK natklasa za klase SEKRETAR i INŽENJER, a ova druga je natklasa za klasu SW\_INŽENJER. Atributi nisu prikazani na dijagramu. Zbog nasleđivanja, SW\_INŽENJER bi imao sve attribute svojih natklasa, INŽENJERa i RADNIKa.



Slika 3-31. Hijerarhija klasa

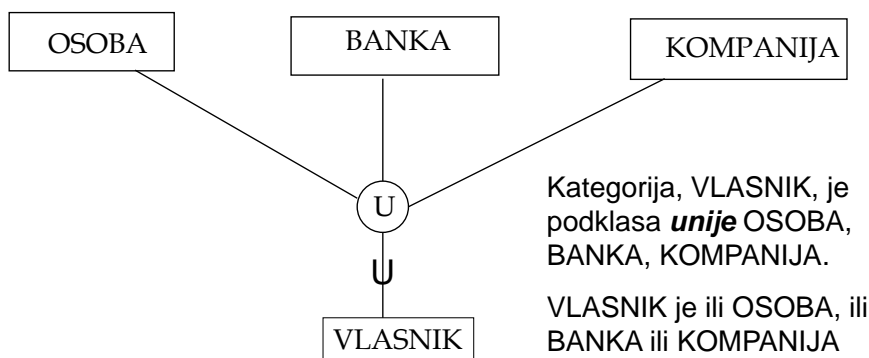
Kod EER modela mogu da se grade i **rešetke** (Lattices), koje podrazumevaju da potklasa može da učestvuje u više relacija klasa/potklasa. To su tzv. **deljive potklase**.

Na slici 3-32. prikazana je rešetka, u kojoj deljiva klasa INŽENJER\_MENADŽER nasleđuje klase INŽENJER i MENADŽER. Ona ima attribute obe svoje natklase.



Slika 3-32. Primer deljive klase

**Kategorije** modeliraju odnos klasa/potklasa sa više od jedne natklase ali **različitih** tipova entiteta. Nasleđivanje atributa je selektivno. Na slici 3-33. prikazana je kategorija VLASNIK, koja je potklasa unije tri klase – OSOBA, BANKA i KOMPANIJA. Praktično, VLASNIK može da bude ili OSOBA, ili BANKA ili KOMPANIJA, pa u skladu sa tim ima i odgovarajuće attribute.

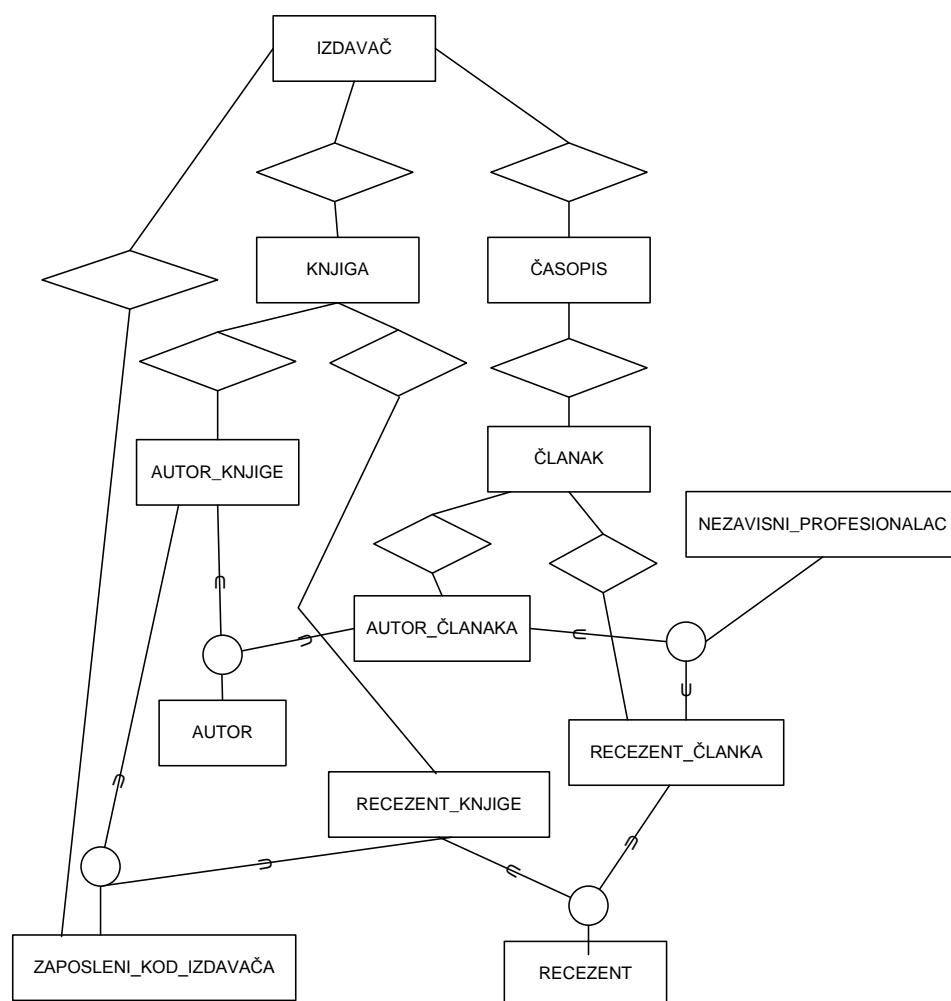


Slika 3-33. Primer kategorije

**Zadatak za vežbu: EER modeliranje**

Dat je EER dijagram na slici 3-34. Dodati neophodne detalje na osnovu sledećih zahteva:

Izdavači mogu da izdaju različite vrste knjiga i časopisa. Neki izdavači izdaju samo knjige, neki samo časopise, a neki i knjige i časopise. Ne postoje knjige i časopisi koje su izdanja više od jednog izdavača. Jedan autor može da piše i knjige i radove u časopisima. Časopis obično sadrži više radova, svaki od njih je napisao jedan ili više autora. Knjige takođe mogu da imaju više autora. Jedan rad se ne može pojaviti u više časopisa. Svaka knjiga i svaki rad se recenziraju od strane više profesionalaca iz naučne oblasti iz koje su autori. Naravno, autori ne mogu da recenziraju svoje radove i svoje knjige. Svaki stručnjak koji vrši recenziju knjige i autor knjige rade za nekog izdavača i plaćeni su od strane tog izdavača. Autori radova i recenzenti nisu plaćeni. Recenzenti radova ne mogu biti recenzenti knjiga. Autori i recenzenti koji nisu plaćeni od strane izdavača, su poznati kao nezavisni profesionalci.



Slika 3-34. EER dijagram za zadatak za vežbu

## 4 RELACIONI MODEL PODATAKA

Relacioni model je najpopularniji i najrasprostranjeniji model podataka danas i predstavlja osnovu za relacione baze podataka koje dominiraju na tržištu. Najveći broj aktuelnih sistema i tehnologija za rad sa bazama podataka baziran je na relacionom modelu koji ima teoretsku osnovu u teoriji skupova i predikatskoj logici prvog reda. Relacioni model je formulisao i predložio E.F. Codd 1970. godine. Ovaj model je jednostavan za razumevanje i zasniva se na korišćenju koncepta matematičke relacije kao osnovnog gradivnog elementa modela.

### 4.1 Osnove relacionog modela podataka

Relacioni model podataka obezbeđuje koncepte za specifikaciju strukture podataka i za manipulaciju podacima, kao i ograničenja kojima se obezbeđuje integritet podataka. Navedeni koncepti su međusobno povezani i oslanjaju se jedan na drugi. Inicijalna specifikacija relacionog modela koju je dao Codd, koja se odnosila na model podataka sa relacijama kao tipom strukture podataka i formalnim jezikom (relacionom algebrom) je tokom godina proširena, tako da danas ne postoji specifikacija jedinstvenog standardizovanog relacionog modela, već postoje klase modela koje se baziraju na relacijama koje sadrže određene specifičnosti u pogledu korišćenih skupova operatora za specifikaciju upita, ažuriranje podataka i specifikaciju ograničenja.

Snažna teorijska osnova relacionog modela obezbeđuje da se projektovanje i evaluacija relacionih baza podataka izvodi sistematskim metodama zasnovanim na apstrakcijama koje su jednostavne za razumevanje, redukuju kompleksnost projektovanja, odnosno omogućavaju projektantu da se koncentriše na sam problem. Relacioni model oslobađa korisnika frustracija oko rukovanja podacima na niskom nivou, zalaženja u detalje smeštanja podataka i metodama pristupa iz korisničkog interfejsa. Relacioni model obezbeđuje sredstva za opis podataka sa njihovom prirodnom strukturom, bez dodatnih struktura za potrebe mašinske reprezentacije. Ovaj model daje osnovu za jezik visokog nivoa za pristup podacima koji obezbeđuje maksimalnu nezavisnost između programa, s jedne strane, i mašinske reprezentacije, s druge strane.

Neformalno se može navesti da baza podataka predstavlja kolekciju više relacija, a svaka relacija je tabela sa vrstama i kolonama. Relacija, kao osnovni koncept relacionog modela je zapravo matematička relacija (što je objašnjeno u

narednom odeljku), i ima jednostavnu i za korisnika vrlo prihvatljivu reprezentaciju u obliku dvodimenzionalne tabele sa podacima, koja se na odgovarajući način može vizuelizovati.

Tabelarna reprezentacija relacije omogućava da se lako razume sadržaj baze podataka, i da se za manipulaciju podacima koristi jednostavan jezik visokog nivoa. Svaka vrsta tabele predstavlja kolekciju vrednosti podataka koje su u nekom odnosu. Intuitivno, vrsta u tabeli predstavlja određeni entitet (objekat, pojavu ili događaj) realnog sveta, dok vrednosti u pojedinim kolonama predstavljaju određena svojstva odgovarajućeg entiteta. Podaci u različitim tabelama mogu da budu u međusobnoj vezi, tako da kreiranje zbirne predstave o objektima realnog sveta koji su u određenom odnosu zahteva povezivanje podataka iz različitih tabela. Važna karakteristika relacionih baza podataka je da se podaci predstavljaju na uniforman način, preko vrednosti na odgovarajućim pozicijama u vrstama/kolonama tabela.

### 4.2 Koncepti relacionog modela podataka

Formalno, u terminologiji relacionog modela podataka, vrste relacije se nazivaju **torke**, kolone **atributi**, a tabela se naziva **relacija**. Vrednost koje mogu da se pojave u nekoj od kolona, odnosno vrednosti atributa, su određene **domenom**. Ova sekcija se bavi formalnim definicijama ovih koncepata.

Koncepti na nivou ekstenzije kod relacionog modela su:

- Domen atributa
- Torka
- Pojava relacije
- Pojava baze podataka

Koncepti na nivou intenzije relacionog modela su:

- Atribut
- Šema relacije
- Šema baze podataka

Svi ostali koncepti se izvode iz osnovnih primenom određenih formalnih, matematičkih pravila, koja nisu deo ovog materijala i osnovnog kursa iz baza podataka.

### 4.3 Domeni, atributi, torke i relacije

U relacionom modelu podataka polazna osnova za definisanje koncepata strukturalne komponente je **univerzalni skup atributa**. Obeležava se sa  $U$ , pri čemu je  $U = \{A_i | i=1, n\}$ . To je skup atributa realnog sistema ili nekog njegovog dela

koji se posmatra (mini sveta). Svaki atribut  $A_i$  opisuje neku odabranu osobinu klase entiteta ili klase poveznika. Svakom atributu je pridružen **domen**, u oznaci  $\text{dom}(A_i)$ , koji predstavlja specifikaciju skupa mogućih vrednosti iz kojeg atribut uzima vrednosti.

U relacionom modelu podataka domen  $D$  je skup atomičnih vrednosti. U ovom slučaju atomičnost znači da su svi elementi domena nedeljivi. Uobičajeni način specifikacije domena je preko tipa podataka. Korisno je da svaki domen ima naziv. Domen se može definisati i kao podopseg vrednosti nekog drugog domena ili nabranjem dopuštenih domenskih vrednosti. U relacionom modelu podataka svaki atribut ima pridružen domen, pri čemu više atributa može imati isti domen. Na primer, atributi koji se odnose na ime studenta i ime nastavnika mogu da imaju isti domen koji definiše da se radi o podacima znakovnog tipa dužine 20.

Koncept domena u relacionom modelu je vrlo važan, zato što omogućava korisniku da definiše na jednom centralnom mestu značenje i izbor vrednosti koje atribut može uzimati. Domen atributa se definiše tipom podataka, dužinom podataka i opsegom vrednosti. Atributi uzimaju vrednosti iz odgovarajućeg domena koji im je dodeljen, što u praksi znači da će vrednosti u tabeli za neku kolonu da budu onog tipa podataka koji smo izabrali za tu kolonu. Međutim, može da se desi da atribut nema dodeljenu vrednost što podrazumeva korišćenje tzv Null vrednosti sa značenjem kako je to objašnjeno u odeljku 3.1.1. Specijalna vrednost Null može biti član svakog domena.

#### Primer: Definicija domena

- Logička definicija domena:
  - Lokalni\_telefonski\_broj: skup 6-cifrenih telefonskih brojeva koji su validni na području lokalne mrežne grupe
  - Ocena\_studenta: celi broj između 6 i 10
  - Kod\_obrazovnog\_profila: skup oznaka profila Elektronskog fakulteta: E,EEN,EMK,MIM,RI,T,US
- Tip podataka ili format domena:
  - Lokalni\_telefonski\_broj: CHARACTER string oblika *ccc-ccc*
  - Prelazna\_ocena\_studenta: INTEGER broj iz opsega 6-10
  - Kod\_obrazovnog\_profila: CHARACTER string iz skupa {E,EEN,EMK,MIM,RI,T,US}

Definicija relacije u relacionom modelu se oslanja na definiciju matematičke relacije, kao podskupa Dekartovog proizvoda skupa domena.

#### Definicija relacije:

Neka je  $A=\{A_1, A_2, \dots, A_n\}$  skup atributa realnog sistema, i neka je svakom atributu  $A_i$  pridružen domen  $D_i$ . Domeni ne moraju biti različiti.

**Relacija  $r$**  na skupovima  $D_1, D_2, \dots, D_n$  je konačan skup **n-torki (torki)** od kojih svaka sadrži prvi element iz  $D_1$ , drugi iz  $D_2, \dots$ , n-ti iz  $D_n$ .

Relacija  $r$  je na osnovu ove definicije podskup Dekartovog proizvoda skupa domena:  $r \subseteq D_1 \times D_2 \times \dots \times D_n$ , gde je  $D_1 \times D_2 \times \dots \times D_n = \{(v_1, v_2, \dots, v_n) \mid v_i \in D_i, i = 1, 2, \dots, n\}$ . Relacija  $r$  u relacionom modelu se sastoji iz zaglavlja i tela relacije. Zaglavlje relacije je  $\{A_1, A_2, \dots, A_n\}$ , čiji elementi  $A_i$  predstavljaju attribute. Svaki atribut je određen imenom i odgovarajućim domenom  $D_i$ . Telo relacije čine n-torke  $(v_1, v_2, \dots, v_n)$ , pri čemu svaka n-torka ima istu kardinalnost kao i zaglavlje, a svaki element  $v_i$  neke n-torke je član domena odgovarajućeg atributa ( $v_i \in \text{dom}(A_i)$ , odnosno  $v_i \in D_i$ ). **Stepen** relacije  $r$  jednak je broju atributa zaglavlja.

**Šema relacije** predstavlja opis relacije i definisana je na osnovu skupa atributa. Zaglavlje relacije opisano u prethodnom pasusu praktično predstavlja šemu relacije.

**Definicija šeme relacije:**

Šema relacije je imenovana dvojka, u oznaci  $\mathbf{N(R, C)}$ , gde je:

- $N$  naziv šeme relacije,
- $R \subseteq U$  skup atributa relacije, a
- $C$  skup ograničenja integriteta relacije

U ovoj definiciji  $N$  je neformalna komponenta koja opisuje semantiku relacije u prirodnom jeziku. Skup ograničenja integriteta  $C$  opisuje odnose između elemenata domena atributa iz  $R$ . Praktično, ova komponenta ograničava koje će se torke pojaviti u instanci ove relacije.

Uobičajeno je da se šema relacije obeležava tako što se navede njen naziv, a nakon toga skup atributa (s tim da se uz svaki atribut može navesti domen i eventualno ograničenje ključa), i eventualno skup ograničenja. Na primer, činjenicu da je  $R$  šema relacije definisana nad atributima  $A_1, A_2, \dots, A_n$ , označavamo sa:

$$R=(A_1, A_2, \dots, A_n),$$

ili alternativno

$$R(A_1, A_2, \dots, A_n).$$

**Primer:** Označavanje relacije

Relacija sa imenom  $R$  i listom atributa  $A_1, A_2, \dots, A_n$ , primarnim ključem  $A_1$ :

$$R(\underline{A_1}, A_2, \dots, A_n)$$

Relacija sa imenom  $R$  i listom atributa  $A_1, A_2, \dots, A_n$ , listom domena atributa  $D_1, D_2, \dots, D_n$  i ključem  $A_1$



$R(\underline{A_1}:D_1, A_2:D_2, \dots, A_n:D_n)$

U oba primera  $\underline{A_1}$  se odnosi na ograničenje tipa “primarni ključ relacije”.

**Primer:** Označavanje relacije uz definisana ograničenja

Posmatraćemo tip entiteta STUDENT sa atributima {IND,IME,PREZIME,BPI}

Pri čemu je u toku projektovanja uočeno da važe ograničenja:

- ( $\gamma_1$ ) svaki student ima broj indeksa i ne postoje dva studenta sa istim brojem indeksa
- ( $\gamma_2$ ) broj položenih ispita (BPI) je veći od nule i manji od 50

Šema relacije koja predstavlja model ove klase entiteta bi imala oblik:

STUDENT({IND,IME,PREZIME,BPI}, { $\gamma_1, \gamma_2$ })

**Primer:** Šema relacije koja se odnosi na radnike preduzeća

Primer dve šeme relacije koje se odnose na radnike, u drugom slučaju su navedeni domen atributa:

RADNIK1 (MBR, LIME, LD, SBR)

RADNIK2 (MBR:integer, LIME:char(15),LD:real, SBR:integer)

**Pojava (instanca) relacije** se odnosi na skup n-torki relacije u nekom trenutku. Ako se napravi analogija relacione šeme sa konceptom tip podataka, tada pojam relacije odgovara konceptu promenljive u programskom jeziku. Kao što promenljiva može da ima različite vrednosti u različitim trenucima, tako i relacija može da sadrži različite n-torke u različitim trenucima.

**Definicija pojave relacije:**

Pojava relacije **r** nad šemom relacije (**R,C**), u oznaci **r(R)**, je skup n-torki **t** = {**t**<sub>1</sub>,**t**<sub>2</sub>,...,**t**<sub>k</sub>} koje zadovoljavaju ograničenja C.

U ovoj definiciji svaka n-torka predstavlja uređenu listu od n-vrednosti, **t**=<**v**<sub>1</sub>,**v**<sub>2</sub>,...**v**<sub>n</sub>>, gde je svaka vrednost iz odgovarajućeg domena, **v**<sub>i</sub>∈**D**<sub>i</sub> ili **v**<sub>i</sub>=null. *i*-ta vrednost torke **t**, koja odgovara atributu **A**<sub>i</sub>, se obeležava sa **t**[**A**<sub>i</sub>]

**Primer:**

Za šemu relacije RADNIK(MBR,LIME,LD,SBR) jedna pojava relacije **radnik** je:

<2203,MIRA,50000,10>

<2817,PERA,40000,20>

<2932,MIRA,35000,10>

<2995,GOCA,18000,30>

Relacija **r** nad šemom relacije **R(A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>)** se predstavlja **tabelom** čije kolone odgovaraju atributima, a vrste pojedinim n-torkama vrednosti ovih atributa. Sve vrste moraju biti različite.

**Primer:** Predstavljanje relacije tabelom

Relacija **radnik** definisana nad šemom relacije **RADNIK(MBR, LIME, LD,SBR)** se predstavlja tabelom kao na slici 4-1.

**radnik**

<u>MBR</u>	LIME	LD	SBR
2203	MIRA	50000	10
2817	PERA	40000	20
2932	MIRA	35000	10
2995	GOCA	18000	30

**Slika 4-1. Tabela sa podacima koja odgovara relaciji *radnik***

#### 4.3.1 Svojstva šeme relacije i relacije

Važno svojstvo šeme relacije je da ne sadrži dva jednaka naziva atributa kao i to da redosled naziva atributa u relaciji nije bitan. Ova dva svojstva proizilaze iz definicije šeme relacije: navedeno je da je šema relacije *skup* naziva atributa, a po definiciji skup ne može sadržati dva međusobno jednaka elementa. Posledica promene redosleda atributa i torki u relaciji neće biti gubitak informacije sadržane u relaciji.

Posledica definicije relacije je svojstvo da relacija ne sadrži dve jednake torke, kao i da redosled torki u relaciji nije bitan. Ova dva svojstva proizilaze iz definicije relacije koja je definisana kao *skup torki*. Pošto su relacije skupovi n-torki, a između elemenata skupa ne postoji uređenost, redosled n-torki u relaciji nije definisan niti je bitan. Ipak, na nivou implementacije ili vizuelizacije sadržaja relacije postoji uređenost n-torki zato što su one smeštene u određenom fizičkom redosledu. Iako taj redosled nije od značaja, korisniku se na logičkom nivou može prikazati različit redosled n-torki, na primer po rastućim vrednostima nekog atributa.

#### 4.4 Ključ šeme relacije

U prethodnom poglavlju je navedeno da kod ER modela za svaki tip entiteta postoji atribut ili skup atributa koji na jedinstven način identifikuje svaki pojedinačni entitet u skupu entiteta. U relacionom modelu podataka sve torke su različite, odnosno ne postoje dve torke koje imaju iste vrednosti za sve attribute relacije. Najčešće se torke razlikuju samo na određenom podskupu atributa. Za specifikaciju ovih ograničenja koristi se, kao kod ER modela, koncept ključa šeme relacije.

**Ključ šeme relacije** je atribut ili skup atributa (tzv kompozitni ključ odnosno ključ od više atributa), čije vrednosti predstavljaju identifikator torke u relaciji, što odgovara identifikatoru pojave tipa entiteta. Svaka šema relacije ima bar jedan ključ koji na jedinstven način identifikuje svaku torku u relaciji. Atributi koji su deo ključa relacije se nazivaju **ključni** odnosno **primarni atributi**.

U relacionom modelu podataka postoji više termina koji se koriste za relacione ključeve, što će nadalje biti uvedeno.

### Definicija ključa

Skup atributa  $X \subseteq R$  predstavlja ključ šeme relacije  $(R, C)$ , ako za svako  $r$  važe sledeća dva uslova:

- 1) U1:  $(\forall u, v \in r) (u[X]=v[X] \Rightarrow u=v)$
- 2) U2:  $(\forall Y \subset X) (\neg U1)$

Uslov U1 definiše *jedinstvenost* vrednosti ključa, dok uslov U2 definiše *minimalnost* skupa atributa ključa.

Na osnovu ove definicije važno je naglasiti da je ključ *minimalni skup atributa* koji na jedinstven način identifikuje svaku svaku torku u relaciji, što znači da uklanjanjem bilo koje komponente ključa, ključ gubi svojstvo identifikacije.

Jedna šema relacije može imati više ključeva, i to su ekvivalentni ključevi ili **ključevi kandidati**.

Jedan od ekvivalentnih ključeva se bira za **primarni ključ** relacije. Za primarni ključ moraju da važe uslovi jedinstvenosti i minimalnosti, odnosno uslovi U1 i U2.

Atribut ili skup atributa koji na jedinstven način identifikuje torku u relaciji se zove **superključ**. Svaka relacija ima najmanje jedan superključ, a to je skup svih atributa te relacije. Kod superključa važi samo uslov jedinstvenosti (uslov U1), odnosno ne važi uslov minimalnosti U2.

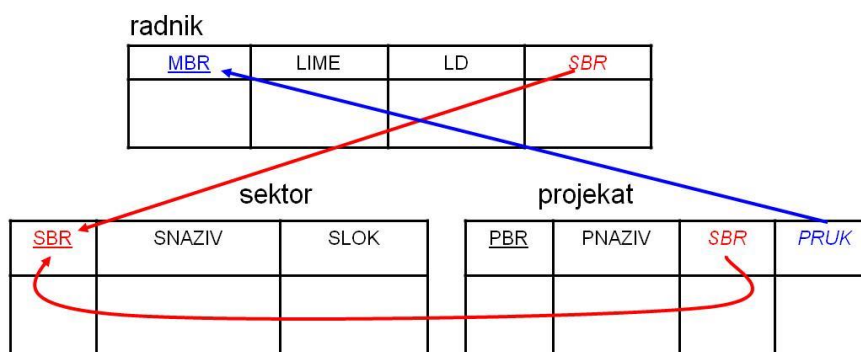
Superključ čiji nijedan pravi podskup nema svojstvo super ključa u toj relaciji je **kandidat za ključ**. Kod kandidata za ključeve važe oba uslova, U1 i U2.

Primarni ključ praktično je jedan od kandidata za ključ koji je izabran od strane projektanta baze podataka da na jedinstven način identifikuje torke relacije. Preko ovog ključa se najčešće vrši traženje u relaciji. On se koristi kao **strani ključ** u drugim šemama relacija, odnosno preko njega se ostvaruju međusobne veze između relacija/tabela.

Pojam **strani ključ** se odnosi na atribut ili skup atributa jedne relacije koji se uparuje sa ključem kandidatom druge ili iste relacije. Skup atributa *SK* relacije *r* je strani ključ relacije ako zadovoljava sledeća dva uslova:

- Ako se relacija  $r$  referencira na relaciju  $q$ , tada atributi iz  $SK$  relacije  $r$  moraju imati isti domen kao atributi primarnog ključa  $PK$  relacije  $q$ .
- Ako se torka  $t_1$  relacije  $r$  referencira na torku  $t_2$  relacije  $q$ , tada vrednost stranog ključa  $SK$  u torki  $t_1$  relacije  $r$  mora biti jednaka vrednosti primarnog ključa  $PK$  u torki  $t_2$  relacije  $q$  ili može imati nedefinisanu (Null) vrednost.

Slika 4-2. prikazuje međusobnu povezanost relacija *radnik*, *sektor* i *projekat* preko stranih ključeva. Relacija *radnik* ima strani ključ SBR, koji je primarni ključ relacije *sektor* zbog veze koja postoji između radnika i sektora u kome radi. Relacija *projekat* takođe ima kao strani ključ SBR zbog veze koja postoji između projekta i sektora koji ga izводи. Projekat sadrži i strani ključ PRUK koji je praktično MBR radnika zbog veze radnika kao rukovodioca projekta.



Slika 4-2. Ostvarivanje veza preko stranih ključeva

#### 4.4.1 Pregled terminologije za ključeve

U ovom odeljku dat je kratak pregled terminologije koja se koristi kod ključeva. Nakon samog termina sledi kratak neformalni opis.

##### Ključ

Pošto su sve torke relacije različite, u relaciji mora postojati atribut ili skup atributa (tzv kompozitni ključ – ključ od više atributa), nazvani relacioni ključevi ili ključevi relacije, koji na jedinstven način identifikuje svaku torku relacije.

##### Superključ

Atribut ili skup atributa koji na jedinstven način identifikuju torku unutar relacije (nema uslova minimalnosti).

##### Ključ kandidat

Superključ čiji nijedan pravi podskup nije superključ relacije. Relacija može imati više ključeva kandidata, kod implementacije se bira jedan od njih kao primarni ključ.

**Primarni ključ**

Ključ kandidat koji je odabran da na jedinstven način identifikuje torke unutar relacije.

**Strani ključ**

Atribut ili skup atributa jedne relacije koji se uparuje sa ključem kandidatom neke druge ili iste relacije. Važan za ostvarivanje međusobnih veza između tabela.

**4.5 Šema i pojava relacione baze podataka**

**Šema relacione baze podataka** predstavlja konačan skup šema relacija baze podataka  $R_i$  ( $i=1, 2, \dots, m$ ) i skup međurelacionih ograničenja IC. Označava se sa  $S(R_1, R_2, \dots, R_m; IC)$ .

**Pojava** (instanca) **baze podataka** nad šemom  $S(R_1, R_2, \dots, R_m; IC)$ , je skup pojava  $r_i$  šema relacija  $R_i$  ( $i=1, 2, \dots, m$ ), pri čemu skup relacija  $r_i$  ( $i=1, 2, \dots, m$ ) zadovoljava ograničenja integriteta IC. Označava se sa  $s$ .

**Primer:** Šema relacione baze podataka FAKULTET

U ovoj šemi je definisano: ime šeme baze podataka (FAKULTET), imena šema relacija (STUDENT, PROFESOR, PREDMET, ZAPISNIK, IZBOR i NASTAVA) i imena atributa (Ind, Ime, Adresa,...). Ograničenja integriteta IC nisu definisana.

FAKULTET({STUDENT, PROFESOR, PREDMET, ZAPISNIK, NASTAVA}; IC)

STUDENT(Ind, Ime, Adresa, Status)

PROFESOR(Id, Ime, KatId)

PREDMET(KatId, PrKod, PrNaziv, Opis)

ZAPISNIK(StudId, PrKod, Rok, Ocena)

IZBOR(StudId, PrKod, Semestar)

NASTAVA(ProfId, PrKod, Semestar)

**Primer:** Šema relacione baze podataka PREDUZEĆE

U ovom primeru Svaka šema relacije sadrži specifikaciju primarnog ključa (podvučeni atribut u šemi relacije) i specifikaciju stranih ključeva (*italic* u šemi relacije)

PREDUZEĆE ({RADNIK, SEKTOR, PROJEKAT}; IC)

RADNIK(MBR, LIME, LD, *SBR*)

SEKTOR(SBR, SNAZIV, *SLOK*)

PROJEKAT(PBR, PNAZIV, *SBR*, *PRUK*)

**Primer:** Jedna pojava baze podataka PREDUZEĆE

Na slici 4-3. prikazana je jedna pojava baze podataka preduzeće čija je šema data u prethodnom primeru. Označeni su primarni ključevi (podvučeni nazivi atributa) i strani ključevi (imena označena *italikom*)

radnik

<u>MBR</u>	LIME	LD	<i>SBR</i>
2203	MIRA	50 000	10
2817	PERA	40 000	20
2932	MIRA	35 000	10
2995	GOCA	18 000	30
3305	LAZA	60 000	40
3515	JOVAN	20 000	20
3819	VLADA	65 000	30

sektor

<u>SBR</u>	SNAZIV	SLOK
10	PROIZVODNJA	NIŠ
20	PROIZVODNJA	BEOGRAD
30	INŽINJERING	NIŠ
40	RAZVOJ	NIŠ

projekat

<u>PBR</u>	PNAZIV	<i>SBR</i>	<i>PRUK</i>
100	PC	10	2203
200	HOST	20	3305
300	LAN	30	3819
400	VIPX	40	2817

**Slika 4-3.** Jedna pojava baze podataka PREDUZEĆE

**Primer:** Šema baze podataka FAKULTET sa definisanim ograničenjima

U ovoj šemi je definisano ograničenje domena i ograničenje ključa (primarni ključ). Data su i ograničenja stranog ključa.

FAKULTET(STUDENT, PROFESOR, PREDMET, ZAPISNIK, NASTAVA, IC)

STUDENT(Ind:INTEGER, Ime:STRING, Adresa:STRING, Status:STRING)

PROFESOR(Id:INTEGER, Ime:STRING, KatId:STRING)

PREDMET(KatId:STRING, PrKod:STRING, PrNaziv:STRING, Opis:STRING)

ZAPISNIK(StudId:INTEGER, PrKod:STRING, Rok:STRING, Ocena:INTEGER)

IZBOR(StudId:INTEGER, PrKod:STRING, Semestar:STRING)

NASTAVA(ProfId:INTEGER, PrKod:STRING, Semestar:STRING)

ZAPISNIK(StudId) referencira STUDENT(Ind)  
 ZAPISNIK(PrKod) referencira PREDMET(PrKod)  
 IZBOR(StudId) referencira STUDENT(Ind)  
 IZBOR(PrKod) referencira PREDMET(PrKod)  
 NASTAVA(ProfId) referencira PROFESOR(Id)  
 NASTAVA(PrKod) referencira PREDMET(PrKod)

#### 4.5.1 Predstavljanje šema relacione baze podataka

Uobičajena konvencija za predstavljanje relacione šeme obuhvata ime relacije iza koga slede imena atributa u malim zagradama. Imena atributa se odvajaju zapetama. Primarni ključ se podvlači, a spoljni ključevi se pišu *italikom*. Niže je prikazana relaciona šema baze podataka PREDUZEĆE:

RADNIK (LIME, SSLOVO, PREZIME, MATBR, DATRODJ, POL, PLATA, ADRESA, *MATBROJS*, *BRSEK*)  
 PROJEKAT (NAZIV, LOKPR, BROJPR, *BRS*)  
 SEKTOR (NAZIV, SBROJ, *MATBRR*, *DATPOST*)  
 CLAN-PORODICE (*MATBRRAD*, IME, POL, SRODSTVO, DATRODJ)  
 LOK-SEK (*BRŠ*, LOKACIJA)  
 RADI-NA (*MBR*, *BRPR*, SATI)

Radi poboljšanja čitljivosti relacione šeme, obično se za imena atributa ili entiteta koriste mala slova, tako da šema dobija jedan od sledećih formata:

RADNIK (Lime, Sslovo, Prezime, MatBr, DatRodj, Pol, Plata, Adresa, *MatBrojS*, *BrSek*)  
 PROJEKAT (Naziv, LokPr, BrojPr, *BrS*)  
 SEKTOR (Naziv, SBroj, *MatBrR*, *DatPost*)  
 CLAN-PORODICE (*MatBrRad*, Ime, Pol, Srodstvo, DatRodj)  
 LOK-SEK (*BrŠ*, Lokacija)  
 RADI-NA (*Mbr*, *BrPr*, Sati)

*radnik* (LIME, SSLOVO, PREZIME, MATBR, DATRODJ, POL, PLATA, ADRESA, *MATBROJS*, *BRSEK*)  
*projekat* (NAZIV, LOKPR, BROJPR, *BRS*)  
 sektor (NAZIV, SBROJ, *MATBRR*, *DATPOST*)

*clan\_porodice* (MATBRRAD, IME, POL, SRODSTVO, DATRODJ)

*lok\_sek* (BRS, LOKACIJA)

*radi\_na* (MBR, BRPR, SATI)

Uobičajena konvencija za predstavljanje relacione šeme je dijagram šeme relacije. Svaka šema relacije se predstavlja jednim izduženim pravougaonikom koji ima onoliko ćelija koliko je atributa u šemi relacije. Ime šeme relacije se ispisuje iznad pravougaonika, a imena atributa u ćelijama, pri čemu ostaje pravilo da se primarni ključ podvlači, a da se spoljni ključevi pišu italikom. Na slici 4-4. je prikazan dijagram relacione šeme baze podataka PREDUZEĆE:

RADNIK

LIME	SSLOVO	PREZIME	MATBR	DATRODJ	POL	...
------	--------	---------	-------	---------	-----	-----

...	PLATA	ADRESA	MATBROJS	BRSEK
-----	-------	--------	----------	-------

PROJEKAT

NAZIV	LOKPR	BROJPR	BRS
-------	-------	--------	-----

SEKTOR

NAZIV	SBROJ	MATBRR	DATPOST
-------	-------	--------	---------

CLAN\_PORODICE

MATBRRAD	IME	POL	SRODSTVO	DATRODJ
----------	-----	-----	----------	---------

LOK\_SEKTOR

BRS	LOKACIJA
-----	----------

RADI\_NA

MBR	BRPR	SATI
-----	------	------

Slika 4-4. Relaciona šema baze podataka PREDUZEĆE

## 4.6 Ograničenja u relacionom modelu

Ograničenja predstavljaju neophodni sastavni deo definicije šeme baze podataka. Koriste se pri formiranju i ažuriranju baze podataka u cilju održavanja sadržaja baze podataka u saglasnosti sa uočenim odnosima između atributa realnog sistema. Neka od ograničenja iz integritetne komponente smo već usvojili u prethodnim odeljcima – na primer, naveli smo da se za svaki atribut u relaciji vezuje određen domen. Ustvari, radi se o **domenskim ograničenjima**, kojima se ograničava skup dozvoljenih vrednosti atributa relacije. Definisan je i pojam **ključa šeme relacije**, koji obezbeđuje da su sve torke u relaciji različite.

Ograničenja se mogu klasifikovati kao:

- **Ograničenja torki** – proveravaju se za svaku torku relacije
- **Relaciona ograničenja** – proveravaju se međusobni odnosi više torki jedne relacije. Ovim se reguliše lokalna konzistentnost – usaglašenost pojave relacije sa definicijom šeme relacije.



- **Međurelaciona ograničenja** – Reguliše se globalna konzistentnost baze podataka, odnosno usaglašenost pojave baze podataka sa definicijom šeme baze podataka.

**Primer:** Ograničenje na nivou torki

Ograničenje  $\gamma_2$  koje je zadato u nekom od prethodnih primera: broj položenih ispita (BPI) je veći od nule i manji od 50:

$$\gamma_2(r) = T \equiv (\forall u \in r)(0 \leq u[\text{BPI}] < 50)$$

**Primer:** Ograničenje na nivou relacije

Ograničenje  $\gamma_1$  koje je zadato u nekom od prethodnih primera predstavlja ograničenje na nivou relacije:

$$\gamma_1(r) = T \equiv (\forall u, v \in r)(u[\text{IND}] = v[\text{IND}] \Rightarrow u = v)$$

U relacionom modelu podataka postoje i druga pravila integriteta, koja ograničavaju ili zabranjuju pojave određenih torki u relaciji. Definišu se ograničenja u odnosu na ključeve, tipove entiteta, međusobno referenciranje i semantiku relacija. Ova ograničenja se nazivaju, redom:

- integritet ključeva
- integritet entiteta
- referencijalni integritet
- semantički integritet

**Integritet ključeva** podrazumeva da vrednosti ključeva kandidata moraju biti jedinstvene u pojavi šeme relacije. Ovim ograničenjem se definiše ograničenje jedinstvenosti. Razlog uvođenja se odnosi na činjenicu da je jedinstvenost vrednosti ključeva kandidata posledica činjenice da relacija predstavlja skup torki, te stoga u relaciji ne mogu postojati dve iste torke.

**Integritet entiteta** podrazumeva da niti vrednost primarnog ključa, niti bilo koje njegove komponente, ne sme imati nepoznatu (NULL) vrednost u pojavi relacije. Pošto primarni ključ služi za identifikaciju pojedinih torki relacije, to nepoznata vrednost primarnog ključa bi onemogućila da se neke torke relacije identifikuju.

**Referencijalni integritet** se odnosi na zahtev da referencirana torka mora postojati. Važan tip referencijalnog integriteta je **ograničenje stranog ključa**. Ovo ograničenje definiše da, ako u relaciji postoji strani ključ, tada vrednost stranog

ključa mora biti jednaka vrednosti ključa kandidata neke torke referencirane relacije ili imati NULL vrednost. To je ograničenje koje se definiše između dve relacije u cilju održavanja konzistentnosti među torkama ovih relacija. Referencijalni integritet obezbeđuje da se svaka torka iz jedne relacije referencira samo na postojeću torku u drugoj ili istoj relaciji

Ograničenja **semantičkog integriteta** se odnosi na opšta, aplikativno bazirana ograničenja. To su pravila koja specificira korisnik, a koja definišu ili ograničavaju neke aspekte realnog sistema. Obično se odnose na neka pravila koja ne mogu da se pokriju ograničenjima na nivou šeme baze podataka, koja su specifična za svaki aplikativni system i koja moraju biti nametnuta i osigurana odgovarajućim mehanizmima. Na primer, ograničenje tipa “Broj zaposlenih u svakom sektoru preduzeća ne može biti veći od 20” ili “Plata rukovodioca mora biti veća od plata njegovih radnika”, se ne može specificirati korišćenjem prethodno opisanih deklarativnih ograničenja na nivou šeme baze podataka. Umesto toga mogu se specificirati procedure sa ciljem da se obezbedi implementacija željenih ograničenja zahtevanih poslovnim pravilima realnog sistema.

### 4.6.1 Specifikacija ograničenja u DBMSu

U relacijama baze podataka i između relacija baze podataka postoji veliki broj ograničenja integriteta koja treba eksplicitno uneti u šemu baze podataka. U šemi relacije se eksplicitno specificiraju sledeća ograničenja:

- Domeni atributa.
- Primarni ključ.
- Strani ključevi i referenca na primarni ključ.
- Da li atribut može imati NULL vrednost.
- Da li atribut mora imati jedinstvenu vrednost.
- Da li atribut ima podrazumevanu vrednost.
- Ostala semantička ograničenja.

Ograničenja integriteta se specificiraju u šemi baze podataka i eksplicitno specificiraju korišćenjem jezika DDL (Data Definition Language, deo SQLa koji je opisan u jednom od narednih poglavlja), tako da ih DBMS može automatski nadzirati. Semantička ograničenja se mogu specificirati i kontrolisati unutar aplikacionih programa za ažuriranje baze podataka ili se mogu specificirati u šemi baze podataka korišćenjem jezika za specificiranje ograničenja.

Većina komercijalnih DBMS-a podržava integritet ključa i entiteta, dok manji broj podržava referencijalni i semantički integritet.

Detalje o jeziku SQL možete pročitati u narednim poglavljima, a u ovoj sekciji je dat pregled dela SQL naredbi koje se koriste za definisanje podataka (iz DDL podskupa naredbi) i njihovo korišćenje za definisanje ograničenja.

Za ilustraciju definisanja navedenih ograničenja, koristiće se date šeme relacija STUDENT i PREDMET:

STUDENT (Ind:INTEGER, Ime:STRING, Adresa:STRING, Status:STRING),  
**Ključ: {Ind}**

PREDMET(KatId:STRING, PrKod:STRING, PrNaziv:STRING, Opis:STRING),  
**Ključevi: {PrKod}, {KatId, PrNaziv}**

U nastavku je najpre prikazana naredba za specificiranje tipa relacije, odnosno kreiranje tabele u bazi podataka, a nakon toga su date odgovarajuće SQL naredbe za definisanje ograničenja.

#### **Specificiranje tipa relacije**

SQL naredba za kreiranje odgovarajuće tabele za šemu relacije STUDENT je:

```
CREATE TABLE STUDENT (
    Ind      INTEGER,
    Ime      CHAR(20),
    Adresa   CHAR(50),
    Status   CHAR(10) )
```

Nakon izvršenja ove naredbe, kreirana je tabela STUDENT sa kolonama Ind, Ime, Adresa i Status, svaka od njih definisana navedenim tipom podatka.

#### **Specificiranje ograničenja ključa**

Ključ relacije STUDENT je atribut Ind, što se u naredbi za kreiranje tabele može specificirati na dva načina.

Prvi način je navođenjem ograničenja PRIMARY KEY:

```
CREATE TABLE STUDENT (
    Ind      INTEGER,
    Ime      CHAR(20),
    Adresa   CHAR(50),
    Status   CHAR(10),
    PRIMARY KEY (Ind ) )
```

Ovaj način je primenjen i u narednoj naredbi, s tim da je kod tabele PREDMET definisano ograničenje UNIQUE za ključ kandidat koji se sastoji od dva atributa KatID i PrNaziv:

```
CREATE TABLE PREDMET (
    KatId    CHAR(6),
    PrKod    CHAR(4),
```

```
PrNaziv CHAR(20),  
Opis CHAR(100),  
PRIMARY KEY (PrKod),  
UNIQUE (KatId,PrNaziv )
```

Generalno, UNIQUE je mehanizam DBMS-a koji služi za (a) deklarisanje ograničenja ključa kandidata (kada su nula vrednosti zabranjene) i (b) deklarisanje ograničenja jedinstvenosti (kada se ne zahteva zabrana nula vrednosti za sva obeležja u UNIQUE).

#### **Specificiranje NULL i podrazumevanih vrednosti**

Za tabelu STUDENT, za atribut Status definisana je podrazumevana vrednost "Brucoš" a ime ne sme da ima NULL vrednosti:

```
CREATE TABLE STUDENT (  
Ind INTEGER,  
Ime CHAR(20) NOT NULL,  
Adresa CHAR(50),  
Status CHAR(10) DEFAULT 'Brucoš',  
PRIMARY KEY (Ind ) )
```

Drugi način zadavanja ograničenja primarnog ključa je navođenjem ključne reči PRIMARY KEY odmah nakon specifikacije odgovarajućeg atributa:

```
CREATE TABLE STUDENT (  
Ind INTEGER PRIMARY KEY,  
Ime CHAR(20) NOT NULL,  
Adresa CHAR(50),  
Status CHAR(10) DEFAULT 'Brucoš' )
```

#### **Specificiranje ograničenja domena**

Za specificiranje ograničenja domena, primer se odnosi na šemu relacije ZAPISNIK:

```
ZAPISNIK(StudId: INTEGER, PrKod: STRING, Rok: STRING, Ocena:  
INTEGER), Ključ: {StudId, PrKod, Rok}
```

Pored primarnog ključa koji se sastoji od tri atributa, definisano je ograničenje domena (navođenjem CHECK kod kreiranja tabele), da Ocena može da ima vrednosti u opsegu od 5 do 10 i vrednost ne može da bude NULL, kao i ograničenje da vrednost StudId može da bude između 0 i 7000:

```
CREATE TABLE ZAPISNIK (  
StudId INTEGER,
```

```

PrKod  CHAR(6),
Rok    CHAR(6),
Ocena  INTEGER NOT NULL,
PRIMARY KEY (StudId,PrKod,Rok),
CHECK (Ocena IN ( 5,6,7,8,9,10) AND VALUE IS NOT NULL),
CHECK (StudId >0 AND StudId <7000) )

```

### Kreiranje korisnički definisanog domena

Za kreiranje korisnički definisanog domena koristi se CREATE DOMAIN. U primeru je navedeno kreiranje domena OCENA. Nakon toga je kod kreiranja tabele ZAPISNIK iskorišćen ovaj domen za specificiranje tipa jedne od kolona:

```

CREATE DOMAIN OCENA INTEGER
CHECK (VALUE IN (5,6,7,8,9,10) AND VALUE IS NOT NULL )
CREATE TABLE ZAPISNIK (
StudId  INTEGER,
PrKod   CHAR(6),
Rok     CHAR(6),
Ocena   OCENA,
PRIMARY KEY (StudId,PrKod,Rok),
CHECK (StudId >0 AND StudId <7000) )

```

### Specificiranje ograničenja stranog ključa

Specificiranje ograničenja stranog ključa je prikazano za zadatu šemu baze podataka, gde su označeni primarni (podvučena imena) i strani ključevi (italic):

```

STUDENT(Ind, Ime, Adresa,Status)
PROFESOR(Id, Ime, KatId)
PREDMET(KatId, PrKod, PrNaziv, Opis)
ZAPISNIK(StudId, PrKod, Rok, Ocena)
IZBOR(StudId, PrKod, Semestar)
NASTAVA(ProfId, PrKod, Semestar)

```

Može da se uoči da su neki atributi delovi primarnog ključa, a ujedno i strani ključevi – praktično, oni mogu da budu posledica preslikavanja veza M:N ili slabih tipova entiteta.

SQL naredba za kreiranje tabele ZAPISNIK sa specifikacijom ograničenja stranih ključeva:

```
CREATE TABLE ZAPISNIK (  
    StudId  INTEGER,  
    PrKod   CHAR(6),  
    Rok     CHAR(6),  
    Ocena   OCENA,  
    PRIMARY KEY (StudId,PrKod,Rok),  
    FOREIGN KEY (StudId) REFERENCES STUDENT(Ind),  
    FOREIGN KEY (PrKod) REFERENCES PREDMET )
```

Ova tabela ima dva strana ključa, StudId i PrKod, koji se referenciraju na odgovarajuće attribute STUDENTA i PREDMETA.

### Unos, brisanje i ažuriranje torki

Za promenu vrednosti torki, kao i za inos novih koriste se sledeće SQL naredbe:

Unos torki:

```
INSERT INTO STUDENT(Ind, Ime, Adresa,Status)  
VALUES (11708, 'Ana', 'Nišavska 11 Piroć', 'apsolvent')
```

Ova SQL naredba omogućava unos jedne torke u relaciju odnosno vrste. Mogu se izostaviti imena atributa (kolona) u klauzuli INTO, ali se vrednosti atributa moraju navesti u redosledu koji je definisan šemom relacije. Dobar stil je da se imena atributa eksplicitno navedu.

Brisanje torki:

```
DELETE FROM STUDENT S  
WHERE S.ime= 'Ana'
```

Naredba DELETE, u ovom slučaju, iz tabele STUDENT briše torku koja u koloni Ime ima vrednost Ana. Ukoliko se uslov u WHERE izostavi, naredba će obrisati sve torke/vrste u tabeli STUDENT. Ova naredba briše podatke a ne i definiciju tabele u bazi podataka.

Ažuriranje vrednosti torki:

```
UPDATE STUDENT S  
SET S.Status= 'poslediplomac'  
WHERE S.ind= 11708
```

Ova SQL naredba omogućava ažuriranje vrednosti navedenih atributa u SET pod uslovima zadatim u WHERE. Navedena naredba u tabeli STUDENT ažurira torku koja u koloni Ind ima vrednost 11708 tako što postavlja novu vrednost

poslediplomac u koloni Status. Ako se ne navede uslov, ažuriraće se vrednosti navedenog atributa u svim torkama.

## 4.7 Operacijska komponenta

U relacionom modelu nad relacijama je definisan skup operacija koje omogućavaju specificiranje upita nad bazom podataka. Jedna od osnovnih karakteristika relacionog modela je da su operandi operacija koji se koriste kod zadavanja upita, cele relacije. Rezultat upita odnosno primene operacija je nova relacija koja može biti formirana od jedne ili više početnih relacija koje se pretražuju.

Postoje dva tipa operacija nad relacionim modelom podataka:

- Operacije relacione algebre
- Operacije relacionog računa

Operacije relacione algebra su opisane u narednom poglavlju. Relacioni račun nije predmet ovog materijala odnosno uvodnog kursa iz baza podataka.

## 4.8 Pregled terminologije osnovnih koncepata

Ovaj odeljak pokušava da ukratko da pregled prethodno opisanih koncepata relacionog modela i time dodatno omogućiti razjašnjenje terminologije.

Ono što je očigledno je da je relacija u relacionom modelu, odgovara pojmu tabela u bazi podataka. U svetu baza podataka, obično se koriste jedni termini kada se govori o relacionom modelu, a drugi kada se govori o bazi podataka, odnosno implementaciji relacionog modela. Uporedni prikaz termina i njihovog značenja dat je na slici 4-5.

Relacioni model	Baza podataka
Relacija	Tabela
Torka	Vrsta
Atribut	Kolona
Domen atributa	Tip podatka kolone
Šema relacije	Opis tabele

**Slika 4-5. Ekvivalentni skup pojmova kod baza podataka**

Na slici 4-6. su prikazane neformalne definicije, odnosno objašnjenja osnovnih koncepata relacionog modela podataka. Više o ovim konceptima i o relacionom modelu podataka možete naći u literaturi koja je navedena na kraju ovog materijala.

Koncept relacionog modela	Objašnjenje (neformalni opis)
<b>Relacija</b>	Tabela sa vrstama i kolonama.
<b>Relaciona baza podataka</b>	Kolekcija normalizovanih relacija sa različitim imenima.
<b>Šema relacije</b>	Opis relacije. Sadrži ime relacije, imena atributa i domene atributa.
<b>Šema relacione baze podataka</b>	Opis baze podataka. Skup šema relacija i skup ograničenja, pri čemu svaka relacija ima različito ime.
<b>Atribut</b>	Odabrana osobina klase entiteta ili klase poveznika, deo šeme relacije; U kontekstu relacije, neformalno se odnosi na imenovanu kolonu relacije.
<b>Domen atributa</b>	Specifikacija skupa dozvoljenih vrednosti za jedan ili više atributa.
<b>Torka relacije</b>	Jedna vrsta relacije odnosno tabele.
<b>Stepen relacije</b>	Broj atributa iz šeme relacije čije vrednosti relacija sadrži.
<b>Kardinalnost relacije</b>	Broj torki koje relacija sadrži.

Slika 4-6. Pregled osnovnih koncepata relacionog modela

## 4.9 Preslikavanje ER/EER modela u relacioni model

Preslikavanje ER modela u relacioni odvija se u sedam sukcesivnih koraka, a za preslikavanje dodatnih koncepata EER modela su neophodna još tri koraka. Nakon preslikavanja, na osnovu zadatog ER/EER dijagrama baze podataka koji je razvijen u fazi konceptualnog projektovanja, dobija se šema baze podataka koja sadrži odgovarajuće šeme relacija. Ove relacije će kasnije biti tabele u bazi podataka koje će čuvati podatke.

Nadalje je opisan postupak preslikavanja korak po korak. Nakon toga dati su primeri koji ilustruju postupak preslikavanja.

### Preslikavanje ER modela u relacioni model

#### Korak 1. Preslikavanje regularnog tipa entiteta

Za svaki regularni tip entiteta E u ER šemi kreira se šema relacije R koja sadrži sve proste attribute i sve proste komponente svih složenih atributa iz E.



Jedan od ključnih atributa iz E se uzima za primarni ključ šeme relacije R. Ako je ključni atribut u E složen, tada se njegov skup prostih atributa uzima zajedno kao primarni ključ u R.

**Korak 2. Preslikavanje slabog tipa entiteta**

Za svaki slabi tip entiteta S u ER šemi čiji je vlasnik tip entiteta E kreira se šema relacije R koja sadrži sve proste attribute iz S i sve proste komponente svih složenih atributa iz S.

Šema relacije R kao spoljni ključ sadrži sve attribute primarnog ključa šeme relacije tipa entiteta E.

Za primarni ključ šeme relacije R uzima se kombinacija primarnog ključa vlasnika E i parcijalnog ključa slabog tipa entiteta S.

**Korak 3. Preslikavanje veza 1:1**

Za svaki binarni tip veza  $R(1:1)$  u ER šemi identifikuju se šeme relacija S i T koje odgovaraju tipovima entiteta koji participiraju u R. Bira se jedna od ove dve šeme relacija, recimo S, i u nju se kao spoljni ključ uključuje primarni ključ šeme relacije T.

Za šemu relacije S treba birati tip entiteta koji totalno participira u R. Kao attribute šeme relacije S treba uključiti sve proste attribute i sve proste komponente složenih atributa tipa veze R.

Može se izabrati i alternativno rešenje po kome se formira zajednička šema relacije za tipove entiteta S i T u koju se uključuju svi atributi iz obe šeme relacije. Ovo je rešenje dobro kada oba tipa entiteta totalno participiraju u R i kada ne participiraju ni u jednom drugom tipu veze.

**Korak 4. Preslikavanje veza 1:N**

Za svaki binarni tip veze  $R(1:N)$  u ER šemi identifikuje se šema relacije S koja participira na N strani. U S se kao spoljni ključ uključuje primarni ključ šeme relacije T koja predstavlja tip entiteta koji participira na 1 strani.

Kao attribute šeme relacije S treba uključiti sve proste attribute i sve proste komponente složenih atributa tipa veze R.

**Korak 5. Preslikavanje veza M:N**

Za svaki binarni tip veze  $R(M:N)$  u ER šemi kreira se nova šema relacije S. U S se kao spoljni ključevi uključuju primarni ključevi šema relacija koje predstavljaju tipove entiteta koji participiraju u R.

Kombinacija ovih spoljnih ključeva formira primarni ključ šeme relacije S. U šemi relacije S se takođe uključuju svi prosti atributi i sve proste komponente složenih atributa tipa veze R.

**Korak 6.** Preslikavanje viševrednosnog atributa

Za viševrednosni atribut A tipa entiteta E ili tipa poveznika P kreira se nova šema relacije R koja sadrži atribut A i primarni ključ K šeme relacije kojom je predstavljen tip entiteta E ili tip veze P.

Za primarni ključ šeme relacije R bira se kombinacija A i K. Ako je viševrednosni atribut složen, uključuju se sve njegove proste komponente.

**Korak 7.** Preslikavanje n-arnog tipa veze

Za svaki n-arni tip veze R,  $n > 2$ , kreira se nova šema relacije S. U S se kao spoljni ključevi uključuju primarni ključevi šema relacija koje predstavljaju tipove entiteta koji participiraju u R.

Kombinacija ovih spoljnih ključeva formira primarni ključ šeme relacije S.

U šemi relacije S se takođe uključuju svi prosti atributi i sve proste komponente svih složenih atributa n-arnog tipa veze.

Ako neki tip entiteta E participira u R sa ograničenjem (min,max) i ako je max=1, tada se kao primarni ključ šeme relacije S može uzeti onaj spoljni ključni atribut kojim se šema relacije S referencira na šemu relacije kojom je predstavljen tip entiteta E.

**Preslikavanje dodatnih koncepata EER modela u relacioni model**

**Korak 8.** Preslikavanje veze potklasa/natklasa (generalizacija/specijalizacija)

Postoje četiri alternative za preslikavanje veze potklasa/natklasa:

- a) Kreirati šeme relacija za svaku natklasu sa svim atributima natklase.  
Kreirati šeme relacija za svaku potklasu sa svim atributima potklase i ključem natklase, koji je istovremeno i ključ potklasa.
- b) Kreirati šeme relacija za sve potklase tako da sadrže sve attribute natklase i sve attribute potklase. Primarni ključ šeme relacije potklase je primarni ključ njegove natklase. Treba uočiti da kod ove opcije ne kreira šema relacije za natklasu.
- c) Kreirati jednu šemu relacije koja sadrži sve attribute natklase i sve attribute potklasa i atribut *tip* koji svojom vrednošću određuje potklasu.
- d) Kao pod (c) s tim što se dodaju atributi  $t_1, t_2, \dots, t_n$  ( $n$  je broj potklasa,  $dom(t_i, i=1,n)=Boolean$ ) koji određuju pripadnost potklasi. Ova opcija je pogodna za preklapajuće potklase, mada se može primeniti i za disjunktne potklase.

**Korak 9.** Preslikavanje deljivih potklasa

Obično ide primena koraka 8(a).

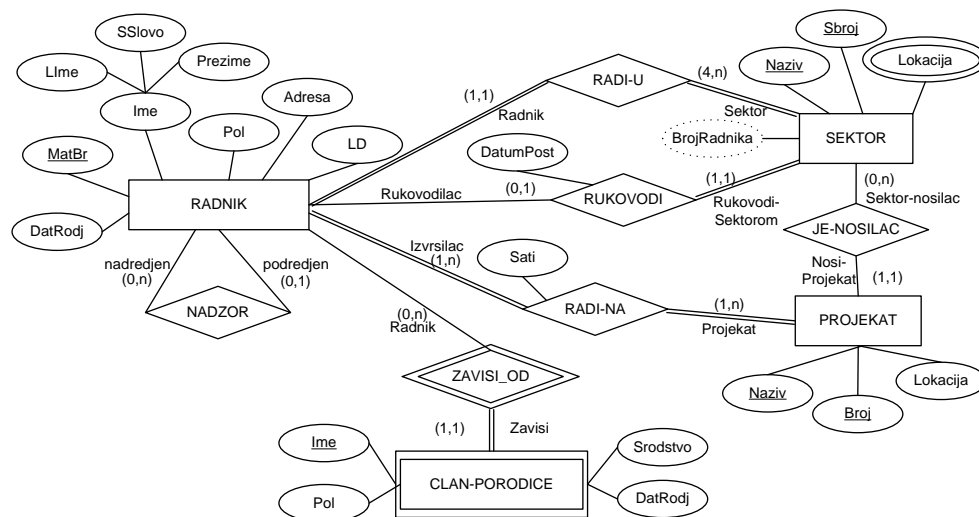
### Korak 10. Preslikavanje kategorija

Kreirati šemu relacije za kategoriju i šeme relacija za njene natklase. Ako su ključevi različiti, za kategoriju se generiše novi atribut koji predstavlja surogat ključ; šema relacije sadrži sve attribute kategorije i surogat ključ. Dodati surogat ključ svakoj šemi relacije koja predstavlja natklasu kategorije koja se preslikava.

### Primeri

#### Primer: Preslikavanje ER modela baze podataka PREDUZEĆE

ER dijagram za bazu podataka PREDUZEĆE dat je na slici 4-7. Ovaj dijagram je dobijen projektovanjem baze podataka na osnovu zahteva iz odgovarajućeg primera u prethodnom poglavlju.



Slika 4-7. ER dijagram baze podataka PREDUZEĆE

Postupak preslikavanja ER modela baze podataka PREDUZEĆE u relacioni model podataka:

#### Korak 1: Preslikavanje regularnih tipova entiteta.

Regularni tipovi entiteta u bazi podataka PREDUZEĆE su RADNIK, SEKTOR i PROJEKAT, tako da se u ovom koraku dobijaju relacije sa istim imenima. Za svaku relaciju označeni su i primarni ključevi.

RADNIK

LIME	SSLOVO	PREZIME	<u>MATBR</u>	DATRODJ	POL	PLATA	ADRESA
------	--------	---------	--------------	---------	-----	-------	--------

SEKTOR

<u>NAZIV</u>	<u>SBROJ</u>
--------------	--------------

PROJEKAT

<u>NAZIV</u>	LOKPR	<u>BROJPR</u>
--------------	-------	---------------

**Korak 2:** Preslikavanje slabih tipova entiteta

Jedini slabi tip entiteta je ČLAN-PORODICE, pa se za njega kreira nova relacija, koja pored atributa slabog tipa entiteta sadrži i ključ relacije vlasnika, MATBRRAD. Primarni ključ je zajedno ključ relacije vlasnika i parcijalni ključ, odnosno MATBRRAD i IME.

CLAN-PORODICE				
<u>MATBRRAD</u>	<u>IME</u>	POL	SRODSTVO	DATROĐ

**Korak 3:** Binarne veze (poveznici) tipa 1:1.

Jedina veza 1:1 je RUKOVODI, u kojoj potpuno učestvuje tip entiteta SEKTOR, kome se dodaje atribut DAT\_POST (datum postavljanja rukovodioca) i spoljni ključ MATBRR (matični broj rukovodioca) koji predstavlja primarni ključ relacije RADNIK.

SEKTOR			
NAZIV	<u>SBROJ</u>	MATBRR	DATPOST

**Korak 4:** Binarne veze tipa 1:N.

Binarne veze tipa 1:N su RADI\_U i NADZOR, u kojima se na N strani javlja RADNIK, kao i veza JE\_NOSILAC u kojoj je na N strani PROJEKAT. Relacijama na N strani se dodaju kao spoljni ključevi primarni ključevi relacije na 1 strani.

RADNIK					
LIME	SSLOVO	PREZIME	<u>MATBR</u>	DATRODJ	POL

	PLATA	ADRESA	MATBROJS	BRSEK
--	-------	--------	----------	-------

PROJEKAT			
NAZIV	LOKPR	<u>BROJPR</u>	BRS

**Korak 5:** Binarna veza M:N

Binarna veza RADI-NA između tipova entiteta RADNIK i PROJEKAT je tipa M:N, tako da se za nju formira posebna relacija RADI-NA u koju se kao spoljni ključevi uključuju primarni ključevi relacija RADNIK i PROJEKAT. Ova dva spoljna ključa formiraju primarni ključ relacije RADI-NA.

RADI-NA		
<u>MBR</u>	<u>BRPR</u>	SATI

**Korak 6:** Viševrednosni atributi

Atribut LOKACIJA tipa entiteta SEKTOR je viševrednosni. Za njega se formira relacija LOK-SEKTOR koja kao spoljni ključ ima primarni ključ relacije SEKTOR.

LOK-SEKTOR	
<u>BRS</u>	<u>LOKACIJA</u>

**Korak 7:** U ER modelu baze podataka PREDUZEĆE ne postoje n-arne veze.

**Koraci 8-10:** U ER modelu baze podataka PREDUZEĆE nisu korišćeni dodatni koncepti proširenog modela.

Kompletni relacioni model baze podataka PREDUZEĆE dobijen nakon preslikavanja ER modela prikazan je na slici 4-8.

**RADNIK**

<u>LIME</u>	SSLOVO	PREZIME	<u>MATBR</u>	DATRODJ	POL	
-------------	--------	---------	--------------	---------	-----	--

	PLATA	ADRESA	MATBROJS	BRSEK
--	-------	--------	----------	-------

**PROJEKAT**

NAZIV	LOKPR	<u>BROJPR</u>	BRS
-------	-------	---------------	-----

**SEKTOR**

NAZIV	<u>SBROJ</u>	MATBRR	DATPOST
-------	--------------	--------	---------

**CLAN\_PORODICE**

<u>MATBRRAD</u>	<u>IME</u>	POL	SRODSTVO	DATROD
-----------------	------------	-----	----------	--------

**LOK\_SEKTOR**

<u>BRS</u>	<u>LOKACIJA</u>
------------	-----------------

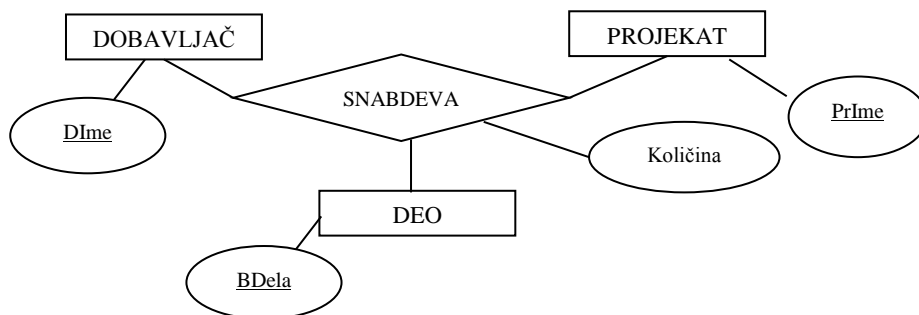
**RADI\_NA**

<u>MBR</u>	<u>BRPR</u>	SATI
------------	-------------	------

**Slika 4-8. Relacioni model baze podataka PREDUZEĆE**

**Primer:** Preslikavanje ternarnog tipa veze

Na slici 4-9. je dat deo ER dijagrama koji sadrži ternarnu vezu i dobijeni relacioni model. U relaciju koja se kreira za ternarnu vezu je pored ključeva svih relacija koje učestvuju u vezi dodat i atribut veze Količina.



**Slika 4-9. Deo šeme koji sadrži ternarnu vezu SNABDEVA**

Preslikavanje:

Za ternarni tip poveznika SNADBEVA relaciona šema sadrži ključeve attribute iz sva tri tipa entiteta odnosno odgovarajućih relacija, kao i atribut veze KOLIČINA:

DOBAVLJAČ		SNABDEVA	
<u>DIME</u>	...	<u>DIME</u>	<u>DEO</u>
		<u>PRIME</u>	KOLIČINA
DEO		PROJEKAT	
<u>BDELA</u>	...	<u>PRIME</u>	...

**Primer:** Preslikavanje ER modela baze podataka VIDEO\_KLUB

ER dijagram za bazu podataka VIDEO\_KLUB dat je slici 53. Ovaj dijagram je dobijen projektovanjem baze podataka na osnovu zahteva iz odgovarajućeg primera u prethodnom poglavlju.

Preslikavanjem ER modela baze podataka VIDEO\_KLUB dobija se relaciona šema ove baze podataka prikazana na slici 4-10.

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

**Slika 4-10. Relacioni model baze podataka VIDEO\_KLUB**

## 5 RELACIONA ALGEBRA

Relaciona algebra je formalni jezik za relacioni model. Zasniva se na matematičkoj teoriji skupova. Jako je važna pošto obezbeđuje **formalnu osnovu** za operacije relacionog modela, a takođe se koristi kao osnova za implementaciju i optimizaciju upita u RDBMS-u. Neki od njenih koncepata su ugrađeni u SQL standardni upitni jezik za relacioni model podataka.

Relaciona algebra je definisana sa ciljem da se obezbedi skup operacija pogodan za iskazivanje upita, odnosno selekciju i dobijanje podataka iz relacione baze podataka. Sekvenca operacija relacione algebre formira **izraze relacione algebre** čiji je rezultat takođe relacija, koja predstavlja rezultat upita nad bazom podataka. Kao što je napomenuto u prethodnom poglavlju, bitna karakteristika operacija u relacionom modelu (samim tim i operacija relacione algebre) jeste da se relacije i operandi i rezultat operacija. Samim tim, kod izraza relacione algebre, koji se sastoje od operatora i operandi su relacije (skupovi torki ili reprezentanti skupova torki), a rezultat primene izraza je takođe relacija.

Za iskazivanje **upita** putem relacione algebre koriste se dve vrste operatora:

- **Specijalni operatori relacione algebre:**
  - selekcija,
  - projekcija,
  - spoj,
  - deljenje.
- **Standardni operatori matematičke teorije skupova:**
  - unija,
  - presek,
  - razlika,
  - Dekartov proizvod.

Osnovni skup operacija relacione algebra čine sledeće operacije:

- Selekcija
- Projekcija
- Unija
- Razlika
- Dekartov proizvod

Preostale operacije su iz takozvanog proširenog skupa pošto se mogu izvesti primenom formalnih pravila iz operacija osnovnog skupa. Prošireni skup operacija relacione algebra čine:

- Presek
- Deljenje
- Spoj i varijante spoja: Unutrašnji, Spoljašnji (Levi, Desni ili Potpuni) i Polu spoj

## 5.1 Selekcija

Selekcija je operacija kojom se iz relacije **izdvajaju** one **torke** koje imaju zadatu vrednost specifikiranih atributa. Atributi i njihove vrednosti po kojima se vrši selekcija se zadaju **uslovom selekcije**. Na slici 5-1. prikazane su dve torke dobijene selekcijom sa zadatim uslovom da radnici rade u sektoru 10.

radnik

MBR	LIME	LD	SBR
2203	MIRA	50 000	<b>10</b>
2817	PERA	40 000	20
2932	MIRA	35 000	<b>10</b>
2995	GOCA	18 000	30
3305	LAZA	60 000	40
3515	JOVAN	20 000	20
3819	VLADA	65 000	30

**Slika 5-1. Rezultat selekcije su izdvojene vrste relacije**

Operacija selekcije se označava na sledeći način:

$$\sigma_{\langle \text{uslov selekcije} \rangle} (\langle \text{ime relacije} \rangle)$$

Selekcija se vrši iz zadate relacije  $\langle \text{ime relacije} \rangle$ . Atributi i njihove vrednosti po kojima se vrši selekcija se zadaju **uslovom selekcije**. Rezultat ove operacije je



relacija koja sadrži one torke početne relacije koje zadovoljavaju zadati <uslov selekcije>.

Redosled atributa rezultatne relacije ekvivalentan je redosledu atributa relacije nad kojom je primenjena selekcija.

Vodite računa da je selekcija unarna operacija! Kao i kod svih ostalih operacija relacione algebre, rezultat primene selekcije nad nekom relacijom je takođe relacija.

Uslov selekcije može da bude prost i složeni.

**Prost uslov selekcije** je logički izraz oblika:

- $A_i \theta A_j$
- $A_i \theta C$
- $C \theta A_i$

gde je:

$\theta \in \{<, =, >, <=, >=, !=\}$ ,  $A_i$  i  $A_j$  imena atributa relacije nad kojom se selekcija izvodi,  $C$  konstanta koja uzima vrednosti iz domena atributa  $A_i$ .

**Složeni uslov selekcije** se sastoji od prostih uslova selekcije koji su povezani operatorima  $\wedge$ ,  $\vee$  i  $\neg$ , odnosno AND, OR i NOT.

**Primer:** Selekcija

Iz relacije *radnik* nad šemom relacije RADNIK (MBR,LIME,LD,SBR) izabрати sve radnike koji ispunjavaju uslov:

- (a)  $LD > 20000$
- (b)  $LD > 20000$  i  $SBR = 10$

Odgovarajuće operacije relacione algebre su:

- (a)  $\sigma_{LD>20000}(\text{radnik})$
- (b)  $\sigma_{LD>20000 \text{ AND } SBR=10}(\text{radnik})$

Rezultat primene ovih operacija je dat na slici 5-2. Data je početna relacija **radnik** i rezultat primene operacije selekcije pod (a) i (b).

radnik

<u>MBR</u>	LIME	LD	SBR
2203	MIRA	50 000	10
2817	PERA	40 000	20
2932	MIRA	35 000	10
2995	GOCA	18 000	30
3305	LAZA	60 000	40
3515	JOVAN	20 000	20
3819	VLADA	65 000	30

 $\sigma_{LD > 20000}$  (radnik)

(a)

<u>MBR</u>	LIME	LD	SBR
2203	MIRA	50 000	10
2817	PERA	40 000	20
2932	MIRA	35 000	10
3305	LAZA	60 000	40
3819	VLADA	65 000	30

 $\sigma_{LD > 20000 \text{ AND } SBR = 10}$  (radnik)

(b)

<u>MBR</u>	LIME	LD	SBR
2203	MIRA	50 000	10
2932	MIRA	35 000	10

Slika 5-2. Primer rezultata primene selekcije

Selekcija je **komutativna** operacija, odnosno važi:

$$\sigma_{\langle \text{uslov1} \rangle} (\sigma_{\langle \text{uslov2} \rangle} (r)) = \sigma_{\langle \text{uslov2} \rangle} (\sigma_{\langle \text{uslov1} \rangle} (r))$$

Posledica komutativnosti je da sekvenca operacija selekcije u nekom upitu relacione algebra može biti primenjena u bilo kom redosledu ili zamenjena jednom operacijom selekcije sa složenim uslovom:

$$\sigma_{\langle \text{uslov1} \rangle} (\sigma_{\langle \text{uslov2} \rangle} (\dots \sigma_{\langle \text{uslovn} \rangle} (r))) =$$

$$\sigma_{\langle \text{uslov1} \rangle \text{ AND } \langle \text{uslov2} \rangle \text{ AND } \dots \text{ AND } \langle \text{uslovn} \rangle} (r)$$

## 5.2 Projekcija

Projekcija je operacija kojom se iz relacije **izdvajaju kolone** koje odgovaraju atributima po kojima se vrši projekcija i kojom se iz tako dobijene relacije eliminišu jednake torke. Na slici 5-3. prikazana je projekcija dve kolone relacije **radnik** – MBR i LD.

radnik

<u>MBR</u>	LIME	LD	SBR
2203	MIRA	50 000	<b>10</b>
2817	PERA	40 000	20
2932	MIRA	35 000	<b>10</b>
2995	GOCA	18 000	30
3305	LAZA	60 000	40
3515	JOVAN	20 000	20
3819	VLADA	65 000	30

Slika 5-3. Primer primene operacije projekcije

Operator projekcije se označava sa:

$$\pi_{\langle \text{lista atributa} \rangle}(\langle \text{ime relacije} \rangle)$$

gde  $\langle \text{lista atributa} \rangle$  definiše za koje attribute se vrši projekcija nad zadatom relacijom  $\langle \text{ime relacije} \rangle$ . Rezultat projekcije je **relacija**. Redosled atributa rezultatne relacije definisan je listom atributa projekcije.

Navedeno je da se kod projekcije eliminišu jednake torke. Ova osobina je posledica toga da operacije relacione algebra tretiraju relacije kao skupove, pa se i u ovom slučaju podrazumeva da se kod projekcije vrednosti jedne ili više kolona prikazuju samo različite vrednosti. Kod upitnog jezika SQL to nije slučaj!

Projekcija nije komutativna operacija.

#### Primer: Projekcija

Projektovati po atributima LIME, SBR relaciju radnik definisanu nad šemom relacije RADNIK(MBR, LIME, LD, SBR):

$$\pi_{\text{LIME, SBR}}(\text{radnik})$$

Projekcija istih atributa, ali u obrnutom redosledu:

$$\pi_{\text{SBR, LIME}}(\text{radnik})$$

Rezultat projekcije u oba slučaja je prikazan na slici 5-4. Iz primera se vidi da kod operacije projekcije navođenja atributa određuje redosled atributa u rezultujućoj relaciji.

radnik				$\pi_{LIME,SBR}(\text{radnik})$		$\pi_{SBR,LIME}(\text{radnik})$	
MBR	LIME	LD	SBR	LIME	SBR	SBR	LIME
2203	MIRA	50 000	10	MIRA	10	10	MIRA
2817	PERA	40 000	20	PERA	20	20	PERA
2932	MIRA	35 000	10	GOCA	30	30	GOCA
2995	GOCA	18 000	30	LAZA	40	40	LAZA
3305	LAZA	60 000	40	JOVAN	20	20	JOVAN
3515	JOVAN	20 000	20	VLADA	30	30	VLAD A
3819	VLADA	65 000	30				

Slika 5-4. Rezultat projekcije nad relacijom *radnik*

### 5.3 Preimenovanje

Preimenovanje je operacija koja omogućava promenu imena relacije i imena atributa u rezultatnoj relaciji. Ova operacija se može koristiti u upitima relacione algebre kako bi rezultati upita bili razumljiviji.

Operator preimenovanja se označava sa:

$$\rho_{\langle \text{novo ime relacije i/ili atributa} \rangle}(\langle \text{ime relacije} \rangle)$$

**Primer:** Operacija Preimenovanje

Varijante operacije preimenovanja, pri čemu je *s* novo ime relacije  $r(A_1, A_2, \dots, A_n)$ ,  $B_1, B_2, \dots, B_n$  su nova imena atributa  $A_1, A_2, \dots, A_n$ .

Preimenovanje imena relacije i imena atributa

$$\rho_{s(B_1, B_2, \dots, B_n)}(r)$$

Preimenovanje imena relacije

$$\rho_s(r)$$

Preimenovanje atributa relacije

$$\rho_{(B_1, B_2, \dots, B_n)}(r)$$

### 5.4 Operacije relacione algebre iz teorije skupova

Operacije relacione algebre iz teorije skupova praktično su standardne matematičke operacije nad skupovima: unija, presek, razlika i Dekartov proizvod. Pošto se radi o operacijama nad skupovima, relacija rezultat ovih operacija sadrži samo različite torke.

**Unija** relacija *r* i *s* je skup torki koje pripadaju relacijama *r* ili *s* ili obema. Uslov za obavljanje ove operacije je da su *r* i *s* **unijski kompatibilne relacije**. To znači da obe relacije imaju **isti stepen** i da su **domeni** korespondentnih atributa u obe relacije **isti**. Rezultujuća relacija ima imena atributa prve relacije.

Unija je komutativna i asocijativna operacija:

$$r \cup s = s \cup r$$

$$(r \cup s) \cup t = r \cup (s \cup t)$$

**Razlika** relacija  $r$  i  $s$  je skup torki koje pripadaju relaciji  $r$  ali ne pripadaju relaciji  $s$ . Relacije  $r$  i  $s$  treba da su **unijski kompatibilne**. Rezultujuća relacija ima imena atributa prve relacije.

Razlika nije komutativna operacija:

$$r - s \neq s - r$$

**Presek** relacija  $r$  i  $s$  je skup torki koje pripadaju obema relacijama. Uslov za obavljanje ove operacije je da su  $r$  i  $s$  **unijski kompatibilne** relacije. Rezultujuća relacija ima imena atributa prve relacije.

Presek se može izraziti preko razlike na sledeći način:

$$r \cap s = r - (r - s)$$

Presek je komutativna i asocijativna operacija:

$$r \cap s = s \cap r$$

$$(r \cap s) \cap t = r \cap (s \cap t)$$

**Primer:** Skupovne operacije

Prikazati rezultat primene operacija razlike, unije i preseka relacija **student** i **kandidat**.

Unija (rezultat primene je prikaza na slici 5-5a):

$$\text{student} \cup \text{kandidat}$$

(zbog komutativnosti je isto što i  $\text{kandidat} \cup \text{student}$ )

Presek (rezultat primene je prikaza na slici 5-5b i 5-5c):

$$\text{student} \cap \text{kandidat}$$

$$\text{kandidat} \cap \text{student}$$

Razlika (rezultat primene je prikaza na slici 5-5d):

$$\text{student} - \text{kandidat} \text{ (ili } \text{kandidat} - \text{student)}$$

student

IND	IME	DATR	GRAD
1211	PERA	010971	NI
1213	VERA	130872	NI
1215	MILA	150171	LE
1217	VERA	220371	SV

kandidat

ID	IME	DATR	GRAD
1213	VERA	130872	NI
1217	VERA	220371	SV
1223	MIKA	101072	LE

(a) student  $\cup$  kandidat

IND	IME	DATR	GRAD
1211	PERA	010971	NI
1213	VERA	130872	NI
1215	MILA	150171	LE
1217	VERA	220371	SV
1223	MIKA	101072	LE

(b) student - kandidat

IND	IME	DATR	GRAD
1211	PERA	010971	NI
1215	MILA	150171	LE

(c) kandidat - student

ID	IME	DATR	GRAD
1223	MIKA	101072	LE

(d) student  $\cap$  kandidat

IND	IME	DATR	GRAD
1213	VERA	130872	NI
1217	VERA	220371	SV

Slika 5-5. Primer primene skupovnih operacija

**Dekartov proizvod** relacija  $r$  i  $s$  nad šemama relacije  $R(A_1, A_2, \dots, A_n)$  i  $S(B_1, B_2, \dots, B_m)$  je relacija  $q$  nad šemom relacije  $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$  koju čine torke dužine  $n+m$ , gde prvih  $n$  komponenti čini torku u  $r$ , a drugih  $m$  torku u  $s$ .

Relacija  $q$  ima po jednu torku za sve kombinacije torki, jedna iz  $r$ , jedna iz  $s$ . Ako  $r$  ima  $K_r$  torki i ako  $s$  ima  $K_s$  torki, tada  $q$  ima  $K_r \times K_s$  torki.

Dekartov proizvod se označava na sledeći način:

$$q = r \times s$$

**Primer:** Dekartov proizvod

Dekartov proizvod relacija  $r$  i  $s$ ,  $r \times s$ , je prikazan na slici 5-6. Rezultat predstavlja skup torki koje su dobijene tako što je napravljena kombinacija prve torke iz  $r$  sa svim torkama iz  $s$  (prve četiri torke u rezultatu), pa nakon toga kombinacija druge torke iz  $r$  sa svim torkama iz  $s$  (poslednje četiri torke u rezultatu).

r			r x s				
A	B	C	A	B	C	D	E
a1	b1	c1	a1	b1	c1	d1	e1
a2	b2	c2	a1	b1	c1	d2	e2
			a1	b1	c1	d3	e3
			a2	b2	c2	d1	e1
			a2	b2	c2	d2	e2
			a2	b3	c2	d3	e3

s	
D	E
d1	e1
d2	e2
d3	e3

Slika 5-6. Primer primene Dekartovog proizvoda

**Deljenje** ili količnik relacija r i s, u oznaci r/s, je skup torki t koje se javljaju u r u kombinaciji sa svim torkama iz s. Deljenje nije iz osnovnog skupa operacija i može se izraziti pomoću operacija projekcija, Dekartov proizvod i razlika ( $\pi$ , x, -) na sledeći način:

$$\pi_y(r) \rightarrow q1$$

$$\pi_y((s \times q1) - r) \rightarrow q2$$

$$q1 - q2 \rightarrow q$$

Deljenje nije ni komutativna, ni asocijativna operacija.

**Primer:** Deljenje

Rezultat deljenja relacija r i s je prikazan na slici 5-7. U rezultatu su vrednosti za atribut A: a1 i a5 pošto se jedino one pojavljuju u r u kombinaciji sa b1 i b2.

r		s	r/s
A	B	B	A
a1	b1	b1	a1
a1	b2	b2	a5
a3	b1		
a4	b2		
a4	b3		
a5	b1		
a5	b2		

Slika 5-7. Primer primene operacije deljenja

## 5.5 $\theta$ -spoj

**$\theta$ -spoj** je spoj relacija  $r$  i  $s$  nad šemama relacija  $R(A_1, A_2, \dots, A_n)$  i  $S(B_1, B_2, \dots, B_m)$  je relacija  $q$  nad šemom  $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$  koja ima po jednu torku za svaku kombinaciju torki, jedna iz  $r$  i jedna iz  $s$ , kad god ova kombinacija zadovoljava **uslov spoja** (što je glavna razlika u odnosu na Dekartov proizvod).

Za spoj se koristi oznaka:

$$r \bowtie_{\langle \text{uslov spoja} \rangle} s$$

gde je **<uslov spoja>** izraz formata:

$$\langle \text{uslov} \rangle \wedge \dots \wedge \langle \text{uslov} \rangle$$

$\langle \text{uslov} \rangle$  je oblika  $A_i \theta B_j$   $\theta \in \{=, <, <=, >, >=, !=\}$

Torke čiji atributi spoja imaju NULL vrednost ne pojavljuju se u rezultatu.

Varijacije  $\theta$ -spoja:

- **Equijoin (ekvi spoj)** je spoj gde je uslov spoja oblika  $A_i = B_j$ .
- **Natural join (prirodni spoj)** je ekvi spoj gde je iz rezultata isključen jedan od dva jednaka atributa ( $A_i$  ili  $B_j$ )

Prva varijanta spoja je na osnovu jednakosti atributa i koristićete je uglavnom u upitima relacione algebre. Ona je osnova za implementaciju spoja kod SQL upitnog jezika.

Prirodni spoj se označava sa “\*” ili se navodi simbol za spoj ali bez navođenja uslova. Da bi prirodni spoj mogao da se primeni, oba atributa u uslovu spoja moraju da **imaju isto ime**. Ukoliko to nije slučaj koristimo operaciju preimenovanja.

**Primer:** Operacija spoja.

Rezultat spoja relacija **radnik** i **projekat** na osnovu uslova spoja koji definiše da su vrednosti broja sektora SBR iste u obe relacije je dat na slici 5-8:

$$\text{radnik} \bowtie_{\text{radnik.SBR}=\text{projekat.SBR}} \text{projekat}$$

Redosled atributa u rezultujućoj relaciji je takav da se navode atributi iz relacije **radnik** (koja je s leve strane operacije), a nakon toga iz relacije **sektor**. Atribut SBR se pojavljuje na dva mesta.



radnik

<u>MBR</u>	LIME	LD	SBR
2203	MIRA	50 000	10
2817	PERA	40 000	20
2932	MIRA	35 000	10
2995	GOCA	18 000	30
3305	LAZA	60 000	40
3515	JOVAN	20 000	20
3819	VLADA	65 000	30

projekat

<u>PBR</u>	PNAZIV	SBR	PRUK
100	PC	10	2203
200	HOST	20	3305
300	LAN	30	3819
400	VIPX	40	2817

radnik  $\bowtie$  radnik.SBR=projekat.SBRprojekat

<u>MBR</u>	IME	LD	SBR	<u>PBR</u>	PNAZIV	SBR	PRUK
2203	MIRA	50 000	10	100	PC	10	2203
2932	MIRA	35 000	10	100	PC	10	2203
2817	PERA	40 000	20	200	HOST	20	3305
3515	JOVAN	20 000	20	200	HOST	20	3305
2995	GOCA	18 000	30	300	LAN	30	3819
3819	VLADA	65 000	30	300	LAN	30	3819
3305	LAZA	60 000	40	400	VIPX	40	2817

Slika 5-8. Primer primene operacije spoja

**Primer:** Prirodni spoj

Rezultat prirodnog spoja relacija **radnik** i **projekat**:

radnik  $\bowtie$  projekat

radnik \* projekat

Atribut na osnovu kog se vrši prirodni spoj je SBR – njegov naziv je isti u obe relacije, a uslov je jednakost vrednosti ovih atributa u obe relacije. Rezultat primene prirodnog spoja ove dve relacije je prikazan na slici 5-9. Redosled atributa u rezultujućoj relaciji je kao kod spoja - navode se najpre atributi iz relacije **radnik** (koja je s leve strane operacije), a nakon toga iz relacije **sektor**. U ovom slučaju se atribut SBR pojavljuje samo na jednom mestu koje mu je određeno pozicijom u relaciji koja se navodi s leve strane operacije prirodnog spoja.

radnik \* projekat  
 radnik ⋈ projekat




<u>MBR</u>	IME	LD	SBR	<u>PBR</u>	PNAZIV	PRUK
2203	MIRA	50 000	10	100	PC	2203
2932	MIRA	35 000	10	100	PC	2203
2817	PERA	40 000	20	200	HOST	3305
3515	JOVAN	20 000	20	200	HOST	3305
2995	GOCA	18 000	30	300	LAN	3819
3819	VLADA	65 000	30	300	LAN	3819
3305	LAZA	60 000	40	400	VIPX	2817

Slika 5-9. Primer primene prirodnog spoja

**Spoljašnji spoj** je operacija kod koje se navodi uslov spoja koji se odnosi na jednakost atributa kao kod klasičnog spoja. Ipak, rezultat primene zavisi od tipa spoljašnjeg spoja, i može da sadrži torke iz neke od relacija iako one ne ispunjavaju uslov jednakosti. Ovaj spoj može da ima sledeće varijante:

- **Levi (Left) spoj** – rezultat zadržava sve torke iz leve relacije bez obzira na to da li ispunjavaju uslov spoja, i one relacije s desne strane koje ispunjavaju zadati uslov. Za one torke s leve strane koje ne ispunjavaju uslov, u desni deo se upisuje NULL vrednost.
- **Desni (Right) spoj** - rezultat zadržava sve torke iz desne relacije bez obzira na to da li ispunjavaju uslov spoja, ali se u levi deo upisuju NULL vrednost za sve one koje ne ispunjavaju uslov.
- **Puni (Full) spoj** - rezultat zadržava sve torke bez obzira na to da li ispunjavaju uslov spoja. Neuparene torke dobijaju NULL vrednost na suprotnoj strani.

Spoljašnji spojevi se obeležavaju na sličan način kao spoj, s tim da se u simbolu dodaju oznake u obliku dve crtice s leve, desne strane ili obostrano, da bi se označilo da li se radi o levom, desnom ili punom spoju:

- Oznaka za levi spoljašnji spoj: 
- Oznaka za desni spoljašnji spoj: 
- Oznaka za puni, odnosno obostrani spoljašnji spoj: 

Spoljašnji spoj se može koristiti da bi se dobila unija torki u slučaju kada relacije nisu unijski kompatibilne. Ova operacija obezbeđuje uniju svih torki relacija koje su parcijalno kompatibilne odnosno kada su samo neki atributi unijski kompatibilni.

## 5.6 Funkcije agregacije

**Funkcije agregacije** su funkcije koje omogućavaju da se na osnovu vrednosti zadatih atributa korišćenjem tih funkcija dobiju odnosno izračunaju nove vrednosti (tzv. agregirane vrednosti).

Označavanje funkcije agregacije u relacionoj algebri:

$$\langle \text{atributi grupisanja} \rangle \mathcal{F} \langle \text{lista funkcija} \rangle (r)$$

gde su:

$\langle \text{atributi grupisanja} \rangle$  lista atributa relacije  $r$

$\langle \text{lista funkcija} \rangle$  je lista parova ( $\langle \text{funkcija} \rangle \langle \text{atribut} \rangle$ )

$\langle \text{funkcija} \rangle$  je funkcija koja se primenjuje i može biti:

SUM - za izračunavanje sume vrednosti navednog atributa,

AVERAGE - za izračunavanje prosečne vrednosti,

MAXIMUM - za određivanje maksimalne vrednosti,

MINIMUM - za određivanje minimalne vrednosti,

COUNT - za brojanje ne-nultih vrednosti zadatog atributa.

**Primer:** Funkcije agregacije

Za svaki sektor naći broj radnika u tom sektoru i prosečan lični dohodak.

$$\text{SBR } \mathcal{F} \text{ COUNT MBR, AVERAGE LD}(\text{radnik})$$

U datom rešenju grupisanje je izvršeno po broju sektora SBR i za svaku grupu torki koja ima istu vrednost SBR (odnosno za svaki sektor) se vrši izračunavanje prosečne vrednosti plate (AVERAGE LD) i broje se radnici (COUNT MBR). Funkcija COUNT broji ne-nulte vrednosti navedenog atributa.

## 5.7 Primeri upita u relacionoj algebri

Upiti u relacionoj algebri se grade korišćenjem osnovnih operacija. Pošto je rezultat primene svake operacije relacija, ona može biti ulaz za narednu operaciju u upitu. Rezultat sekvence primenjenih operatora (odnosno upita relacione algebre) je takođe relacija koja sadrži željene podatke iz početnih relacija. Upiti relacione algebre se mogu napisati na više alternativnih načina, a u navedenim primerima prikazano je samo po jedno rešenje.

Primeri upita se odnose na dve šeme relacionih baza podataka: prva koja sadrži šeme relacija koje se odnosi se na studente, predmete i vezu između njih koja

definiše koje je predmete neki student izabrao, a druga se odnosi na bazu podataka PREDUZEĆE.

Šema baze podataka o studentima:

STUDENT(Indeks, Ime, Adresa, Mentor)

PREDMET(ID, Naziv)

IZABRAO(Indeks, IDP)

**Primer 1.** Prikazati Indekse i Imena studenata

Indeksi i imena studenata su podaci koji se čuvaju u relaciji STUDENT pa se upit svodi na primenu operacije projekcije koja vraća različite vrednosti parova Indeks, Ime:

$$\pi_{\text{Indeks, Ime}}(\text{STUDENT})$$

**Primer 2.** Prikazati ID-ove onih predmeta koje je izabrao neki student (izabrani predmeti)

Pošto se u ovom primeru ne traže dodatni podaci o predmetima, već samo ID, a informacije o predmetima koji su izabrani od studenata su u relaciji IZABRAO, upit se svodi na jednostavnu primenu projekcije:

$$\pi_{\text{IDP}}(\text{IZABRAO})$$

**Primer 3.** Podaci o studentima čiji je mentor Petar Petrović

Svi podaci o studentima su u relaciji STUDENT, uključujući i podatke o mentoru. Potrebno je izdvojiti one torke iz ove relacije koje zadovoljavaju uslov na je vrednost atributa Mentor “Petar Petrović”. Primer se odnosi na očiglednu primenu selekcije:

$$\sigma_{\text{Mentor} = \text{“Petar Petrović”}}(\text{STUDENT})$$

**Primer 4.** Prikazati nazive izabranih predmeta.

Informacije o izabranim predmetima su u relaciji IZABRAO, a nazivi predmeta su u relaciji PREDMET. Da bi ste prikazali nazive izabranih predmeta potrebno je uraditi spoj ove dve relacije i nakon toga projekcijom izdvojiti samo tražene attribute:

$$\pi_{\text{Naziv}}(\text{PREDMET} \bowtie_{\text{ID} = \text{IDP}} \text{IZABRAO})$$

**Primer 5.** Prikazati ID-ove predmeta koje nije izabrao nijedan student

U ovom slučaju projekcijom ID iz relacije PREDMET dobićemo različite vrednosti ID-ova svih predmeta. Projekcijom IDP iz relacije IZABRAO dobićemo različite identifikatore samo izabranih predmeta. Razlika ova dva skupa (svi predmeti i izabrani predmeti) su predmeti koje nije izabrao nijedan student:

$$\pi_{ID}(PREDMET) - \pi_{IDP}(IZABRAO)$$

**Za vežbu:** Studenti koji nisu izabrali nijedan predmet.

**Primer 6.** Nazivi predmeta koje nije izabrao nijedan student.

Deo rešenja ovog upita je rešenje prethodnog upita. Rezultat smeštamo u privremenu relaciju **r**:

$$r \leftarrow \pi_{ID}(PREDMET) - \pi_{IDP}(IZABRAO) \text{ - prethodni primer}$$

Da bi prikazali nazive predmeta potrebno je uraditi spoj sa relacijom PREDMET koja čuva nazive, i nakon toga projekcijom izdvojiti samo različite nazive predmeta:

$$\pi_{Naziv}(r \bowtie_{r.ID=Predmet.ID} PREDMET)$$

**Primer 7.** Imena studenata i nazivi predmeta koje su izabrali.

Privremena relacija  $r_1$  sadrži predmete koje su studenti izabrali;  $r_2$  sadrži podatke o studentima i predmetima koje su izabrali (praktično ova dva spoja su veza studenata i predmeta preko relacije IZABRAO). Na kraju, projekcijom su prikazana imena studenata i nazivi predmeta:

$$r_1 \leftarrow PREDMET \bowtie_{ID=IDP} IZABRAO$$

$$r_2 \leftarrow r_1 \bowtie_{r_1.Indeks=Student.Indeks} STUDENT$$

$$r \leftarrow \pi_{Ime,Naziv}(r_2)$$

**Primer 8.** Indeksi studenata koji su izabrali predmet “Baze podataka”.

Pomoćna relacija  $r_1$  sadrži podatke o predmetu Baze podataka, koji se spojem sa IZABRAO povezuju sa studentima koji su ga izabrali:

$$r_1 \leftarrow \sigma_{Naziv = \text{“Baze podataka”}}(PREDMET)$$

$$r_2 \leftarrow r_1 \bowtie_{ID=IDP} IZABRAO$$

$$r \leftarrow \pi_{Indeks}(r_2)$$

**Za vežbu:** Indeksi studenata koji su izabrali predmet “Baze podataka” ILI predmet “Strukture podataka”.

Ideja 1: promena uslova kod selekcije

Ideja 2: rešenje iz prethodnog primera (2 puta) + unija

**Za vežbu:** Indeksi studenata koji su izabrali oba predmeta: “Baze podataka” i “Strukture podataka”.

Šema baze podataka PREDUZEĆE:

RADNIK(MBR,LIME,PREZIME,DRODJ,ADRESA,LD,SBR,SEF)

SEKTOR(SBR,SNAZIV,SLOK,MBR)

PROJEKAT(PBR,PNAZIV,SBR,PRUK)

RADINA(MBR,PBR,SATI)

ČLAN\_PORODICE(MBR,IME,DRODJ,SRODSTVO)

**Primer 1:** Naći ime, prezime i adresu svih radnika koji rade u sektoru 5

RADNIK(MBR,LIME,PREZIME,DRODJ,ADRESA,LD,SBR,SEF)

SEKTOR(SBR,SNAZIV,SLOK,MBR)

Rešenje:

$$r \leftarrow \pi_{LIME, PREZIME, ADRESA} \sigma_{SBR=5} (radnik)$$

**Primer 2:** Naći imena, prezimena i adrese svih radnika koji rade u sektoru ‘PROIZVODNJA’ u Nišu

$$r1 \leftarrow \sigma_{SNAZIV='PROIZVODNJA' \text{ AND } SLOK='NIŠ'} (sektor)$$

$$r2 \leftarrow r1 \bowtie_{f1.SBR=radnik.SBR} radnik$$

$$r \leftarrow \pi_{MBR, LIME, LD}(r2)$$

**Primer 3:** Naći ime, prezime i adresu svih radnika koji rade u sektoru ‘RAZVOJ’

RADNIK(MBR,LIME,PREZIME,DRODJ,ADRESA,LD,SBR,SEF)

SEKTOR(SBR,SNAZIV,SLOK,MBR)

Rešenje

$r1 \leftarrow \sigma_{SNAZIV='RAZVOJ'}(sektor)$

$r2 \leftarrow radnik \bowtie_{radnik.SBR=r1.SBR} r1$

$r \leftarrow \pi_{LIME, PREZIME, ADRESA}(r2)$

Drugi oblik upita

$r \leftarrow \pi_{LIME, PREZIME, ADRESA}(radnik \bowtie_{radnik.SBR=sektor.SBR} (\sigma_{SNAZIV='RAZVOJ'}(sektor)))$

**Primer 4:** Naći imena radnika koji rade na svim projektima koje nadgleda sektor broj 5

RADNIK(MBR,LIME,PREZIME,DRODJ,ADRESA,LD,SBR,SEF)

PROJEKAT(PBR,PNAZIV,SBR,PRUK)

RADINA(MBR,PBR,SATI)

Rešenje:

$r1 \leftarrow \sigma_{SBR=5}(projekat)$  projekti koje nadgleda sektor 5

$r2 \leftarrow r1 \bowtie_{r1.PBR = radina.PBR} radina$

$r3 \leftarrow r2 \bowtie_{r2.MBR = radnik.MBR} radnik$  - radnici koji rade na projektima sek 5

$r4 \leftarrow \pi_{LIME,PREZIME}(r3)$

**Primer 5:** Naći imena radnika koji imaju 2 i više izdržavanih lica

RADNIK(MBR,LIME,PREZIME,DRODJ,ADRESA,LD,SBR,SEF)

ČLAN\_PORODICE(MBR,IME,DRODJ,SRODSTVO)

Rešenje:

$r1(MBR, BROJ\_IZL) \leftarrow_{MBR} \mathcal{F} COUNT IME(ČLAN\_PORODICE)$

$r2 \leftarrow \sigma_{BROJ\_IZL \geq 2}(r1)$

$r \leftarrow \pi_{LIME,PREZIME}(r2 \bowtie_{r2.MBR = radnik.MBR} radnik)$





## 6 NORMALIZACIJA I NORMALNE FORME

Jedan od ciljeva relacionog modela je da se obezbedi zaštita podataka od čestih i potencijalno opasnih reorganizacija baze podataka, što se može postići tako što se sprečavaju anomalije održavanja podataka u bazi podataka i omogućava bolje korišćenje informacija iz baze podataka. U ostvarivanju tih ciljeva značajnu ulogu ima normalizacija i normalne forme.

Da bi se obezbedilo da relacija nema neželjena svojstva koja mogu da postoje među atributima, u procesu normalizacije se ograničavaju odnosi među atributima na određene tipove zavisnosti. Za relaciju u kojoj takvi odnosi važe kažemo da je normalizovana, odnosno da se nalazi u normalnoj formi.

Normalizacija se oslanja na pojmove i definicije iz relacionog modela, kao što su funkcionalne zavisnosti, zatvarač skupa funkcionalnih zavisnosti i sl. Zbog toga je u ovom poglavlju, u prvom odeljku dat pregled definicija koje se odnose na funkcionalne zavisnosti. Nakon toga, nakon kratkog odeljka koji navodi potencijalne probleme koji mogu da se jave zbog postojanja anomalija u ažuriranju i razloga zašto je potrebna normalizacija, sledi odeljak u kome su date definicije normalnih formi i postupka normalizacije.

### 6.1 Funkcionalne zavisnosti

Funkcionalna zavisnost (FZ) je prvi tip ograničenja definisan u okviru relacionog modela podataka. FZ specificira ograničenja između dva skupa atributa u relacionoj bazi podataka i koriste se za specifikaciju određenih, u praksi vrlo značajnih, normalnih formi organizacije baze podataka, kao i za specifikaciju algoritama transformacije postojećih šema relacija u željene normalne forme.

#### Podsetnik:

Šema relacije  $R(A;F)$  je definisana na osnovu:

- skupa atributa  $A$ , i
- skupa funkcionalnih zavisnosti  $F$  kojima se specificiraju ograničenja

Funkcionalna zavisnost je izraz oblika  $f: X \rightarrow Y$ , gde **f** predstavlja **naziv** funkcionalne zavisnosti, a **X** i **Y** su skupovi atributa iz posmatrane šeme relacije.

Često se koristi skraćena notacija:  $X \rightarrow Y$ . Značenje navedenog izraza  $X \rightarrow Y$  se može interpretirati na dva načina:

- da Y funkcionalno zavisi od X ili
- da X funkcionalno određuje Y

To dalje znači da se svakom elementu iz domena atributa X, **dom(X)** može pridružiti najviše jedan element iz domena atributa Y, **dom(Y)**.

**Primer:** Funkcionalna zavisnost atributa

Data je funkcionalna zavisnost:

Grad, Adresa  $\rightarrow$  PoštanskiKod

Ako u relaciji postoje atributi Grad, Adresa i Poštanski kod, očigledna funkcionalna zavisnost je da naziv grada i adresa zajedno funkcionalno određuju poštanski kod, odnosno da poštanski kod funkcionalno zavisi od naziva grada i adrese.

### Definicija funkcionalne zavisnosti

Funkcionalna zavisnost se može definisati na sledeći način:

Preduslov: Neka je **r** relacija nad šemom relacije **R(A;C)** i neka su **X** i **Y** podskupovi skupa atributa **A={A1, A2, ..., An}**

U šemi relacije **R** postoji funkcionalna zavisnost **f: X  $\rightarrow$  Y** ako i samo ako je svakom elementu iz **dom(X)** pridružen najviše jedan element iz **dom(Y)**, odnosno, na osnovu vrednosti atributa X može se odrediti odgovarajuća vrednost atributa Y.

Posmatrano na nivou torki, funkcionalna zavisnost  $X \rightarrow Y$  znači da za bilo koje dve torke **t1** i **t2** iz **r** važi implikacija:

$$t1[X] = t2[X] \Rightarrow t1[Y] = t2[Y]$$

Kao posledica toga, ako je **t1[X] = t2[X]** (odnosno vrednost atributa X u torci t1 jednaka je vrednosti istih atributa u torci t2), tada mora biti **t1[Y] = t2[Y]**, odnosno vrednosti atributa Y u odgovarajućim torkama moraju da budu iste. Praktično, vrednost komponente Y torke u relaciji **r** zavisi od vrednosti komponente X, odnosno vrednost komponente X funkcionalno određuje vrednost komponente Y.

**Primer:** Funkcionalne zavisnosti u relaciji

Zadata je šema relacije POSETILAC({IME,ADRESA};{IME $\rightarrow$ ADRESA})

Funkcionalna zavisnost **IME**→**ADRESA** može se tumačiti na sledeći način:

- Vrednost atributa **IME** na jedinstven način određuje vrednost atributa **ADRESA**
- Ako u instanci relacije **posetilac(POSETILAC)** postoji torka  
 $\langle \text{GAGA}, \text{Nišavska 14} \rangle$ ,

tada kad god se u nekoj instanci relacije **posetilac(POSETILAC)** pojavi vrednost **GAGA** u koloni **IME**, u koloni **ADRESA** ove torke mora biti vrednost **Nišavska 14**

**Primer:** Značenje funkcionalne zavisnosti

U šemi relacije:

**RADPROJ**(**MBR**,**RIME**,**SATI**,**PBR**,**PNAZIV**,**PLOK**,**SBR**,**PRUK**)

na osnovu semantike atributa mogu da se prepoznaju sledeće funkcionalne zavisnosti:

1.  $\text{MBR} \rightarrow \text{RIME}$
2.  $\text{MBR} \rightarrow \text{SBR}$
3.  $\text{PBR} \rightarrow \{\text{PNAZIV}, \text{PLOK}\}$
4.  $\{\text{MBR}, \text{PBR}\} \rightarrow \text{SATI}$
5.  $\text{PBR} \rightarrow \text{PRUK}$

Značenje navedenih FZ, redom, se može protumačiti na sledeći način:

1. Matični broj radnika jednoznačno određuje ime radnika
2. Matični broj radnika jednoznačno određuje broj sektora u kome radnik radi
3. Broj projekta jednoznačno određuje naziv i lokaciju projekta
4. Matični broj radnika i broj projekta jednoznačno određuju vreme (SATI) angažovanja radnika na projektu
5. Broj projekta jednoznačno određuje rukovodioca projekta

Semantika ovih ograničenja je:

1. Svaki radnik ima jedinstven matični broj (**MBR**)
2. Radnik može raditi samo u jednom sektoru (**SBR**)
3. Svaki projekat ima jedinstven broj projekta (**PBR**)
4. Radnik može biti angažovan na više projekata određeni broj sati
5. Svaki projekat ima samo jednog rukovodioca

## 6.2 Izvođenje funkcionalnih zavisnosti

Projektanti baze podataka specificiraju funkcionalne zavisnosti koje su semantički očigledne u toku projektovanja baze podataka. U svim relacijama nad šemom relacije  $R$  u kojoj važi skup funkcionalnih zavisnosti  $F$  postoje i mnoge druge FZ. U realnom sistemu nije moguće specificirati sve moguće FZ za datu situaciju, zato što je neke teško prepoznati, a druge mogu biti na neki način prikrivene.

Ove FZ se mogu **izvesti** iz FZ koje postoje u  $F$  i koje su definisane u fazi projektovanja baze podataka. Za dobijanje tih novih funkcionalnih zavisnosti na osnovu postojećih je potreban **alat za izvođenje**. Da bismo uveli takav alat potrebno je upoznati sledeće koncepte:

- zatvarač skupa FZ
- pravila izvođenja FZ
- zatvarač skupa atributa

### 6.2.1 Zatvarač skupa funkcionalnih zavisnosti

#### Definicija:

**Zatvarač** ili **zatvaranje** skupa funkcionalnih zavisnosti  $F$ , u oznaci  $F^+$ , je skup funkcionalnih zavisnosti sadržan u  $F$  i skup svih funkcionalnih zavisnosti koje se mogu izvesti iz  $F$ .

Za potrebe izvođenja novih FZ iz postojećih definišu se **pravila** ili **aksiome izvođenja** funkcionalnih zavisnosti. Kardinalni broj zatvarača je vrlo veliki čak i za mali broj FZ u  $F$  i za mali broj atributa.

**Pravila (aksiome)** izvođenja funkcionalnih zavisnosti su:

P1 (**Refleksivnost**) : Ako je  $Y \subseteq X$ , tada važi  $X \rightarrow Y$

P2 (**Proširenje**) : Ako je  $X \rightarrow Y$  i  $Z \subseteq W$ , tada važi  $XW \rightarrow YZ$

P3 (**Tranzitivnost**) : Ako je  $X \rightarrow Y$  i  $Y \rightarrow Z$ , tada važi  $X \rightarrow Z$

P4 (**Dekompozicija**) : Ako je  $X \rightarrow YZ$ , tada važi  $X \rightarrow Y$  i  $X \rightarrow Z$

P5 (**Unija**) : Ako  $X \rightarrow Y$  i  $X \rightarrow Z$ , tada važi  $X \rightarrow YZ$

P6 (**Pseudotranzitivnost**) : Ako  $X \rightarrow Y$  i  $YW \rightarrow Z$ , tada važi  $XW \rightarrow Z$

gde su  $X, Y, W$  i  $Z$  podskupovi skupa atributa  $A$  šeme relacije  $R(A;F)$  i gde  $XY$  označava uniju skupova  $X$  i  $Y$ .

Pravila P1-P3 se nazivaju **Armstrongove aksiome**. Pravilo 1 dovodi do definisanja tzv. **trivijalnih FZ**. To su zavisnosti za koje važi da je desna strana FZ podskup leve strane.

U pravilu P2, ako je  $Z=W$ , tada ovo pravilo postaje:

Ako je  $X \rightarrow Y$ , tada važi  $XZ \rightarrow YZ$

Ako je  $W$  prazan skup u Pravilu 6, tada ono prelazi u tranzitivnost.

Na osnovu pravila unije (P5) sledi da se svaki skup funkcionalnih zavisnosti  $F$  može zameniti ekvivalentnim skupom  $G$  u kojem se na desnoj strani svake FZ nalazi samo 1 atribut.

Iscrpnom primenom pravila izvođenja P1-P6 na skup funkcionalnih zavisnosti  $F$  dobija se skup funkcionalnih zavisnosti  $F^+$ .

U nastavku je dat pseudokod algoritma za nalaženje zatvarača  $F^+$ . Algoritam kao ulaz ima skup atributa šeme relacije i skup inicijalnih funkcionalnih zavisnosti, a rezultat je skup  $F^+$ . Inicijalno, zatvarač ima vrednost početnog skupa FZ, i u narednim koracima se proširuje. Algoritam za svaku FZ iz aktulene vrednosti  $F^+$  (do tog koraka generisane FZ kao i početni skup) primenjuje pravilo refleksivnosti i dodaje rezultat u  $F^+$  (koraci 4 i 5). Za svaki par FZ pokušava da primenom tranzitivnosti generiše nove FZ i da ih doda u  $F^+$ . Postupak se ponavlja sve dok se generišu nove FZ.

#### Algoritam za nalaženje $F^+$

**Ulaz:** Skup atributa  $A=\{A_1, A_2, \dots, A_n\}$ , i

Skup funkcionalnih zavisnosti  $F=\{f_1, f_2, \dots, f_m\}$  šeme relacije  $R(A;F)$

**Izlaz:**  $F^+$  (funkcionalno zatvaranje skupa  $F$ )

1.  $F^+ := F$ ;
2. *repeat*
3.   za svaku funkcionalnu zavisnost  $f$  u  $F^+$
4.     primeniti pravilo refleksivnosti i pravilo proširenja na  $f$ ;
5.     dodati rezultujuće funkcionalne zavisnosti u  $F^+$ ;
6.   za svaki par funkcionalnih zavisnost  $f_i$  i  $f_j$  u  $F^+$
7.     ako se  $f_i$  i  $f_j$  mogu kombinovati korišćenjem tranzitivnosti
8.     dodati rezultujuću funkcionalnu zavisnost u  $F^+$ ;
9. *until* (ne menja se  $F^+$ )

### 6.2.2 Zatvarač skupa atributa

#### Definicija:

**Zatvarač skupa atributa**  $X$  (gde je  $X \subseteq A$ ) u odnosu na skup funkcionalnih zavisnosti  $F$  je skup atributa  $Y$  koje funkcionalno određuje  $X$ :

$$X_F^+ = \{Y \mid X \rightarrow Y \in F^+\}$$

Zatvarač skupa atributa ima više primena:

- Za proveru da li je  $X$  super ključ šeme relacije: Proveravamo da li  $X^+$  sadrži sve attribute šeme relacije.
- Za proveru da li funkcionalna zavisnost  $X \rightarrow Y$  važi u  $F$ : Proveravamo da li je  $Y \subseteq X^+$ .
- Kao alternativni način za izračunavanje  $F^+$ : Za svako  $X \subseteq A$  šeme relacije  $R(A;F)$  nalazimo  $X^+$  i za svako  $Y \subseteq X^+$  dodajemo u  $F^+$  funkcionalnu zavisnost  $X \rightarrow Y$ .

Algoritam za nalaženje zatvarača  $X^+$  skupa atributa  $X$  zadate šeme relacije  $R$  podrazumeva da postupak krene od skupa  $X$ , i vrši se provera svih funkcionalnih zavisnosti iz  $F$ . Ako leva strana posmatrane FZ predstavlja podskup do tada generisanog zatvarača skupa atributa, onda se zatvarač proširuje skupom atributa sa desne strane te FZ. Postupak se ponavlja sve dok postoje promene, odnosno sve dok se dodaju novi atributi u  $X^+$ .

#### Algoritam za nalaženje $X^+$

**Ulaz:** Konačan skup atributa  $A$  šeme relacija  $R(A;F)$ , skup funkcionalnih zavisnosti  $F$  nad  $A$  i podskup  $X$  skupa  $A$

**Izlaz:**  $X^+$  (zatvaranje  $X$  u odnosu na  $F$ ).

1.  $X^+ := X$ ;
2. *repeat*
3.    $\text{staro}X^+ := X^+$ ;
4.   za svaku FZ  $Y \rightarrow Z$  u  $F$
5.     ako  $Y \subseteq X^+$  tada  $X^+ := X^+ \cup Z$ ;
6. *until* ( $X^+ = \text{staro}X^+$ )

**Primer:** Nalaženje  $X^+$ 

U šemi relacije (ključ je prikazan na uobičajeni način, podvlačenjem imena ključnih atributa):

RADPROJ = (MBR, RIME, SATI, PBR, PNAZIV, PLOK)

postoje sledeće funkcionalne zavisnosti :

f1: {MBR, PBR}  $\rightarrow$  SATI

f2: MBR  $\rightarrow$  RIME

f3: PBR  $\rightarrow$  {PNAZIV, PLOK}

Prema gornjem algoritmu dobija se redom, za zatvarače skupa atributa  $X_1$ ,  $X_2$  i  $X_3$ , gde je:  $X_1 = \{MBR\}$ ,  $X_2 = \{PBR\}$ ,  $X_3 = \{MBR, PBR\}$ :

Za  $X_1 = \{MBR\}$ , polazimo od skupa  $\{MBR\}$  i posmatramo sve FZ. Jedino f2 s leve strane ima atribut MBR koji je „podskup“ početnog skupa, pa zatvaraču dodajemo attribute s desne strane ove FZ, a to je RIME. Nakon ovog koraka nema više promena u zatvaraču, pa je to kraj postupka:

$X_1 = \{MBR\}$ ,  $X_1^+ = \{MBR, RIME\}$

Za  $X_2 = \{PBR\}$ , polazimo od skupa  $\{PBR\}$  i posmatramo sve FZ. Jedino f3 s leve strane ima atribut PBR koji je „podskup“ početnog skupa, pa zatvaraču dodajemo attribute s desne strane ove FZ, a to su PNAZIV i PLOK. Nakon ovog koraka nema više promena u zatvaraču, pa je to kraj postupka:

$X_2 = \{PBR\}$ ,  $X_2^+ = \{PBR, PNAZIV, PLOK\}$

Za  $X_3 = \{MBR, PBR\}$ , polazimo od skupa  $\{MBR, PBR\}$  i posmatramo sve FZ. Sve date FZ sa leve strane imaju attribute koji su „podskup“ početnog skupa, pa zatvaraču dodajemo attribute s desne strane sve tri FZ (tri prolaza kroz *repeat* petlju algoritma):

$X_3 = \{MBR, PBR\}$ ,  $X_3^+ = \{MBR, PBR, SATI, RIME, PNAZIV, PLOK\}$

### 6.3 Ključ šeme relacije

**Definicija:**

Neka je  $R(\{A_1, A_2, \dots, A_n\}; F)$  šema relacije i neka je  $X \subseteq \{A_1, A_2, \dots, A_n\}$

$X$  je **ključ šeme relacije  $R$**  ako i samo ako ima sledeća svojstva:

**S1. Svojstvo identifikacije:**  $X$  funkcionalno određuje sve attribute šeme relacije, tj.  $\{X \rightarrow A_i \mid i=1, 2, \dots, n\}$

**S2. Svojstvo minimalnosti:** Nijedan pravi podskup od  $X$  nema tu osobinu, tj. za  $X' \subset X$  važi  $\{X' \rightarrow A_i \mid i=1,2,\dots,n\}$

Iz definicije ključa relacije sledi da svaka relacija ima barem jedan ključ, a to je skup svih atributa  $A = \{A_1, A_2, \dots, A_n\}$ .

Ako za neko  $X \subseteq \{A_1, A_2, \dots, A_n\}$  važi samo svojstvo S1 iz definicije ključa, tada  $X$  predstavlja **super ključ** relacije  $R$ . Treba uočiti da super ključ sadrži ključ relacije. Može se reći da je ključ relacije njen minimalni i stoga neredundantni super ključ.

Ako skup ključeva  $K$  šeme relacije  $R(R;F)$  ima više od jednog elementa, tada su svi oni **ključevi kandidati**. Jedan od ključeva kandidata se bira za **primarni ključ** šeme relacije.

**Primer:** Ključ šeme relacije

U šemi relacije  $RADNIK(\underline{MBR}, LIME, LD, SBR)$ ,  $\{MBR\}$  je jedini ključ kandidat, tako da je on istovremeno i primarni ključ.

U šemi relacije  $RADNIK(\underline{MBR}, JMB, LIME, LD, SBR)$ , ključevi kandidati su  $\{MBR, JMB\}$  ali je  $MBR$  izabran za primarni ključ

**Primarni ključ** je ključ koji je izabran da jedinstveno identifikuje entitete predstavljene relacijom. Stoga nijedan od njegovih atributa nesme nikada imati nedefinisanu (**null**) vrednost. Svaka šema relacije mora imati primarni ključ.

Atribut  $A_i$  šeme relacije  $R$  je **primarni (ključni) atribut** ako je član primarnog (bilo kog) ključa relacije zadate šemom  $R$ . Ostali atributi  $A_j \neq A_i$  šeme relacije  $R$  su **neprimarni (sporedni, neključni)**.

**Primer:** Primarni i neprimarni atributi

U šemi relacije  $RADPROJ$  atributi  $MBR$  i  $PBR$  su primarni (ključni), dok su svi ostali neprimarni (sporedni, neključni)

**Zavisnost ključa**

Ako je  $K$  jedan ključ šeme relacije  $(A;F)$ , tada važi funkcionalna zavisnost  $K \rightarrow A$ . Ova se funkcionalna zavisnost naziva **zavisnošću ključa**.

**Primer:** Zavisnost ključa

Ako je zadata šema relacije  $R(\{A,B,C,D,E\}; \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\})$  i ako je  $A$  ključ ove šeme relacije, tada u ovoj relaciji postoji sledeća zavisnost ključa:  $A \rightarrow A,B,C,D,E$



Algoritam za nalaženje jednog ključa je jednostavan i svodi se na primenu odnosno određivanje zatvarača skupa atributa. Algoritam inicijalno počinje sa skupom svih atributa šeme relacije  $R$  i nakon toga za svaki atribut vrši proveru da li je deo ključa ili se može izbaciti. Ova provera se vrši tako što se odredi zatvarač aktuelnog skupa atributa bez posmatranog atributa  $(K-X)$ . Ako se za zatvarač  $(K-X)^+$  dobije skup svih atributa relacije  $R$ , tada se posmatrani atribut  $X$  može izbaciti iz ključa. Naredni koraci podrazumevaju ponavljanje postupka tako što se u svakom od njih posmatra po jedan od preostalih atributa. Na kraju postupka određen je jedan ključ zadate šeme relacije.

#### Algoritam za nalaženje jednog ključa

**Ulaz:** Šema relacije  $(R;F)$ ; skup atributa  $R$  i skup funkcionalnih zavisnosti  $F$  nad skupom atributa  $R$

**Izlaz:** Ključ  $K$  šeme relacije  $(R;F)$

**Metoda:** Algoritam nalazi samo jedan ključ  $K$  za zadati skup atributa  $R$  i zadati skup  $FZ F$  nad skupom  $R$

1.  $K := R$ ;
2. Za **svaki** atribut  $X$  u  $K$ 
  - { izračunati  $(K-X)^+$  u odnosu na  $F$ ;
  - ako  $(K-X)^+$  sadrži sve attribute u  $R$
  - tada  $K := K - \{X\}$  };

**Primer:** Određivanje jednog ključa

Za zadatu šemu relacije  $(R;F)$  naći jedan ključ, gde je:

$R=(A,B,C,D,E)$ , i

$F=( A \rightarrow BC,$

$CD \rightarrow E,$

$B \rightarrow D,$

$E \rightarrow A)$

Primena algoritma, opisana po koracima:

1.  $K=\{A,B,C,D,E\}$  – inicijalno, potencijalni ključ se sastoji od svih atributa
2. Iz  $K$  izbacujemo attribute redom
  - Iz  $K$  izbacujemo atribut  $A$ , proveru da li je  $A$  deo ključa:

Izračunavamo  $\{K-A\}^+_F$ , tj.  $\{K-A\}^+_F = \{B, C, D, E, A\}$ ;

(CD određuje E, ali E je već u „ključu“; B određuje D koji je takođe u „ključu“, jedino se zbog poslednje FZ doda A)

$\{K-A\}^+$  sadrži sve attribute iz R,

tako da je  $K := K-A := \{B, C, D, E\}$ ;

Iz  $K = \{B, C, D, E\}$  izbacujemo B

(sada u K nema atributa A koji je izbačen u prethodnom koraku!)

Izračunavamo  $\{K-B\}^+_F$ , tj.  $\{K-B\}^+_F = \{C, D, E, A, B\}$ ;

$\{K-B\}^+$  sadrži sve attribute iz R,

$K := K-B$ , tj.  $K = \{C, D, E\}$ ;

Iz K izbacujemo atribut C

Izračunavamo  $\{K-C\}^+_F$ , tj.  $\{K-C\}^+_F = \{D, E, A, B, C\}$ ;

$\{K-C\}^+$  sadrži sve attribute iz R,

$K := K-C$ , tj.  $K = \{D, E\}$ ;

Iz K izbacujemo atribut D

Izračunavamo  $\{K-D\}^+_F$ , tj.  $\{K-D\}^+_F = \{E, A, B, C, D\}$ ;

$\{K-D\}^+$  sadrži sve attribute iz R, pa

$K := K-D$ , tj.  $K = \{E\}$ ;

Iz K izbacujemo atribut E

Izračunavamo  $\{K-E\}^+_F$ , tj.  $\{K-E\}^+_F = \emptyset$ ;

$\{K-E\}^+$  ne sadrži sve attribute iz R, pa K ostaje nepromenjeno, tj.  $K = \{E\}$ ;

3. KRAJ algoritma: Ključ zadate šeme relacije je  $\{E\}$

Algoritam za nalaženje svih ključeva šeme relacije se takođe oslanja na zatvarač skupa atributa i odgovara prethodnom algoritmu, ali je zahtevniji – postupak je potrebno ponoviti za svaki podskup akupa atributa šeme relacije R.

#### **Algoritam za nalaženje svih ključeva šeme relacije**

**Ulaz:** Šema relacije (R;F)

**Izlaz:** Skup ključeva SK šeme relacije (R;F)

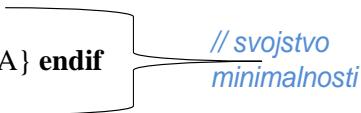
1.  $SK := \emptyset$ ;
2. **do** traži\_ključ  $K_i (\forall X \subseteq R)$  // za **svaki podskup** skupa atributa R
 

**if**  $R = X_F^+$  **then** // svojstvo identifikacije
 

**do** redukcija ( $\forall A \in X$ )
 

**if**  $A \in \{X-A\}^+$  **then**  $X = \{X-A\}$  **endif**

**enddo** redukcija



 $K_i = X$   
 Dodati  $K_i$  u SK  
**endif**

- 3. **enddo** traži\_ključ  $K_i$

**Primer:** Određivanje svih ključeva

Zadata je šema relacije (R;F), gde je

$R = (A, B, C, D, E)$  i  $F = (A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A)$

Naći sve ključeve ove relacije.

**Ulaz:**  $R = (A, B, C, D, E)$  i  $F = (A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A)$

1.  $SK = \emptyset$
2. Ispitujemo da li svojstvo ključa ima svaki podskup skupa atributa R, tj.:
 

$\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{A,B\}, \{A,C\}, \{A,D\}, \{A,E\}, \{B,C\}, \{B,D\},$   
 $\{B,E\}, \{C,D\}, \{C,E\}, \{D,E\}, \{A,B,C\}, \{A,B,D\}, \{A,B,E\}, \{A,C,D\}$   
 $\{A,C,E\}, \{A,D,E\}, \{B,C,D\}, \{B,C,E\}, \{B,D,E\}, \{C,D,E\}, \{A,B,C,D\},$   
 $\{A,B,C,E\}, \{A,B,D,E\}, \{A,C,D,E\}, \{B,C,D,E\}$
3. KRAJ algoritma.

**Izlaz:**  $SK = \{A, E, BC, CD\}$

Važni pojmovi koje ćemo u narednom poglavlju kod normalizacije pomenuti i koristiti, a odnose se na FZ su potpuna i parcijalna funkcionalna zavisnost, kao i tranzitivna zavisnost.

### Potpuna i parcijalna funkcionalna zavisnost

Neka su X i Y skupovi atributa. Funkcionalna zavisnost  $X \rightarrow Y$  je **potpuna** ako izbacivanjem bilo kog atributa A iz X ova zavisnost više ne postoji.

Ako postoji atribut A koji se može izbaciti iz X, a da se funkcionalna zavisnost  $X \rightarrow Y$  zadrži, onda je ova funkcionalna zavisnost **parcijalna**.

**Primer:** Potpuna i parcijalna FZ

U šemi relacije STUDENT(BRIND, JMB, IME, GODINA) funkcionalna zavisnost:

$$\{BRIND, JMB\} \rightarrow IME$$

nije potpuna budući da važi i funkcionalna zavisnost  $BRIND \rightarrow IME$ , odnosno da broj indeksa funkcionalno određuje ime studenta.

### Tranzitivna funkcionalna zavisnost

Neka su X i Y skupovi atributa šeme relacije R. Funkcionalna zavisnost  $X \rightarrow Y$  u šemi relacije R je **tranzitivna** zavisnost ako u R postoji skup atributa Z, koji nije podskup nijednog ključa relacije R, za koji:

- važi FZ gde  $X \rightarrow Z$ ,
- ne važi FZ da  $Z \rightarrow X$
- važi FZ gde  $Z \rightarrow Y$ .

Tranzitivna zavisnost postoji kod relacija sa najmanje tri atributa. Kod ovakve zavisnosti, u odnosu na primarni ključ, X treba da bude primarni ključ, a Z može biti ili kandidat za ključ ili njegov deo.

**Primer:** Tranzitivna FZ

U šemi relacije:

RADSEK(RIME, RMBR, DATRODJ, ADRESA, SBR, SNAZIV, SRUK)

postoji tranzitivna funkcionalna zavisnost:

$$RMBR \rightarrow SBR \text{ i } SBR \rightarrow SRUK,$$

a atribut SBR nije podskup nijednog ključa relacije.

## 6.4 Analiza šeme relacione baze podataka

Analiza šeme relacione baze podataka je proces određivanja kvaliteta projektovanih šema relacija. U teoriji baza podataka definisani su formalizmi za analizu kvaliteta šeme relacije i prevođenje šeme relacije u ekvivalentnu, bolju šemu. Postoje dva takva formalizma:

- Logičke zavisnosti i
- Normalne forme

Kvalitet projektovane šeme relacije se može posmatrati sa logičkog (sa strane korisnika) i fizičkog nivoa (sa strane sistema). Na **logičkom nivou** od šeme se zahteva da bude jasna i laka za razumevanje, a na **fizičkom nivou** od šeme se zahteva da obezbedi ažurni skup podataka uz minimalan utrošak računarskih resursa.

Logičke zavisnosti izražavaju činjenicu da jedan objekat zavisi od drugih objekata. U šemi relacije  $R(R;F)$  nad skupom atributa  $R=(A_1, A_2, \dots, A_n)$  mogu postojati različita ograničenja zavisnosti kao posledica osobina i pravila funkcionisanja realnog sistema koji se ovom relacijom modelira. Najčešće se sreću **funkcionalne** i **više značajne** zavisnosti.

U procesu povećanja kvaliteta šeme relacije mogu da se koriste različiti faktori, kao što su redukovanje redundantnih vrednosti u torkama, redukovanje NULL vrednosti u torkama ili isključivanje lažnih torki.

Pri obradi relacija mogu da se pojave različite anomalije. Nepoželjno ponašanje relacija pri izvođenju operacija ažuriranja naziva se anomalijom ažuriranja. Na osnovu operacije koja indukuje anomalije ažuriranja razlikujemo:

- anomalije unosa
- anomalije brisanja
- anomalije modifikacije

Anomalije ažuriranja ćemo razmotriti na primeru relacije *radsek* nad šemom relacije RADSEK(**RMBR**, RIME, RLD, SBR, SNAZIV, **SRUK**)

Očigledno je da se radi o loše projektovanoj šemi relacije (slika 6-1.). Ova relacija sadrži attribute dva entiteta: RADNIK i SEKTOR. Entitet RADNIK je opisan atributima {RMBR, RIME, RLD}, a entitet SEKTOR atributima {SBR, SNAZIV, SRUK}.

**RADSEK**

RMBR	RIME	RLD	SBR	SNAZIV	SRUK
111	Laza	100000	10	Projektni biro	333
222	Mara	60000	10	Projektni biro	333
333	Nada	150000	10	Projektni biro	333
444	Djura	50000	20	Kontrola	null

**Slika 6-1. Primer loše projektovane relacije**

U šemi relacije RADSEK postoje sledeće funkcionalne zavisnosti nad atributima:

f1: RMBR  $\rightarrow$  {RIME, RLD, SBR}

f2: SBR  $\rightarrow$  {SNAZIV, SRUK}

Pri izvođenju operacija nad ovom relacijom javljaju se sve tri vrste anomalija.

Relacija RADSEK(**RMBR**, RIME, RLD, SBR, SNAZIV, SRUK) **posедује anomalije unosa** koje se ogledaju u sledećem :

- U relaciju RADSEK se ne mogu uneti podaci o sektoru ukoliko u taj sektor nije raspoređen nijedan radnik. Ukoliko bi se takav unos dozvolio tada bi atributi RMBR, RIME i RLD imali nedefinisanu (NULL) vrednost, što nije dozvoljeno budući da je RMBR primarni ključ relacije
- Pri unosu podataka o novom radniku potrebno je, uz konkretne vrednosti za sve attribute entiteta RADNIK, uneti konkretne vrednosti za sve attribute entiteta SEKTOR. Pri svakom unosu moramo voditi računa da podaci o sektoru budu konzistentni sa već unetim podacima o tom sektoru. Ponavljanje podataka o sektoru u torkama svih radnika tog sektora govori o prisustvu redundance u relaciji.

Opisane anomalije su posledica sledeće **tranzitivne funkcionalne zavisnosti** među atributima relacije:

RMBR  $\rightarrow$  {RIME, RLD, SBR},

SBR  $\rightarrow$  {SNAZIV, SRUK},

Pri čemu SBR nije podskup ključa relacije.

Ove anomalije se mogu otkloniti **dekompozicijom** šeme relacije RADSEK tako da se svaka funkcionalna zavisnost predstavi posebnom relacijom. Rezultat ove dekompozicije su sledeće šeme relacija:

RADNIK(**RMBR**, RIME, RLD, SBR)

SEKTOR(**SBR**, SNAZIV, SRUK)

Može se uočiti da svaka nova šema relacije sadrži samo attribute jednog entiteta.

Relacija RADSEK(**RMBR**, RIME, RLD, SBR, SNAZIV, SRUK) **posедује anomalije brisanja** koje se ogledaju u sledećem:

- Ako iz relacije RADSEK obrišemo sve radnike koji rade u nekom sektoru, tada ćemo obrisati i podatke o tom sektoru. Na taj način se iz baze podataka gubi informacija o sektorima koji trenutno nemaju raspoređene radnike. Ova pojava se naziva anomalijom brisanja, a posledica je ranije pomenute tranzitivne zavisnosti među atributima relacije.

- U slučaju dekompozicije relacije RADSEK na RADNIK i SEKTOR ne postoji takav problem. Sada se podaci o svim sektorima memorišu u posebnoj relaciji i to bez redundance.

Relacija RADSEK(**RMBR**, RIME, RLD, SBR, SNAZIV, *SRUK*) poseduje **anomalije modifikacije** koje se ogledaju u sledećem:

- Pri promeni vrednosti nekog od atributa entiteta SEKTOR (na primer, SRUK) potrebno je da se izmeni vrednost atributa SRUK u torkama svih radnika koji rade u tom sektoru. Dok proces izmene traje postoji nekonzistentnost u podacima. U našem primeru svi radnici koji rade u sektoru čiji atribut SRUK menjamo neće imati istu vrednost za SRUK. Ova pojava se naziva anomalijom modifikacije, a posledica je ranije pomenute tranzitivne zavisnosti među atributima relacije.
- Nakon dekompozicije relacija RADSEK na RADNIK i SEKTOR ova anomalija se gubi. Sada je potrebno da se izmeni vrednost atributa SRUK samo u tabeli SEKTOR

Razmatrane anomalije ažuriranja su posledica izvesnih **neželjenih struktura funkcionalnih zavisnosti** u šemi relacije. Eliminisanjem svih nepoželjnih struktura funkcionalnih zavisnosti eliminišu se i anomalije ažuriranja. Za šeme relacija koje su formirane pod ovim restrikcijama kaže se da su u **normalnoj formi**, a postupak kojim se šema relacije prevodi u neku normalnu formu naziva se **normalizacija**.

Pri normalizaciji se koristi tehnika **dekompozicije**. Dekompozicija šeme relacije nije proizvoljna i po pravilu se zasniva na određenoj strukturi funkcionalnih zavisnosti među atributima relacije. Dekompozicijom se svaka funkcionalna zavisnost među atributima šeme relacije izoluje u posebnu šemu relacije. O normalizaciji i normalnim formama možete da nađete više detalja u narednim odeljcima.

**Dekompozicija relacija** je proces zamene šeme relacije  $R(R;F)$  skupom šema relacija  $D=\{R_1(R_1;F_1), \dots, R_m(R_m;F_m)\}$  pri čemu moraju biti ispunjeni sledeći uslovi:

1. **Očuvanje atributa:** Nijedan atribut ne sme biti izgubljen, što znači da se svaki atribut iz  $R$  mora naći bar u jednoj šemi  $R_i$  dekompozicije  $D$
2. **Spoj-bez-gubitaka (neaditivni spoj)** - osigurava da se pri prirodnom spoju relacija dobijenih dekomponovanjem ne jave lažne torke. Svojstvo spoj-bez-gubitaka odnosi se na gubitak informacije, a ne na gubitak torki
3. **Očuvanje zavisnosti.** Dekompozicija obezbeđuje očuvanje zavisnosti ako su sve funkcionalne zavisnosti šeme  $(R;F)$  zadržane u  $D$

**Dekompozicija je dobra** ako je skup šema relacija  $D$  ekvivalentan šemi  $R$  i ako su isključene anomalije.

Dekompozicija  $D=\{\mathbf{R1},\mathbf{R2}\}$  šeme relacije  $\mathbf{R(R;F)}$  ima svojstvo *spoj-bez-gubitaka* u odnosu na skup funkcionalnih zavisnosti  $F$  u  $R$  ako i samo ako važi:

- (Uslov 1)  $R1 \cap R2 \rightarrow R1 \in F^+$  ili
- (Uslov 2)  $R1 \cap R2 \rightarrow R2 \in F^+$

gde je:

- $Z=R1 \cap R2$  skup zajedničkih atributa obe komponente
- (Uslov 1)  $Z$  je super ključ komponente  $R1$
- (Uslov 2)  $Z$  je super ključ komponente  $R2$

Praktično, ako je  $R1 \cap R2$  super ključ šeme relacije  $R1$  ili  $R2$ , tada binarna dekompozicija ima svojstvo spoj-bez-gubitaka. Za proveru da li je  $R1 \cap R2$  super ključ treba koristiti algoritam za nalaženje zatvarača skupa atributa

## 6.5 Normalne forme i normalizacija

Nakon projektovanja baze podataka sledi analiza projektovane šeme i donošenje odluke da li je projektovana šema dobra ili je treba dalje doterivati. Da bismo doneli takvu odluku potrebno je da shvatimo koje probleme, ako ih ima, može da proizvede ta šema. Kao vodič za to služe **normalne forme** koje nam obezbeđuju informaciju o tome koji se problemi neće javiti ako je šema u nekoj normalnoj formi. U teoriji relacionog modela podataka je predloženo više normalnih formi.

**Normalizacija** je postupak kojim se dobija skup relacija sa željenim osobinama tj ograničenjima definisanim normalnom formom, kako bi se izbegle anomalije ažuriranja. Proces normalizacije je formalna metoda koja se oslanja na funkcionalnim zavisnostima, odnosno na zavisnostima između ključeva (primarnog i kandidata) i ostalih atributa relacije.

Osnovne karakteristike funkcionalnih zavisnosti kod normalizacije koje se podrazumevaju su da:

- Definišu *jedan-na-jedan* relacije između atributa sa leve i atributa sa desne strane FZ;
- Postoje sve vreme;
- Netrivijalne su.

### Proces normalizacije

Normalizacija se izvršava u više koraka. Svaki korak odgovara jednoj normalnoj formi sa željenim osobinama definisanim normalnom formom. Svaka sledeća normalna forma je stroža i uvodi više restrikcija, samim tim i smanjuje mogućnost pojave anomalija.

Normalne forme, od onih sa najmanje restrikcija ka onima sa više restrikcija:



1. Prva normalna forma (1NF)
2. Druga normalna forma (2NF)
3. Treća normalna forma (3NF)
4. Boyce-Coddova normalna forma (BCNF)
5. Četvrta normalna forma (4NF)
6. Peta normalna forma (5NF) ili normalna forma projekcija-spoj (PJNF)
7. Normalna forma domen-ključ (DKNF)

Prve četiri se definišu na osnovu funkcionalnih zavisnosti (FZ) među atributima. 4NF, 5NF i DKNF nisu predmet ovog materijala i uvodnog kursa iz baza podataka. 4NF se zasniva na višeznačnim zavisnostima, dok se 5NF zasniva na zavisnosti spoja. DKNF uzima u obzir sve moguće zavisnosti i ograničenja.

### 6.5.1 Prva normalna forma

#### Definicija 1NF:

Šema relacije  $R(R;F)$  je u prvoj normalnoj formi (1NF) ako i samo ako domeni relacije  $R$  sadrže samo proste (atomične) vrednosti.

Proste vrednosti su vrednosti koje se ne mogu dalje deliti ili ako u konkretnoj situaciji nisu rastavljene na komponente.

U 1NF **nisu dozvoljeni** viševrednosni i kompozitni atributi i njihove kombinacije, tj. nisu dopuštene relacije u relacijama (tzv. ugneždene relacije) ili relacije kao atributi torki. Ovaj uslov je ukinut kod ugneždenog relacionog modela i kod objektno-relacionih sistema. Istorijski je 1NF uvedena da bi se onemogućila pojava viševrednosnih atributa, kompozitnih atributa i njihovih kombinacija.

#### Primer: 1NF

Nenormalizovana relacija (NNF), data na slici 6-2., se odnosi na klijente, koji iznajmljuju (rentiraju) objekte na nekoj adresi, plaćaju neku rentu vlasniku za koga se čuva ID i ime.

Ova relacija odnosno tabela sadrži više vrednosti atributa koje se ponavljaju – vrednosti atributa *ObjekatID*, zato što klijent može da rentira više objekata, a samim tim i vrednosti ostalih atributa o objektu, rentiranju i vlasniku. Očigledno se radi o loše projektovanoj relaciji koja je ovde navedena da bi se ilustrovala prva normalna forma.

KlijentID	ImeK	ObjekatID	AdresaO	rentStart	rentKraj	renta	VlasnikID	ImeVlasnika
CR76	Petar Perić	PG4	Cvetna, 8 Niš	1-Jul-00	31-Aug-01	150	CO40	Tina Tinkić
		PG16	Glavna, 10 Beograd	1-Sep-02	1-Sep-02	250	CO93	Toša Manić
CR56	Aca Stanić	PG4	Cvetna, 8 Niš	1-Sep-99	10-Jun-00	150	CO40	Tina Tinkić
		PG36	Kej, 12 Piroć	10-Oct-00	1-Dec-01	70	CO93	Toša Manić
		PG16	Glavna, 10 Beograd	1-Nov-02	1-Aug-03	250	CO93	Toša Manić

Slika 6-2. Nenormalizovana relacija

Test za 1NF treba da izvrši proveru da li postoje ovakve anomalije kao u primeru, odnosno da li su vrednosti svih atributa atomične. Uslov prve normalne forme definiše da kod relacije koja je u 1NF presek jedne kolone i jedne vrste sadrži samo jednu vrednost. Ukoliko nisu, treba izvršiti normalizaciju do 1NF.

**Test za 1NF:**

- Proveriti da li su svi atributi šeme relacije prosti (atomični).
- Ako jesu, tada je šema relacije u 1NF.
- Ako nisu, tada šema relacije nije u 1NF. Takvu šemu treba prevesti u 1NF, tj. treba izvršiti **1NF normalizaciju**.

Normalizacija do 1NF obezbeđuje da se uklone horizontalne redundance u podacima, odnosno da ne postoje dve kolone koje čuvaju istu informaciju, i ne postoji kolona koja čuva više od jedne vrednosti. Svaka vrsta u relaciji treba da bude jedinstvena, što se ostvaruje primarnim ključem.

**1NF normalizacijom** treba izdvojiti grupu koja se ponavlja u novu relaciju, koja sadrži taj atribut kao i primarni ključ originalne relacije.

Ako je relacija u 1NF obezbeđuje se:

- Lakše zadavanje upita/uređenja
- Skalabilnost
- Svaka vrsta se može identifikovati kod ažuriranja

Prva normalna forma je istorijski značajna, ali u praksi se često može zanemariti. Naime, ako se konceptualno projektovanje uradi kako treba, relacija je sigurno u 1NF pa o proveru i normalizaciji ne morate da vodite računa! Osim toga,

kod objektno-relacionih sistema uslov 1NF je ukinut (o tome u narednim kursevima na višim godinama studija).

### 6.5.2 Druga normalna forma (2NF)

Druga normalna forma (2NF) se bazira na konceptu **potpune funkcionalne zavisnosti**.

Podsetnik:

Funkcionalna zavisnost  $X \rightarrow Y$  je **potpuna funkcionalna zavisnost** ako izbacivanjem bilo kog atributa A iz X ova zavisnost više ne važi.

Funkcionalna zavisnost  $X \rightarrow Y$  je **parcijalna funkcionalna zavisnost** ako se neki atribut  $A \in X$  može izbaciti iz X i da zavisnost i dalje važi.

#### Definicija 2NF:

Šema relacije  $R(R;F)$  je u drugoj normalnoj formi (2NF) ako svaki atribut A u R ispunjava jedan od sledeća dva uslova:

- 1) javlja se u nekom ključu kandidatu
- 2) nije parcijalno zavisn od nekog ključa kandidata

Interpretacija definicije:

1. Šema relacije  $R(R;F)$  je u drugoj normalnoj formi (2NF) ako svaki njen neključni atribut **nema parcijalnu zavisnost** od nekog ključa šeme relacije
2. Šema relacije  $R(R;F)$  je u drugoj normalnoj formi (2NF) ako svaki njen neključni atribut potpuno funkcionalno zavisi od svakog ključa šeme relacije R

#### Test za 2NF

- Treba proveriti da li je **svaki neključni atribut potpuno** funkcionalno zavisi od svakog ključa kandidata.

Kod testiranja na 2NF mogu se iskoristiti specijalni slučajevi koji se odnose na ključeve kandidate od jednog atributa. Ako se u šemi relacije R svi ključevi kandidati sastoje samo od **jednog atributa** ili ako su svi atributi ključni, onda je šema relacije R u 2NF.

U praksi, 2NF relacija mora da bude i u 1NF, uz dodatni uslov da svaki neprimarni atribut potpuno zavisi od ključa. Drugom normalnom formom se povećava efikasnost čuvanja podataka i smanjuje ponavljanja podataka.

**Primer:** Test za 2NF

Data je šema relacije  $R(R;F)$ . Proveriti da li je šema relacije  $R(R;F)$  u 2NF?

$R=(A,B,C,D)$

$F=\{AB \rightarrow C, B \rightarrow DC, BC \rightarrow A\}$

2NF test

1. Naći sve ključeve šeme relacije (postupak po algoritmu za određivanje svih ključeva šeme relacije koji je dat u odeljku 6.3):

$K=\{B\}$

2. Kako jedini ključ  $K=\{B\}$  sadrži jedan atribut,  $R$  je u 2NF (specijalni slučaj, nema parcijalne zavisnosti).

3. KRAJ TESTA

**Primer:** Test za 2NF

Da li je data šema relacije  $R(R,F)$  u 2NF?

$R=(A,B,C,D,E)$

$F=\{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$

2NF test

1. Naći sve ključeve šeme relacije (postupak po algoritmu za određivanje svih ključeva šeme relacije koji je dat u odeljku 6.3):

$K1=A, K2=E, K3=BC, K4=CD$

2. Naći sve ključne attribute (atributi koji se deo ključeva,  $A$  kao deo  $K1$ ,  $E$  iz  $K2$ ,  $BC$  iz  $K3$  i  $CD$  iz  $K4$ ):

$KA = \{A, B, C, D, E\}$

3. Naći sve neključne attribute (svi iz šeme relacije koji nisu deo nekog ključa – u ovom slučaju ih nema):

$NKA = \{ \}$

4. Kako su svi atributi ključni,  $R$  je u 2NF (drugi specijalni slučaj).

5. KRAJ TESTA

**Primer:** Test za 2NF

Da li je data šema relacije  $R(R,F)$  u 2NF?

$$R=(S,A,I,P)$$

$$F=\{SI \rightarrow P, S \rightarrow A\}$$

2NF test

1. Naći sve ključeve šeme relacije

$$K=SI$$

2. Naći sve ključne attribute

$$KA = \{S,I\}$$

3. Naći sve neključne attribute

$$NKA = \{A,P\}$$

4. Proveriti da li A parcijalno zavisi od ključa SI

Kako je za **FZ**:  $S \rightarrow A$ ,  $S \subset K$ , to neključni atribut A parcijalno zavisi od ključa SI i šema relacije **nije u 2NF**.

KRAJ TESTA

Kod normalizacija do druge normalne forme treba eliminisati vertikalnu redundancu u podacima, odnosno da se ista vrednost ne sme ponavljati u nekoj vrsti. Podrazumeva se da sve kolone u vrsti moraju da se referenciraju na sve delove ključa (kompozitni ključevi).

**2NF Normalizacija** ima preduslov da relacija bude u 1NF, a nakon toga se vrši eliminacija parcijalnih zavisnosti: ako postoji parcijalna zavisnost, vršimo **dekompoziciju** relacije na osnovu te FZ. U posebnu relaciju izdvajamo parcijalno zavisne attribute i atribut koji ih određuje (koji postaje ključ nove relacije).

**Primer:** Normalizacija do 2NF

Pokazali smo da šema relacije data u prethodnom primeru:

$$R=(S,A,I,P)$$

$$F=\{SI \rightarrow P, S \rightarrow A\}$$

- nije u 2NF, zbog parcijalne zavisnosti  $S \rightarrow A$ .
- ključ ove relacije je SI.

Normalizacija dekompozicijom se vrši na osnovu FZ  $S \rightarrow A$ , tako što se od izdvajaju atributi S i A u novu relaciju R2 (S postaje ključ te relacije), a iz početne se izbacuju samo parcijalno zavisni atributi (u ovom slučaju samo A):

$R1=(S,I,P)$

$R2=(S,A)$

**Primer:** Normalizacija do 2NF

Šema relacije (pojava je data na slici 6-2):

KlijentRentira(KlijentID, ObjekatID, rentStart, rentKraj, ImeK,  
AdresaO, renta, VlasnikID, ImeVl )

ima sledeće FZ:

- |     |  |                         |
|-----|--|-------------------------|
| fz1 | KlijentID, ObjekatID → rentStart, rentKraj                                     | (Primarni Ključ)        |
| fz2 | KlijentID → ImeK   | (Parcijalna zavisnost)  |
| fz3 | ObjekatID → AdresaO, renta,<br>VlasnikID, ImeVl                                | (Parcijalna zavisnost)  |
| fz4 | VlasnikID → ImeVl  | (Tranzitivna zavisnost) |
| fz5 | KlijentID, rentStart → ObjekatID, AdresaO,<br>rentKraj, rent, VlasnikID, ImeVl | (Ključ kandidat)        |
| fz6 | ObjekatID, rentStart →<br>KlijentID, ImeK, rentKraj                            | (Ključ kandidat)        |

Eliminacija parcijalnih zavisnosti fz2 i fz3, podrazumeva kreiranje 3 nove relacije (slika 6-3.), dve na osnovu datih FZ i originalna iz koje se izbaciju parcijalno zavisni atributi:

Klijent (KlijentID, ImeK), po fz2.

VlasnikObjekta (ObjekatID, AdresaO, renta, VlasnikID, ImeVl), po fz3.

Rentira (KlijentID, ObjekatID, rentStart, rentKraj), ostatak originalne relacije

Klijent		Rentira			
KlijentID	ImeK	KlijentID	ObjekatID	rentStart	rentKraj
CR76	Petar Perić	CR76	PG4	1-Jul-00	31-Aug-01
CR56	Aca Stanić	CR76	PG16	1-Sep-02	1-Sep-02
		CR56	PG4	1-Sep-99	10-Jun-00
		CR56	PG36	10-Oct-00	1-Dec-01
		CR56	PG16	1-Nov-02	1-Aug-03

VlasnikObjekta				
ObjekatID	AdresaO	renta	VlasnikID	ImeVI
PG4	Cvetna 8, Niš	150	CO40	Tina Tinkić
PG16	Glavna 10, Beograd	250	CO93	Toša Manić
PG36	Kej 12, Pirot	70	CO93	Toša Manić

Slika 6-3. Relacije nakon normalizacije do 2NF

### 6.5.3 Treća normalna forma (3NF)

#### Definicija 3NF:

Šema relacije  $R(R;F)$  je u trećoj normalnoj formi (3NF) u odnosu na skup funkcionalnih zavisnosti  $F$  ako za svaku funkcionalnu zavisnost u  $F^+$  oblika  $X \rightarrow A$  ( $X \subseteq R$ ,  $A$  jedan atribut iz  $R$ ) važi :

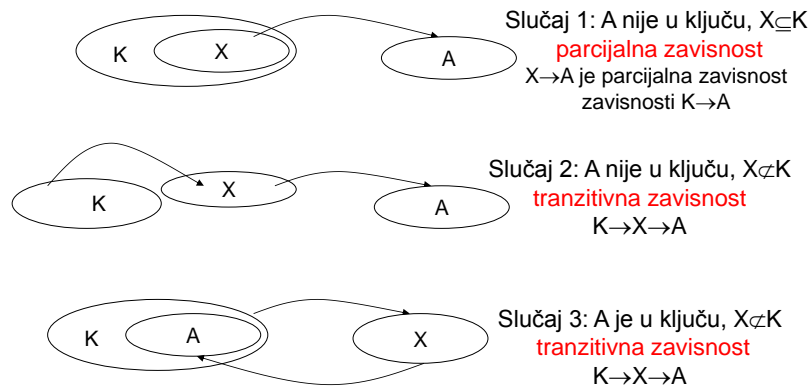
- 1)  $A \subseteq X$  ( $X \rightarrow A$  je trivijalna funkcionalna zavisnost), ili
- 2)  $X$  je super ključ šeme relacije  $R(R;F)$ , ili
- 3)  $A$  je ključni atribut šeme relacije  $R(R;F)$

Treća normalna forma definiše strože uslove u odnosu na drugu. Podrazumeva se da je svaka 3NF relacija takođe u 2NF, ali da postoje relacije koje su u 2NF, a nisu u 3NF.

Treći uslov 3NF definicije je zanimljiv za analizu pošto vodi ka anomalijama unosa, brisanja i modifikacije. Ako FZ  $X \rightarrow A$  ne zadovoljava ovaj uslov, razlozi mogu biti:

1.  $X$  je pravi podskup nekog ključa  $K$ :
  - $X \rightarrow A$  je parcijalna zavisnost i par  $(X,A)$  je redundantan
2.  $X$  nije pravi podskup nijednog ključa
  - $X \rightarrow A$  je tranzitivna zavisnost zbog lanca zavisnosti  $K \rightarrow X \rightarrow A$  ( ne možemo pridružiti  $X$  vrednost ključu  $K$ , a da ne pridružimo  $A$  vrednost  $X-u$ ).

Na slici 6-4. su prikazani slučajevi koji se odnose na varijante parcijalne i tranzitivne zavisnosti.



**Slika 6-4. Ilustracija parcijalnih i tranzitivnih zavisnosti**

Slučaj 1 se odnosi na parcijalnu zavisnost koja je, kako smo videli, važna za 2NF i 3NF. Za  $X \rightarrow A$ , ako A nije u ključu i X je podskup ključa K, očigledna je parcijalna zavisnost A od dela ključa. Slučaj 2 i 3 se odnose na dve varijante tranzitivne zavisnosti koja je važna za 3NF. Slučaj 2 pokazuje tranzitivnu zavisnost A od ključa K preko X (za koji postoji FZ  $X \rightarrow A$ ), koji nije deo ključa ali je određen ključem preko FZ  $K \rightarrow X$ . Slučaj 3 se odnosi na varijantu kada je A deo ključa, a tranzitivno zavisi od njega preko X.

### 3NF test

- Treba proveriti da li svaka zavisnost iz  $F^+$  zadovoljava uslov 1) ili 2) ili 3) iz 3NF definicije

Može se dokazati da je dovoljno proveriti samo funkcionalne zavisnosti iz  $F$  (ne iz  $F^+$ ) tako da ako nijedna od FZ iz  $F$  ne narušava 3NF ograničenja, tada to neće činiti nijedna zavisnost iz  $F^+$ .

Ako  $F$  sadrži FZ čije desne strane sadrže više od jednog atributa, tada primenom pravila dekompozicije skup  $F$  treba transformisati u skup  $F'$  gde sve FZ sadrže po jedan atribut sa desne strane.

Treća normalna forma obezbeđuje da sve kolone moraju da direktno zavise od primarnog ključa. U praksi, instanca šeme koja nije u 3NF sadrži redundantne informacije koje su posledica FZ. Da bi šema relacije bila u 3NF ona mora da bude u 2NF. Često, ako je relacija u 2NF, postoje dobre šanse da je i u 3NF. Ako ste dobro odradili konceptualno projektovanje u preslikavanje na relacioni model, relacije su uglavnom u 3NF. Normalizacijom do 3NF se obezbeđuje da nema redundanse u podacima odnosno nema anomalija koje su posledica tranzitivnih zavisnosti.



### Primer: 3NF normalizacija

**Klijent** (KlijentID, ImeK)

**Rentira** (KlijentID, ObjekatID, rentStart, rentKraj)

fz6      ObjekatID, rentStart  $\rightarrow$  KlijentID, rentKraj      (Ključ kandidat)

fz3      ObjekatID → AdresaO, renta,  
VlasnikID, ImeVl      (Primarni Ključ)

Tranzitivna zavisnost fz4 kod šeme relacije VlasnikObjekta narušava 3NF. Normalizacija dekompozicijom na osnovu fz4, VlasnikID  $\rightarrow$  ImeVl, uvodi novu šemu relacije **Vlasnik** (slika 6-5) na osnovu ove FZ koja sadrži sve attribute FZ (ključ je leva strana te FZ, u ovom slučaju VlasnikID), a iz originalne relacije VlasnikObjekta se izbacuju atributi koji su s leve strane posmatrane FZ (ImeVl):

VlasnikObjekta (ObjekatID, AdresaO, renta, VlasnikID) - ostatak originalne relacije, bez atributa ImeVI

133

**Klijent**

KlijentID	ImeK
CR76	Petar Perić
CR56	Aca Stanić

**Rentira**

KlijentID	ObjekatID	rentStart	rentKraj
CR76	PG4	1-Jul-00	31-Aug-01
CR76	PG16	1-Sep-02	1-Sep-02
CR56	PG4	1-Sep-99	10-Jun-00
CR56	PG36	10-Oct-00	1-Dec-01
CR56	PG16	1-Nov-02	1-Aug-03

**VlasnikObjekta**

ObjekatID	AdresaO	renta	VlasnikID
PG4	Cvetna 8, Niš	150	CO40
PG16	Glavna 10, Beograd	250	CO93
PG36	Kej 12, Pirot	70	CO93

**Vlasnik**

VlasnikID	ImeVI
CO40	Tina Tinkić
CO93	Toša Manić

**Slika 6-5. Relacije nakon normalizacije do 3NF**

### 6.5.4 Boyce-Codd-ova normalna forma (BCNF)

#### Definicija:

Šema relacije  $R(R;F)$  je u Boyce-Codd-ovoj normalnoj formi (BCNF) u odnosu na skup funkcionalnih zavisnosti  $F$  ako za svaku funkcionalnu zavisnost u  $F^+$  oblika  $X \rightarrow Y$  ( $X \subseteq R, Y \subseteq R$ ) važi:

- 1)  $Y \subseteq X$  ( $X \rightarrow Y$  je trivijalna funkcionalna zavisnost), ili
- 2)  $X$  je super ključ šeme relacije

U BCNF sve netrivialne FZ su oblika:

super ključ  $\rightarrow$  skup atributa

Iz definicije je očigledan odnos 3NF i BCNF - 3NF sadrži treći uslov koji ne postoji u BCNF, što predstavlja relaksaciju BCNF uslova. Svaka BCNF relacija je takođe u 3NF, a postoje relacije koje su u 3NF, a nisu u BCNF.

#### BCNF test

- Treba proveriti da li svaka zavisnost iz  $F^+$  zadovoljava ograničenje 1) ili 2) iz definicije BCNF

Može se pokazati da je dovoljno proveriti samo funkcionalne zavisnosti iz  $F$ , pa ako nijedna od FZ iz  $F$  ne narušava BCNF ograničenja, tada to neće činiti nijedna zavisnost iz  $F^+$ .

U praksi, prvi uslov je jednostavno proveriti, pa se metoda za BCNF svodi na proveru da li sve netrivialne FZ  $X \rightarrow Y$  iz  $F$  zadovoljavaju BCNF uslov 2), odnosno da li je **X super ključ**.

### Algoritam za BCNF test

**Ulaz:** Šema relacije  $R(R;F)$

**Izlaz:** TRUE ako je šema relacije  $R(R;F)$  u BCNF ili

FALSE ako nije

1. **Za svaku** netrivialnu FZ  $X \rightarrow Y$  iz F **do**
2.     Izračunati  $X_F^+$  // zatvarač skupa atributa X u odnosu na F
3.     **if** ( $X_F^+ \neq R$ ) **then return(FALSE)** // X nije super ključ, šema  
   // relacije nije u BCNF
4. **enddo**
5. **return(TRUE)** // šema relacije je u BCNF

### Primer: Test za BCNF

Proveriti da li je data šema relacije **R(R;F)** u BCNF, ako je:

$$R = \{A, B, C, D\}$$
$$F = \{AB \rightarrow C, BC \rightarrow A, B \rightarrow D\}$$

BCNF test:

Uslov (1): Sve FZ iz  $F$  su netrivialne i treba ih proveriti na Uslov (2).

Uslov (2), za svaku FZ:

1. Proveriti da li FZ  $AB \rightarrow C$  zadovoljava BCNF ograničenje
  - Naći  $(AB)^+_F$ 
    - $(AB)^+_F = (A, B, C, D)$
  - Kako je  $(AB)^+_F = R$ , zaključujemo da ova FZ **zadovoljava** BCNF ograničenje (AB je superključ relacije!)
2. Proveriti da li FZ  $BC \rightarrow A$  zadovoljava BCNF ograničenje
  - Naći  $(BC)^+_F$ 
    - $(BC)^+_F = (B, C, A, D)$
  - Kako je  $(BC)^+_F = R$  zaključujemo da ova FZ **zadovoljava** BCNF ograničenje
3. Proveriti da li FZ  $B \rightarrow D$  zadovoljava BCNF ograničenje

- Naći  $(B)_F^+$ 
  - $(B)_F^+ = (B, D)$
- Kako je  $(B)_F^+ \neq R$  zaključujemo da ova FZ **NE zadovoljava** BCNF ograničenje, ova šema **nije u BCNF** (zbog  $B \rightarrow D$ )

□ KRAJ TESTIRANJA

**Primer:** Test za BCNF

Da li je data šema relacije  $R(R;F)$  u BCNF?

$R = (A, B, C, D)$

$F = \{AB \rightarrow C, B \rightarrow DC, BC \rightarrow A\}$

BCNF test

1. Proveriti da li FZ  $AB \rightarrow C$  zadovoljava BCNF uslov 2)

$(AB)^+ = (A, B, C, D)$

Kako je  $(AB)^+ = R$ , to je AB super ključ, zadovoljava uslov 2)

2. Proveriti da li FZ  $B \rightarrow D$  zadovoljava BCNF uslove

$(B)^+ = (A, B, C, D)$

Kako je  $(B)^+ = R$ , to je B super ključ, zadovoljava uslov 2)

3. Proveriti da li FZ  $B \rightarrow C$  zadovoljava BCNF uslove

$(B)^+ = (A, B, C, D)$

Kako je  $(B)^+ = R$ , to je B super ključ, zadovoljava uslov 2)

4. Proveriti da li FZ  $BC \rightarrow A$  zadovoljava BCNF uslove

$(BC)^+ = (A, B, C, D)$

Kako je  $(BC)^+ = R$ , to je BC super ključ, zadovoljava 2)

5. **R je u BCNF, KRAJ TESTA**

**Primer:** Test za BCNF

Da li je data šema relacije  $R(R;F)$  u BCNF?

$R = (A, B, C, D, E)$

$F = \{A \rightarrow B, BC \rightarrow E, ED \rightarrow A\}$

BCNF test: Za individualni rad

(Odgovor: nije u BCNF)

**Primer:** Test za BCNF

Da li je data šema relacije  $R(R;F)$  u BCNF?

$R=(C,S,Z)$

$F= \{CS \rightarrow Z, Z \rightarrow C\}$

BCNF test: Za individualni rad

(Odgovor: nije u BCNF)

Ukoliko je šema relacije u BCNF, njena instanca ne sadrži redundantne informacije koje su posledica FZ. Anomalije modifikacije i brisanja se ne javljaju u BCNF šemama. Ipak, relacije sa više od jednog ključa još uvek mogu da imaju anomalije unosa. Da bi se izbegao ovaj problem jedan od ključeva se označava za primarni i zabranjuje se da njegovi atributi imaju NULL vrednosti.

Kod praktičnog rada, kao posledica definicije može da se iskoristi činjenica da je šema relacije u BCNF ako svaki skup atributa koji određuje neki drugi skup atributa je ključ kandidat.

Razlika između 3NF i BCNF je u tome što za zavisnost  $A \rightarrow B$ :

- 3NF dozvoljava da B može da bude primarni atribut, dok A nije kandidat za ključ
- BCNF zahteva da takva zavisnost može da postoji samo ako je A kandidat za ključ

**BCNF normalizacija** se vrši na isti način – prepoznaju se FZ koje su razlog da relacija nije u BCNF i na osnovu njih se vrši dekompozicija. Kreira se nova relacija za svaku takvu FZ i iz početne se izbacе zavisni atributi iz tih FZ.

**Primer:** Normalizacija do BCNF

Neka je data relacija *KlijentIntervju* (slika 6-6.) i skup funkcionalnih zavisnosti:

*KlijentIntervju* (klijentID, intervjuDatum, intervjuVreme, zaposlID, prostorijaBr)

fz1: klijentID, intervjuDatum  $\rightarrow$  intervjuVreme, zaposlID, prostorijaBr

fz2: zaposlID, intervjuDatum, intervjuVreme  $\rightarrow$  klijentID

fz3: prostorijaBr, intervjuDatum, intervjuVreme  $\rightarrow$  klijentID, zaposlID

fz4: zaposlID, intervjuDatum  $\rightarrow$  prostorijaBr

**KlijentIntervju**

<b>klijentID</b>	<b>intervjuDatum</b>	<b>intervjuVreme</b>	<b>zaposlID</b>	<b>prostorijaBr</b>
CR76	13-May-02	10.30	SG5	G101
CR76	13-May-02	12.00	SG5	G101
CR74	13-May-02	12.00	SG37	G102
CR56	1-Jul-02	10.30	SG5	G102

**Slika 6-6. Relacija KlijentIntervju**

Funkcionalna zavisnost fz1 se odnosi na primarni ključ relacije, dok fz2 i fz3 definišu kandidate za ključ. Problematična FZ je fz4, pošto leva strana nije ključ kandidat i posledica su potencijalne anomalije ažuriranja (na primer, ako treba ažurirati dve torke tako da se menja vrednost *prostorijaBr* za *zaposlID* SG5 i 13-May-02).

Da bi izvršili normalizaciju, moramo da eliminišemo FZ koja narušava BCNF (u ovom slučaju fz4) tako što ćemo da uradimo dekompoziciju po toj FZ na dve relacije (slika 6-7) **Intervju** (originalna relacije bez zavisnih atributa koji su s desne strane posmatrane FZ, u ovom slučaju *prostorijaBR*) i **ZaposlProstorija** (nova relacija na osnovu fz4):

Intervju (klijentID, intervjuDatum, intervjuVreme, zaposlID)

ZaposlProstorija (zaposlID, intervjuDatum, prostorijaBr)

**Intervju**

<b>klijentID</b>	<b>intervjuDatum</b>	<b>intervjuVreme</b>	<b>zaposlID</b>
CR76	13-May-02	10.30	SG5
CR76	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-Jul-02	10.30	SG5

**ZaposlProstorija**

<b>zaposlID</b>	<b>intervjuDatum</b>	<b>prostorijaBr</b>
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	1-Jul-02	G102

**Slika 6-7. Relacija KlijentIntervju nakon normalizacije****6.5.5 Denormalizacija**

Denormalizacija je postupak koji podrazumeva spajanje relacija u jednu kako bi se povećala efikasnost nekih upita/pretraga. U praksi se koristi ako se proceni da je dobitak kod pretrage veći od problema sa anomalijama. Ipak, u tom slučaju se

mora dodatno brinuti o integritetu podataka i anomalijama, kao posledici denormalizacije. Cilj je da se dođe do logički i dalje korektnih, a tehnički prihvatljivih rešenja.

Preporuka: izvršiti najpre normalizaciju **kako** treba, pa tek onda denormalizaciju **ako** treba. Koristiti jedino ako ne postoji drugi način optimizacije izvršenja upita. Pokušajte sa *temp* tabelama, UNION-ma, VIEW-ma, ugnježenim upitima i sl.





## 7 UPITNI JEZIK SQL

**Structured Query Language (SQL)** predstavlja programski jezik koji je projektovan za potrebe pretraživanja i upravljanja podacima u Sistemima za upravljanje relacionim bazama podataka (Relational Database Management Systems – RDBMS), za kreiranje i modifikacija šema relacione baze podataka i za kontrolu pristupa objektima baze podataka.

Prva verzija programskog jezika SQL razvijena je od strane kompanije IBM krajem 1970-ih godina. Ova verzija, inicijalno nazvana SEQUEL, je razvijena za potrebe manipulacije podacima System R sistemu za upravljanje relacionim bazama podataka.

SQL je formalno standardizovan 1986 godine, od strane American National Standards Institute (ANSI), pod nazivom SQL-86. Ovaj standard je kasnije prihvaćen id od strane International Organization for Standardization (ISO). Svi SQL standardi koji su naknadno usvojeni predstavljaju zajedničke standarde organizacija ANSI i ISO.

SQL je programski jezik razvijen sa specifičnom namenom . Njegova osnovna karakteristika je da se radi o deklarativnom jeziku za upravljanje i manipulacijom podacima u relacionim bazama podataka. SQL omogućava pribavljanje, dodavanje, ažuriranje i brisanje podataka u relacionoj bazi podataka. Većina proizvođača RDBMS-a proširuje SQL standard dodajući proceduralne elemente, kontrolne strukture, korisnički definisane tipove podataka i druge elemente. SQL standard SQL-99 mnoga od ovih proširenja su formalno prihvaćena kao delovi SQL jezika u vidu SQL Persistent Stored Modules (SQL/PSM) (Tabela 1).

Većina proizvođača relacionih DBMS-ova obezbeđuje alat za kreiranje i izvršavanje naredbi SQL jezika. Ovaj alat može biti običan Command-line Interface (SQL/CLI) ili alat sa bogatim grafičkim interfejsom. Takođe, u većini slučajeva, je obezbeđena programksa podrška koja omogućava da se SQL naredbe kreiraju i koriste iz drugih programskih jezika.

Sve naredbe SQL jezika se mogu podeliti u dve velike grupe:

1. Data Definition Language (DDL) – jezik koji se koristi za definisanje strukture relacione baze podataka

2. Data Manipulation Language (DML) – jezik za pribavljanje i ažuriranje podataka u relacionoj bazi podataka.

U nastavku je najpre dat kratak opis baze podataka PREDUZEĆE koja je korišćena u primerima.

**Tabela 1 : Proširenja SQL jezika koja su postala deo standarda**

Proizvođač	Ime	Puno ime
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase/Firebird	PSQL	Procedural SQL
IBM	SQL PL	SQL Procedural Language (implements SQL/PSM)
Microsoft/Sybase	T-SQL	Transact-SQL
MySQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
Oracle	PL/SQL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language (bazira se na Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules

## 7.1 Baza podataka PREDUZEĆE

Baza podataka PREDUZEĆE je jedan od uobičajenih primera koji se koristi u literaturi o bazama podataka. ER dijagram ove baze podataka je dat u poglavlju 3, na slici 3-25, a šema relacione baze podataka dobijena preslikavanjem sa ER modela na relacioni, na slici 4-4. Na slici 7-1. prikazana je pojava ove baze podataka koja prikazuje podatke korišćene u primerima nadalje.

Tabela **RADNIK** čuva podatke o radnicima preduzeća. Za svakog radnika se pamti matični broj, ime, srednje slovo, prezime, datum rođenja, pol, plata, adresa, matični broj neposrednog rukovodioca i broj sektora u kome radnik radi. Za primarni ključ je izabran matični broj radnika odnosno kolona **MATBR**. Tabela ima dva strana ključa: **MATBROJS** i **BRSEK**. Kolona **MATBROJS** referencira kolonu **MatBr** u istoj tabeli jer rukovodilac je jedan od radnika (rekurzivna veza). Kolona **BRSEK** referencira kolonu **SBROJ** u tabeli **SEKTOR**.

Preduzeće je podeljeno na veći broj sektora. U tabeli **SEKTOR** se pamte sledeći podaci o sektorima: broj sektora, naziv sektora, matični broj šefa sektora i datum kada je šef sektora postavljen na svoju dužnost. Za primarni ključ je izabran broj sektora odnosno kolona **SBROJ**. Jedini strani ključ je kolona **MATBR**.

(predstavlja matični broj rukovodioca) koja referencira kolonu **MATBR** u tabeli **RADNIK** (šef sektora je jedan od radnika u preduzeću).

**RADNIK**

IME	SSLOVO	PREZIME	MATBR	DATROD	POL	PLATA	ADRESA	MATERS	BRSEK
Marko	J	Petrović	123456789	09-JAN-65	M	30000	Obilićev Venac 11	333445555	5
Sima	F	Todorović	333445555	08-DEC-55	M	40000	Dušanova 32	888665555	5
Valentina	D	Kovačević	999887777	19-JAN-68	Ž	25000	Knjeginje Ljubice 12/34	987654321	4
Aleksandra	S	Petrović	987654321	20-JUN-41	Ž	43000	Knjaževačka 11	888665555	4
Velibor	T	Jovanović	666884444	15-SEP-62	M	36000	Knjaževačka 132/12	333445555	5
Jelena	P	Janković	453453453	31-JUL-72	Ž	25000	Vizantijski bulevar 123/12	333445555	5
Stanko	L	Manojlović	987987987	29-MAR-69	M	25000	Nemanjina 23	987654321	4
Jovan	S	Obradović	888665555	10-NOV-47	M	55000	Nikole Kopernika 11	(null)	1

**SEKTOR**

NAZIV	SBROJ	MATBR	DATPOST
Razvoj	5	333445555	22-MAY-88
Administracija	4	987654321	01-JAN-85
Uprava	1	888665555	19-JUN-81

**PROJEKAT**

NAZIV	LOKPR	BROJPR	BRS
ProizvodX	Niš	1	5
ProizvodY	Pirot	2	5
ProizvodZ	Niš	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

**CLAN\_PORODICE**

MATBRAD	IME	POL	DATROD
333445555	Nevena	Ž	05-APR-86
333445555	Teodora	Ž	03-MAY-58
123456789	Ana	Ž	30-DEC-88
123456789	Jelena	Ž	05-MAY-66
333445555	Milan	M	25-OCT-83
987654321	Veselin	M	28-FEB-42
123456789	Mihajlo	M	04-JAN-88

**RADI\_NA**

MBR	BRPR	SATI
888665555	20	10
123456789	1	32
123456789	2	7
666884444	3	40
453453453	1	20
453453453	2	20
333445555	2	10
333445555	3	10
333445555	10	10
333445555	20	10
999887777	30	30
999887777	10	10
987987987	10	25
987987987	20	5
987654321	30	20
987654321	20	15

**LOK\_SEK**

BRS	LOKACIJA
1	Niš
4	Leskovac
4	Niš
5	Niš
5	Pirot

**Slika 7-1. Relacija KlijentIntervju nakon normalizacije**

Tabela **PROJEKAT** čuva podatke o projektima koji su aktivni unutar preduzeća. Za svaki projekat se pamti: broj projekta, naziv projekta, lokacija projekta i broj sektora koji je zadužen za projekat. Za primarni ključ je izabran broj projekta odnosno kolona **BROJPR**. Jedini strani ključ je kolona **BRS** (broj sektora koji je zadužen za projekat) koja referencira kolonu **SBROJ** u tabeli **SEKTOR**.

Tabela **CLAN\_PORODICE** čuva sledeće podatke o članovima porodica radnika: matični broj radnika, ime člana porodice, pol, srodstvo sa radnikom i datum rođenja. Primarni ključ je kompozitni i čine ga kolone **MATBRAD** i **IME**. Strani ključ je kolona **MATBRAD** (matični broj radnika za koga je član porodice vezan) koja referencira kolonu **MATBR** u tabeli **RADNIK**.

Tabela **LOK\_SEK** čuva podatke o lokacijama na kojima se nalazi određeni sektor: broj sektora i naziv lokacije. Primarni ključ je kompozitni i čine ga kolone

**BRS** i **LOKACIJA**. Strani ključ je kolona **BRS** (broj sektora za koji je lokacija vezana) koja referencira kolonu **SBROJ** u tabeli **SEKTOR**.

Tabela **RADI\_NA** predstavlja evidenciju angažovanja radnika na različitim projektima. Za svako angažovanje se pamti: matični broj angažovanog radnika, broj projekta na kome je angažovan i broj radnih sati nedeljno koliko je angažovan na projektu. Primarni ključ je kompozitni i čine ga kolone **MATBRRAD** i **BRPR**. Tabela ima dva strana ključa: **MATBRRAD** i **BRPR**. Kolona **MATBRRAD** (predstavlja matični broj radnika na koga se odnosi angažovanje) referencira kolonu **MATBR** u tabeli **RADNIK**, a kolona **BRPR** (broj projekta na kome je radnik angažovan) referencira kolonu **BROJPR** u tabeli **PROJEKAT**.

## 7.2 SQL naredbe za kreiranje podataka

SQL DDL naredbe se koriste za kreiranje, izmenu i brisanje same relacije baze podataka kao i objekata koji čine relacionu bazu podataka. Centralni objekat svake relacije baze podataka jeste tabela. Za kreiranje tabele koristi se SQL naredba **CREATE TABLE**.

```
CREATE TABLE <ime_tabele>
    (<lista_deklaracija_kolona>
    [, <lista_deklaracija_ograničenja_tabele>]);
<ime_kolone><tip_podatka>[( <dužina>)] [<ograničenje_kolone>],
```

Deklaracija kolone sadrži ime kolone, tip podatka kolone, opciono dužinu (ako je neophodna, u zavisnosti od tipa podatka) i opciono, ograničenja za kolonu. Deklaracije kolona se razdvajaju zapetama. U odeljku **Tipovi podataka** navedeni su tipovi podataka koje možete koristiti. Ograničenja za kolonu su opciona i mogu da sadrže specifikaciju predefinisane vrednosti i niz specifikacija ograničenja, što je detaljnije opisano u odeljku **Ograničenja kolona**.

Nakon deklaracije svih kolona navode se ograničenja koja važe za celu tabelu. Specifikacije ograničenja za tabelu su navedena u odeljku **Ograničenja tabele**.

Navedena sintaksa naredbe **CREATE TABLE** nije kompletna. SQL standard obezbeđuju korišćenje velikog broja dodatnih klauzula koje korisnicima omogućavaju preciznu kontrolu procesa kreiranja tabele. Osim toga proizvođači RDBMS-ova proširuju **CREATE TABLE** naredbu klauzulama koje su posledica specifičnih osobina njihovih proizvoda.

### 7.2.1 Tipovi podataka

U Tabeli 2 su dati generički tipovi podataka koje definiše SQL standard. Ne podržavaju svi RDBMS-ovi tipove podataka definisane u SQL standardu.

**Tabela 2: Generički SQL tipovi podataka**

Tip podataka	Opis
integer	Celobrojna vrednost
smallint	Celobrojna vrednost
numeric(p,s)	Argument p definiše ukupan broj cifara broja dok argument s definiše broj cifara desno od decimalnog zareza. Npr. numeric(6, 2) je broj koji ima 4 cifre ispred decimalnog zareza i 2 cifre iza decimalnog zareza.
decimal(p,s)	Argument p definiše ukupan broj cifara broja dok argument s definiše broj cifara desno od decimalnog zareza. Npr. numeric(6, 2) je broj koji ima 4 cifre ispred decimalnog zareza i 2 cifre iza decimalnog zareza.
real	Broj u pokrenom zarezu jednostruke preciznosti
double precision	Broj u pokretnom zarezu dvostruke preciznosti
float(p)	Argument p definiše preciznost broja.
char(x)	Argument x predstavlja maksimalan broj karaktera koji kolona može da prihvati. Ovaj tip definiše tekstualne podatke fiksne dužine, odnosno podataka se dopunjuje blanko znacima sa desne strane kako bi se obezbedila specificirana dužina.
varchar2(x)	Argument x predstavlja maksimalan broj karaktera koji kolona može da prihvati. Ovaj tip definiše tekstualne podatke promenljive dužine (nema dopunjavanja blanko znacima).
bit(x)	Argument x definiše maksimalan broj bitova koje podataka može da prihvati. Podaci su fiksne dužine.
bit varying(x)	Argument x definiše maksimalan broj bitova koje podataka može da prihvati. Podaci su promenljive dužine.
date	Datum.
time	Vreme.
timestamp	Datum i vreme.
time with time zone	time koji uključuje i informaciju o vremenskoj zoni.
timestamp with time zone	timestamp koji uključuje i informaciju o vremenskoj zoni..
Interval	Vremenski interval.

U nastavku su dati tipovi podataka koje na osnovu SQL standarda implementira ORACLE RDBMS. Dobar deo ORACLE tipova podataka se poklapa sa generičkim SQL tipovima ali postoje i odstupanja.

### **CHAR(n)**

Ovaj tip definiše znakovne podatke fiksne dužine. Obavezna se mora specificirati dužina kolone (u bajtovima ili karakterima) od 1 do 2000 bajtova. Podrazumevana dužina je 1 bajt. Dužina se podrazumevano zadaje kao broj bajtova.

Ukoliko je podatak u koloni kraći od specificirane dužine on se dopunjuje blanko znacima sa desne strane. Ukoliko je podataka veći od specificirane dužine Oracle

DBMS vraća grešku. Prilikom poređenja CHAR vrednosti uzimaju se u obzir i dodati blanko znaci.

### **VARCHAR(n) i VARCHAR2(n)**

Ovaj tip podataka definiše znakovne podatke promenljive dužine. Obavezna se mora specificirati maksimalna dužina kolone (u bajtovima ili karakterima) od 1 do 4000 bajtova. Podrazumevana dužina je 1 bajt. Maksimalna dužina se podrazumevano zadaje kao broj bajtova.

Ukoliko je podataka veći od specificirane maksimalne dužine Oracle DBMS vraća grešku. U koloni se pamti podatak bez dopunjavanja blanko znacima.

VARCHAR i VARCHAR2 su sinonimi (preporučuje se korišćenje VARCHAR2 tipa da bi se izbegli problemi sa budućom kompatibilnošću).

**Napomena:** Kod različitih kodnih rasporeda za predstavljanje karaktera se koristi različiti broj bajtova. Na primer, kod ASCII kodnog raporeda za predstavljanje svakog karaktera se koristi tačno jedan bajt. Za razliku od toga, kod UTF8 kodnog rasporeda za predstavljanje karaktera se može koristiti više od jednog bajta (maksimalno četiri). Za svaku ORACLE bazu podataka prilikom kreiranja je definisan podrazumevani kodni raspored koji će se koristiti za predstavljanje znakovnih podataka u toj bazi podataka.

### **NCHAR(n) i NVARCHAR(n)**

Ova dva tipa podataka omogućavaju memorisanje znakovnih podataka korišćenjem UTF kodnog rasporeda bez obzira šta je izabrano kao podrazumevani kodni raspored prilikom kreiranja ORACLE baze podataka. Ova dva tipa podataka omogućavaju memorisanje znakovnih podataka korišćenjem UTF8 (ili opciono AL16UTF16) kodnog rasporeda.

NCHAR pamti UTF8 znakovne podatke fiksne dužine, dok NVARCHAR pamti UTF8 znakovne podatek promenljive dužine. Dužina se uvek specificira kao broj karaktera. Maksimalna dužina za NCHAR kolone je 2000 karaktera a za NVARCHAR kolone je 4000 karaktera.

### **NUMBER(p, s)**

Tip podataka koji se koristi za predstavljanje brojeva i u fiksnom i u pokretno zarezu. Korišćenjem ovog tipa mogu se predstaviti sledeće vrednosti:

- pozitivni brojevi  $1 \times 10^{-130}$  do  $9.99...9 \times 10^{125}$  sa 38 značajnih cifara
- negativni brojevi  $-1 \times 10^{-130}$  do  $-9.99...9 \times 10^{125}$  sa 38 značajnih cifara
- nula
- pozitivna i negativna beskonačna vrednost

Argument *p* (*precision*) definiše ukupan broj cifara broja dok argument *s* (*scale*) definiše broj cifara desno od decimalnog zareza. Kombinacija vrednosti (\*, s) specificira da je maksimalan broj cifara 38 dok se broj cifara desno od decimalnog zareza održava u skladu sa navedenom vrednošću. Negativna vrednost za *scale* nalaže Oracle-u da izvrši zaokruživanje numeričke vrednosti na specificirani broj pozicija levo od decimalne tačke.

### **INTEGER, INT, SMALL INT**

Tipovi podataka koji se koriste za memorisanje celobrojnih podataka. Umesto ovih tipova preporučuje se korišćenje tipa **NUMBER**.

### **FLOAT, REAL, BINARY\_FLOAT, BINARY\_REAL**

Tipovi podataka koji se koriste memorisanje razlomljenih brojnih vrednosti. Umesto ovih tipova preporučuje se korišćenje tipa **NUMBER**.

### **LOB tipovi podataka**

Ovi tipovi podataka se koriste za pamćenje velikih blokova nestruktuiranih podataka:

- **BLOB** se koristi za pamćenje binarnih podataka
- **CLOB** i **NCLOB** se koriste za pamćenje znakovnih podataka
- **BFILE** omogućava pamćenje podataka van baze podataka u file sistemu

Maksimalna količina podataka koja se može smestiti u kolonu ovog tipa je 128TB.

### **DATE, TIMESTAMP**

Ovi tipovi podataka se koriste za memorisanje informacija o datumu i vremenu. Pamti se informacija o godini, mesecu, danu, satu, minuti i sekundama:

- standardni format za datume je DD-MMM-YYYY (Primer: '13-NOV-92')
- standardni format za pamćenje vremena je HH:MI:SS. (Primer: '13-NOV-92 11:12:32')

Ukoliko se navede samo datum podrazumevani podatak za vreme je 00:00:00 odnosno ponoć. Ukoliko se navede samo vreme podrazumevani datum je prvi dan tekućeg meseca.

**TIMESTAMP** može da uključuje i delove sekundi i informaciju o vremenskoj zoni.

Za razliku od SQL standarda ORACLE DBMS ne predviđa postojanje posebnih tipova za memorisanje vremena i datuma.

**Primer:** SQL DDL naredbe za kreiranje tabela RADNIK i SEKTOR

```
CREATE TABLE RADNIK
(
    LIME VARCHAR2(15),
    SSLOVO VARCHAR2(2),
    PREZIME VARCHAR2(15),
    MATBR NUMBER(10),
    DATRODJ DATE,
    POL CHAR(1),
    PLATA NUMBER(8, 2),
    ADRESA VARCHAR2(30),
    MATBRS NUMBER(10),
    BRSEK NUMBER(2)
);

CREATE TABLE SEKTOR
(
    NAZIV VARCHAR2(15),
    SBROJ NUMBER(2),
    MATBRR NUMBER(10),
    DATPOST DATE
);
```

### 7.2.2 Ograničenja kolone

Ograničenja koja možete da definišete za kolonu prilikom kreiranja tabela su:

- **NULL** ili **NOT NULL** –definiše da kolona može ili ne može imati NULL vrednosti.
- **UNIQUE** – definiše da kolona ima jedinstvene vrednosti (kandidati za ključeve).
- **PRIMARY KEY** – definiše da kolona predstavlja primarni ključ tabele (može da se primeni na samo jednu kolonu u tabeli).
- **CHECK** –definiše ograničenje za proveru vrednosti kolone (koristi se kod upisa ili ažuriranja vrednosti).
- **DEFAULT** – definiše podrazumevanu vrednost za kolonu (kolona uzima ovu vrednost, ako vrednost kolone nije navedena).



- **REFERENCES** –definiše da kolona predstavlja spoljni ključ tabele.

Za vrednost kolone se mogu specificirati ograničenja **NULL** ili **NOT NULL** čime se dozvoljava ili zabranjuje **NULL** vrednost kolone. Strože ograničenje je **UNIQUE**, koje ne dozvoljava ponavljanje vrednosti u koloni.

Za kolonu podrazumevano ograničenje je **NULL** vrednost. To znači da navodimo samo ograničenje **NOT NULL**, ako je definisano za konkretnu kolonu.

Kod navođenja **PRIMARY KEY** podrazumeva se **NOT NULL** ograničenje za tu kolonu, tako da ga ne treba posebno navoditi.

**Napomena:** Prilikom navođenja **PRIMARY KEY** ograničenja, podrazumevaju se **UNIQUE** i **NOT NULL** ograničenje za tu kolonu, tako da se ne moraju posebno navoditi.

Pri kreiranju tabele u kojoj je atribut A primarni ključ, a atribut B spoljni ključ možete koristiti oblik naredbe sledeći **CREATE TABLE**.

```
CREATE TABLE <imeTabele>
(
  A <tip_podatka> PRIMARY KEY,
  B <tip_podatka> REFERENCES <ime_ref_tabele>(<ime_ref_kolone>),
  ostali atributi
);
```

Uočite da kod deklaracije spoljnog ključa tabele treba navesti iza ključne reči **REFERENCES** ime referencirane tabele i opcionalno, u maloj zagradi, ime referencirane kolone u toj tabeli. ORACLE SQL očekuje da je referencirana kolona primarni ključ (kolona deklarirana sa **PRIMARY KEY**) u referenciranoj tabeli ili da je nad njom podignuto ograničenje **UNIQUE**.

Na taj način atribut iza koga stoji klauzula **REFERENCES** definisan je kao spoljni ključ u odnosu na primarni ključ tabele čije je ime navedeno iza klauzule **REFERENCES**.

Primer: SQL DDL naredba za kreiranje tabele PROJEKAT

```
CREATE TABLE PROJEKAT
(
  NAZIV VARCHAR(25) NOT NULL,
  LOKPR VARCHAR(15) DEFAULT 'Niš',
  BROJPR NUMBER(3) PRIMARY KEY,
  BRS NUMBER(2) NOT NULL REFERENCES SEKTOR(SBROJ)
);
```

### 7.2.3 Ograničenja tabele

Za definisanje ograničenja koja važe za tabelu u celini možete koristiti:

- **PRIMARY KEY** – definiše koja kolona ili koje kolone čine primarni ključ tabele.
- **UNIQUE** – definiše koja kolona ili koje kolone imaju jedinstvene vrednosti (kandidati za ključeve).
- **FOREIGN KEY** – definiše koja kolona ili koje kolone čine spoljni ključ tabele.
- **CHECK** – definiše ograničenja vrednosti kolone ili kolona koje DBMS proverava kod upisa ili ažuriranja vrednosti te ili tih kolona.

Za kreiranje tabele u kojoj je skup atributa predstavlja primarni ključ možete koristiti oblik sledeći oblik CREATE TABLE naredbe:

```
CREATE TABLE < ime_tabele >
(
    <atributi i njihovi tipovi podataka>,
    CONSTRAINT <ime ograničenja>
        PRIMARY KEY (<lista_atributa_koji_čine_primarni_ključ>)
);
```

Podrazumeva se da su atributi koji čine primarni ključ prethodno definisani u sekciji < atributi i njihovi tipovi podataka >.

**Primer:** DDL naredba za kreiranje tabele RADI\_NA

```
CREATE TABLE RADI_NA
(
    MBR NUMBER(10),
    BRPR NUMBER(3),
    SATI NUMBER(3) NOT NULL CHECK (SATI > 0),
    CONSTRAINT RADI_NA_PK PRIMARY KEY (MBR, BRPR)
);
```

**Napomena:** Prilikom kreiranja kompozitnog primarnog ključa nije moguće iskoristiti ograničenje kolone. **Za slučaj kompozitnog primarnog ključa mora se iskoristiti ograničenje tabele!!!**

**Primer:** Primer **POGREŠNO** napisane CREATE TABLE naredbe

Kod ove naredbe se kreira kompozitni primarni ključ. Ovako napisanu CREATE TABEL naredbu ORACLE SQL tumači kao pokušaj kreiranja tabele koja ima dva primarna ključa. Pošto to nije moguće, dobićete odgovarajuću poruku o grešci.

```
CREATE TABLE RADI_NA
(
    MBR INTEGER PRIMARY KEY,
    BRPR INTEGER PRIMARY KEY,
    SATI NUMBER(2,0)
);
```

Za definisanje ograničenja stranog ključa u tabeli u kojoj skup atributa čini strani ključ može se koristiti sledeći oblik **CREATE TABLE** naredbe:

```
CREATE TABLE <naziv_tabele>
(
    <atributi i njihovi tipovi podataka>,
    FOREIGN KEY (<lista_atributa_koji_čine_spoljni_ključ>)
    REFERENCES (<lista_referenciranih_atributa>)
    [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
    [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
    CHECK (<uslovni_izraz>)
);
```

Podrazumeva se da su atributi koji čine strani ključ prethodno definisani u sekciji <atributi i njihovi tipovi podataka>.

Kod deklaracije spoljnog ključa tabele iza ključne reči **REFERENCES** navodi se ime referencirane tabele i opciono, u maloj zagradi, ime referencirane kolone u toj tabeli. ORACLE SQL očekuje da referencirane kolone predstavljaju primarni ključ u referenciranoj tabeli ili je nad njima definisano ograničenje **UNIQUE**. Ukoliko je izostavljena lista referenciranih kolona, ORACLE SQL automatski podrazumeva primarni ključ referencirane tabele.

U deklaraciji kolone ili tabele, nakon klauzule **REFERENCES**, mogu se navesti klauzule **ON DELETE** ili **ON UPDATE** koje specificiraju aktivnosti u slučaju narušavanja integriteta.

**ON DELETE** omogućava specifikaciju aktivnosti nad torkama relacije, odnosno vrstama tabele u kojoj je navedeno ovo ograničenje, u slučaju brisanja torki u referenciranoj tabeli.

**ON UPDATE** omogućava specifikaciju aktivnosti nad torkama u tabeli gde je **REFERENCES** ograničenje specificirano u slučaju ažuriranja (promene) referenciranih kolona.

**Napomena:** ORACLE SQL ne podržava klauzulu **ON UPDATE**.

U oba slučaja, iza ovih klauzula se navodi jedna od klauzula koja definiše aktivnost koja će se izvršiti nad torkama u tabeli u kojoj je ograničenje navedeno u slučaju narušavanja referencijalnog integriteta (pokušava se brisanje vrsta u referenciranoj tabeli ili se ažuriraju vrednosti referenciranih kolona):

- **NO ACTION** – ORACLE SQL definiše poruku o grešci u slučaju da akcija brisanja ili ažuriranja mogu da dovedu do narušavanja

referencijalnog integriteta. Ovo je podrazumevano ponašanje ograničenja tipa stranog ključa.

- **CASCADE** – kaskadno izvršenje aktivnosti brisanja (brišu se torke u tabeli u kojoj se ograničenje nalazi) kod **ON DELETE** ili aktivnosti ažuriranja (ažuriraju se vrednosti atributa koji čine primarni ključ) kod **ON UPDATE**.
- **SET DEFAULT** – vrednosti atributa stranog ključa se postavljaju na podrazumevanu vrednost.
- **SET NULL** – vrednost atributa stranog ključa se postavlja na NULL.

**Primer:** DDL naredba za kreiranje tabele CLAN\_PORODICE

```
CREATE TABLE CLAN_PORODICE
(
    MATBRRAD NUMBER(10),
    IME VARCHAR(15),
    POL CHAR(1) DEFAULT 'M' CHECK (POL IN ('M', 'Ž')),
    DATRODJ DATE,
    CONSTRAINT CLAN_PORODICE_PK PRIMARY KEY (MATBRRAD, IME),
    CONSTRAINT RODITELJ_FK FOREIGN KEY (MATBRRAD)
        REFERENCES RADNIK (MATBR)
);
```

**Primer:** DDL naredbe za kreiranje baze podataka PREDUZEĆE

```
CREATE TABLE RADNIK
(
    LIME VARCHAR(15) NOT NULL,
    SSLOVO VARCHAR(2) NOT NULL,
    PREZIME VARCHAR(15) NOT NULL,
    MATBR NUMBER(10),
    DATRODJ DATE,
    POL CHAR(1) DEFAULT 'M' CHECK (POL IN ('M', 'Ž')),
    PLATA NUMBER(8, 2) CHECK (PLATA > 1000),
    ADRESA VARCHAR(30),
    MATBRS NUMBER(10),
    BRSEK NUMBER(2) NOT NULL,
    CONSTRAINT RADNIK_PK PRIMARY KEY (MATBR),
    CONSTRAINT SEKTOR_FK FOREIGN KEY (BRSEK)
        REFERENCES SEKTOR (SBROJ),
    CONSTRAINT RUKOVODI_FK FOREIGN KEY (MATBRS)
        REFERENCES RADNIK (MATBR)
);

CREATE TABLE SEKTOR
(
    NAZIV VARCHAR(15) NOT NULL,
    SBROJ NUMBER(2),
    MATBRR INT NOT NULL,
    DATPOST Date,
```

```
        CONSTRAINT SEKTOR_PK PRIMARY KEY (SBROJ),
        CONSTRAINT SEF_FK FOREIGN KEY (MATBRR)
            REFERENCES RADNIK (MATBR)

    );

CREATE TABLE PROJEKAT
(
    NAZIV VARCHAR(25) NOT NULL,
    LOKPR VARCHAR(15) DEFAULT 'Niš',
    BROJPR NUMBER(3),
    BRS NUMBER(2) NOT NULL,
    CONSTRAINT PROJEKATPK PRIMARY KEY (BROJPR),
    CONSTRAINT NADLEZAN_FK FOREIGN KEY (BRS)
        REFERENCES SEKTOR (SBROJ)
);

CREATE TABLE CLAN_PORODICE
(
    MATBRRAD NUMBER(10),
    IME VARCHAR(15),
    POL CHAR(1) DEFAULT 'M' CHECK (POL IN ('M', 'Ž')),
    DATRODJ Date,
    CONSTRAINT CLAN_PORODICE_PK PRIMARY KEY (MATBRRAD, IME),
    CONSTRAINT RODITELJ_FK FOREIGN KEY (MATBRRAD)
        REFERENCES RADNIK (MATBR)
);

CREATE TABLE LOK_SEK
(
    BRS NUMBER(2),
    LOKACIJA VARCHAR(15),
    CONSTRAINT LOKACIJA_PK PRIMARY KEY (BRS, LOKACIJA),
    CONSTRAINT SEKTOR_FK2 FOREIGN KEY (BRS)
        REFERENCES SEKTOR (SBROJ)
);

CREATE TABLE RADI_NA
(
    MBR NUMBER(10),
    BRPR NUMBER(3),
    SATI NUMBER(3) NOT NULL CHECK (SATI > 0),
    CONSTRAINT RADI_NA_PK PRIMARY KEY (MBR, BRPR),
    CONSTRAINT RADNIK_FK FOREIGN KEY (MBR)
        REFERENCES RADNIK (MATBR),
    CONSTRAINT PROJEKAT_FK FOREIGN KEY (BRPR)
        REFERENCES PROJEKAT (BROJPR)
);
```

## 7.3 Modifikacija šeme relacione baze podataka

Postoji veliki broj SQL DDL naredbi koje omogućavaju izmenu šeme relacione baze podataka. Prilikom korišćenja ovih naredbi potrebno je da budemo jako oprezni kako greškom, ili usled nepažnje ne bi obrisali neke objekte ili podatke koji su nam od značaja.

### 7.3.1 Brisanje tabele

Za brisanje tabele iz relacione baze podataka koristi se naredba **DROP TABLE**. Ova naredba briše strukturu tabele zajedno sa svim podacima koji se u tabeli nalaze. ORACLE SQL definiše sledeći oblik **DROP TABLE** naredbe:

```
DROP TABLE <ime_tabele> [CASCADE CONSTRAINTS]
```

Opcija **CASCADE CONSTRAINTS** definiše da se prilikom brisanja tabele automatski brišu i sva ograničenja koja referenciraju tabelu koja se briše.

**Primer:** SQL naredba koja briše tabelu RADNIK zajedno sa svim podacima koji se u tabeli nalaze.

```
DROP TABLE RADNIK;
```

DBMS će sprečiti brisanje tabele u slučaju da to dovodi do narušavanja referencijalnog integriteta, odnosno brisanje tabele nije moguće ukoliko u bazi podataka postoje ograničenja stranog ključa koja referenciraju tabelu koju želimo da obrišemo. U tom slučaju je moguće iskoristiti opciju **CASCADE CONSTRAINTS** koja će najpre obrisati sva ograničenja a zatim će obrisati i samu tabelu. U tom slučaju naredba za brisanje tabele **RADNIK** ima sledeći oblik:

```
DROP TABLE RADNIK CASCADE CONSTRAINTS;
```

### Modifikacija tabele

Za modifikaciju strukture tabele koristi se naredba **ALTER TABLE**. ORACLE SQL definiše sledeći oblik ove naredbe:

```
ALTER TABLE <ime_tabele>  
  ADD <definicija atributa> |  
  ADD <definicija ograničenja> |  
  DROP <ime atributa> |  
  DROP CONSTRAINT <ime_ograničenja> |
```

```
MODIFY < definicija atributa > |  
DROP PRIMARY KEY  
DROP UNIQUE <ime atributa>
```

**Primer:** SQL naredba koja u tabelu PROJEKAT dodaje novu kolonu.

```
ALTER TABLE PROJEKAT  
ADD VREDNOST_PROJEKTA NUMBER(10, 2);
```

Prilikom dodavanja nove kolone ne možemo odmah primeniti NOT NULL ograničenje jer bi u tom slučaju ograničenje odmah bilo narušeno. Zbog toga se kolona dodaje bez NOT NULL ograničenja, dodaju se neophodni podaci pa se eventualno naknadno dodaje ograničenje.

**Primer:** SQL naredba koja modifikuje kolonu i uvodi NOT NULL ograničenje nad kolonom.

```
ALTER TABLE PROJEKAT  
MODIFY VREDNOST_PROJEKTA NUMBER(12, 2) NOT NULL;
```

Prilikom promene tipa neke kolone treba voditi računa o podacima koji već postoje u toj koloni. ORACLE DBMS će pokušati da automatski izvrši konverziju postojećih podataka ukoliko je to moguće. Ukoliko nije moguće ORACLE DBMS će prijaviti poruku o grešci.

**Primer:** SQL naredba koja u tabelu dodaje novo ograničenje kojim se obezbeđuje da vrednost projekta uvek mora biti pozitivan broj.

```
ALTER TABLE PROJEKAT  
ADD CONSTRAINT VREDNOST_PROJEKTA_CK  
CHECK (VREDNOST_PROJEKTA > 0);
```

**Primer:** SQL naredba kojom se iz tabele PROJEKAT briše ograničenje.

```
ALTER TABLE PROJEKAT  
DROP CONSTRAINT VREDNOST_PROJEKTA_CK;
```

**Primer:** SQL naredba kojom se briše kolona iz tabele.

```
ALTER TABLE PROJEKAT  
DROP COLUMN VREDNOST_PROJEKTA;
```

## 7.4 Pretraživanje podataka

Pretraživanje i pribavljanje podataka su najčešće operacije koje korisnici izvršavaju u relacionoj bazi podataka. Za pretraživanje i pribavljanje podataka SQL programski jezik obezbeđuje naredbu **SELECT**. Naredba **SELECT** pribavlja podatke iz jedne tabele ili više povezanih tabela koje se nalaze u relacionoj bazi podataka. U svom osnovnom obliku naredba **SELECT** ne može ni na koji način da izmeni podatke koji se nalaze u relacionoj bazi podataka.

Naredba **SELECT** je deklarativna naredba. Korišćenjem ove naredbe korisnici imaju mogućost samo da specificiraju rezultate koje žele. Sa druge strane RDBMS je zadužen da isplanira, optimizuje i izvrši fizičke operacije neophodne za generisanje specificiranih rezultata.

Rezultat **SELECT** naredbe je uvek relacija. Naredba **SELECT** koristi podatke iz jedne ili većeg broja tabela, manipuliše tim podacima i kao rezultat generiše tabelu. Čak i kada je rezultat obrade skalarna vrednost ona se tretira kao tabela sa jednom vrstom i jednom kolonom.

### 7.4.1 Naredba **SELECT**

Naredba **SELECT** je jedna od najkompleksnijih naredbi SQL programskog jezika. Uključuje veći broj ključnih reči klauzula:

- **SELECT** – definiše listu kolona koje će biti uključene u rezultujuću tabelu
- **FROM** – definiše tabele iz kojih se pribavljaju podaci za potrebe generisanja rezultujuće tabele. Klauzula **FROM** može da uključi jednu ili više opcionih **JOIN** klauzula za povezivanje tabela na osnovu kriterijuma zadatih od strane korisnika.
- **WHERE** – definiše predikat na osnovu koga se ograničava broj vrsta u rezultujućoj tabeli. Ova klauzula iz rezultata eliminiše sve vrste za koje specificirani predikat ne vraća vrednost **TRUE**.
- **GROUP BY** – grupiše vrste koje u određenim kolonama imaju identične vrednosti.
- **HAVING** – definiše predikat na osnovu koga se elimiše vrste nakon što je klauzula **GROUP BY** primenjena na rezultujuću tabelu.
- **ORDER BY** – koristi se za sortiranje rezultujuće tabele. Korisnici specificiraju kolone po kojima se vrši sortiranje kao i smer sortiranja.



### 7.4.2 Kaluzule SELECT i FROM

Klauzule **SELECT** i **FROM** su jedine obavezne u okviru **SELECT** naredbe. klauzula **FROM** specificira tabele iz kojih se pribavljaju podaci. Ukoliko se navede više tabela potrebno je specificirati način spajanja tabela. Spajanje tabela će biti detaljno objašnjeno u narednoj sekciji. Primeri u ovom odeljku će biti ograničeni samo na na pribavljanje podataka iz jedne tabele.

Klauzula **SELECT** specificira kolone koje treba uključiti u rezultujuću tabelu. Mogu se koristiti sledeće opcije:

- **ALL** – u rezultujućoj tabeli prikazuju se sve vrste koje zadovoljavaju navedeni predikat
- **DISTINCT** – iz rezultujuće tabele izbacuju se duplikati vrsta
- **\*** - rezultujuća tabela uključuje sve kolone tabele ili tabela iz kojih se pribavljaju podaci
- **tabela.\*** - rezultujuća tabela uključuje sve kolone specificirane tabele
- **izraz** - ime kolone ili funkcije nad kolonama koja će biti uključena u rezultujuću tabelu
- **AS pseudonim** - novo ime kolone ili funkcije nad kolonama koje im se dodeljuje u rezultujućoj tabeli

**Primer:** U nastavku je dat SQL upit koji prikazuje kompletan sadržaj tabele RADNIK

```
SELECT * FROM RADNIK;
```

LIME	SSLOVO	PREZIME	MATBR	DATRODJ	POL	PLATA	ADRESA	MATBRS	BRSEK
Marko	J	Petrović	123456789	09-JAN-65	M	30000	Obilićev Venac 11	333445555	5
Sima	F	Todorović	333445555	08-DEC-55	M	40000	Dušanova 32	888665555	5
Valentina	D	Kovačević	999887777	19-JAN-68	Ž	25000	Knjeginje Ljubice 12/34	987654321	4
Aleksandra	S	Petrović	987654321	20-JUN-41	Ž	43000	Knjaževačka 11	888665555	4
Velibor	T	Jovanović	666884444	15-SEP-62	M	36000	Knjaževačka 132/12	333445555	5
Jelena	P	Janković	453453453	31-JUL-72	Ž	25000	Vizantijski bulevar 123/12	333445555	5
Stanko	L	Manojlović	987987987	29-MAR-69	M	25000	Nemanjina 23	987654321	4
Jovan	S	Obradović	888665555	10-NOV-47	M	55000	Nikole Kopernika 11	(null)	1

Rezultat bi bio ekvivalenta da smo napisali upit kod koga su umesto \* navedena imena svih kolona u tabeli.

```
SELECT LIME, SSLOVO, PREZIME, MATBR, DATRODJ,
       POL, PLATA, MATBRS, BRSEK
FROM RADNIK;
```

**Primer:** Ukoliko želimo da prikazemo samo određene kolone iz tabele **RADNIK** posle **SELECT** klauzule navešćemo imena kolona koje su od interesa. U nastavku je dat SQL upit koji prikazuje samo imena i prezimena radnika.

```
SELECT Ime, Prezime FROM RADNIK;
```

IME	PREZIME
Marko	Petrović
Sima	Todorović
Valentina	Kovačević
Aleksandra	Petrović
Velibor	Jovanović
Jelena	Janković
Stanko	Manojlović
Jovan	Obradović

Redosled kojim su kolone navedene u klauzuli **SELECT** definiše redosled kolona u rezultujućoj tabeli. U nastavku je dat SQL upit koji prikazuje imena i prezimena svih radnika ali u nešto drugačijem redosledu.

```
SELECT Prezime, Ime FROM RADNIK;
```

PREZIME	IME
Petrović	Marko
Todorović	Sima
Kovačević	Valentina
Petrović	Aleksandra
Jovanović	Velibor
Janković	Jelena
Manojlović	Stanko
Obradović	Jovan

**Primer:** U nastavku je dat SQL upit koji za svakog radnik određuje matični broj njegovog neposrednog rukovodioca.

```
SELECT MATBRS FROM RADNIK;
```

MATBRS
333445555
888665555
987654321
888665555
333445555
333445555
987654321
(null)

Možemo da se primeti da se u rezultujućoj tabeli neki matični brojevi javljaju više puta. To je posledica činjenice da veći broj radnika može imati istog rukovodioca.

Ekvivalentan SQL upit koristi ključnu reč **ALL**. Ključna reč **ALL** nalaže ORACLE RDBMS-u da prikaže sve vrste rezultujuće tabele (uključujući i duplikate). Ključna reč **ALL** je podrazumevana pa se najčešće izostavlja.

```
SELECT ALL MATBRS FROM RADNIK;
```

Ukoliko želimo da eliminišemo duplikate koristićemo ključnu reč **DISTINCT**.

```
SELECT DISTINCT MATBRS FROM RADNIK;
```

MATBRS
333445555
(null)
987654321
888665555

### 7.4.3 Klauzula WHERE

Klauzula WHERE specificira uslov na osnovu koga se kreira rezultujuća tabela. U rezultujuću tabelu će biti uključene samo one vrste koje zadovoljavaju specificirani uslov, odnosno za koje specificirani uslov ima vrednost TRUE. U uslovu se mogu javiti:

1. Relacioni operatori
2. Logički operatori
3. Operator **BETWEEN**
4. Operator **IN**
5. Operator **LIKE**
6. Operator **IS NULL**

SQL podržava šest relacionih operatora koji imaju sledeće značenje:

1. = Jednako
2. <> Nije jednako (različito)
3. < Manje od
4. > Veće od
5. <= Manje ili jednako a
6. >= Veće ili jednako

**Primer:** U ovom primeru dat je SQL upit koji prikazuje podatke o radnicima koji se prezivaju *Petrović*.

```
SELECT *
FROM RADNIK
WHERE PREZIME = 'Petrović';
```

LIME	SSLOVO	PREZIME	MATR	DATRODJ	POL	PLATA	ADRESA	MATBR	BRSEK
Marko	J	Petrović	123456789	09-JAN-65	M	30000	Obilićev Venac 11	333445555	5
Aleksandra	S	Petrović	987654321	20-JUN-41	Ž	43000	Knjaževačka 11	888665555	4

**Napomena:** Treba primetiti da se tekstualni podaci zadaju korišćenjem jednostrukih apostrofa: 'Petrović'.

**Primer:** Naredni primer sadrži SQL upit koji prikazuje imena i prezimena radnika čija je plata jednaka ili veća od 40000.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE PLATA >= 40000;
```

LIME	PREZIME
Sima	Todorović
Aleksandra	Petrović
Jovan	Obradović

SQL omogućava korišćenje standardnih logičkih operatore **AND**, **OR** i **NOT**, ali i operatore **IN** i **BETWEEN** koji omogućavaju jednostavnije korišćenje prethodno navedenih operatora u nekim slučajevima.

Prioritet logičkih operatora je sledeći:

1. **NOT**
2. **AND**
3. **OR**

**Napomena:** Logički operatori **AND** i **OR** se koriste na standardni način. Međutim, kod SQL-a, logički operator negacije **NOT** se navodi na početku logičkog izraza, a ne ispred operatora poređenja. Na primer, **NOT A = B** je validni **WHERE** uslov, ali **A NOT = B** nije.

**Primer:** Za prikaz podataka o radnicima koji se prezivaju Petrović i čija je plata jednaka ili veća od 40000 može se koristiti SQL upit koji je dat u nastavku.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE PREZIME = 'Petrović' AND PLATA >= 40000;
```

A	LIME	A	PREZIME
	Aleksandra		Petrović

**Primer:** U nastavku je dat SQL upit koji prikazuje podatke o radnicima koji se prezivaju Petrović i čija je plata manja od 40000 (nije jednaka ili veća od 40000).

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE PREZIME = 'Petrović' AND NOT PLATA >= 40000;
```

A	LIME	A	PREZIME
	Marko		Petrović

**Primer:** U nastavku je dat primer SQL upita koji prikazuje podatke o radnicima koji se prezivaju *'Petrović'* ili se prezivaju *'Jovanović'*.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE PREZIME = 'Petrović' OR PREZIME = 'Jovanović';
```

A	LIME	A	PREZIME
	Marko		Petrović
	Aleksandra		Petrović
	Velibor		Jovanović

Ključna reč **IN** zamenjuje višestruku upotrebu operatora **OR** i **=**. Operator **NOT IN** prikazuje sve vrste osim onih određenih **IN** listom.

**Primer:** Naredni primer predstavlja SQL upit koji korišćenjem operatora **IN** izdvaja se samo radnike koji se prezivaju *'Petrović'* ili *'Jovanović'*.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE PREZIME IN ('Petrović', 'Jovanović');
```

A	LIME	A	PREZIME
	Marko		Petrović
	Aleksandra		Petrović
	Velibor		Jovanović

**Primer:** Naredni SQL upit pribavlja podatke o svim radnicima osim onih koji se prezivaju *'Petrović'* ili *'Jovanović'*.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE PREZIME NOT IN ('Petrović', 'Jovanović');
```

LIME	PREZIME
Sima	Todorović
Valentina	Kovačević
Jelena	Janković
Stanko	Manojlović
Jovan	Obradović

Operator **BETWEEN** zamenjuje višestruku upotrebu operatora **AND** i **=**. Ovaj operator omogućava ispitivanje da li je vrednost atributa/kolone u zadatom opsegu (uključujući i granice opsega).

**Primer:** U nastavku je dat SQL upit koji koristi operator **BETWEEN** za prikazivanje podataka o radnicima čija je plata u opsegu od 30000 do 40000 (uključujući i granice opsega).

```
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE PLATA BETWEEN 30000 AND 40000;
```

MATBR	LIME	PREZIME
123456789	Marko	Petrović
333445555	Sima	Todorović
666884444	Velibor	Jovanović

Alternativni SQL upit koji vraća identičan rezultat bez korišćenja **BETWEEN** operatora ima sledeći oblik:

```
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE PLATA >= 30000 AND PLATA <= 40000;
```

Operator **IS NULL** se koristi za poređenje sa **NULL** vrednostima, odnosno omogućava proveru da li kolona ima **NULL** vrednost.

**Primer:** U nastavku je dat SQL upit koji izdvaja podatke o radnicima koji nemaju neposrednog rukovodioca odnosno kod kojih kolona **MATBRS** ima vrednost **NULL** (postoji samo jedan takav radnik koji se nalazi na funkciji glavnog rukovodioca, odnosno on nema nikoga iznad sebe u hijerarhiji).

```
SELECT MATBR, LIME, PREZIME, MATBRS
FROM RADNIK
WHERE MATBRS IS NULL;
```

MATBR	LIME	PREZIME	MATBRS
888665555	Jovan	Obradović	(null)

Za slučaj da želimo da pronademo sve radnike koji imaju neposredne rukovodioce (kod kojih kolona MATBRS ima vrednost koja nije NULL) odgovarajući SQL upit se može napisati na jedan od naredna dva načina:

```
SELECT *
FROM RADNIK
WHERE NOT MATBRS IS NULL;
```

```
SELECT *
FROM RADNIK
WHERE MATBRS IS NOT NULL;
```

**Napomena:** Treba voditi računa da se na NULL vrednosti ne može primentiti ni jedan relacioni operator (sve NULL vrednosti su jedinstvene). Može se samo proveravati da li kolona ima NULL vrednost ili nema. Iz tog razloga naredni SQL upit ne vraća ni jedan slog u rezultujućoj tabeli jer poređenje sa NULL vrednošću nije moguće. Pri tome ORACLE DBMS ne prijavljuje grešku.

```
SELECT *
FROM RADNIK
WHERE MATBRS = NULL;
```

Operator **LIKE** omogućava poređenje znakovnih vrednosti (tekstualnih podataka) sa zadatim šablonom. Za definisanje šablona koriste se simboli procenat (%) i donja crta (\_). Procenat (%) predstavlja bilo koji znak (broj, slovo, interpunkcijski znak) ili skup znakova. Procenat (%) može da zameni i prazan string. Donja crta (\_) zamenjuje samo jedan znak (Pojavljivanje znaka je obavezno). Operator **NOT LIKE** prikazuje sve vrste koje ne odgovaraju prethodno zadatom opisu

**Primer:** U narednom SQL upitu % iza slova 'J' označava proizvoljan broj znakova, odnosno predstavlja uzorak za poklapanje koji sadrži na početku slovo 'J' i proizvoljan broj znakova iza njega. Za nalaženje svih radnika koji imaju 'J' na početku prezimena, može se koristiti šablon 'J%'.

```
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE PREZIME LIKE 'J%';
```

MATBR	LIME	PREZIME
666884444	Velibor	Jovanović
453453453	Jelena	Janković

**Napomena:** ORACLE DBMS kod tekstualnih podataka pravi razliku između malih i velikih slova. Zbog toga, predikat **LIKE** 'j%' u prethodnom upitu ne bi vratio nijednu rezultujuću vrstu.

Za nalaženje svih radnika koji sadrže slovo *j* negde u prezimenu mogao bi se koristiti šablon '%j%'.

**Primer:** Ukoliko je neophodno prikazate podatke o radnicima koji su rođeni toko meseca septembra, odgovarajući SQL upit ima sledeći oblik:

```
SELECT MATBR, LIME, PREZIME, DATRODJ
FROM RADNIK
WHERE DATRODJ LIKE '%SEP%';
```

MATBR	LIME	PREZIME	DATRODJ
666884444	Velibor	Jovanović	15-SEP-62

Alternativni SQL upit koji vraća identičan rezultat ima sledeći oblik:

```
SELECT MATBR, LIME, PREZIME, DATRODJ
FROM RADNIK
WHERE DATRODJ LIKE '___SEP___';
```

Operator **NOT LIKE** prikazuje sve vrste koje ne odgovaraju prethodno zadatom šablonu.

**Primer:** SQL upit koji pronalazi podatke o svim radnicima koji nisu rođeni tokom meseca septembra ima sledeći oblik:

```
SELECT MATBR, LIME, PREZIME, DATRODJ
FROM RADNIK
WHERE DATRODJ NOT LIKE '%SEP%';
```



MATBR	LIME	PREZIME	DATRODJ
123456789	Marko	Petrović	09-JAN-65
333445555	Sima	Todorović	08-DEC-55
999887777	Valentina	Kovačević	19-JAN-68
987654321	Aleksandra	Petrović	20-JUN-41
453453453	Jelena	Janković	31-JUL-72
987987987	Stanko	Manojlović	29-MAR-69
888665555	Jovan	Obradović	10-NOV-47

#### 7.4.4 Klauzula ORDER BY

Klauzula **ORDER BY** specificira redosled prikazivanja vrste rezultujuće tabele, sortiranjem po vrednosti nekih kolona u rastući (**ASC**) (predefinisana vrednost) ili opadajući redosled (**DESC**). Ukoliko klauzula **ORDER BY** nije navedena vrste u rezultujućoj tabeli su poređane po slučajnom principu i ne postoji nikakva garancije da će isti upit uvek generisati rezultujuću tabelu čije su vrste poređane na isti način.

**Primer:** U nastavku je dat SQL upit koji prikazuje podatke o radnicima i sortira ih prema prezimenu u rastućem redosledu.

```
SELECT MATBR, LIME, SSLOVO, PREZIME
FROM RADNIK
ORDER BY PREZIME ASC;
```

MATBR	LIME	SSLOVO	PREZIME
453453453	Jelena	P	Janković
666884444	Velibor	T	Jovanović
999887777	Valentina	D	Kovačević
987987987	Stanko	L	Manojlović
888665555	Jovan	S	Obradović
123456789	Marko	J	Petrović
987654321	Aleksandra	S	Petrović
333445555	Sima	F	Todorović

**Napomena:** Ukoliko se vrste sortiraju u rastućem redosledu (**ASC**) po vrednosti neke kolone, nije potrebno eksplicitno navesti smer sortiranja. Rastući redosled je podrazumevani redosled sortiranja u **ORDER BY** klauzuli.

**Primer:** Alternativni SQL upit koji prikazuje podatke o radnicima sortirane prema prezimenu u opadajućem redosled ima sledeći oblik:

```
SELECT MATBR, LIME, SSLOVO, PREZIME
FROM RADNIK
ORDER BY PREZIME DESC;
```

MATBR	LIME	SSLOVO	PREZIME
333445555	Sima	F	Todorović
123456789	Marko	J	Petrović
987654321	Aleksandra	S	Petrović
888665555	Jovan	S	Obradović
987987987	Stanko	L	Manojlović
999887777	Valentina	D	Kovačević
666884444	Velibor	T	Jovanović
453453453	Jelena	P	Janković

**Napomena:** Ukoliko se vrste sortiraju u opadajućem redosledu (**DESC**) po vrednosti neke kolone, uvek je potrebno eksplicitno navesti smer sortiranja.

Sortiranje je moguće vršiti prema vrednostima većeg broja kolona. U tom slučaju se sve kolone po kojima se rezultat sortira navode u **ORDER BY** klauzuli zajedno sa smerom sortiranja. Redosled navođenja kolona definiše redosled po kome će se njihove vrednosti uzimati u obzir prilikom sortiranja rezultujuće tabele.

Primer: Naredni SQL upit prikazuje podatke o radnicima i sortira ih prema broju sektora u opadajućem redosledu a prema prezimenu u rastućem redosledu.

```
SELECT MATBR, LIME, SSLOVO, PREZIME, BRSEK
FROM RADNIK
ORDER BY BRSEK DESC, PREZIME ASC;
```

MATBR	LIME	SSLOVO	PREZIME	BRSEK
453453453	Jelena	P	Janković	5
666884444	Velibor	T	Jovanović	5
123456789	Marko	J	Petrović	5
333445555	Sima	F	Todorović	5
999887777	Valentina	D	Kovačević	4
987987987	Stanko	L	Manojlović	4
987654321	Aleksandra	S	Petrović	4
888665555	Jovan	S	Obradović	1

### 7.4.5 Aritmetičke funkcije

SQL dozvoljava korišćenje matematičkih funkcija u **SELECT** i **WHERE** klauzulama. Na taj način se kao rezultat pretraživanja mogu prikazati rezultati izračunavanja nekog matematičkog izraza.

Deo aritmetičkih funkcija koje podržava ORACLE DBMS dat je u nastavku:

SQL dozvoljava korišćenje matematičkih funkcija u **SELECT** i **WHERE** klauzulama. Na taj način se kao rezultat pretraživanja mogu prikazati rezultati izračunavanja nekog matematičkog izraza.

- +, -, \*, /
- **ROUND**(broj[,d]) - zaokruživanje na d decimala
- **POWER**(broj,e) - e-ti stepen zadatog broja
- **TRUNC**(broj[,d]) - odsecanje na d decimala

- **ABS**(broj) - apsolutna vrednost broja
- **SIGN**(broj) - znak broja
- **MOD**(broj1, broj2) - broj1 mod broj2
- **SQRT**(broj) - kvadratni koren broja
- **LEAST**(izraz, ...) - najmanji navedeni izraz
- **GREATEST**(izraz, ...) - najveći navedeni izraz

**Primer:** U nastavku je dat SQL upit koji prikazuje imena i prezimena radnika kao i njihove plate uvećane za bonus od 5000.

```
SELECT MATBR, LIME, SSLOVO, PREZIME,
       PLATA + 5000 AS PLATA_SA_BONUSOM
FROM RADNIK;
```

MATBR	LIME	SSLOVO	PREZIME	PLATA_SA_BONUSOM
123456789	Marko	J	Petrović	35000
333445555	Sima	F	Todorović	45000
999887777	Valentina	D	Kovačević	30000
987654321	Aleksandra	S	Petrović	48000
666884444	Velibor	T	Jovanović	41000
453453453	Jelena	P	Janković	30000
987987987	Stanko	L	Manojlović	30000
888665555	Jovan	S	Obradović	60000

**Napomena:** Rezultatu matematičke funkcije (PLATA + 5000) je dodeljeno ime korišćenjem sintakse pseudonima: **AS PLATA\_SA\_BONUSOM**. U rezultujućoj tabeli rezultat ove aritmetičke funkcije se pojavljuje kao nova kolona.

**Primer:** Naredni SQL upit prikazuje podatke o radnicima kod kojih iznos plate uvećan za 5000 ima vrednost veću od 40000.

```
SELECT MATBR, LIME, SSLOVO, PREZIME,
       PLATA + 5000 AS PLATA_SA_BONUSOM
FROM RADNIK
WHERE PLATA + 5000 > 40000;
```

MATBR	LIME	SSLOVO	PREZIME	PLATA_SA_BONUSOM
333445555	Sima	F	Todorović	45000
987654321	Aleksandra	S	Petrović	48000
666884444	Velibor	T	Jovanović	41000
888665555	Jovan	S	Obradović	60000

**Primer:** U nastavku je dat SQL upit koji demonstrira način primene mehanizma pseudonima za imenovanje kolona u rezultujućoj tabeli.

```
SELECT LIME AS IME, PREZIME,
       PLATA + 5000 AS PLATA_SA_BONUSOM,
       1 AS KONSTANTA, NULL NOVA_KOLONA
FROM RADNIK;
```

IME	PREZIME	PLATA_SA_BONUSOM	KONSTANTA	NOVA_KOLONA
Marko	Petrović	35000	1 (null)	
Sima	Todorović	45000	1 (null)	
Valentina	Kovačević	30000	1 (null)	
Aleksandra	Petrović	48000	1 (null)	
Velibor	Jovanović	41000	1 (null)	
Jelena	Janković	30000	1 (null)	
Stanko	Manojlović	30000	1 (null)	
Jovan	Obrovčić	60000	1 (null)	

#### 7.4.6 Funkcije za rad sa stringovima

Funkcije za rad sa stringovima omogućavaju manipulaciju znakovnim podacima. U nastavku je dat deo fnkcija za rad sa stringovma koji je podržan od strane ORACLE DBMS-a:

- `string1 || string2` - konkatencija stringova
- **LENGTH**(string) - dužina stringa
- **SUBSTR**(s, i, j) - podniz od s dužine j od pozicije i
- **INSTR**(s1, s2[,k]) - traži s2 u s1 od pozicije k
- **UPPER**(s) - prevodi s u velika slova
- **LOWER**(s) - prevodi s u mala slova
- **TO\_NUM**(s) - prevodi s u numerički tip
- **TO\_CHAR**(n) - prevodi numerički u znakovni tip
- **LPAD**(s, l[,c]) - popunjava s sa leve strane sa l znakova c
- **RPAD**(s, l[,c]) - popunjava s sa desne strane sa l znakova c
- **NVL**(s1, s2) - ako važi s1 IS NULL, vraća s2; u suprotnom vraća s1.

**Primer:** U nastavku je dat SQL upit koji konkatencijom formira puno ime svih radnika.

```
SELECT LIME || ' ' || SSLOVO || '.' || ' ' || PREZIME AS PUNO_IME
FROM RADNIK;
```

PUNO_IME
Marko J. Petrović
Sima F. Todorović
Valentina D. Kovačević
Aleksandra S. Petrović
Velibor T. Jovanović
Jelena P. Janković
Stanko L. Manojlović
Jovan S. Obradović

**Primer:** U nastavku je dat SQL upit koji prikazuje informacije o radnicima uključujući i matične brojeve njihovih neposrednih rukovodioca. Za radnika koji nema neposrednog rukovodioca ispisuje se poruka NEMA ŠEFA.

```
SELECT LIME, PREZIME,
       NVL(TO_CHAR(MATBR), 'NEMA ŠEFA') SEF
FROM RADNIK;
```

LIME	PREZIME	SEF
Marko	Petrović	333445555
Sima	Todorović	888665555
Valentina	Kovačević	987654321
Aleksandra	Petrović	888665555
Velibor	Jovanović	333445555
Jelena	Janković	333445555
Stanko	Manojlović	987654321
Jovan	Obradović	NEMA ŠEFA

#### 7.4.7 Funkcije agregacije

Funkcije agregacije su dobile naziv po tome što vrše agregaciju rezultata upita. Korišćenje ovih funkcija je jednostavno, pošto se navode u listi kolona **SELECT** klauzule koje se prikazuju. Značenje funkcija je sledeće:

- **AVG(kolona)** - izračunava srednju vrednost datog atributa
- **SUM(kolona)** - izračunava sumu svih vrednosti atributa
- **MIN(kolona)** - nalazi minimalnu vrednost atributa
- **MAX(kolona)** - nalazi najveću vrednost atributa
- **COUNT(\*)** - nalazi broj vrsta u tabeli (grupi)
- **COUNT(kolona)** - nalazi broj broj vrsta sa ne NULL vrednostima kolone
- **COUNT (DISTINCT kolona)** - nalazi broj vrsta sa različitim vrednostima zadate kolone

**Primer:** Funkcija COUNT određuje broj vrsta u rezultujućoj tabeli. SQL upit koji je definisan u nastavku određuje broj radnika o kojima se čuvaju podaci u bazi podataka.

```
SELECT COUNT(*) AS UKUPNO_RADNIKA  
FROM RADNIK;
```

UKUPNO_RADNIKA
8

**Primer:** Sledeći SQL upit određuje maksimalnu, minimalnu, prosečnu i ukupnu platu svih radnika u preduzeću.

```
SELECT MAX(PLATA) AS MAX_PLATA,  
       MIN(PLATA) AS MIN_PLATA,  
       AVG(PLATA), SUM(PLATA)  
FROM RADNIK;
```

MAX_PLATA	MIN_PLATA	AVG(PLATA)	SUM(PLATA)
55000	25000	34875	279000

Napomena: Funkcije agregacije nije moguće koristiti u **WHERE** klauzuli. To je posledica činjenice da se rezultat funkcija agregacija izračunava nakon što se odrede vrste koje ulaze u sastav rezultujuće tabele, odnosno nakon obrade predikta koji je zadat u **WHERE** klauzuli. U nastavku je dat SQL upit koji se **NE MOŽE IZVRŠITI** i koji će **GENERISATI GREŠKU**.

```
SELECT LIME, PREZIME, PLATA  
FROM RADNIK  
WHERE PLATA > AVG(PLATA);
```

## 7.5 Napredne klauzule SELECT naredbe i spajanje tabela

### 7.5.1 Spajanje tabela

Svi SQL upiti koji su razmatrani u prethodnoj sekciji su koristili podatke iz samo jedne tabele. Često se javlja situacija da se tražena informacija nalazi u većem broju tabela. U takvim situacijama potrebno je izvršiti spajanje vrsta iz različitih tabela i generisanje rezultujuće tabele.

Za pribavljanje podataka iz većeg broja tabela dovoljno je u klauzuli **FROM** navesti imena tabela iz kojih želimo da pribavimo podatke. Da bi spajanje tabela bilo uspešno, osim u nekim specijalnim slučajevima, potrebno je da navedemo

uslov spoja, odnosno da navedemo kolone na osnovu čijih vrednosti se vrši spajanje vrsta iz različitih tabela. Spajanje tabela se vrši tako što se najčešće uparuje strani ključ iz jedne tabele sa primarnim ključem koji referencira u drugoj tabeli. Uslov spajanja može da se zada u okviru **WHERE** klauzule ili korišćenjem ključne reči **JOIN** u okviru **FROM** klauzule.

**Spoj na jednakost (equi-join)** obezbeđuje spajanje podataka iz dve ili više tabela na osnovu jednakosti odgovarajućih atributa, obično na osnovu primarnih i stranih ključeva. Najjednostavniji slučaj navođenja spoja je kada se u **WHERE** klauzuli specificira uslov spoja po jednakosti.

**Dekartov proizvod** je slučaj kada u **WHERE** klauzuli ne postoji uslov spoja, a u **FROM** klauzuli je navedeno više tabela. U tom slučaju nema spajanja vrsta po vrednosti nekog atributa, već se pravi kombinacija svake vrste iz jedne tabele sa svakom vrstom iz druge tabele (u slučaju Dekartovog proizvoda dve tabele)

**Spoljni spoj (outer-join)** omogućava spajanje dve tabele po vrednosti nekog atributa (kao kod equi-join), ali i uključivanje onih torki (vrsta) iz jedne ili druge tabele (ili iz obe), koje ne zadovoljavaju uslov jednakosti.

SQL standard definiše sledeće tipove spojeva:

- *cross join*
- *inner join*
- *outer join*
  - *left outer join*
  - *right outer join*
  - *full outer join*

Zbog jednostavnost za primere u nasatavku biće korišćene nešto jednostavnije verzije tabela RADNIK i SEKTOR. Ove pojednostavljene verzije ove dve tabele su prikazane u nastavku.

RADNIK

IME	SSLOVO	PREZIME	MATER	BRSEK
Borivoje	P	Veljković	11111111	10
Marko	J	Petrović	123456789	5
Sima	F	Todorović	333445555	5
Valentina	D	Kovačević	999887777	4
Aleksandra	S	Petrović	987654321	4
Velibor	T	Jovanović	666884444	5
Jelena	P	Janković	453453453	5
Stanko	L	Manojlović	987987987	4
Jovan	S	Obrovović	888665555	1

SEKTOR

NAZIV	SBROJ
Razvoj	5
Administracija	4
Uprava	1
Prodaja	12

**Napomena:** U pojednostavljenu verziju tabele RADNIK dodate su informacije o radniku *Borivoju Veljkoviću* koji radi u nepostojećem sektoru čiji je broj 10.

**Napomena:** U pojednostavljenu verziju tabele SEKTOR dodate su informacije o sektoru *Prodaja* u kome ne radi nijedan radnik.

### 7.5.2 Dekartov proizvod (cross-join)

Cross join predstavlja Dekartov proizvod vrsta iz tabela koje se spajaju. Dekartov proizvod dve tabele (A **CROSS JOIN** B) se dobija tako što se svaka vrsta iz jedne tabele kombinuje sa svakom vrstom iz druge tabele.

Ne koristi se često ali predstavlja osnovu za definisanje ostalih tipova spoja. Ukoliko se u FROM klauzuli navede veći broj tabela a ne navede se tip i uslov spoja podrazumeva se cross join.

```
SELECT *
FROM SEKTOR CROSS JOIN RADNIK;
```

```
SELECT *
FROM SEKTOR, RADNIK;
```

NAZIV	SBROJ	LIME	SSLOVO	PREZIME	MATBR	BRSEK
Razvoj	5	Borivoje	P	Veljković	111111111	10
Razvoj	5	Marko	J	Petrović	123456789	5
Razvoj	5	Sima	F	Todorović	333445555	5
Razvoj	5	Valentina	D	Kovačević	999887777	4
Razvoj	5	Aleksandra	S	Petrović	987654321	4
Razvoj	5	Velibor	T	Jovanović	666884444	5
Razvoj	5	Jelena	P	Janković	453453453	5
Razvoj	5	Stanko	L	Manojlović	987987987	4
Razvoj	5	Jovan	S	Obradović	888665555	1
Administracija	4	Borivoje	P	Veljković	111111111	10
Administracija	4	Marko	J	Petrović	123456789	5
Administracija	4	Sima	F	Todorović	333445555	5
Administracija	4	Valentina	D	Kovačević	999887777	4
Administracija	4	Aleksandra	S	Petrović	987654321	4
Administracija	4	Velibor	T	Jovanović	666884444	5
Administracija	4	Jelena	P	Janković	453453453	5

### 7.5.3 Unutrašnji spoj (inner-join)

Unutrašnji spoj (Inner join) predstavlja najčešće korišćeni tip spoja. Ovaj tip spoja, u osnovi, definiše presek vrsta iz tabela koje učestvuju u spoju.

Prilikom spajanja dve tabele (A **INNER JOIN** B) uzimaju se sve vrste iz tabele A i pronalazi im se odgovarajuća vrsta u tabeli B. Ukoliko vrsta iz tabele A nema odgovarajuću vrstu u tabeli B ne uključuje se u rezultat. Ukoliko vrsti iz tabele A odgovara više vrsta tabele B ona se u rezultatu ponavlja više puta (po jednom za svaku odgovarajuću vrstu u tabeli B).

**Napomena:** Prilikom spajanja tabela treba voditi računa o tome da kolone u različitim tabelama mogu imati ista imena. U takvim situacijama se koristi sintaksa IME\_TABELE.IME\_KOLONE. U SQL upitima koji su dati u nastavku taj pristup



je iskorišćen za kolone SEKTOR.SBROJ i RADNIK.BRSEK mada nije bilo neophodno jer kolone imaju različita imena. Isti pristup treba primeniti i za klauzulu SELECT ukoliko se javi sličan problem.

```
SELECT *
FROM SEKTOR INNER JOIN RADNIK
      ON SEKTOR.SBROJ = RADNIK.BRSEK;
```

```
SELECT *
FROM SEKTOR JOIN RADNIK
      ON SEKTOR.SBROJ = RADNIK.BRSEK;
```

```
SELECT *
FROM SEKTOR, RADNIK
WHERE SEKTOR.SBROJ = RADNIK.BRSEK;
```

NAZIV	SBROJ	IME	SSLOVO	PREZIME	MATBR	BRSEK
Razvoj	5	Marko	J	Petrović	123456789	5
Razvoj	5	Sima	F	Todorović	333445555	5
Administracija	4	Valentina	D	Kovačević	999887777	4
Administracija	4	Aleksandra	S	Petrović	987654321	4
Razvoj	5	Velibor	T	Jovanović	666884444	5
Razvoj	5	Jelena	P	Janković	453453453	5
Administracija	4	Stanko	L	Manojlović	987987987	4
Uprava	1	Jovan	S	Obradović	888665555	1

#### 7.5.4 Levi spoljašnji spoj (left-outer join)

Levi spoljašnji spoj (Left-outer Join) u osnovi predstavlja prošireni inner-join. Levi spoljašnji spoj (A **LEFT OUTER JOIN** B) pored vrsta koje uključuje unutrašnji spoj uključuje i vrste iz tabele A (leve tabele) koje nemaju odgovarajuću vrstu u tabeli B (desnoj tabeli). U vrstama koje iz tabele A koje ne mogu da se upare ni sa jednom vrstom iz tabele B, kolone iz tabele B imaju vrednost NULL.

```
SELECT *
FROM SEKTOR LEFT OUTER JOIN RADNIK
      ON SEKTOR.SBROJ = RADNIK.BRSEK;
```

```
SELECT *
FROM SEKTOR LEFT JOIN RADNIK
      ON SEKTOR.SBROJ = RADNIK.BRSEK;
```

```
SELECT *
FROM SEKTOR, RADNIK
WHERE SEKTOR.SBROJ = RADNIK.BRSEK(+);
```

NAZIV	SBROJ	LIME	SSLOVO	PREZIME	MATR	BRSEK
Razvoj	5	Marko	J	Petrović	123456789	5
Razvoj	5	Sima	F	Todorović	333445555	5
Administracija	4	Valentina	D	Kovačević	999887777	4
Administracija	4	Aleksandra	S	Petrović	987654321	4
Razvoj	5	Velibor	T	Jovanović	666884444	5
Razvoj	5	Jelena	P	Janković	453453453	5
Administracija	4	Stanko	L	Manojlović	987987987	4
Uprava	1	Jovan	S	Obradović	888665555	1
Prodaja	12	(null)	(null)	(null)	(null)	(null)

U rezultatu levog spoljašnjeg spoja pojednostavljenih verzija tabela SEKTOR i RADNIK pojavljuju se i podaci o sektoru *Prodaja*. U ovom sektoru nema radnika pa se u slučaju unutrašnjeg spoja ne pojavljuje u rezultujućoj tabeli. U slučaju levog spoljašnjeg spoja, pojavljuje se jednom a nedostajeće kolone iz desne tabele (u ovom slučaju tabela RADNIK) postavljene su na vrednost NULL.

### 7.5.5 Desni spoljašnji spoj (right-outer join)

Desni spoljašnji spoj (Right outer join) funkcioniše kao i left outer join samo je sada uloga tabela promenjena. Desni spoljašnji spoj (**A RIGHT OUTER JOIN B**) pored vrsta koje uključuje unutrašnji spoj u rezultat uključuje vrste iz tabele B (desne tabele) koje nemaju odgovarajuću vrstu u tabeli A (levoj tabeli). U vrstama koje iz tabele B koje ne mogu da se upare ni sa jednom vrstom iz tabele A, kolone iz tabele A imaju vrednost NULL.

```
SELECT *
FROM SEKTOR RIGHT OUTER JOIN RADNIK
ON SEKTOR.SBROJ = RADNIK.BRSEK;

SELECT *
FROM SEKTOR RIGHT JOIN RADNIK
ON SEKTOR.SBROJ = RADNIK.BRSEK;

SELECT *
FROM SEKTOR, RADNIK
WHERE SEKTOR.SBROJ (+) = RADNIK.BRSEK;
```

NAZIV	SBROJ	LIME	SSLOVO	PREZIME	MATR	BRSEK
Razvoj	5	Jelena	P	Janković	453453453	5
Razvoj	5	Velibor	T	Jovanović	666884444	5
Razvoj	5	Sima	F	Todorović	333445555	5
Razvoj	5	Marko	J	Petrović	123456789	5
Administracija	4	Stanko	L	Manojlović	987987987	4
Administracija	4	Aleksandra	S	Petrović	987654321	4
Administracija	4	Valentina	D	Kovačević	999887777	4
Uprava	1	Jovan	S	Obradović	888665555	1
(null)	(null)	Borivoje	P	Veljković	111111111	10

U rezultatu desnog spoljašnjeg spoja pojednostavljenih verzija tabela SEKTOR i RADNIK pojavljuju se i podaci o radniku *Borivoju Veljkoviću*. Ovaj radnik radi u nepostojećem sektoru pa se u slučaju unutrašnjeg spoja ne pojavljuje u rezultujućoj

tabeli. U slučaju desnog spoljašnjeg spoja, pojavljuje se jednom a nedostajeće kolone iz leve tabele (u ovom slučaju tabela SEKTOR) postavljene su na vrednost NULL.

### 7.5.6 Potpuni spoljašnji spoj (full-outer join)

Potpuni spoljašnji spoj (Full outer join) predstavlja kombinaciju rezultata koje vraćaju left outer i right outer join.

Potpuni spoljašnji spoj (A **FULL OUTER JOIN** B) pored vrsta koje uključuje unutrašnji spoj u rezultat uključuje i vrste iz obe tabele (i iz A i iz B) koje nemaju odgovarajuće slogove u drugoj tabeli. U vrstama koje ne mogu da se upare, nedostajuće kolone imaju vrednost NULL.

```
SELECT *
FROM SEKTOR FULL OUTER JOIN RADNIK
ON SEKTOR.SBROJ = RADNIK.BRSEK;
```

```
SELECT *
FROM SEKTOR FULL JOIN RADNIK
ON SEKTOR.SBROJ = RADNIK.BRSEK;
```

NAZIV	SBROJ	LIME	SSLOVO	PREZIME	MATBR	BRSEK
(null)	(null)	Borivoje	P	Veljković	111111111	10
Razvoj	5	Marko	J	Petrović	123456789	5
Razvoj	5	Sima	F	Todorović	333445555	5
Administracija	4	Valentina	D	Kovačević	999887777	4
Administracija	4	Aleksandra	S	Petrović	987654321	4
Razvoj	5	Velibor	T	Jovanović	666884444	5
Razvoj	5	Jelena	P	Janković	453453453	5
Administracija	4	Stanko	L	Manojlović	987987987	4
Uprava	1	Jovan	S	Obradović	888665555	1
Prodaja	12	(null)	(null)	(null)	(null)	(null)

U rezultatu poptunog spoljašnjeg spoja pojednostavljenih verzija tabela SEKTOR i RADNIK pojavljuju se i podaci o radniku *Borivoju Veljkoviću* ali i podaci o sektoru *Prodaja*.

**Primer:** Za slučaj da je neophodno kreirati spisak radnika koji rade u sektoru 'Razvoj', može se uočiti da se informacije o radnicima nalaze u tabeli RADNIK a da se naziv sektora čuva u tabelu SEKTOR. Zahvaljujući tome što u tabeli RADNIK postoji strani ključ BRSEK koji referencira primarni ključ tabele SEKTOR, moguće je ove dve tabele povezati kako bi se dobili željeni podaci. Odgovarajući SQL upita je dat u nastavku:

```
SELECT LIME, PREZIME, NAZIV
FROM RADNIK INNER JOIN SEKTOR
ON RADNIK.BRSEK = SEKTOR.SBROJ
WHERE NAZIV = 'Razvoj';
```

IME	PREZIME	NAZIV
Velibor	Jovanović	Razvoj
Marko	Petrović	Razvoj
Jelena	Janković	Razvoj
Sima	Todorović	Razvoj

Alternativni SQL upit koji vraća identičan rezultat ima sledeći oblik:

```
SELECT IME, PREZIME, NAZIV
FROM RADNIK, SEKTOR
WHERE NAZIV = 'Razvoj'
      AND RADNIK.BRSEK = SEKTOR.SBROJ;
```

Ovaj oblik upita podrazumeva da se u **WHERE** klauzuli osim **uslova selekcije** *NAZIV = 'Razvoj'* (koji selektuje vrste koje će ući u rezultujuću tabelu) uključuje i **uslov spoja** *RADNIK.BRSEK = SEKTOR.SBROJ* koji definiše kako će tabele iz **FROM** klauzule biti spojene.

**Primer:** Sledeći SQL upit za radnike ženskog pola određuje imena projekata na kojima su angažovane. Spajanje tabela u ovom upitu izvedeno je korišćenjem **WHERE** klauzule.

Za potrebe pribavljanja traženih informacija podatke izvlačimo iz tri tabela: RADNIK, RADNA i PROJEKAT. Broj tabela iz kojih izvlačimo podatke nije ničim ograničen. Potrebno je voditi računa da spojevi između tabela budu definisani na odgovarajući način kako bi dobili željene podatke. Kako je već napomenuto spoj se najčešće definiše između spoljašnjeg ključa u jednoj tabeli i primarnog ključa koji se referencira u drugoj tabeli (RADNA.Radnik i RADNIK.MatBr, RADNA.Projekat i PROJEKAT.Broj). Za spajanje različitih tabela mogu se kombinovati različiti tipovi spoja.

```
SELECT R.MATBR, R.IME, R.PREZIME, P.NAZIV
FROM RADNIK R, RADNA RN, PROJEKAT P
WHERE R.MATBR = RN.MBR
      AND RN.BRPR = P.BROJPR
      AND R.POL = 'Ž';
```

MATBR	IME	PREZIME	NAZIV
453453453	Jelena	Janković	ProizvodX
453453453	Jelena	Janković	ProizvodY
999887777	Valentina	Kovačević	Reorganizacija
987654321	Aleksandra	Petrović	Informacioni sistem
999887777	Valentina	Kovačević	Godišnji izveštaj
987654321	Aleksandra	Petrović	Godišnji izveštaj

**Napomena:** Potrebno je obratiti pažnju da je prilikom definisanja SQL upita iskorišćena mogućnost da se tabelama dodele pseudonimi radi jednostavnijeg

zapisa. U konkretnom slučaju tabelama RADNIK, RADNA i PROJEKAT su dodeljeni pseudonimi R, RN i P respektivno.

Alternativni oblikSQL naredbe koja vraća identične rezultate bi imao sledeći oblik:

```
SELECT R.MATBR, R.LIME, R.PREZIME, P.NAZIV
FROM RADNIK R INNER JOIN RADNA RN
      ON R.MATBR = RN.MBR
      INNER JOIN PROJEKAT P
      ON RN.BRPR = P.BROJPR
WHERE R.POL = 'Ž';
```

**Primer:** SQL upit koji za sve projekte locirane u Nišu prikazuje broj projekta, broj sektora koji ga izvodi i ime, prezime i datum rođenja rukovodioca ima sledeći oblik:

```
SELECT BROJPR, BRS, LIME,PREZIME, DATRODJ
FROM PROJEKAT INNER JOIN SEKTOR
      ON SBROJ = BRS
      INNER JOIN RADNIK
      ON MATBR = MATBRR
WHERE LOKPR = 'Niš';
```

BROJPR	BRS	LIME	PREZIME	DATRODJ
1	5	Sima	Todorović	08-DEC-55
3	5	Sima	Todorović	08-DEC-55
10	1	Jovan	Obradović	10-NOV-47
30	4	Aleksandra	Petrović	20-JUN-41

Alternativni oblikSQL naredbe koja vraća identične rezultate bi imao sledeći oblik:

```
SELECT BROJPR, BRS, LIME,PREZIME,DATRODJ
FROM PROJEKAT, SEKTOR, RADNIK
WHERE SBROJ = BRS
      AND MATBR = MATBRR
      AND LOKPR = 'Niš';
```

**Primer:** U nastavku je dat SQL upit koji za svakog radnika prikazuje ime i prezime kao i ime i prezime njegovog šefa:

```
SELECT X.LIME, X.PREZIME, Y.LIME, Y.PREZIME
FROM RADNIK X INNER JOIN RADNIK Y
      ON X.MATBRS = Y.MATBR;
```

LIME	PREZIME	LIME_1	PREZIME_1
Velibor	Jovanović	Sima	Todorović
Marko	Petrović	Sima	Todorović
Jelena	Janković	Sima	Todorović
Sima	Todorović	Jovan	Obradović
Aleksandra	Petrović	Jovan	Obradović
Valentina	Kovačević	Aleksandra	Petrović
Stanko	Manojlović	Aleksandra	Petrović

Ovaj SQL upit je karakterističan po tome što se javlja spoj koji je posledica postojanja rekurzivne veze. Tabela RADNIK sadrži strani ključ MATBRS koji referencira primarni ključ u istoj tabeli odnosno u tabeli RADNIK. Zbog toga se u upitu tabela RADNIK pojavljuje dva puta. Da bi to bilo moguće, obavezno je korišćenje pseudonima tj. svakoj kopiji tabele RADNIK potrebno je obezbediti poseban sinonim.

Alternativni oblik SQL naredbe koja vraća identične rezultate bi imao sledeći oblik:

```
SELECT X.LIME, X.PREZIME, Y.LIME, Y.PREZIME
FROM RADNIK X, RADNIK Y
WHERE X.MATBRS = Y.MATBR;
```

### 7.5.7 Klauzule GROUP BY i HAVING

Funkcije agregacije imaju zadatak da omoguće generisanje sumarnih informacija na osnovu podataka u relacionoj bazi podataka.

Klauzula **GROUP BY** ima zadatak da omogući grupisanje vrsta u rezultujućoj tabeli na osnovu zajedničkih vrednosti. Time se povećava vrednost funkcija agregacije jer se u kombinaciji sa **GROUP BY** klauzulom mogu primenjivati na određene grupe vrsta a ne samo na čitavu rezultujuću tabelu

Potrebno je voditi računa, da ukoliko ne postoji GROUP BY klauzula, u SELECT klauzuli nije moguće kombinovati funkcije agregacije sa imenima kolona. U nastavku je dat SQL upit koji **NE MOŽE DA SE IZVRŠI** i koji će dovesti do **POJAVE GREŠKE**.

```
SELECT MATBR, LIME, PREZIME, SUM(Plata)
FROM RADNIK;
```

Ovaj upit je moguć samo uz upotrebu GROUP BY klauzule.

Klauzula **GROUP BY** zahteva od DBMS-a da izvrši sortiranje rezultujuće tabele prema specificiranim kolonama i izvrši grupisanje vrsta koje imaju iste vrednosti za specificirane kolone. Ukoliko su prisutne funkcije agregacije one će se primeniti na tako dobijene grupe. Tek uz prisustvo **GROUP BY** klauzule moguće je u **SELECT** klauzuli kombinovati imena kolona i funkcije agregacije.

Bitno je da napomenuti da se klauzula **GROUP BY** izvršava nakon klauzule **WHERE** odnosno da se grupisanje vrši tek nakon što su određene vrste koje treba da uđu u sastav rezultujuće tabele.

**Primer:** U nastavku je dat primer SQL upita koji za svaki sektor računa broj radnika koji rade u njemu. Za grupisanje radnika po broju sektora u kome radi iskorišćena je **GROUP BY** klauzula.

```
SELECT BRSEK, COUNT(*)
FROM RADNIK
GROUP BY BRSEK;
```

BRSEK	COUNT(*)
1	1
5	4
4	3

Klauzula **HAVING** se koristi za filtriranje podataka rezultata dobijenih korišćenjem **GROUP BY** klauzule i funkcija agregacije. Ova klauzula primenjuje uslov filtriranje na formirane grupe.

**Primer:** SQL upit iz prethodnog primera je modifikovan, korišćenjem **HAVING** klauzule, tako da su prikazani podaci samo o sektorima koji imaju više od jednog radnika.

```
SELECT BRSEK, COUNT(*)
FROM RADNIK
GROUP BY BRSEK
HAVING COUNT(*) > 1;
```

BRSEK	COUNT(*)
5	4
4	3

Klauzule **GROUP BY** i **HAVING** je moguće kombinovati sa **WHERE** klauzulom. Pri tome treba voditi računa o redosledu izvršavanja (isti je redosled po kome se klauzule ređaju prilikom pisanja **SELECT** naredbe):

1. **WHERE** – primenjuje se predikat koji određuje vrste koje ulaze u sastav rezultujuće tabele
2. **GROUP BY** - vrši se grupisanje vrsta u rezultujućoj tabeli
3. **HAVING** – primenjuje se predikat koji određuje vrste koje će ostati u rezultatu upita
4. **ORDER BY** – sortiranje rezultata se vrši tek na kraju

**Primer:** SQL upit koji prikazuje brojeve sektora u kojima radi više od jednog radnika ženskog pola ima sledeći oblik:

```
SELECT BRSEK, COUNT(*)
FROM RADNIK
WHERE POL = 'Ž'
GROUP BY BRSEK
HAVING COUNT(*) > 1;
```

BRSEK	COUNT(*)
4	2

Klauzulu **GROUP BY** je moguće primeniti istovremeno na veći broj kolona. Pri tome su kriterijum za formiranje grupa zajedničke vrednosti u specificiranim kolonama. Prilikom formiranja grupa vodi se računa i o redosledu po kome su kolone za grupisanje navedene (kao da se formiraju grupe sa podgrupama u okviru njih).

**Primer:** U nastavku je dat SQL upit koji za svaki sektor prikazuje prosečnu zaradu po polovima.

```
SELECT BRSEK, POL, ROUND(AVG(PLATA), 2) AS PROSEK_PLATA
FROM RADNIK
GROUP BY BRSEK, POL
ORDER BY AVG(PLATA);
```

BRSEK	POL	PROSEK_PLATA
5	Ž	25000
4	M	25000
4	Ž	34000
5	M	35333.33
1	M	55000

**Napomena:** Prilikom korišćenja klauzule GROUP BY, sve kolone koje su navedene u klauzuli SELECT a na koje nije primenjena neka funkcija agregacije, **MORAJU BITI NAVEDENE U GROUP BY KLAUZULI**. U suprotnom SQL upit neće moći da se izvrši.

**Primer:** SQL upit koji prikazuje naziv sektora i broj radnika koji rade u njima dat je u nastavku:

```
SELECT S.SBROJ, S.NAZIV, COUNT(*) AS BROJ_RADNIKA
FROM SEKTOR S INNER JOIN RADNIK R
ON S.SBROJ=R.BRSEK
GROUP BY S.SBROJ, S.NAZIV;
```



SBROJ	NAZIV	BROJ_RADNIKA
5	Razvoj	4
4	Administracija	3
1	Uprava	1

**Primer:** Naredni SQL upit za svaki sektor daje ukupan broj radnih sati koje radnici provode na projektima za koje je taj sektor zadužen.

```
SELECT S.SBROJ, S.NAZIV, SUM(SATI) AS SATI_UKUPNO
FROM SEKTOR S INNER JOIN PROJEKAT P
    ON S.SBROJ = P.BRS
    INNER JOIN RADNIKA RN
    ON P.BROJPR = RN.BRPR
GROUP BY S.SBROJ, S.NAZIV;
```

SBROJ	NAZIV	SATI_UKUPNO
5	Razvoj	139
4	Administracija	90
1	Uprava	45

**Primer:** U nastavku je dat SQL upit koji za sve projekte koji imaju više od dva angažovana radnika prikazuje broj projekta, ime projekta i broj radnika koji na njemu rade.

```
SELECT BROJPR, NAZIV, COUNT(*)
FROM PROJEKAT, RADNIKA
WHERE BROJPR = BRPR
GROUP BY BROJPR, NAZIV
HAVING COUNT(*) > 2;
```

BROJPR	NAZIV	COUNT(*)
20	Informacioni sistem	4
10	Reorganizacija	3
2	ProizvodY	3

### 7.5.8 Kombinovanje rezultata više SQL upita

Programski jezik SQL dozvoljava kombinovanje rezultata većeg broj SQL upit korišćenjem operacija za rad sa skupovima: unija (**UNION**), presek (**INTERSECT**) i razlika (**MINUS**).

Klauzula **UNION** kombinuje rezultate dva ili više upita u jednu rezultujuću tabelu. Rezultati upita koji se kombinuju moraju imati kolone koje se slažu po broju (isti broj kolona), redosledu (odgovarajuće kolone se nalaze na istim pozicijama) i tipu (odgovarajuće kolone moraju da imaju kompatibilne tipove).

Klauzula **UNION** kombinuje rezultate dva ili više upita u jednu rezultujuću tabelu. U rezultujuću tabelu ulaze svi slogovi iz svih rezultujućih tabela. Prilikom izvršavanje operacije **UNION** elimiše se duplikati. To znači da se slog koji se

pojavljuje u više rezultujućih tabela, u rezultatu unije pojavljuje samo jednom. Zbog toga je potrebno voditi računa da rezultujuće tabele koje se spajaju moraju da uključuju atribute neophodne za jednoznačnu identifikaciju slogva.

**Primer:** Potrebno je napisati SQL upit koji vraća nazive sektora u kojima rade radnici koji se prezivaju *'Petrović'* i *'Jovanović'*. Ovaj problem može da se reši tako što se podeli na dva manja problema. Najpre se odrede nazivi sektora u kojima rade radnici koji se prezivaju *'Petrović'*, a zatim nazivi sektora u kojima rade radnici koji se prezivaju *'Jovanović'*. Kombinovanjem ova dva rezultata dobijaju se željeni podaci.

SQL upit koji određuje nazive sektora u kojima rade radnici koji se prezivaju *'Petrović'*.

```
SELECT DISTINCT NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Petrović';
```

NAZIV
Administracija
Razvoj

SQL upit koji određuje nazive sektora u kojima rade radnici koji se prezivaju *'Jovanović'*.

```
SELECT DISTINCT NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Jovanović';
```

NAZIV
Razvoj

Kombinovanjem ova dva upita korišćenjem klauzule **UNION** dobija se traženi rezultat.

```
SELECT DISTINCT NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Petrović'
UNION
SELECT DISTINCT NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Jovanović';
```

NAZIV
Administracija
Razvoj

**Napomena:** Bitno je uočiti da sektor '*Razvoj*' koji se javlja kao rezultat i jednog i drugog upita u uniji se pojavljuje samo jednom. To je posledica činjenice da unija eliminiše sve duplikate iz rezultujuće tabele.

**Primer:** SQL upit u nastavku prikazuje kako se korišćenjem klauzule **UNION** može implementirati full-outer join spoj.

```
SELECT *
FROM SEKTOR LEFT OUTER JOIN RADNIK
        ON SEKTOR.SBROJ=RADNIK.BRSEK
UNION
SELECT *
FROM SEKTOR RIGHT OUTER JOIN RADNIK
        ON SEKTOR.SBROJ=RADNIK.BRSEK;
```

**Primer:** U nastavku su dati **POGREŠNO NAPISANI UPITI**, koji ne mogu da se kombinuju jer **nema slaganja po tipu kolona**.

```
SELECT DISTINCT NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Petrović'
UNION
SELECT DISTINCT SBROJ
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Jovanović';
```

**Primer:** U nastavku su dati **POGREŠNO NAPISANI UPITI**, koji ne mogu da se kombinuju jer **nema slaganja po broju kolona**.

```
SELECT DISTINCT NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Petrović'
UNION
SELECT DISTINCT SBROJ, NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Jovanović';
```

**Primer:** U nastavku su dati **POGREŠNO NAPISANI UPITI**, koji ne mogu da se kombinuju jer **nema slaganja po redosledu i tipu kolona**.

```
SELECT DISTINCT NAZIV, SBROJ
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Petrović'
UNION
```

## Uvod u baze podataka

---

```
SELECT DISTINCT SBROJ, NAZIV
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND PREZIME = 'Jovanović';
```

Klauzula **INTERSECT** vraća samo vrste koje se javljaju u rezultujućim tabelama svih SQL upita koji se kombinuju.

**Primer:** Za kreiranje spiska radnika koji rade u sektoru 'Administracija' i koji imaju platu veću od 40000 moguće je iskoristiti dva SQL upita čiji su rezultati iskombinovani korišćenjem klauzule **INTERSECT**.

U nastavku je dat SQL upit koji vraća podatke o radnicima koji rade u sektoru 'Administracija'.

```
SELECT MATBR, LIME, PREZIME
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND NAZIV = 'Administracija';
```

MATBR	LIME	PREZIME
987654321	Aleksandra	Petrović
987987987	Stanko	Manojlović
999887777	Valentina	Kovačević

Naredni SQL upit vraća podatke o radnicima čija je plata veća od 40000.

```
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE PLATA > 40000;
```

MATBR	LIME	PREZIME
987654321	Aleksandra	Petrović
888665555	Jovan	Obradović

Traženi rezultat se dobija kombinovanjem rezultata ova dva upita korišćenjem klauzule **INTERSECT**. U rezultatu se nalaze samo vrste koji se pojavljuju u rezultujućim tabelama oba pojedinačna upita.

```
SELECT MATBR, LIME, PREZIME
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND NAZIV = 'Administracija'
INTERSECT
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE PLATA > 40000;
```

MATBR	LIME	PREZIME
987654321	Aleksandra	Petrović

Klauzula **MINUS** vraća samo one vrste koje se javljaju u rezultatu provg SQL upita ali se ne javljaju i u rezultatima ostalih SQL upita.

**Primer:** Ukoliko je potrebno pribaviti podatke podake o radnicima koji rade u sektoru 'Administracija' a nemaju platu veću od 40000, moguće je iskoristiti upit iz prethodnog primera. SQL upit iz prethodnog primera treba modifikovati tako da se umesto klauzule **INTERSECT** koristi klauzula **MINUS**.

```
SELECT MATBR, LIME, PREZIME
FROM SEKTOR, RADNIK
WHERE SBROJ = BRSEK AND NAZIV = 'Administracija'
MINUS
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE PLATA > 40000;
```

MATBR	LIME	PREZIME
987987987	Stanko	Manojlović
999887777	Valentina	Kovačević

### 7.5.9 Ugnježdjeni upiti

ORACLE SQL nudi različite mehanizme za ugnježdavanje upita. Jedan od mehanizama su takozvani **virtuelni pogledi**. Ovaj mehanizam omogućava ugnježdavanje jednog SQL upita u **FROM** klauzuli drugog SQL upita. Ugnježdjeni SQL upit se u tom slučaju ponaša kao i bilo koja druga tabela iz baze podataka.

**Primer:** Za formiranje spiska radnika čija je plata, nakon uvećanja, veća od 40000 moguće je iskoristiti mehanizam virtuelnih pogleda.

```
SELECT MATBR, LIME, PREZIME, UVECANA_PLATA
FROM (SELECT MATBR, LIME, PREZIME, PLATA + 5000 AS UVECANA_PLATA
      FROM RADNIK)
WHERE UVECANA_PLATA > 40000;
```

Rezultat ugnježdenog upita koji kreira spisak radnika sa platama uvećanim za 5000:

MATBR	LIME	PREZIME	UVECANA_PLATA
123456789	Marko	Petrović	35000
333445555	Sima	Todorović	45000
999887777	Valentina	Kovačević	30000
987654321	Aleksandra	Petrović	48000
666884444	Velibor	Jovanović	41000
453453453	Jelena	Janković	30000
987987987	Stanko	Manojlović	30000
888665555	Jovan	Obradović	60000

Spisak radnika čija je plata, nakon povećanja od 5000, veća od 40000:

MATBR	IME	PREZIME	UVECANJA_PLATA
333445555	Sima	Todorović	45000
987654321	Aleksandra	Petrović	48000
666884444	Velibor	Jovanović	41000
888665555	Jovan	Obradović	60000

Za ugnježdavanje upita moguće je iskoristiti i operator **IN**. U tom slučaju ugnježdjeni upit dinamički generiše skup vrednosti koje **IN** operator proverava.

**Primer:** Spisak radnika koji se nalaze na poziciji rukovodioca sektora može se kreirati koiršćenjem SQL upita koji je dat u nastavku.

```
SELECT MATBR, IME, PREZIME
FROM RADNIK
WHERE MATBR IN (SELECT MATBR FROM SEKTOR);
```

MATBR	IME	PREZIME
333445555	Sima	Todorović
888665555	Jovan	Obradović
987654321	Aleksandra	Petrović

Alternativno upit se može napisati i na sledeći način:

```
SELECT MATBR, IME, PREZIME
FROM RADNIK
WHERE MATBR IN (333445555, 888665555, 987654321);
```

Prednost korišćenja ugnježdenog upita je ogromna jer se skup vrednosti generiše dinamički, a samim tim će uvek uzimati u obzir ažurne podatke iz baze podataka.

Za ugnježdavanje upita moguće je iskoristiti i klauzulu **EXISTS**. Ova klauzula proverava da li ugnježdjeni upit vraća rezultujuću tabelu koja sadrži vrste ili ne.

**Primer:** SQL upit koji je dat u nastavku pribavlja podatke o radnicima koji imaju članove porodice.

```
SELECT MATBR, IME, PREZIME
FROM RADNIK
WHERE EXISTS (SELECT IME
FROM CLAN_PORODICE
WHERE CLAN_PORODICE.MATBRAD = RADNIK.MATBR);
```

MATBR	IME	PREZIME
123456789	Marko	Petrović
333445555	Sima	Todorović
987654321	Aleksandra	Petrović

Da bi operator EXISTS funkcionisao na pravilan način vrlo često se javlja potreba da ugnježdjeni upit povežemo (korelišemo) sa spoljašnjim upitom. U ovom konkretno slučaju koreliisanje se obavlja uz pomoć uslova `CLAN_PORODICE.MATBRAD = RADNIK.MATBR`. Zahvaljujući dodatnom uslovu spoljašnji i ugnježdjeni upit su povezani i ugnježdjeni upit vraća podatke samo za konkretnog radnika odnosno vraća članove porodice (ukoliko postoje) samo za konkretnog radnika. U spoljašnjem upitu se za svakog radnika uz pomoć operatora EXISTS proverava da li ugnježdjeni upit vraća rezultujuću tabelu sa vrstama odnosno da li radnik ima članove porodice ili ne.

Ukoliko se izostavi uslov koreliisanja dobio bi se upit koji **NE VRAĆA TRAŽENE PODATKE**.

```
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE EXISTS (SELECT IME FROM CLAN_PORODICE);
```

MATBR	LIME	PREZIME
123456789	Marko	Petrović
333445555	Sima	Todorović
999887777	Valentina	Kovačević
987654321	Aleksandra	Petrović
666884444	Velibor	Jovanović
453453453	Jelena	Janković
987987987	Stanko	Manojlović
888665555	Jovan	Obradović

Ukoliko se analiziraju rezultati nekoreliisanog upita, može da se zaključi da su vraćeni podaci o svim radnicima bez obzira na to da li imaju članove porodice ili ne. Problem je nepostojanje korelacije između spoljašnjeg upita i ugnježdenog upita. U ovom slučaju, kad god se proverava da li radnik ima članove porodice ili ne, ugnježdjeni upit uvek vraća isti rezultat (sve članove porodice). Zbog toga **EXISTS** ima vrednost TRUE kod svakog radnika odnosno rezultujuća tabela sadrži podatke o svim radnicima.

**Primer:** Modifikacijom SQL upita iz prethodnog primera može se dobiti upit koji pribavlja podatke o radnicima koji nemaju članove porodice.

```
SELECT MATBR, LIME, PREZIME
FROM RADNIK
WHERE NOT EXISTS (SELECT 0
FROM CLAN_PORODICE
WHERE CLAN_PORODICE.MATBRAD = RADNIK.MATBR);
```

MATBR	LIME	PREZIME
453453453	Jelena	Janković
666884444	Velibor	Jovanović
888665555	Jovan	Obradović
987987987	Stanko	Manojlović
999887777	Valentina	Kovačević

**Napomena:** Potrebno je uočiti da ugnježdeni upit vraća samo konstantu vrednost 0. Ovakvo rešenje se primenjuje jako često kada nisu neophodni podaci iz ugnježdenog upita već samo želimo da proverimo da li oni postoje ili ne.

**Primer:** Za pribavljanje podataka o radnicima koji imaju više od dva člana porodice moguće je iskoristiti SQL upit koji je dat u nastavku.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE (SELECT COUNT(*)
      FROM CLAN_PORODICE
      WHERE RADNIK.MATBR = CLAN_PORODICE.MATBRRAD) >= 2;
```

LIME	PREZIME
Marko	Petrović
Sima	Todorović

Neophodno je uočiti da su ugnježdeni i spoljašnji upit korelisani.

**Primer:** U nastavku je dat SQL upit koji kreira listu projekata u koje je uključen radnik sa imenom *'Petrović'* kao radnik ili rukovodilac sektora koji izvodi projekte.

```
SELECT DISTINCT NAZIV
FROM PROJEKAT
WHERE BROJBR IN (SELECT BROJPR
                 FROM PROJEKAT, SEKTOR, RADNIK
                 WHERE BRS = SBROJ
                 AND MATBR = MATBRR
                 AND PREZIME = 'Petrović')
OR BROJPR IN (SELECT BRPR
              FROM RADNA, RADNIK
              WHERE MBR = MATBR
              AND PREZIME = 'Petrović');
```

NAZIV
Informacioni sistem
ProizvodX
ProizvodY
Godišnji izveštaj



**Primer:** Naredni SQL upit pribavlja podatke o radnicima koji su na poziciji rukovodioca sektora i imaju bar jednog člana porodice.

```
SELECT LIME, PREZIME
FROM RADNIK
WHERE EXISTS (SELECT *
               FROM CLAN_PORODICE
               WHERE MATBR = MATBRAD)
AND EXISTS (SELECT *
            FROM SEKTOR
            WHERE MATBR = MATBRR);
```

LIME	PREZIME
Sima	Todorović
Aleksandra	Petrović

#### 7.5.10 Pseudo kolona ROWNUM

ORACLE DBMS svim rezultujućim tabelama koje se dobijaju izvršavanjem SQL upita pridružuje pseudo kolona pod nazivom **ROWNUM**. Ova pseudo kolona sadrži redni broj vrste u rezultujućoj tabeli. Brojanje počinje počinje od 1. Oracle dodeljuje redni broj nakon filtriranja koje se vrši na osnovu uslova selekcije iz **WHERE** klauzule. **ROWNUM** kolona se može koristiti po sličnom principu kao i ostale kolone iz rezultujuće tabelle.

```
SELECT ROWNUM, LIME, PREZIME
FROM RADNIK
WHERE ROWNUM < 5;
```

ROWNUM	LIME	PREZIME
1	Marko	Petrović
2	Sima	Todorović
3	Valentina	Kovačević
4	Aleksandra	Petrović

ORACLE DDBMS dodeljuje vrstama vrednosti **ROWNUM** kolone u trenutku pribavljanja. Nepoznavanje ove činjenice može da dovede do zabuna i neočekivanih rezultata.

**Primer:** SQL upit koji je dat u nastavku neće vratiti ni jednu rezultujuću vrstu zbog neadekvatnog načina na koji se koristi **ROWNUM** kolona.

```
SELECT *
FROM RADNIK
WHERE ROWNUM > 1;
```

Prilikom pribavljanja podataka, prva rezultujuća vrsta koju treba pribaviti ima **ROWNUM** = 1, pa samim tim ne zadovoljava uslov. Kao posledica dolazi do toga da vrst anije pribavljena, odnosno **ROWNUM** i dalje ostaje 1. Naredna vrsta koju treba pribaviti dobija **ROWNUM** 1 i neće biti pribavljen pošto ne zadovoljava uslov a **ROWNUM** i dalje ostaje 1. Zbog toga ni jedan red ne može da zadovolji uslov odnosno nema redova u rezultatu upita.

Pseudo kolona **ROWNUM** se često koristi za kreiranje takozvanih top-N upita odnosno SQL upita koji treba da vrte prvih N vrsta iz neke rezultujuće tabele.

**Primer:** Tipičan primer top-N upita može da bude pribavljanje podataka o pet radnika koji imaju najveću platu u firmi. Zbog neadekvatnog korišćenja **ROWNUM** kolone, upit koji je dat u nastavku **NEĆE PRIBAVITI** očekivane rezultate.

```
SELECT ROWNUM, LIME, PREZIME, PLATA
FROM RADNIK
WHERE ROWNUM <= 5
ORDER BY PLATA DESC;
```

Rezultujuća tabela sa izmešanim vrednostima ROWNUM kolone:

ROWNUM	LIME	PREZIME	PLATA
4	Aleksandra	Petrović	43000
2	Sima	Todorović	40000
5	Velibor	Jovanović	36000
1	Marko	Petrović	30000
3	Valentina	Kovačević	25000

Rezultat koji se dobija je netačan a posebno su interesantne izmešane vrednosti u ROWNUM koloni. Oov je posledica činjenice da se vrednost ROWNUM kolone dodaje prilikom pribavljanja a pre sortiranja.

U nastavku je dato SQL upit koji na pravilan način koristi ROWNUM kolonu i pribavlja tačne rezultate.

**Primer:** Podaci o pet radnika koji imaju najveću platu

```
SELECT ROWNUM, LIME, PREZIME, PLATA
FROM (SELECT LIME, PREZIME, PLATA
      FROM RADNIK
      ORDER BY PLATA DESC)
WHERE ROWNUM <= 5;
```

ROWNUM	LIME	PREZIME	PLATA
1	Jovan	Obradović	55000
2	Aleksandra	Petrović	43000
3	Sima	Todorović	40000
4	Velibor	Jovanović	36000

U unutrašnjem upitu će **ROWNUM** vrednosti biti izmešane zato što su generisane pre sortiranja. Ova **ROWNUM** kolona se **NE KORISTI**! Spoljašnji upit pribavlja redove iz rezultata ugnježdenog upita i u tom procesu se generišu nove vrednosti za **ROWNUM** kolonu i te vrednosti se koriste u spoljašnjem upitu.

**Primer:** U nastavku je dat SQL upit koji pribavlja podatke o radniku koji ima najviše članova porodice.

```
SELECT LIME, PREZIME
FROM (SELECT LIME, PREZIME, COUNT(*) AS BR_CLANOVA
      FROM RADNIK, CLAN_PORODICE
      WHERE RADNIK.MATBR = CLAN_PORODICE.MATBRAD
      GROUP BY LIME, PREZIME
      ORDER BY BR_CLANOVA DESC)
WHERE ROWNUM = 1;
```

LIME	PREZIME
Sima	Todorović

Ovako rešenje ima jedan potencijalni nedostatak. To je situacija u kojoj veći broj radnika koji zadovoljavaju uslov odnosno imaju isti najveći broj članova porodice. Zbog toga u nekim situacijama treba iskoristiti alternativno rešenje koje je dato u nastavku:

```
SELECT LIME, PREZIME, COUNT(*) AS BR_CLANOVA
FROM RADNIK, CLAN_PORODICE
WHERE RADNIK.MATBR = CLAN_PORODICE.MATBRAD
GROUP BY LIME, PREZIME
HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                  FROM RADNIK, CLAN_PORODICE
                  WHERE RADNIK.MATBR = CLAN_PORODICE.MATBRAD
                  GROUP BY LIME, PREZIME);
```

Rezultat su svi radnici koji zadovoljavaju uslov da imaju najveći broj članova porodice:

LIME	PREZIME	BR_CLANOVA
Sima	Todorović	3
Marko	Petrović	3

Kada se pogleda ova rezultujuća tabela, može se uočiti da su sada uključeni svi radnici koji zadovoljavaju uslov bez obzira na njihov broj.

### 7.5.11 Klauzula CONNECT BY...START WITH

Klauzula CONNECT BY...START WITH se koristi za pribavljanje podataka koji imaju hijerarhijske relacije tipa roditelj->dete (npr. šef->radnik, uređaj->deo).

Ovakve relcije su posledica postojanja rekurzivnih veza u okviru ER modela. Klauzula **START WITH** određuje čvor od koga se kreće sa formiranjem hijerarhije odnosno određuje koren hijerarhije. Klauzula **PRIOR** određuje uslov povezivanja između roditeljskih čvorova i čvorova dece u hijerarhiji.

**Primer:** U nastavku je dat SQL upit koji prikazuje hijerarhijsku organizaciju preduzeća.

```
SELECT LIME, PREZIME, LEVEL
FROM RADNIK
CONNECT BY PRIOR MATBR = MATBRS
START WITH LIME = 'Jovan';
```

LIME	PREZIME	LEVEL
Jovan	Obradović	1
Sima	Todorović	2
Marko	Petrović	3
Jelena	Janković	3
Velibor	Jovanović	3
Aleksandra	Petrović	2
Stanko	Manojlović	3
Valentina	Kovačević	3

U prethodnom primeru klauzulom **START WITH** je definisano da će koren hijerarhije biti radnik čije je ime '*Jovan*' a uz pomoć klauzule **PRIOR** je definisano da se hijerarhija uspostavlja od radnika na više nivou (od šefova) ka radnika na nižem nivou. Pored toga iskorišćena je i pseudo kolona **LEVEL** da se prikaže hijerarhijski nivo na kome se radnik nalazi.

**Primer:** Naredni SQL upit promenom uslova u klauzuli **PRIOR** menja način organizovanja hijerarhije. U ovom slučaju hijerarhija se organizuje od radnika ka nadređenim šefovima.

```
SELECT LIME, PREZIME, LEVEL
FROM RADNIK
CONNECT BY PRIOR MATBRS = MATBR
START WITH LIME = 'Stanko';
```

LIME	PREZIME	LEVEL
Stanko	Manojlović	1
Aleksandra	Petrović	2
Jovan	Obradović	3

## 7.6 SQL naredbe za manipulaciju podacima

Ova sekcija se odnosi na ostatak DML naredbi odnosno obradićemo naredbe koje omogućavaju modifikaciju podataka u relacionoj bazi podataka.

Postoje tri moguće operacije za modifikovanje podataka:

1. Dodavanje novih podataka (dodavanje novih vrsta u tabelu)
2. Ažuriranje podataka (izmena vrednosti kolona u postojećim vrstama tabele)
3. Brisanje podataka (brisanje vrsta iz tabele)

### 7.6.1 Dodavanje novih podataka

Za dodavanje podataka u tabelu koristi se SQL naredba **INSERT...INTO**. Naredba **INSERT...INTO** dodaje nove vrste u tabelu relacione baze podataka. Vrednosti kolona se definišu zadavanjem vrednosti u obliku konstanti ili korišćenjem rezultata SQL upita. U zavisnosti od načina zadavanja vrednosti kolona postoje različiti oblici **INSERT..INTO** naredbe.

Svi primeri u nastavku će biti razmatrani za slučaj tabele PROJEKAT. U nastavku je data **CREATE TABLE** naredba koja je iskorišćena za kreiranje ove tabele kao i test podaci dati u prethodnim odeljcima ovog poglavlja.

```
CREATE TABLE PROJEKAT
(
    NAZIV VARCHAR(25) NOT NULL,
    LOKPR VARCHAR(15) DEFAULT 'Niš',
    BROJPR NUMBER(3),
    BRS NUMBER(2) NOT NULL,
    CONSTRAINT PROJEKATPK PRIMARY KEY (BROJPR),
    CONSTRAINT NadlezanFK FOREIGN KEY (BRS)
        REFERENCES SEKTOR(SBROJ)
);
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodX	Niš	1	5
ProizvodY	Pirot	2	5
ProizvodZ	Niš	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

U nastavku je dat oblik **INSERT..INTO** naredbe koji se koristi u situacijama kada se vrednosti kolona zadaju korišćenjem konstanti.

```
INSERT INTO <ime_tabele>
[(<ime_kolone1> [,<ime_kolone2>]...)]
VALUES (<vrednost_kolone1> [{, <vrednost_kolone2>}...]);
```

Iz definicije možemo zaključiti da je lista kolona opcionalna. Ukoliko je lista kolona izostavljena, u listi vrednosti kolona moraju se navesti vrednosti za svaku kolonu koja postoji u tabeli u koju se dodaje nova vrsta. U tom slučaju lista vrednosti kolona mora da odgovara redosledu kojim su kolone navedene prilikom kreiranja tabele (u **CREATE TABLE** naredbi). Takođe, vrednosti kolona, moraju biti kompatibilne po tipu sa tipovima podataka koji su za kolone navedeni prilikom kreiranja tabele.

**Primer:** U nastavku je data SQL naredba kojom se dodaje nova vrsta u tabelu PROJEKAT. Dodaju se informacije o projektu čiji je broj 100, zove se ProizvodA, lociran je u Prokuplje i za njega je zadužen sektor čiji je broj 5.

```
INSERT INTO PROJEKAT
VALUES ('ProizvodA', 'Prokuplje', 100, 5);
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodA	Prokuplje	100	5
ProizvodX	Niš	1	5
ProizvodY	Pirot	2	5
ProizvodZ	Niš	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

Ukoliko se u **INSERT...INTO** naredbi zadaje lista kolona, moguće je promeniti redosled zadavanja kolona u odnosu na onaj koji je specificiran u **CREATE TABLE** naredbi.

**Primer:** Za dodavanje novog projekta čiji je broj 101, zove se ProjekatB, lociran je u Svrljigu i za njega je zadužen sektor broj 4 može se iskoristiti naredna SQL naredba.

```
INSERT INTO PROJEKAT
(NAZIV, BROJPR, BRS, LOKPR)
VALUES ('ProizvodB', 101, 4, 'Svrljig');
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodA	Prokuplje	100	5
ProizvodB	Svrljig	101	4
ProizvodX	Niš	1	5
ProizvodY	Pirot	2	5
ProizvodZ	Niš	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

Prilikom dodavanja nove vrste u tabelu biće proverena sva ograničenja koja su definisana nad tabelom: tip podataka, dužina polja, primarni ključ, **NOT NULL**, **CHECK...** Ukoliko bar jedno ograničenje nije zadovoljeno DBMS će prijaviti poruku o grešci i neće izvršiti naredbu.

Lista kolona u pojedinim slučajevima ne mora biti kompletna. Iz liste se mogu izostaviti kolone kod kojih nije definisano **NOT NULL** ograničenje i kolone koje imaju definisano **DEFAULT** ograničenje. Za istovljene kolone se upisuje podrazumevana vrednost definisana **DEFAULT** ograničenjem ili se upisuje **NULL** vrednost ukoliko **DEFAULT** ograničenje ne postoji.

**Primer:** U narednom primeru iz liste kolona je izbačena kolona LOKPR.

```
INSERT INTO PROJEKAT
(NAZIV, BROJPR, BRS)
VALUES ('ProizvodC', 102, 5);
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodA	Prokuplje	100	5
ProizvodB	Svrljig	101	4
ProizvodC	Niš	102	5
ProizvodX	Niš	1	5
ProizvodY	Pirot	2	5
ProizvodZ	Niš	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

Pošto je u tabeli za kolonu LOKPR definisano **DEFAULT** ograničenje, prilikom dodavanja nove vrste u tu kolonu će biti upisana vrednost 'Niš'. Da **DEFAULT** ograničenje ne postoji u kolonu Lokacija bi bila upisana vrednost **NULL**. Za slučaj da je nad kolonom LOKPR definisano **NOT NULL** ograničenje, DBMS bi generisao odgovarajuću grešku i naredba za dodavanje novog projekta ne bi imala efekta.

Ukoliko se vrednosti kolona zadaju korišćenjem upita, naredba **INSERT...INTO** ima nešto drugačiji oblik.

```
INSERT INTO <ime_tabele>
[(<ime_kolone1> [,<ime_kolone2>]...)]
<upit>;
```

U ovom slučaju za listu kolona važe ista pravila kao i u prethodnim situacijama. Vrednosti kolona se sada ne zadaju kako konstante već se zadaju kao rezultujuća tabela nekog upita. Redosled i tip kolona u rezultujućoj tabeli upita mora da odgovara redosledu i tipu kolona u listi.

**Primer:** U nastavku je dat primer **INSERT...INTO** naredbe koja koristi SQL upit da bi u fiktivnu tabelu **PROJEKAT\_NEW** dodala podatke iz tabele **PROJEKAT**.

```
CREATE TABLE PROJEKAT_NEW
(
    BROJ NUMBER(3),
    NAZIV VARCHAR(25) NOT NULL,
    LOKACIJA VARCHAR(15)
);
INSERT INTO PROJEKAT_NEW
(BROJ, NAZIV, LOKACIJA)
SELECT BROJPR, NAZIV, LOKPR
FROM PROJEKAT;
```

**Primer:** U nastavku je dat SQL upit za kreiranje nove tabele na osnovu podataka koji vraća naredba SELECT. Bitno je uočiti da kreirana tabela nema nijedno ograničenja (ni primarni ključ).

```
CREATE TABLE SEK_INFO
AS SELECT NAZIV, COUNT(*), AVG(PLATA)
FROM SEKTOR, RADNIK
WHERE BRSEK = SBROJ
GROUP BY NAZIV;
```

### 7.6.2 Ažuriranje podataka

Za ažuriranje podataka se koristi SQL naredba **UPDATE...SET**. Osnovni oblik ove komande je dat u nastavku.

```
UPDATE <ime_tabele>
SET <ime_kolone> = <izraz> [,<ime_kolone> = <izraz>...]
[WHERE <uslov>];
```

Klauzula **SET** definiše u kojoj se koloni menja vrednost definisana zadatim izrazom. Izraz može biti konstantna vrednost, vrednost nekog izraza ili vrednost koju vraća SQL upit.

Ako se navede **WHERE** klauzula, ažuriranje se vrši samo za kolone koje ispunjavaju navedenost nekog navedenog uslova.

**Primer:** U nastavku je dat SQL upit koji lokacije projekta čiji je broj 101 menja na vrednost 'Beograd'.

```
UPDATE PROJEKAT
SET LOKPR = 'Beograd'
WHERE BROJPR = 101;
```

NAZIV	LOKPR	BROJPR	BR5
ProizvodA	Prokuplje	100	5
ProizvodB	Beograd	101	4
ProizvodC	Niš	102	5
ProizvodX	Niš	1	5
ProizvodY	Pirot	2	5
ProizvodZ	Niš	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

**Napomena:** Treba biti jako oprezan prilikom korišćenja **UPDATE...SET** naredbe. UKOLIKO SE U PRETHODNOM PRIMERU IZOSTAVI WHERE



KLAUZULA ILI USLOV NIJE DOBRO DEFINISAN, LAKO MOŽEMO DA IZMENIMO I VRSTE KOJE NISMO ŽELELI DA MENJAMO, ODNOSNO DA IZGUBIMO NEKE DRAGOCENE PODATKE.

Korišćenjem UPDATE...SET naredbe moguće je istovremeno menjati vrednosti većeg broja kolona.

**Primer:** U sledećem primeru svi projekti koji u nazivu sadrže reč '*Proizvod*' se premeštaju u Niš u nadležnost sektora broj 4.

```
UPDATE PROJEKAT
SET LOKPR = 'Niš', BRS = 4
WHERE NAZIV LIKE '%Proizvod%';
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodA	Niš	100	4
ProizvodB	Niš	101	4
ProizvodC	Niš	102	4
ProizvodX	Niš	1	4
ProizvodY	Niš	2	4
ProizvodZ	Niš	3	4
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

**Primer:** U nastavku je dat SQL upit kojim se projekat broj 3 prebacuje u Beograd u nadležnost sektora broj 5.

```
UPDATE PROJEKAT
SET LOKPR = 'Beograd', BRS = 5
WHERE BROJPR= 3;
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodA	Niš	100	4
ProizvodB	Niš	101	4
ProizvodC	Niš	102	4
ProizvodX	Niš	1	4
ProizvodY	Niš	2	4
ProizvodZ	Beograd	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	4
Godišnji izveštaj	Niš	30	4

**Primer:** Izmene nad podacima kojima se projekti za koje je zadužen sektor broj 4 prebacuju u nadležnost sektora '*Razvoj*'.

```
UPDATE PROJEKAT
SET BRS = (SELECT SBROJ
           FROM SEKTOR
           WHERE NAZIV = 'Razvoj')
WHERE BRS = 4;
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodA	Niš	100	5
ProizvodB	Niš	101	5
ProizvodC	Niš	102	5
ProizvodX	Niš	1	5
ProizvodY	Niš	2	5
ProizvodZ	Beograd	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	5
Godišnji izveštaj	Niš	30	5

### 7.6.3 Brisanje podataka

Za brisanje podataka iz relacione baze podataka koristi se naredba **DELETE**. U svom osnovnom obliku naredba **DELETE** ima sledeću sintaksu:

```
DELETE FROM <ime_tabele>
[WHERE <uslov>];
```

**Primer:** SQL naredba kojom se iz tabele PROJEKAT brišu podaci o svim projektima čiji se naziv završava slovom 'A' data je u nastavku.

```
DELETE
FROM PROJEKAT
WHERE NAZIV LIKE '%A';
```

NAZIV	LOKPR	BROJPR	BRS
ProizvodB	Niš	101	5
ProizvodC	Niš	102	5
ProizvodX	Niš	1	5
ProizvodY	Niš	2	5
ProizvodZ	Beograd	3	5
Reorganizacija	Niš	10	1
Informacioni sistem	Leskovac	20	5
Godišnji izveštaj	Niš	30	5

Uslov koji navedete u **WHERE** definiše kriterijume za selekciju torki koje treba obrisati iz zadate tabele. **AKO SE NE NAVEDE WHERE KLAUZULA, NAREDBA DELETE BRIŠE SVE VRSTE IZ TABELA ČIJE SE IME NAVEDE U FROM KLAUZULI.** Zbog toga treba biti jako oprezan prilikom korišćenja naredbe DELETE. Ukoliko izostavite uslov ili je uslov neadekvatno definisan može doći do trajnog brisanja podataka koje nismo želeli da obrišemo.

**Primer:** U nastavku je data SQL naredba koja briše podatke o svim projektima.

```
DELETE
FROM PROJEKAT;
```

**Primer:** SQL naredba koja je data u nastavku briše podatke o svim projektima za koje je zadužen sektor broj 1.

```
DELETE
FROM PROJEKAT
WHERE BRS = 1;
```

Primer: Za brisanje podataka o svim projektima koji su locirani u 'Beogradu' ili 'Novom Sadu' i za koje je zadužen sektor 'Administracija' može se iskoristiti SQL naredba koja je data u nastavku.

```
DELETE FROM PROJEKAT
WHERE LOKPR IN ('Beograd', 'Novi Sad')
AND BRS = (SELECT SBROJ
           FROM SEKTOR
           WHERE NAZIV = 'Administracija');
```

**Primer:** U nastavku je data SQL naredba koja briše podatke o svim projektima na kojima nije angažovan nijedan radnik.

```
DELETE FROM PROJEKAT
WHERE BROJPR NOT IN (SELECT BRPR
                    FROM RADNA);
```

Takođe, kao alternativno rešenje koje ima isti efekat moguće je iskoristiti i sledeću SQL naredbu.

```
DELETE FROM PROJEKAT
WHERE NOT EXISTS (SELECT 0
                 FROM RADNA
                 WHERE RADNA.BRPR = PROJEKAT.BROJPR);
```

Prilikom korišćenja naredbe DELETE treba voditi računa i o efektima koji mogu da se jave kao posledica postojanja ograničenja stranog ključa. Možemo za primer uzmemo tabele RADNIK i SEKTOR iz baze podataka PREDUZEĆE, i da pokušamo da obrišemo podatke o sektoru čiji je naziv 'Administracija'.

```
DELETE
FROM SEKTOR
WHERE Naziv = 'Administracija';
```

Izvršavanjem ove naredbe moguće je da se javi jedna od naredne tri situacije:

1. Ukoliko u tabeli radnik **ne postoje** radnici koji rade u sektoru 'Administracija', iz tabele SEKTOR **biće obrisani podaci** o sektoru čiji je naziv 'Administracija'.

2. Ukoliko u tabeli RADNIK **postoje** radnici koji rade u sektoru 'Administracija' i strani ključ je definisan korišćenjem opcije **ON DELETE NO ACTION**, DBMS će **prijaviti grešku i neće izvršiti brisanje**. Brisanje u ovom slučaju nije moguće jer bi u tabeli RADNIK dobili vrste koje referenciraju nepostojeći sektor čime bi bio narušen referencijalni integritet.
3. Ukoliko u tabeli radnik **postoje** radnici koji rade u sektoru 'Administracija' i strani ključ je definisan korišćenjem opcije **ON DELETE CASCADE**, iz tabele SEKTOR **biće obrisani podaci** o sektoru čiji je naziv 'Administracija' ali će i iz tabele RADNIK **biti obrisani podaci o radnicima** koji rade u sektoru 'Administracija'. Za razliku od prethodnog slučaja gde će DBMS sprečiti brisanje da ne bi došlo do narušavanja referencijalnog integriteta, **u ovom slučaju DBMS automatski briše sve podatke kod kojih može doći do narušavanja referencijalnog integriteta**. Ovo je još jedan razlog više zbog čega treba biti jako oprezan prilikom korišćenja DELETE naredbe.

## 7.7 Naredbe za rad sa pogledima

Pogled u bazi podataka predstavlja logičku tabelu koja se kreira na osnovu jedne ili više tabela ili drugih pogleda. Pogled se tretira kao virtuelna tabela koja sadrži podatke koji predstavljaju rezultat izvršavanja uskladištenog upita. Pogled ne postoji fizički, odnosno podaci se ne dupliraju.

Prilikom svakom pristupanja pogledu njegovi podaci se formiraju dinamički izvršavanjem uskladištenog upita.

Osnovni oblik naredbe za kreiranje pogleda dat je u nastavku:

```
CREATE VIEW <ime_pogled>  
AS <upit>;
```

Kreirani pogled može se obrisati korišćenjem naredne SQL naredbe:

```
DROP VIEW <ime_pogled>;
```

U nastavku su date neke od prednosti korišćenja pogleda:

- pogled može da sadrži samo podskup podataka iz tabele
- pogled može da spoji podatke iz većeg broja tabela u jednu virtuelnu tabelu

- pogled može da sadrži podatke koji predstavljaju rezultat izvršavanja funkcija (aritmetičke funkcije, funkcije za rad sa stringovima, funkcije agregacije).
- pogledi mogu da sakriju kompleksnost podataka
- povećava se bezbednost baze podataka
- sakrivaju detalje o šemi baze podataka od korisnika.

**Primer:** U nastavku je data SQL naredba koja kreira pogled RADNIK\_PROJEKAT koji sadrži imena i prezimena radnika, imena projekata na kojima rade kao i broj radnih časova koje tokom nedelje provode na tom projektu.

```
CREATE VIEW RADNIK_PROJEKAT
AS SELECT LIME, PREZIME, NAZIV, SATI
FROM RADNIK, PROJEKAT, RADNA
WHERE MATBR = MBR AND BRPR = BROJPR;
```

Nakon kreiranja, pogled se može koristiti u SQL upitima kao i bilo koja druga tabela. Naredna SQL naredba prikazuje podatke iz kreiranog pogleda RADNIK\_PROJEKAT.

```
SELECT * FROM RADNIK_PROJEKAT;
```

LIME	PREZIME	NAZIV	SATI
Marko	Petrović	ProizvodY	7
Marko	Petrović	ProizvodX	32
Sima	Todorović	Informacioni sistem	10
Sima	Todorović	Reorganizacija	10
Sima	Todorović	ProizvodZ	10
Sima	Todorović	ProizvodY	10
Valentina	Kovačević	Godišnji izveštaj	30
Valentina	Kovačević	Reorganizacija	10
Aleksandra	Petrović	Godišnji izveštaj	20
Aleksandra	Petrović	Informacioni sistem	15
Velibor	Jovanović	ProizvodZ	40
Jelena	Janković	ProizvodY	20
Jelena	Janković	ProizvodX	20
Stanko	Manojlović	Informacioni sistem	5
Stanko	Manojlović	Reorganizacija	25
Jovan	Obradović	Informacioni sistem	10

**Primer:** SQL naredba koja je data u nastavku kreira pogled SEK\_INFO koji sadrži informacije o nazivu sektora, o broju radnika koji rade u sektoru i prosečnoj zaradi radnika u sektoru.

```
CREATE VIEW SEK_INFO
AS
SELECT NAZIV NAZIV_SEKTORA,
COUNT(*) BROJ_RADNIKA,
```

## Uvod u baze podataka

---

```
AVG (PLATA) PROSECNA_PLATA
FROM SEKTOR, RADNIK
WHERE BRSEK = SBROJ
GROUP BY NAZIV;
```

NAZIV_SEKTORA	BROJ_RADNIKA	PROSECNA_PLATA
Administracija	3	31000
Uprava	1	55000
Razvoj	4	32750

Ukoliko SQL upit sadrži kolone čija vrednost se računa a kojima nije dodeljen pseudonim, prilikom kreiranja pogleda je takvim kolonama neophodno dodeliti ime. U tom slučaju bi se za prethodni primer isoristila alternativna naredba data u nastavku.

```
CREATE VIEW SEK_INFO
(NAZIV_SEKTORA, BROJ_RADNIKA, PROSECNA_PLATA)
AS
SELECT NAZIV, COUNT(*), AVG (PLATA)
FROM SEKTOR, RADNIK
WHERE BRSEK = SBROJ
GROUP BY NAZIV;
```

Pogledi se mogu koristiti za ažuriranje podataka u tabelama koje se nalaze u pozadini (u **FROM** klauzuli uskladištenog upita). Da bi ažuriranje podataka korišćenjem tabela bilo moguće, DBMS mora da bude u stanju da mapira strukturu pogleda na strukturu tabela u pozadini (kolone i vrste). Za ažuriranje podataka korišćenjem pogleda koriste se standardne SQL naredbe za ažuriranje podataka: **INSERT...INTO**, **UPDATE...SET**, **DELETE FROM**. Pri tome treba voditi računa da nije moguće ažuriranje vrednosti kolona pogleda koje su dobijene kao rezultat SQL funkcija (aritmetičkih funkcija, funkcija za rad sa stringovima ili funkcija agregacije).

**Primer:** U nastavku je data SQL naredba koja kreira pogled RAD\_SEK\_1 koji sadrži matične brojeve, puna imena i plate svih radnika koji rade u sektoru 1.

```
CREATE VIEW RAD_SEK_1
AS
SELECT MATBR, LIME || ' ' || PREZIME PUNO_IME, PLATA
FROM RADNIK
WHERE BRSEK = 1;
```

Ovako kreiran pogled se može iskoristiti za ažuriranje tabele RADNIK na osnovu koje je pogled kreiran. U nastavku je data SQL naredba koja ažurira platu radnika korišćenjem kreiranog pogleda.

```
UPDATE RAD_SEK_1
SET PLATA = 1.1 * PLATA;
```

**Napomena:** Potrebno je uočiti da se kreirani pogled ne može iskoristiti za ažuriranje podataka o punom imenu radnika.

## 7.8 Naredbe za rad sa indeksima

Indeks je struktura podataka koja poboljšava performanse pretraživanje u relacionoj bazi podataka i najčešće se implementiraju kao stabla traženja. Mogu se kreirati nad jednom ili većim brojem kolona tabele relacione baze podataka. Indeks sadrži kopiju vrednosti kolona nad kojima se kreira (ključevi indeksa). Pojedini DBMS-ovi omogućavaju da se indeksi kreiraju i nad rezultatima operacija nad kolonama tabela. U opštem slučaju, indeksi se ne mogu kreirati nad pogledima.

Svaki indeks fizički postoji odnosno neophodan je prostor na disku u kome će se čuvati struktura indeksa.

Prilikom kreiranja primarnog ključa automatski se kreira i indeks nad kolonama koje čine primarni ključ. Svi ostali indeksi moraju da se kreiraju eksplicitno. Obično se indeksiraju strani ključevi u tabeli (da bi se ubrzale operacije spajanja tabela) i kolone (ili grupe kolona) koje se koriste za postavljanje uslova u upitima.

Osnovna namena indeksa je da ubrzaju operacije za pretragu podataka. Međutim, sa druge strane indeksi usporavaju operacije ažuriranja podataka. Zbog toga nije dobro rešenje kreirati indekse nad svim kolonama u bazi podataka. Jako je bitno da se pravilno izbalansira korišćenje indeksa u bazi podataka kako bi se postigli optimalni rezultati.

U SELECT naredni nije potrebno posebno specificirati indekse. Analizom SQL upita DBMS sam određuje koje indekse treba koristiti za optimalno izvršavanje upita.

U nastavku je dat osnovni oblik SQL naredbe za kreiranje indeksa:

```
CREATE [UNIQUE] INDEX <ime_indeksa>
ON {ime_tabele (ime_kolone[ASC I DESC]
               [, ime_kolone[ASC I DESC]]...)};
```

Opcija **UNIQUE** se koristi kada se kreira indeks koji nameće i ograničenje jedinstvenost, odnosno zahteva da svi ključevi indeksa (kombinacija vrednosti kolona koje čine indeks) bude jedinstvena. Indeksi koji se kreiraju prilikom kreiranja primarnog ključa imaju uključenu ovu opciju.

Sintaksa SQL naredbe za brisanje indeksa:

```
DROP INDEX <ime_indeksa>;
```

**Primer:** SQL naredba koja kreira indeks nad kolonom lokacija projekta.

```
CREATE INDEX PROJEKAT_LOK_IND ON PROJEKAT (LOKPR);
```

**Primer:** SQL naredba koja kreira indeks nad kolonama ime i prezime radnika projekta.

```
CREATE INDEX RADNIK_IME_IND ON RADNIK (IME DESC, PREZIME ASC);
```

**Primer:** SQL naredba koja kreira indeks za potrebe primarnog ključa (prilikom kreiranja ograničenja primarnog ključa DBMS automatski kreira i UNIQUE indeks).

```
CREATE UNIQUE INDEX RADNIK_MATBR_IND ON RADNIK (MATBR);
```



## 8 KREIRANJE KORISNIČKIH APLIKACIJA: ADO.NET

### 8.1 Osnove ADO.NET-a

Većina trenutno prisutnih korisničkih aplikacija vrši obradu podataka, a najčešći način skladištenja i korišćenja ovih podataka je preko baza podataka. U najjednostavnijem slučaju, aplikacije koje pristupaju bazi podataka omogućavaju svojim korisnicima da pretražuju podatke i prikazuju ih u tabelarnom obliku. Za pristup podacima projektanti aplikacija mogu koristiti različite tehnologije.

U prošlosti je razvijeno više standardnih interfejsa za pristup bazama podataka. Svaki sistem za upravljanje bazama poseduje sopstveni programski interfejs (eng. *application programming interface-API*) čijim korišćenjem je moguće iz programskog koda odnosno iz aplikacije, vršiti manipulaciju podacima u bazi podataka. Programski interfejs (API) predstavlja kolekciju objekata i metoda koji omogućavaju pozivanje funkcija DBMS-a iz programskog koda. Svaki DBMS poseduje svoj programski interfejs pa je bilo neophodno razviti standarde za pristup bazama podataka kako projektanti aplikacija ne bi morali da koriste različite interfejse u zavisnosti od konkretnog DBMS-a koji koriste.

Open Database Connectivity (ODBC) standard je razvijen sa ciljem da obezbedi načine za manipulaciju podacima u relacionim bazama podataka koji bi bili nezavisni od konkretnog DBMS-a. Microsoft je razvio OLE DB, objektno-orijentisani interfejs koji enkapsulira funkcionalnosti servera baza podataka. OLE DB je razvijen ne samo za relacione baze podataka već ima i mogućnost korišćenje drugih tipova podataka. Ovaj interfejs nisu mogli koristiti projektanti koji su svoje aplikacije razvijali korišćenjem Visual Basic-a i script jezika pa je Microsoft razvio Active Data Object (ADO) interfejs. ADO koristi funkcionalnosti OLE DB interfejsa i može biti korišćen iz bilo kog programskog jezika.

ADO.NET je naslednik ADO-a i deo je Microsoft-ove .NET platforme. ADO.NET ima niz karakteristika koje ga razlikuju u odnosu na prethodne tehnologije za pristup podacima. Funkcionalnosti ADO.NET-a baziraju se na korišćenju novog objekta pod nazivom DataSet. DataSet predstavlja lokalnu kopiju podataka pribavljenih iz baze podataka i može sadržati više od jedne tabele. Verovatno najbitnija karakteristika ovog objekta je činjenica da pruža mogućnost

manipulacije nad podacima bez potrebe da veza sa bazom podataka bude u svakom trenutku otvorena. Prethodne tehnologije za pristup podacima su pretpostavljale da je veza sa bazom podataka aktivna za vreme izvršenja koda koji vrši obradu podataka. Stalno aktivna konekcija dozvoljavala je trenutne izmene podataka i nadgledanje promena za vreme izvršenja koda. Problem je predstavljao ograničeni broj konekcija koje je server baze podataka mogao da pruži korisnicima pa su nakon zauzeća svih dostupnih konekcija ostali korisnici morali da čekaju da se neka od konekcija oslobodi.

ADO.NET ima u potpunosti drugačiji pristup u odnosu na prethodnike. Konekcija sa bazom podataka se i dalje kreira ali je moguće mnogo ranije osloboditi konekciju i učiniti je dostupnom ostalim korisnicima. Razlog je mogućnost pribavljanja kopije podataka iz baze i skladištenja ovih podataka u DataSet objektu. Nakon pribavljanja podataka, moguće je zatvoriti konekciju pre početka obrade podataka. Naravno, nakon završetka obrade podataka izmenjena je jedino lokalna kopija podataka pa je neophodno ponovo otvoriti konekciju ka bazi podataka kako bi bilo moguće snimiti izmene.

## 8.2 ADO.NET Data Provider-i

Za razliku od ADO-a, ADO.NET ne poseduje jedinstveni skup tipova objekata koji komuniciraju sa različitim sistemima za upravljanje bazama podataka (DBMS). ADO.NET poseduje više skupova tipova objekata koji komuniciraju sa DBMS-im. Ove skupove tipova objekata nazivamo data provider-ima. Svaki od data provider-a optimizovan je za interakciju sa konkretnim DBMS-om. Prednost ovakvog pristupa je mogućnost pojedinačnih data provider-a da imaju mogućnost manipulacije objektima koji su specifični za posmatrani DBMS. Još jedna od prednosti je način komunikacije između data provider-a i DBMS-a. Naime, s obzirom da je svaki data provider optimizovan za rad sa konkretnim DBMS-om, on komunicira direktno sa DBMS-om tj. ne postoji međusloj koji bi prilagodio zahteve korisnika konkretnom DBMS-u.

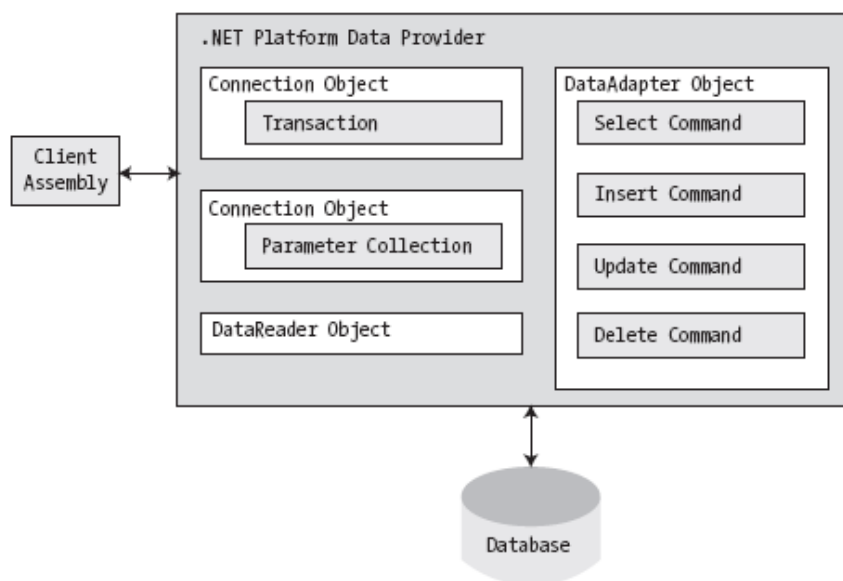
Data provider je najlakše posmatrati kao skup tipova objekata definisanih u određenom prostoru imena tipova (eng. *namespace*) koji imaju mogućnost direktne komunikacija sa konkretnim DBMS-om. Svaki od data provider-a poseduje skup klasa koje omogućavaju izvršenje osnovnih funkcionalnosti. Sve klase konkretnih data provider-a imaju zajedničke roditeljske klase tj. izvedene su iz istog skupa klasa i interfejsa, koje se nalaze u System.Data.Common prostoru imena odnosno u System.Data prostoru imena. Osnovni objekti ADO.NET Data Provider tipa objekta prikazani su u nastavku, u tabeli 3.

Iako će se imena konkretnih objekata svakog od konkretnih Data Provider objekata razlikovati (npr. SqlConnection, OracleConnection, OdbcConnection ili MySqlConnection), svi su izvedeni iz iste klase i implementiraju iste interfejse pa je nakon savladavanja korišćenja jednog od data provider-a relativno jednostavno

koristiti sve ostale. Struktura data provider-a .NET platforme prikazana je na slici 8-1.

**Tabela 3: Struktura data provider-a .NET platforme**

Objekat	Roditeljska klasa	Implementira interfejs	Značenje
Connection	DbConnection	IDbConnection	Omogućava otvaranje i zatvaranje konekcije ka bazi podataka
Command	DbCommand	IDbCommand	Predstavlja SQL upit ili uskladištenu proceduru. Omogućava pristup DataReader objektu konkretnog data provider-a
DataReader	DbDataReader	IDataReader, IDataRecord	Omogućava čitanje podataka korišćenjem kursora na serverskoj strani
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Prenosi DataSet objekte između klijenta i izvora podataka. Posедуje konekciju i skup od četiri osnovne operacije za selektovanje, dodavanje, izmenu i brisanje podataka u izvoru podataka
Parameter	DbParameter	IDataParameter, IDbDataParameter	Predstavlja imenovani parametar u parametrizovanom upitu
Transaction	DbTransaction	IDbTransaction	Enkapsulira transakciju baze podataka



**Slika 8-1. Struktura Data Provider-a .NET platforme**

Microsoft .NET platforma poseduje niz ugrađenih data provider-a za različite DBMS-ove. Spisak ugrađenih data provider-a prikazan je u nastavku u tabeli 4.

**Tabela 4: Spisak ugrađenih data provider-a**

Data Provider	Prostor imena tipova (namespace)	Biblioteka
OLE DB	System.Data.OleDb	System.Data.dll
Microsoft SQL Server	System.Data.SqlClient	System.Data.dll
Microsoft SQL Server Mobile	System.Data.SqlServerCe	System.Data.SqlServerCe.dll
ODBC	System.Data.Odbc	System.Data.dll
Oracle	System.Data.OracleClient	System.Data.OracleClient.dll

Pored data provider-a koji su ugrađeni u .NET platformu, moguće je koristiti data provider-e koje obezbeđuju pojedinačni proizvođači besplatnih ili komercijalnih DBMS-ova poput SQLite, DB2, MySQL, PostgreSQL ili Sybase.

Pored predstavljenih prostora imena, .NET platforma poseduje skup dodatnih prostora imena koji svojim funkcionalnostima pripadaju skupu ADO.NET prostora imena. Neki od ovih prostora imena prikazani su u nastavku u tabeli 5.

**Tabela 5: Spisak prostora imena ADO.NET-a**

Prostor imena tipova (namespace)	Značenje
System.Data	Definiše osnovne ADO.NET tipove koje koriste svi data provider-i
System.Data.Common	Sadrži tipove koje dele svi ADO.NET data provider-i
System.Data.Sql	Sadrži tipove koji omogućavaju otkrivanje instanci MS SQL Server-a u lokalnoj mreži
System.Data.SqlTypes	Sadrži tipove podataka koje koristi Microsoft SQL Server

Od svih ADO.NET prostora imena (namespace-a), System.Data je najopštiji. Svaka aplikacija koja želi da pristupa podacima korišćenjem ADO.NET-a mora koristiti klase definisane ovim prostorom imena. System.Data sadrži klase koje su zajedničke za sve data provider-e. U nastavku, u tabeli 6, prikazani su osnovni članovi System.Data prostora imena.

**Tabela 6: Osnovni članovi System.Data prostora imena**

Klasa	Značenje
Constraint	Predstavlja ograničenje primenjeno na DataColumn objekat
DataColumn	Predstavlja jednu kolonu DataTable objekta
DataRelation	Predstavlja roditelj/dete odnos između dva DataTable objekta
DataRow	Predstavlja jedan red u DataTable objektu
DataSet	Predstavlja lokalnu kopiju podataka u memoriji klijentskog računara koji se sastoji od niza povezanih DataTable objekata
DataTable	Predstavlja lokalnu kopiju tabele baze podataka
DataTableReader	Omogućava čitanje podataka iz DataTable objekta red po red
DataRowView	Predstavlja pogled na tabelu baze podataka i koristi se za sortiranje, filtriranje, pretraživanje i izmenu podataka

### 8.3 Direktan pristup podacima korišćenjem ADO.NET-a

Najlakši način izvršenja svih operacija nad bazom podataka je direktno izvršenje svih operacija pri čemu se ne vodi računa o lokalnim kopijama podataka. Ovakav model je najbliži tradicionalnom ADO programiranju i otklanja probleme konkurentnog izvršenja operacija nad bazom podataka koji se dešavaju kada više korisnika istovremeno izvršava operacije nad istim podskupom podataka. Ovakav način izvršenja operacija nad bazom podataka je dobro rešenje kada je potrebno pročitati podatke ili izmeniti podake u jednom redu neke od tabela relacione baze podataka. Ovakav pristup nije efikasan ukoliko je potrebno modifikovati više različitih redova iz jedne ili više tabela.

Uobičajeni redosled operacija prilikom pribavljanja podataka korišćenjem ovakvog pristupa sastoji se iz sledećih koraka:

- Kreirati **Connection**, **Command** i **DataReader** objekte
- Otvoriti konekciju
- Koristiti **DataReader** za čitanje podataka iz baze podataka
- Zatvoriti konekciju

### 8.4 Kreiranje konekcije ka bazi podataka

Pre pribavljanja ili izmene podataka neophodno je kreirati konekciju ka izvoru podataka. Broj raspoloživih konekcija je ograničen pa je potrebno držati konekciju otvorenom što je kraće moguće. Konekcija ka izvoru podataka u ADO.NET-u enkapsulirana je klasom **Connection**. Prilikom kreiranja instance klase **Connection** neophodno je definisati **ConnectionString** atribut ove klase. **ConnectionString** atribut predstavlja formatirani niz karaktera sastavljen od niza ime/vrednost parova odvojenih međusobno karakterom **‘;’**. Ovaj atribut sadrži informacije o imenu mašine kojoj pristupamo, načinu autentifikacije korisnika, imenu baze kojoj pristupamo i sl. U nastavku su dati primeri kreiranja konekcije korišćenjem Microsoft SQL Server data provider-a i Oracle data provider-a.

#### Microsoft SQL Server

```
SqlConnection conn = new SqlConnection();  
conn.ConnectionString = "Data Source=localhost;Initial  
Catalog=PREDUZECE;Integrated Security=SSPI";
```

#### Oracle

```
OracleConnection conn = new OracleConnection();  
conn.ConnectionString = "Data  
Source=160.99.9.139/gislab.elfak.ni.ac.rs;User  
Id=S1010;Password=S1010";
```

**Data Source** ukazuje na ime servera na kome se nalazi baza podataka kojoj pristupamo. **Initial Catalog** predstavlja ime baze podataka kojoj ćemo pristupati

korišćenjem kreirane konekcije. **Integrated Security** ukazuje na način autentifikacije korisnika. U posmatranom slučaju pristupa se korišćenjem Windows korisničkog naloga. Alternativno, moguće je proslediti identifikator korisnika (eng. *User Id*) i lozinku (eng. *Password*) naloga koji želimo da koristimo za pristup bazi podataka, kako je to učinjeno u primeru za korišćenje Oracle data provider-a.

Pre korišćenja konekcije neophodno je eksplicitno otvoriti konekciju pozivom metode *Open* objekta *Connection*:

```
conn.Open();
```

Nakon eksplicitnog poziva funkcije za otvaranje konekcije, otvorena je i aktivna konekcija ka bazi podataka. Nakon završetka obrade podataka neophodno je zatvoriti konekciju pozivom funkcije *Close* ili *Dispose* za zatvaranje konekcije:

```
conn.Close();
```

```
conn.Dispose();
```

Objekti klase **Connection** poseduju skup atributa kojima je moguće upravljati tokom izvršenja transakcija nad bazom podataka i pribaviti informacije vezane za izvor podataka kome se pristupa.

## 8.5 Kreiranje SQL komande

Nakon kreiranja konekcija ka bazi podataka, neophodno je izvršiti SQL naredbe za manipulaciju nad podacima. Ukoliko koristimo Microsoft SQL Server data provider, SQL naredbe se izvršavaju korišćenjem instanci klase **SqlCommand** tj korišćenjem **SqlCommand** objekta. Ukoliko koristimo Oracle data provider, SQL naredbe se izvršavaju korišćenjem instanci klase **OracleCommand** tj korišćenjem **OracleCommand** objekta. Objekti klase **SqlCommand** i **OracleCommand** predstavljaju objektno-orijentisanu reprezentaciju SQL upita, imena tabela ili uskladištenih procedura. Dakle, komande mogu biti različitog tipa. Tip komande specificiran je *CommandType* atributom objekata klase **SqlCommand** ili **OracleCommand** i može imati neku od vrednosti iz *ComandType* enumeracije.

```
public enum CommandType
{
    StoredProcedure,
    TableDirect,
    Text
}
```

Prilikom kreiranja komande moguće je navesti SQL upit kao argument konstruktora objekta ili korišćenjem **CommandText** atributa **SqlCommand** ili **OracleCommand** objekta. Prilikom kreiranja komande neophodno je navesti konekciju koja će se koristiti za izvršenje komande. Konekciju je moguće navesti

kao argument konstruktora objekta ili korišćenjem **Connection** atributa **SqlCommand** ili **OracleCommand** objekta.

#### Microsoft SQL Server

```
SqlConnection conn = new SqlConnection();
String strSQL = "Select * from RADNIK";
SqlCommand comm = new SqlCommand(strSQL, conn);

SqlCommand newComm = new SqlCommand();
newComm.Connection = conn;
newComm.CommandText = strSQL;
```

#### Oracle

```
OracleConnection conn = new OracleConnection();
String strSQL = "Select * from RADNIK";
OracleCommand comm = new OracleCommand(strSQL, conn);

OracleCommand newComm = new OracleCommand();
newComm.Connection = conn;
newComm.CommandText = strSQL;
```

Prosto kreiranje **SqlCommand** ili **OracleCommand** objekta, ili bilo koje druge komande ako se koristi neki drugi data provider, ne znači da je SQL upit sadržan u komandi automatski prosleđen na izvršenje. Objekat koji predstavlja komandu je nakon kreiranja samo pripremljen za dalju upotrebu. Najbitnije metode članice **SqlCommand** i **OracleCommand** objekta prikazane su u nastavku, u tabeli 7.

Tabela 7: Osnovni članovi System.Data prostora imena

Metoda	Značenje
Cancel()	Poništava izvršenje komande
ExecuteReader()	Povratna vrednost funkcije je DataReader objekat izabranog data provider-a
ExecuteNonQuery()	Izvršava komandu od koje se ne očekuje da kao povratne vrednosti daje podatke
ExecuteScalar()	Varijanta ExecuteNonQuery() koja kao povratnu vrednost vraća jedan podatak

## 8.6 Korišćenje DataReader objekta

Nakon uspostavljanja konekcije ka bazi podataka i kreiranja SQL upita, neophodno je izvršiti kreirani upit kako bi se pribavili podaci. Ukoliko se koristi SQL Server data provider ili Oracle data provider, najlakši i najbrži način za pribavljanje podataka je korišćenje **DataReader** objekata. **DataReader** objekti mogu samo da čitaju podatke i to red po red unapred. Imajući ovo u vidu, jasno je

da je upotreba **DataReader** objekata korisna samo u situacijama kada je kreirani upit SELECT SQL naredba. **DataReader** objekti posebno su korisni kada je potrebno brzo iterirati kroz veliku količinu podataka pri čemu nije potrebno posedovati lokalne kopije jednom očitanih podataka. **DataReader** objekti kreiraju se pozivom **ExecuteReader()** metode kreirane komande. Nakon kreiranja **DataReader** objekta, jedan red rezultujuće tabele podataka pribavlja se korišćenjem **Read()** metode.

Ukoliko pretpostavimo da je kreirana konekcija ka bazi, da bi se očitali podaci iz baze podataka neophodno je napisati SQL upit koji selektuje potrebne informacije, kreirati komandu koja će izvršiti upit i kreirati **DataReader** objekat koji će pribaviti podatke. U nastavku će biti prikazan deo programskog koda koji izvršava selektovanje svih redova iz tabele RADNIK u bazi podataka PREDUZECE.

### Microsoft SQL Server

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=localhost;User
Id=admin;Password=admin;Initial Catalog=PREDUZECE;";
String strSQL = "Select * from RADNIK";
SqlCommand comm = new SqlCommand(strSQL, conn);
SqlDataReader reader;
reader = comm.ExecuteReader();
while(reader.Read())
{
    //programski kod koji vrši obradu pribavljenih podataka
}
```

### Oracle

```
OracleConnection conn = new OracleConnection();
conn.ConnectionString = "Data
Source=160.99.9.139/gislab.elfak.ni.ac.rs;User
Id=S1010;Password=S1010; ";
String strSQL = "SELECT * FROM RADNIK";
OracleCommand comm = new OracleCommand(strSQL, conn);
OracleDataReader reader;
reader = comm.ExecuteReader();
while(reader.Read())
{
    //programski kod koji vrši obradu pribavljenih podataka
}
```



Pored izvršenja SQL upita koji vrše selektovanje podataka, moguće je izvršiti SQL upite koji vrše dodavanje, izmenu ili brisanje podataka iz baze podataka. U nastavku će biti prikazan deo programskog koda koji vrši dodavanje novog radnika, izmenu podataka o radniku i briše podatke o radniku u tabeli RADNIK baze podataka PREDUZECE.

### Microsoft SQL Server

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=localhost;User
Id=admin;Password=admin;Initial Catalog=PREDUZECE;";
try
{
    conn.Open();
    //dodavanje novog radnika
    String strSQL1 = "Insert into RADNIK values
('123456789','Marko',          'J', 'Petrovic',
'1/9/1965','Obilićev Venac', 'M', '30000')";
    SqlCommand comm1 = new SqlCommand(strSQL1, conn);
    comm1.ExecuteNonQuery();
    //izmena podataka o radniku
    String strSQL2 = "Update RADNIK set Plata=50000 where
MatBr='123456789'";
    SqlCommand comm2 = new SqlCommand(strSQL2, conn);
    comm2.ExecuteNonQuery();
    //brisanje podataka o radniku
    String strSQL3 = "Delete from RADNIK where
MatBr='123456789'";
    SqlCommand comm3 = new SqlCommand(strSQL3, conn);
    comm3.ExecuteNonQuery();
}
catch (Exception exc)
{
}
finally
{
    conn.Close();
}
```

## Oracle

```
OracleConnection conn = new OracleConnection();
conn.ConnectionString = "Data
Source=160.99.9.139/gislab.elfak.ni.ac.rs;User
Id=S1010;Password=S1010; ";
try
{
    conn.Open();
    //dodavanje novog radnika
    String strSQL1 = "INSERT INTO RADNIK VALUES
('123456789','Marko',          'J', 'Petrovic',
'1/9/1965','Obilićev Venac', 'M', '30000')";
    OracleCommand comm1 = new OracleCommand(strSQL1, conn);
    comm1.ExecuteNonQuery();
    //izmena podataka o radniku
    String strSQL2 = "UPDATE RADNIK SET PLATA=50000 WHERE
MATBR='123456789'";
    OracleCommand comm2 = new OracleCommand(strSQL2, conn);
    comm2.ExecuteNonQuery();
    //brisanje podataka o radniku
    String strSQL3 = "DELETE FROM RADNIK WHERE
MATBR='123456789'";
    OracleCommand comm3 = new OracleCommand(strSQL3, conn);
    comm3.ExecuteNonQuery();
}
catch (Exception exc)
{
}
finally
{
    conn.Close();
}
```

Često je neophodno parametrizovati upite koji se prosleđuju bazi podataka na osnovu podataka koje su korisnici uneli u aplikaciju. Kako bi prosleđivanje podataka koje su korisnici uneli bilo što sigurnije, ADO.NET poseduje mogućnost kreiranja parametrizovanih komandi. Svaka ADO.NET komanda poseduje kolekciju individualnih parametara. Svaki od parametara predstavlja nezavisni objekat koji je moguće kreirati i dodati u kolekciju ADO.NET **Command** objekta. Parametri zauzimaju određeno mesto u SQL upitu koji komanda izvršava. Kako bi se ukazala pozicija u SQL upitu na kojoj će se naći određeni parametar, koristi se

prefix “@” praćen imenom odgovarajućeg parametra u slučaju korišćenja Microsoft SQL Server data provider-a, odnosno prefix “:” u slučaju korišćenja Oracle data provider-a. U nastavku će biti prikazan primer kreiranja i izvršenja parametrizovane komande. U slučaju korišćenja SQL Server data provider-a, parametri su instance klase **SqlParameter**, dok su u slučaju korišćenja Oracle data provider-a parametri instance klase **OracleParameter**.

### Microsoft SQL Server

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=localhost;User
Id=admin;Password=admin;Initial Catalog=PREDUZECE;";
//kreiranje parametra
SqlParameter param = new SqlParameter();
param.ParameterName = "@parameter";
param.Value = "Marko";
param.SqlDbType = SqlDbType.NVarChar;
//kreiranje parametrizovanog upita
String strSQL = "Select * from RADNIK where
Ime=@parameter";
SqlCommand comm = new SqlCommand(strSQL, conn);
//dodavanje parametra u kolekciju parametara komande
comm.Parameters.Add(param);
try
{
    conn.Open();
    SqlDataReader dr = comm.ExecuteReader();
    while (dr.Read())
    {
        //obrada podataka
    }
}
catch (Exception exc)
{
}
finally
{
    conn.Close();
}
```

## Oracle

```
OracleConnection conn = new OracleConnection();
conn.ConnectionString = "Data Source=160.99.9.139-
gislab.elfak.ni.ac.rs;User Id=S1010;Password=S1010; ";
//kreiranje parametra
OracleParameter param = new SqlParameter();
param.ParameterName = "parameter";
param.Value = "Marko";
param.OracleDbType = OracleDbType.NVarchar2;
//kreiranje parametrizovanog upita
String strSQL = "SELECT * FROM RADNIK WHERE
Ime=:parameter";
OracleCommand comm = new OracleCommand(strSQL, conn);
//dodavanje parametra u kolekciju parametara komande
comm.Parameters.Add(param);
try
{
    conn.Open();
    OracleDataReader dr = comm.ExecuteReader();
    while (dr.Read())
    {
        //obrada podataka
    }
}
catch (Exception exc)
{
}
finally
{
    conn.Close();
}
```

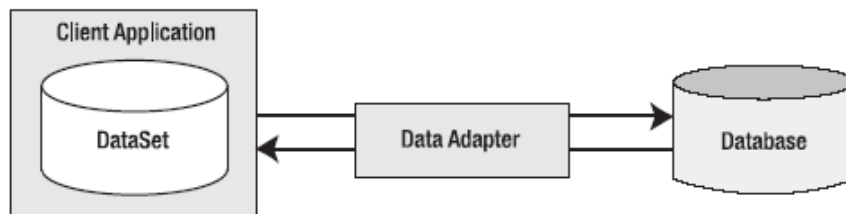
## 8.7 Korišćenje bezkonekcionog sloja ADO.NET-A za pristup podacima

Korišćenje **Connection**, **Command** i **DataReader** objekata podrazumeva da je veza ka izvoru informacija (bazi podataka) otvorena za sve vreme trajanja obrade podataka. Prednosti i mane ovakvog načina pristupanja podacima opisane su u prethodnim poglavljima. ADO.NET pored ovakvog načina pristupa podacima poseduje i drugačiji mehanizam, tzv. beskonekcion pristup podacima.

Korišćenje bezkonekcionog načina pristupa podacima zasnovano je na kreiranju lokalne kopije podataka koji se čuvaju u operativnoj memoriji klijentskog računara. Korišćenjem tipova objekata iz System.Data ADO.NET prostora imena moguće je kreirati lokalni model podataka koji će pored “sirovih” podataka posedovati veze između tabela, ograničenja primenjena na kolone, primarne ključeve, poglede i sve druge karakteristike modela podataka korišćenog za kreiranje baze podataka. Lokalni model podataka dozvoljava korisnicima kreiranje i izvršavanje upita nad lokalnim podacima, njihovo filtriranje, sortiranje i snimanje u bazu podataka.

U većini slučajeva, čak i prilikom korišćenja bezkonekcionog sloja ADO.NET-a, neophodno je kreirati objekte koji predstavljaju konekciju i komandu. Za pribavljanje i ažuriranje podataka se u slučaju bezkonekcionog pristupa podacima koristi tzv. data adapter objekat koji predstavlja instancu **DataAdapter** klase. Za razliku od direktnog pristupa podacima, podaci pribavljeni korišćenjem data adapter objekta se ne pribavljaju korišćenjem **DataReader** objekata. Data adapter objekti koriste **DataSet** objekte za prenos podataka od i ka bazi podataka. Svaki **DataSet** objekat može biti sastavljen od većeg broja **DataTable** (tabela podataka) objekata koji poseduju kolekcije **DataRow** (red tabele) i  **DataColumn** (kolona tabele) objekata.

Data adapter objekat koji se koristi upravlja konekcijom ka bazi podataka automatski. Kako bi se poboljšala funkcionalnost, data adapter objekat će konekciju ka bazi držati otvorenom minimalni period vremena. Kada se na klijentskoj strani kreira **DataSet** objekat, data adapter će prekinuti konekciju sa bazom podataka ostavljajući potpuno samostalnu kopiju podataka na klijentskom računaru. Klijentska aplikacija može nakon toga nesmetano vršiti izmene nad pribavljenom kopijom podataka. Ovaj proces prikazan je na slici 8-2.



Slika 8-2. Bezkonekcioni pristup podacima

Da bi kreirali lokalnu kopiju podataka korišćenjem data adapter-a i **DataSet** objekata, neophodno je najpre kreirati konekciju ka bazi podataka i komandu na osnovu koje će biti kreirana lokalna kopija podataka. U nastavku je prikazan primer programskog koda koji vrši učitavanje svih radnika iz tabele radnik.

### Microsoft SQL Server

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=localhost;User
Id=admin;Password=admin;Initial Catalog=PREDUZECE;";
String strSQL = "Select * from RADNIK";
SqlCommand comm = new SqlCommand(strSQL, conn);
SqlDataAdapter adapter = new SqlDataAdapter(comm);
DataSet ds = new DataSet();
try
{
    conn.Open();
    adapter.Fill(ds, "Radnici");
}
catch(Exception exc)
{
    //obrada izuzetka
}
finally
{
    conn.Close();
}
```

### Oracle

```
OracleConnection conn = new OracleConnection();
conn.ConnectionString = " Data
Source=160.99.9.139/gislab.elfak.ni.ac.rs;User
Id=S1010;Password=S1010; ";
String strSQL = "SELECT * FROM RADNIK";
OracleDataAdapter adapter = new OracleDataAdapter(strSQL,
comm);
DataSet ds = new DataSet();
try
{
    conn.Open();
    adapter.Fill(ds, "RADNICI");
}
catch(Exception exc)
{
    //obrada izuzetka
}
```

```
finally
{
    conn.Close();
}
```

Nakon kreiranja lokalne kopije podataka tj **DataSet** objekta, moguće je ažurirati pribavljene podatke. U nastavku je prikazan primer programskog koda koji vrši ažuriranje podataka o svim radnicima čije je ime Miloš postavljajući im platu na vrednost 30000.

#### Microsoft SQL Server

```
foreach(DataRow dr in ds.Tables["Radnici"].Rows)
{
    if(dr["Ime"].ToString() == "Miloš")
    {
        dr["Plata"] = 30000;
    }
}
```

#### Oracle

```
foreach(DataRow dr in ds.Tables["RADNICI"].Rows)
{
    if(dr["IME"].ToString() == "MILOŠ")
    {
        dr["PLATA"] = 30000;
    }
}
```

Moguće je takođe i obrisati neki od redova u lokalnoj kopiji podataka ili dodati novi red.

#### Microsoft SQL Server

```
int i = 0;
foreach(DataRow dr in ds.Tables["Radnici"].Rows)
{
    if(dr["Ime"].ToString() == "Miloš")
    {
        dr.Delete();
    }
}
foreach(DataRow dr in ds.Tables["Radnici"].Rows)
{
```

```
if(dr["Ime"].ToString() == "Miloš")
{
    dr["MatBr"] = 123456789;
    dr["Ime"] = "Strahinja";
    dr["Prezime"] = "Bogdanović";
    dr["SSlovo"] = "M";
    dr["DatRodj"] = "1/1/2011";
    dr["Adresa"] = "Bulevar Nemanjića 5/11";
    dr["Plata"] = 50000;
    ds.Tables["Radnici"].Rows.Add(row);
}
}
```

### Oracle

```
int i = 0;
foreach(DataRow dr in ds.Tables["RADNICI"].Rows)
{
    if(dr["IME"].ToString() == "MILOŠ")
    {
        dr.Delete();
    }
}
foreach(DataRow dr in ds.Tables["RADNICI"].Rows)
{
    if(dr["IME"].ToString() == "MILOŠ")
    {
        dr["MATBR"] = 123456789;
        dr["IME"] = "Strahinja";
        dr["PREZIME"] = "Bogdanović";
        dr["SSLOVO"] = "M";
        dr["DATRODJ"] = "1/1/2011";
        dr["ADRESA"] = "Bulevar Nemanjića 5/11";
        dr["PLATA"] = 50000;
        ds.Tables["RADNICI"].Rows.Add(row);
    }
}
```

Sve izmene učinjene na lokalnoj kopiji podataka mogu se snimiti u bazu podataka korišćenjem *Update()* metode **DataAdapter** objekta.



**Microsoft SQL Server**

```
conn.Open();
adapter.UpdateCommand = new SqlCommand("UPDATE RADNIK SET
Ime='" + dr["Ime"].ToString() + "' WHERE MatBr=" +
dr["MatBr"].ToString(), conn);
int izmenjeniRedovi = adapter.Update(ds, "Radnici");
conn.Close();
```

**Oracle**

```
conn.Open();
adapter.UpdateCommand = new SqlCommand("UPDATE RADNIK SET
IME='" + dr["IME"].ToString() + "' WHERE MATBR=" +
dr["MATBR"].ToString(), conn);
int izmenjeniRedovi = adapter.Update(ds, "RADNICI");
conn.Close();
```

ADO.NET održava informacije o originalnim i trenutnim vrednostima svih podataka koji se nalaze u **DataSet** objektu. Prilikom ažuriranja podataka, ADO.NET traži redove koji u potpunosti odgovaraju originalnim vrednostima podataka u redovima **DataSet** objekta i po pronalaženju ih zamenjuje novim redovima tj. podacima iz redova lokalne kopije. U ovom trenutku lako je uočiti da će problem nastati ukoliko neki drugi korisnik izmeni podatke pre nego što naša aplikacija snimi izmenjene podatke. Naime, drugi korisnici mogu izmeniti podatke u bazi pa prilikom upoređivanja originalnih vrednosti lokalnih kopija podataka i podataka u bazi neće doći do poklapanja. U ovakvim situacijama desiće se izuzetak. Ovakve situacije moguće je preduprediti korišćenjem **DataAdapter.RowUpdated** događaja. Ovaj događaj dešava se prilikom izvršenja svake insert, update ili delete operacije ali pre nego što se desi izuzetak pa nam daje mogućnost da sprečimo dešavanje izuzetka. **DataAdapter.RowUpdated** događaj dešava se u trenucima dok aplikacija radi sa DataSet objektom i otvara novu konekciju ka bazi podataka pa je preporučljivo uneti da programski kod koji se izvršava u ovim situacijama bude što manji.

**Zadaci za vežbu:**

1. Korišćenjem ADO.NET direktnog pritupa podacima pronaći radnika čiji je matični broj 123456789 i izmeniti njegovo ime na “Aleksandar”. Za pristup podacima koristiti SQL Server data provider. Podaci o radnicima čuvaju se u tabeli RADNIK baze podataka PREDUZECE.

2. Korišćenjem bes konekcionog sloja ADO.NET-a kreirati lokalnu kopiju podataka o svim radnicima, povećati svim radnicima platu za 15% i snimiti izmenjene podatke. Za pristup podacima koristiti SQL Server data provider. Podaci o radnicima čuvaju se u tabeli RADNIK baze podataka PREDUZECE.



## 9 LITERATURA

1. R. Elmasri & S. Navathe, *Fundamentals of Database Systems*, Pearson International Education, Addison Wesley, 6th edition, 2010.
2. S.Đorđević-Kajan, L.Stoimenov, *Baze podataka*, praktikum za vežbe na računaru, Elektronski fakultet u Nišu, Edicija pomožni udžbenici, 2004.
3. D.Kroenke, D.Auer, *Database Concepts*, Third Edition, Pearson Prentice Hall, 2008.
4. T.M.Connolly, C.E.Begg, *Database Systems: A Practical Approach to Design, Implementation and Management*, Fourth Edition, Pearson International Education, Addison Wesley, 2004.
5. Mogin P, Luković I, Govedarica M, 2000, Principi projektovanja baza podataka, Univerzitet u Novom Sadu, Novi Sad.

**Univerzitet u Nišu**  
**Elektronski fakultet**

**Leonid Stoimenov**  
**Uvod u baze podataka**

---

**Edicija: Udžbenici**  
**ISBN: 978-86-6125-099-6**  
**Niš, 2013**