Univerzitet Singidunum

Baze podataka



Mladen Veinović
Goran Šimić
Aleksandar Jevremović
Milan Tair

Beograd, 2025.

UNIVERZITET SINGIDUNUM

Mladen Veinović
Goran Šimić
Aleksandar Jevremović
Milan Tair

BAZE PODATAKA

Četvrto izdanje

Beograd, 2025.

BAZE PODATAKA

Autori:
dr Mladen Veinović
dr Goran Šimić
dr Aleksandar Jevremović
msr Milan Tair

Recenzenti:
dr Milan Milosavljević
dr Branko Kovačević
dr Vladislav Miškovic

Izdavač:
UNIVERZITET SINGIDUNUM
Beograd, Danijelova 32
www.singidunum.ac.rs

Za izdavača:
dr Milovan Stanišić

Priprema za štampu:
Jelena Petrović

Dizajn korica:
Aleksandar Mihajlović

Tiraž:
2500 primeraka

Štampa:
Caligraph, Beograd

ISBN: 978-86-7912-684-9

Copyright:
© 2025. Univerzitet Singidunum
Izdavač zadržava sva prava.
Reprodukcija pojedinih delova ili celine ove publikacije nije dozvoljena.

SADRŽAJ

Predgovor za novo izdanje.....XI
Predgovor stari XII

1. UVOD.....1
1.1 Istorijat razvoja baza podataka 3
1.2 Vrste baza podataka.....6
1.2.1 Lične baze podataka.....7
1.2.2 Baze podataka za radne grupe.....8
1.2.3 Baze podataka odeljenja.....8
1.2.4 Baze podataka organizacija.....9
1.2.5 Internet, Intranet i Extranet baze podataka.....10

2. OSNOVNI KONCEPTI I DEFINICIJE 13
2.1 Podatak 13
2.2 Informacija.....14
2.3 Metapodaci - podaci o podacima (metadata).....16
2.4 Sistem za upravljanje bazama podataka 17
2.5 Primena baza podataka 19

3. KLASIČNI SISTEMI SA DATOTEKAMA I BAZE PODATAKA.....20
3.1 Nedostaci sistema zasnovanog na datotekama 21
3.2 Pristup zasnovan na bazama podataka.....22
3.3 Prednosti pristupa zasnovanog na bazama podataka 23
3.4 Troškovi i rizici pristupa zasnovanog na bazama podataka.....25

4. MODELOVANJE 27
4.1 Razvoj konceptualnih modela.....28
4.1.1 Entiteti.....29
4.1.2 Veze između entiteta.....29
4.1.3 Troslojna arhitektura za apstrakciju podataka.....31
4.2 Modeli baza podataka.....33

4.2.1 Hijerarhijski model....34
4.2.2 Mrežni model....35
4.2.3 Relacioni model....36
4.2.4 Objektni model....39
4.3 Model objekti veze....40
4.3.1 Dijagrami objekti-veze (DOV)....41
4.3.2 Objekti....42
4.3.3 Atributi....42
4.3.4 Veze....43
4.3.5 Prošireni model objekti veze (PMOV)....45
4.3.6 Primer....46

5. STRUKTURNA SISTEMSKA ANALIZA (SSA)....49
5.1 Funkcionalna dekompozicija....51
5.1.1 Jacksonovi dijagrami....51
5.1.2 Pravila u kreiranju Jacksonovih dijagrama....52
5.2 Dijagrami tokova podataka (DTP)....54
5.2.1 Kontekstualni dijagrami....59
5.2.2 Dijagram toka podataka 0. nivoa....61
5.2.3 Kompletan primer dekompozicije kroz DTP....62
5.3 Rečnik podataka....66
5.3.1 Definisanje struktura....66
5.3.2 Definisanje polja....68
5.3.3 Definisanje domena....69

6. RELACIONI MODEL PODATAKA....71
6.1 Istorija i razvoj....71
6.2 Ključni koncepti....72
6.3 Komponente u relacionom modelu podataka....74
6.3.1 Šema relacije....74
6.3.2 Relacija....75
6.3.3 Relaciona baza podataka....75

6.3.4 Super ključ 76
6.3.5 Kandidat ključ 76
6.3.6 Primarni ključ 76
6.3.7 Spoljni ključ 77
6.3.8 NULL vrednost 77
6.4 Integritet podataka u relacionom modelu 78
6.4.1 Korisnička pravila integriteta 78
6.4.2 Identifikacioni integritet 78
6.4.3 Referencijalni integritet 79

7. RELACIONA ALGEBRA 81
7.1 Restrikcija (σ) 82
7.2 Projekcija (π) 83
7.3 Unija (∪) 86
7.4 Razlika (-) 88
7.5 Presek - ∩ 90
7.6 Dekartov proizvod (×) 91
7.7 Spajanje (><) 93
7.7.1 θ spajanje 94
7.7.2 Ekvi spajanje 94
7.7.3 Prirodno spajanje 94
7.8 Deljenje (/) 96

8. SQL 97
8.1 MySQL server 98
8.1.1 Postupak instalacije MySQL RDBMS 98
8.1.1.1 Instalacija MySQL na Windows operativnom sistemu 98
8.1.1.2 Instalacija MySQL na GNU/Linux operativnom sistemu 98
8.1.2 Interakcija sa serverom 101
8.1.2.1 Konekcija na server na Windows operativnom sistemu 101
8.1.2.2 Konekcija na server na GNU/Linux operativnom sistemu 101

8.1.3 Konzola MySQL servera....102
8.2 Sintaksa SQL jezika....102
8.2.1 Neosetljivost na velika i mala slova....102
8.2.2 Beline....103
8.2.3 Komentari....103
8.3 Tipovi podataka....105
8.3.1 Tekstualni tipovi podataka....105
8.3.2 Numerički tipovi vrednosti....106
8.3.3 Vremenski tipovi podataka....107
8.3.4 Specijalni tipovi podataka....107
8.3.5 Odabir tipa podatka....108
8.4 SQL kroz praktičan primer....109
8.4.1 Analiza i dekompozicija poslovnog procesa....109
8.4.2 Identifikovanje entiteta....110
8.4.3 Identifikovanje izričitih atributa entitetâ....111
8.4.4 Identifikovanje relacija između entiteta....112
8.4.5 Objedinjavanje entiteta i atributa....119
8.4.6 Usklađivanje sa KONVENCIJOM IMENOVANJA....119
8.4.7 Preslikavanje entiteta u relacioni model....122
8.4.8 Grafički prikaz tabela i relacija....123
8.4.9 Model baze podataka....124
8.4.10 SQL naredba za pravljenje baze podataka - CREATE TABLE....126
8.4.11 SQL naredba za odabir baze podataka - USE....126
8.4.12 SQL naredba za pravljenje tabele - CREATE TABLE....127
8.4.13 SQL naredba za brisanje tabele - DROP TABLE....132
8.4.14 SQL naredba za pravljenje indeksa - CREATE INDEX....132
8.4.15 SQL naredba za brisanje indeksa - DROP INDEX....133
8.4.16 SQL naredba za izmenu tabele - ALTER TABLE....133
8.4.17 SQL naredba za pravljenje referenci - FOREIGN KEY....136
8.4.18 Manipulacija podacima....139
8.4.19 SQL naredba za dodavanje podataka - INSERT....139

8.4.20 SQL naredba za dopremanje podataka - SELECT 147
8.4.21 Spajanje setova rezultata - UNION 154
8.4.22 Filtriranje rezultata - WHERE 154
8.4.23 Grupisanje rezultata i funkcije agregacije - GROUP BY 157
8.4.24 Drugi stepen filtriranja - HAVING 160
8.4.25 Mehanizam sortiranja rezultata - ORDER BY 161
8.4.26 Ograničavanje eta rezultata - LIMIT 162
8.4.27 Ugnježđeni upiti 164
8.4.28 Rad sa pogledima - VIEW 165
8.4.29 SQL naredba za izmenu podataka - UPDATE 168
8.4.30 SQL naredba za brisanje podataka - DELETE 170
8.4.31 Rad sa promenljivama 172
8.4.32 Rad sa transakcijama - TRANSACTION 173
8.4.33 Rad sa okidačima - TRIGGER 175
8.4.34 Ugrađene funkcije - FUNCTION 176
8.4.35 Ugrađene procedure - PROCEDURE 177
8.4.36 Zakazivanje odloženih poslova - EVENT 179
8.4.37 Često korišćene funkcije 181
8.4.38 Upravljanje korisnicima - USER MANAGEMENT 183
8.4.39 Upravljanje pravima korisnika - PRIVILEGE 185

9. RELACIJE LOŠE STRUKTURE I NORMALIZACIJA 186
9.1 Redundansa i anomalije ažuriranja 187
9.1.1 Višestruko unošenje podataka 188
9.1.2 Višestruko brisanje podataka 188
9.1.3 Višestruke izmene podataka 188
9.2 Funkcionalna zavisnost 190
9.3 Postupak normalizacije 193
9.3.1 Prva normalna forma (1NF) 194
9.3.2 Druga normalna forma (2NF) 195
9.3.3 Treća normalna forma (3NF) 196

9.3.4 Boyce-Codd normalna forma (BCNF) 196
9.4 Denormalizacija 197

10. TRANSAKCIJE 199
10.1 Definicija transakcije 200
10.2 Osobine transakcija 200
10.3 COMMIT i ROLLBACK 202
10.4 Konkurentno izvršavanje transakcija 203
10.5 Protokol zaključavanja 204
10.6 Oporavak baze podataka 206

11. BAZE PODATAKA I APLIKACIJE 207
11.1 Slojevita struktura aplikacija 208
11.1.1 Troslojni model kao osnovni model 208
11.1.2 Višeslojni modeli 209
11.1.3 Aplikacije servisi (nezavisne softverske komponente) 210
11.2 Specifičnosti pristupa BP iz različitih aplikacionih slojeva 212
11.2.1 Pristup podacima iz prezentacionog sloja 212
11.2.2 Pristup podacima iz sloja poslovne logike 218
11.2.3 Pristup iz sloja podataka (poziv ugnježdenih procedura) 220
11.3 Tehnologije koje omogućavaju razmenu podataka između BP i aplikacija 222
11.3.1 ODBC (Object Database Connectivity) 222
11.3.2 ADO (ActiveX Data Objects) 224
11.3.3 DAO (Data Access Objects) 225
11.3.4 JDBC (Java Database Connectivity) 226
11.3.5 Enterprise Java Beans 227
11.3.6 Okruženja za indirektan pristup SUBP (ORM Frameworks) 229

12. XML (I) BAZE PODATAKA 231
12.1 Opšte o XML-u 231
12.2 XML proširenje za SQL (SQL/XML) 235

12.2.1 Funkcije za kreiranje XML dokumenta 236
12.2.2 XML tip podataka u SUBP 239
12.2.3 Pravila za mapiranje 240
12.3 Manipulacija nad BP koje sadrže XML formatIrane podatke (Native XML BP) 241
12.3.1 XPath 241
12.3.2 XQuery 244

13. SKLADIŠTENJE PODATAKA (DATA WAREHOUSING) 250
13.1 Arhitektura skladišta podataka 251
13.1.1 Proširena arhitektura 252
13.2 Logički dizajn skladišta podataka 253
13.2.1 Formiranje dimenzija 253
13.2.2 Shema zvezde 253
13.2.3 Shema pahuljice 257
13.3 Izdvajanje, transport, transformacija i učitavanje podataka (ETL proces) 258
13.3.1 Izdvajanje (ekstrakcija) podataka 259
13.3.2 Transport podataka 260
13.3.3 Transformacija i učitavanje podataka 261
13.4 Promena (ažuriranje) podataka u skladištu 264
13.4.1 Primer ažuriranja tabele skladišta 265
13.5 Agregiranje podataka u skladištu 265
13.5.1 ROLLUP ekstenzija 266
13.5.2 CUBE ekstenzija 267
13.5.3 Funkcije i skupovi za grupisanje 270
13.6 Analitičke operacije 271
13.6.1 OLAP funkcije 271
13.6.2 Napredno pretraživanje podataka 272

14. NoSQL BP 274
14.1 Uvod 274

14.2 Motivi nastanka NoSQL DBMS 274
14.3 Osnovni principi NoSQL BP 277
14.3.1 Agregacioni model podataka 277
14.4 NoSQL DBMS organizacija podataka 279
14.4.1 Skladišta ključ – vrednost parova 279
14.4.2 Skladišta dokumenta 281
14.4.3 Skladišta familija kolona 282
14.4.4 Grafovska skladišta podataka 284
14.5 NoSQL DBMS – modeli distribuiranja podataka 286
14.6 Map Reduce koncept 289
14.7 Primeri NoSQL DBMS 291

15. PRIMERI ZA MODELOVANJE 292
15.1 Adresar na mobilnom telefonu 292
15.2 Biblioteka 294
15.3 Magacin 297
15.4 Banka 299
15.5 Marketinška agencija 300
15.6 Elektronsko glasanje 301
15.7 Elektronska prodavnica 304

Literatura 307

PREDGOVOR ZA NOVO IZDANJE

U ovom izdanju udžbenika Baze podataka izvršene su neophodne izmene u odnosu na prethodno izdanje, koje su nastale kako zbog konceptualnih napredaka u oblasti modelovanja baza, tako i na osnovu iskustva autora u realizaciji ovog predmeta. Osnovni tekst je unapređen, grafičke ilustracije su poboljšane, a sve uočene greške su ispravljene. Najveća izmena se odnosi na potpuno novu glavu 8, gde je kroz jedan konkretan i praktičan primer predstavljen način upotrebe upitnog jezika SQL. Pored toga, potpuno nova je i glava 14 na temu NoSQL baza podataka, koje danas dominiraju na Web-u. To su skalabilni sistemi otvorenog koda, omogućavaju manipulaciju heterogenim sadržajima koji ne mogu da se normalizuju i koji su distribuirani na Web-u. NoSQL baze imaju osnovnu prednost da im se performase ne umanju (ne smanjuje se vreme odziva) kada tokom eksploatacije dolazi do nagomilavanja podataka, što je karakteristično za podatke na Internetu. U ovom izdanju izostavljena su uputstva za korišćenje konkretnih softvera za rad sa bazama podataka, pošto se nove verzije veoma brzo uvode u operativnu upotrebu, čime osnovni tekst knjige brzo zastareva. Takođe, pojavljuju se novi softveri koji rade direktno na Web-u pozivanjem kroz brauzer, bez potrebe da se vrši instalacija.

Autori se nadaju da će tekst dat u ovom izdanju udžbenika biti prihvaćen od strane studenata i da je konkretniji i upotrebljivi za savladavanje baza podataka. Sve primedbe na tekst su dobro došle.

Beograd, septembar 2018. godine

PREDGOVOR STARI

Knjiga Baze podataka je namenjena studentima Univerziteta Singidunum za pripremu ispita iz predmeta Baze podataka, ali može biti od koristi i svima onima koji kroz praksu i teoriju savladavaju baze podataka. U toku izlaganja materije autori se nisu vezivali za konkretan softverski sistem za upravljanje bazama podataka, već su nastojali da na jednom mestu prikažu sve koncepte u bazama podataka u skladu sa savremenim kretanjima u ovoj oblasti. Dominantno su razmatrani koncepti relacionog modela koji jeste osnova za najveći broj poslovnih aplikacija.

Udžbenik je podeljen u nekoliko delova. Na početku se iznose osnovni koncepti i definicije koje se koriste u bazama podataka. Čitalac se polako vodi od klasičnih sistema za rad sa podacima, koji su zasnovani na programskim jezicima i datotekama, ka pravim bazama podataka koje se zasnivaju na sistemu za upravljanje bazama podataka. U nastavku se razmatra modelovanje i uvode različiti modeli podataka, onako kako su se istorijski pojavljivali. Baze podataka ne postoje same za sebe. One su osnova informacionih sistema, kako velikih tako i malih organizacija, a projektuju se sa ciljem dobijanja pravovremenih informacija za donošenje odluka. Zbog toga je posebna pažnja posvećena strukturnoj sistemskoj analizi, kao početnom koraku u projektovanju baza podataka. Detaljno je razmotren relacioni model podataka, a u okviru njega posebna pažnja je posvećena integritetskim komponentama, koje omogućavaju održavanje konzistentnosti jedne baze podataka. U nastavku je ukratko razmatrana relaciona algebra kao osnova za upitne jezike, a zatim su prikazane mogućnosti MySQL-a sa svim njegovim specifičnostima za rad sa relacionim bazama podataka. Na primerima relacija loše strukture diskutovani su karakteristični problemi koji se odnose na anomalije ažuriranja, a zatim je objašnjen postupak normalizacije ovakvih relacija. Detaljno je diskutovan i postupak denormalizacije koji se, uprkos teorijskim postavkama, često koristi u cilju povećanja efikasnosti posebno kod izveštavanja. Danas se baze podataka uglavnom koriste u mrežnom okruženju, gde se više transakcija konkurentno izvršava nad istom bazom podataka. Diskutovani su mehanizmi koji omogućavaju nesmetan paralelan rad više transakcija. Detaljno su razmotreni mehanizmi zaključavanja koji omogućavaju pouzdanost kod paralelnog izvršavanja više transakcija nad istom bazom podataka i istim podacima. U skladu sa razvojem savremenih Internet aplikacija i Web servisa, sistematično su prikazane različite tehnike povezivanja baza podataka i aplikacija. Posebno se razmatra raslojavanje aplikacija čime se postiže veća funkcionalnost samih aplikacija i lakše se zadovoljavaju naknadni zahtevi korisnika.

Posebno poglavlje koje ovaj udžbenik čini aktuelnim je deo oko baza podataka i XML-a (*Exten-sible Markup Language*). Usled čestih promena u preduzećima, postoji potreba i za udruživanjem informacionih sistema u čijoj osnovi su baze podataka. Format i struktura podataka u BP ostaju nepromenjeni, menja se samo način njihovog predstavljanja - njihova prezentacija van BP. Drugim rečima - različite aplikacije u okviru istog, ili različitih informacionih sistema razmenjuju podatke kao XML dokumenta, čiji je sadržaj hijerarhijski struktuiran i opisan rečnikom podataka tako da su moguće brze transformacije sadržaja između XML tekstualnog formata i formata ciljne BP. Proizvođači sistema za upravljanje BP su otišli i korak napred - omogućeno je skladištenje podataka u XML formatu, omogućeni su direktni upiti, manipulacija i transakcije nad XML formatiranim podacima. Ovakvo rešenje je prepoznato naročito kao potreba unutar kompanija (unutar istog IS) jer se izbegava dvostruka transformacija podataka (između aplikacija - učesnika u razmeni).

Na kraju, izložen je teorijski deo oko skladištenja podataka. U savremenim poslovnim sistemima postoji potreba korišćenja podataka iz različitih izvora. Kao posledica, podaci bitni za procese upravljanja u takvim složenim sistemima nisu više na jednom mestu, na jednoj BP, niti su deo istog IS. Proizvođači SUBP i poslovnih IS su podržali opisane potrebe fleksibilnim softverskim rešenjima namenjenim za podršku u odlučivanju. Ovi sistemi nisu namenjeni za transakcionu obradu podataka kao što je to slučaj konvencionalnih IS već su namenjeni za ekstrakciju, transport, transformaciju podataka iz različitih izvora, zatim njihovu analitičku obradu u realnom vremenu. Ovi procesi su obuhvaćeni jednim terminom - "Skladištenje podataka" (*Data Wareho-using*). Kao rezultat navedenih procesa dobijaju se skladišta podataka koja se razlikuju od konvencionalnih BP.

Na kraju udžbenika dati su karakteristični primeri za modelovanje i programiranje, a konstruisani su od lakših ka težim problemima. U dodatku su prikazani karakteristični softveri koji se široko koriste za modelovanje i rad sa bazama podataka.

Autori su se potrudili da izlaganje bude što jasnije, da ne opterete čitaoca kompleksnim matematičkim aparatom niti konkretnim softverskim alatima, a sve u cilju što jasnijeg predstavljanja baza podataka. Svi saveti i eventualne primedbe na tekst su dobrodošli.

Beograd, 2013. godine

Autor

1 UVOD

Baze podataka su namenjene za prikupljanje, organizovanje, čuvanje i manipulaciju podacima na osnovu kojih se dobijaju informacije koje su potrebne krajnjim korisnicima. Obavezan su deo savremenih distribuiranih aplikacija i omogućavaju nesmetano izvršavanje konkurentnih aplikacija koje rade u realnom vremenu i imaju potrebu za istovremenim pristupom podacima (unos, brisanje, menjanje, čitanje). Služe kao podrška poslovanju kao neizostavni deo informacionih sistema različite namene. Imaju razvijene mehanizme za oporavak podataka u slučajevima otkaza sistema ili narušavanja samih podataka u bazi. Bazama podataka se može upravljati na način da se različitim korisnicima mogu definisati specifične privilegije za pristup i korišćenje podataka.

Bazama podataka pristupaju udaljeni korisnici primenom različitih tehnologija kao što su Web čitači (*browser*-i), digitalni telefoni, govorni automati i sl. Zbog velike konkurencije u svim oblastima poslovanja, može se očekivati da tehnologija baza podataka dobije još veći značaj. Menadžeri traže način da iz baze podataka brže dođu do novih saznanja kako bi bili u prednosti u odnosu na svoju konkurenciju. Na primer, detaljna baza podataka o prodaji se može iskoristiti kako bi se saznalo koji kupci kupuju koje proizvode, što se koristi kao osnova za reklamu i marketinšku kampanju. Organizacije mogu da uključe u svoje baze podataka procedure koje se zovu okidači - trigeri (*alerts*) koji upozoravaju o mogućim vanrednim događajima (kao što su predstojeći nedostatak zaliha neke robe ili šansa za prodaju dodatne količine robe) i na osnovu kojih mogu nastati odgovarajuće reakcije. Mnoge organizacije danas prave posebne baze podataka koje se zovu „skladišta podataka“ (*data werehouses*) koje služe za aplikacije za podršku u odlučivanju.

Izučavanje baza podataka i sistema za upravljanje bazama podataka jesu osnova za izučavanje informacionih sistema. Stručnjak za informacione sisteme mora da bude spreman da analizira potrebe preduzeća i da dizajnira i implementira baze podataka u okviru razvoja informacionog sistema jedne organizacije. Takođe, mora da bude spreman da se konsultuje sa krajnjim korisnicima i da im pokaže kako se korišćenjem baza podataka može imati bolja podrška za odlučivanje, čime se stvara prednost nad konkurencijom. Široko rasprostranjeno korišćenje baza podataka vezanih za Internet sajtove, koji vraćaju dinamičke informacije korisnicima Web sajta, zahteva od projektanta da razume ne samo kako da poveže bazu podataka sa sajtom već i kako da je zaštiti tako da se njenom sadržaju može pristupiti, ali ne i izmeniti od strane spoljnih korisnika.

Postoji puno načina kako se može definisati baza podataka. U osnovi to je skup podataka koji su organizovani prema potrebama korisnika, koji se održavaju i koji se koriste za dobijanje informacija. Moderne baze podataka su čuvaju na računaru, ali to nije bitno za samu definiciju. Na primer, adrese poznanika i prijatelja, kolekcija

filmova na CD-ovima, telefonski imenik itd. su takođe baze podataka (mada ih većina ljudi tako ne zove). Međutim, smeštanje baze podataka na računar omogućava lakšu i bržu obradu podataka i dobijanje željene informacije. Karakterističan je primer sa telefonskim imenikom koji se nalazi na papiru. Jednostavno je pronaći telefonski broj željene osobe, ali je znatno teže pronaći ime osobe na osnovu telefonskog broja. Ako je telefonski imenik veći (više uskladištenih podataka) prethodni problem se dodatno usložnjava. Računarski zasnovane baze podataka omogućavaju jednostavno i brzo dobijanje informacija. Pored osnovnih informacija iz odgovarajuće baze podataka se mogu dobiti i posebne informacije. Na primeru telefonskog imenika mogu se izlistati podaci za sve osobe po imenu npr. Marko, mogu se izlistati sve osobe kojima telefonski broj počinje npr. sa 2, osobe kojima se telefonski broj završava sa 45 i još mnogo toga.



Slika 1.1: Baze podataka nekada i danas

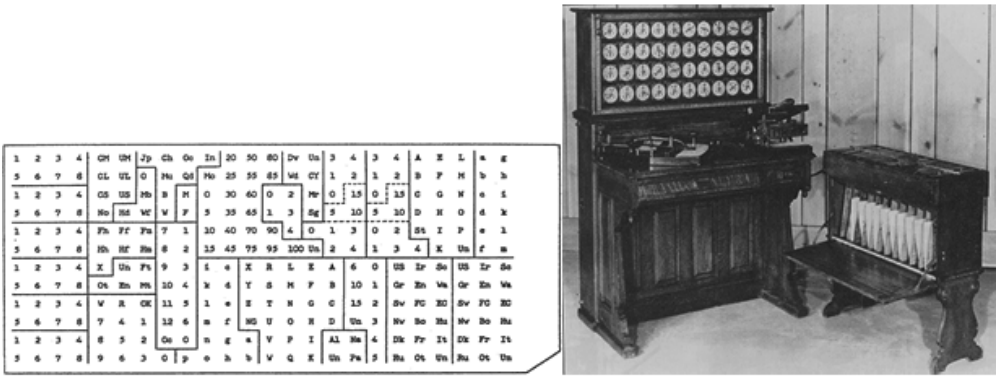
Na razvoj baza podataka presudno je uticao razvoj računara, računarskih mreža, kao i klijent/server obrade. Istraživanje, projektovanje i upotreba baza podataka su vrlo brzo pokazali niz svojih dobrih strana kao što su: smanjeni troškovi održavanja; smanjena potreba za mrežnim resursima; poboljšan integritet podataka; donošenje ispravnih odluka na osnovu objektivnih informacija, brža reakcija na tržištu, itd.

Baza podataka smeštena u računaru, predstavljena je nizom bita, organizovanih u bajtove, a sa više bajtova u odgovarajućem formatu zapisuju se vrednosti pojedinih podataka i predstavljaju jedno polje baze podataka. Niz polja se organizuje u zapise (rekorde) koji imaju značenje jer mogu da predstavljaju opis nekog objekta iz realnog sveta ili neke veze između objekata realnog sveta. Zapisi istog formata se slažu i čine datoteke, koje su fizički zapisane na disku. Baza podataka obuhvata više povezanih datoteka i može biti centralizovana na jednom računaru ili distribuirana na više međusobno udaljenih računara. Pored podataka koji su zapisani u bazi podataka, postoje i posebni podaci kojima se opisuju pojedinačne datoteke, njene osobine,

karakteristični parametri iz datoteka, uspostavljene međusobne veze, pravila koja se odnos na pojave koje postoje i u realnom svetu i sl. Takvi podaci se zovu meta-podaci (*metadata*), tj. podaci o podacima. Koriste se pri pokretanju rada sa bazom podataka, kako bi se mogli učitati svi konfiguracioni podaci odgovarajuće baze (*self-describing*).

1.1 ISTORIJAT RAZVOJA BAZA PODATAKA

Nastanak baza podataka se vezuje za *Herman*-a *Holerith*-a koji je 1884. godine prijavio patent – sistem za automatsku obradu podataka (AOP) o popisu stanovništva u SAD. Podaci na bušenim karticama su ručno unošeni u uređaj za očitavanje, a obrada podataka se odnosila na prebrojavanje. Programiranje se svodilo na izbor vrste prebrojavanja, a radilo se ručnim prespajanjem kontakata. Dotadašnja obrada podataka popisa trajala je 10-tak godina, a sa *Holerith*-ovim izumom vreme obrade bilo je smanjeno na šest nedelja. *Herman Hollerith* je osmislio ideju po kojoj se svaki stanovnik SAD predstavlja nizom od 80 karaktera – ime, godište itd. popunjenih praznim prostorima da bi se za sva imena obezbedila ista dužina, tako da baza podataka bude „poravnata“. On je uspeo da proda koncept svoje mašine i bušene kartice koje su služile za čuvanje podataka u statističkom birou SAD. Tako je popis stanovništva iz 1890. godine bio prva automatizovana baza podataka, koja se u suštini sastojala od hiljada kutija punih bušenih kartica. Od *Hollerith*-ove kompanije nastao je današnji IBM.



Slika 1.2: Izgled Hollerith-ove bušene kartice i mašine za očitavanje kartica

Nakon Drugog svetskog rata, u kompanijama i vladinim institucijama počeli su se pojavljivati prvi elektronski računari. Oni su se često koristili upravo za jednostavne linearne baze podataka, najčešće za računovodstvo. Ipak, vrlo brzo, bogati kupci su počeli da zahtevaju više od njihovih ekstremno skupih mašina. Sve je to vodilo do ranih baza podataka. Zanimljivo, ove rane aplikacije su nastavile da koriste Hollerith-ove bušene kartice, neznatno modifikovane u odnosu na originalni dizajn.

Nefleksibilnost polja iste dužine, baze podataka pokretane 80 kolonskim bušenim karticama, učinile su rane računare metom napada i šala i potpunom misterijom za običnog čoveka.

Većina prvobitnih baza podataka se odnosila na specifične programe napisane za specifične baze podataka. Za razliku od modernih sistema koji mogu biti primenjeni na potpuno različite baze podataka, ovi sistemi su bili usko povezani za bazu podataka da bi osigurali brzinu na uštrb fleksibilnosti. Sistemi upravljanja bazama podataka su se prvi put pojavili tokom 1960-tih godina i nastavili su da se razvijaju tokom sledećih decenija. U većini slučajeva, period uvođenja je dugo trajao, skoro deceniju pre navedene godine početka upotrebe. Na primer, relacioni model je prvi put definisan od strane E.F.Codd u tekstu objavljenom 1970 godine. Međutim, relacioni model nije bio široko prihvaćen sve do 1980-tih godina.

1960-te

Tokom ovog perioda, sistemi zasnovani na datotekama su i dalje imali dominantnu ulogu. Pa ipak, prvi sistemi za upravljanje bazom podataka su uvedeni u ovoj deceniji, i korišćeni su najpre samo kod velikih i složenih projekata, kao što je to bio projekat sletanja Apolo-a na Mesec. Ovaj period možemo posmatrati kao period ’dokazivanja’, u kom je demonstrirana sposobnost ovih sistema da upravljaju ogromnim količinama podataka. Takođe, prvi napor da se standardizuju preduzet je sa formiranjem DBTG grupe (*Data Base Task Group*), tokom kasnih ’60-ih godina.

1970-te

Tokom ove decenije, upotreba sistema za upravljanje bazom podataka je postajala komercijalna stvarnost. Hijerarhijski i mrežni sistemi za upravljanje podacima su uvedeni, velikim delom zbog potrebe za sistemom koji će moći da upravlja složenim strukturama podataka, kao što su računi fabrika pri nabavci sirovina, kojima je bilo izuzetno teško upravljati preko konvencionalnih metoda. Ovi modeli se i suštinski smatraju prvom generacijom sistema za upravljanje bazom podataka. Oba pristupa su se široko primenjivala, zapravo, mnogi od tih sistema su u upotrebi i danas. Pa ipak, imali su nekoliko velikih nedostataka:

- Težak pristup podacima. Za pristup i najjednostavnijim podacima bili su potrebni izuzetno složeni programi.
- Veoma ograničena nezavisnost podataka, tako da se programi nisu mogli izolovati od promena u formatu podataka.
- Nije postojala nijedna široko prihvaćena teorijska podloga za bilo koji od ovih modela, za razliku od relacionog modela podataka.

1980-te

Da bi se prevazišla ova ograničenja, E. F. Codd, kao i mnogi drugi, razvili su model relacionih podataka tokom ’70-ih godina. Ovaj model, koji se smatra drugom generacijom DBMS-a, doživeo je široku komercijalnu upotrebu u poslovnom svetu tokom ’80-ih godina. Sa relacionim modelom svi podaci su predstavljeni u formi tabele. Relativno jednostavan programski jezik četvrte generacije, nazvan SQL, korišćen je za dobijanje informacija. Ovaj model je obezbedio jednostavan pristup podacima i onim ljudima koji nisu bili programeri, prevazilazeći na taj način jednu od najvećih primedbi koja je pratila sisteme prvih generacija. Model je takođe pokazao i svoju pogodnost za komunikaciju na relaciji klijent/server, paralelni prenos podataka, i upotrebu grafičkog korisničkog interfejsa (GUI).

1990-te

Ove godine se označavaju kao nova računarska era, najpre na nivou računarske komunikacije na relaciji klijent/server, a potom sa stvaranjem skladišta za podatke i upotrebom Internet aplikacija, koji su dobijali sve više na važnosti u ovom periodu. Kao što je upravljanje podacima od strane DBMS-a postalo visoko primenjivo (npr., u računovodstvu) tokom osamdesetih godina, multimedijalni podaci (uključujući i grafiku, zvuk, slike i video zapis) su postali uobičajena stvar tokom devedesetih godina. Kako bi se izborili sa sve složenijim podacima, tokom devedesetih su uvedeni sistemi okrenuti ka objektu, koji se smatraju trećom generacijom. Zbog velike potrebe za organizacijom ogromne količine podataka kako struktuiranih, tako i ne struktuiranih podataka, ovaj i prethodni sistem su u upotrebi i danas. Neki proizvođači čak rade na razvoju kombinovanih sistema za upravljanje bazama podataka, kako bi mogli da upravljaju obema vrstama istih.

Od 2000. godine

Naravno, postavlja se pitanje u kom pravcu će da krene razvoj DBMS tehnologija u narednom periodu. Iako će nesumnjivo doći do novih iznenađenja, možemo očekivati nastavak dobro uspostavljenih trendova:

- Mogućnost upravljanja sve složenijim tipovima podataka. Ovi tipovi uključuju i više dimenzione podatke, koji su već dobili na važnosti u aplikacijama skladištenja podataka.
- Nastavak razvoja ’univerzalnih servera’. Zasnovani na sistemu treće generacije DBMs-a, to su serveri koji mogu da upravljaju širokom lepezom raznih tipova podataka, tako da budu transparentni svim korisnicima. Biće naročito važni kod Internet aplikacija.
- Dok su u potpunosti distribuirane baze podataka postale realnost, trenutni trend ka centralizaciji istih će se nastaviti. Kako se troškovi komunikacije sve više smanjuju, nasuprot porastu tipova podataka, vrednost lociranja i pristupa centralizovanoj bazi podataka takođe se smanjuje. Manji troškovi, a visoke performanse svakako ohrabruju ovaj trend.

- Skladišta sa adresiranim sadržajem će postajati sve popularnija. Sa ovakvim pristupom, korisnik može da izvuče bilo kakav podatak specifikacijom kakvu vrstu podatka želi, umesto kako da dođe do njega. Na primer, korisnik može da skenira fotografiju i da traži od kompjutera pretragu, kako bi pronašao istu takvu, ili njoj sličnu.
- Baza podataka i druge tehnologije, poput veštačke inteligencije i televizije, kao informacionog servisa, olakšaće pristup podacima neobučenim korisnicima. Na primer, korisnik će biti u mogućnosti da zahteva podatak na više jezika, a tehnologija baza podataka će da uključuje potrebe korisnika za podacima, na osnovu upita koji se čuvaju, i menjati se na taj način.
- Rad na tehnologijama algoritama za tehniku analize podataka, koji teže ka upravljanju veoma velikim paketima podataka, kako bi organizacije što lakše analizirale svoja ogromna skladišta podataka. To će u velikoj meri olakšati u planiranju strategije organizacija za njihovo poslovanje za duže vremenske periode.
- I na kraju skale se nalazi dalje širenje PDA, što će dovesti do poboljšane sinhronizacije malih baza podataka i poboljšanje brzine bežičnog prenosa. Bluetooth bežični standard će u velikoj meri ubrzati razvoj bežične konekcije na Internet. Ali će i nametnuti pitanje daljeg razvoja zaštite podataka.

1.2 VRSTE BAZA PODATAKA

DBMS može da podrži različite vrste baza podataka. Baze podataka se klasifikuju prema broju korisnika, prema lokaciji baze podataka, kao i na osnovu načina i obima korišćenja.

Broj korisnika određuje da li je baza podataka jedno-korisnička (*single-user*) ili višekorisnička (*multi-user*). Jednokorisničke baze podataka podržavaju samo jednog korisnika u datom trenutku. Drugim rečima, ako korisnik A koristi bazu podataka, korisnici B i C moraju da čekaju dok korisnik A ne završi. Jednokorisničke baze podataka koje su realizovane za PC računare često se nazivaju desktop baze podataka. Nasuprot tome, višekorisničke baze podataka podržavaju istovremeni rad više korisnika. Podrazumeva se da su realizovane kao server u mrežnom okruženju i da se bazi pristupa iz same lokalne mreže ili preko Interneta. Kada višekorisničke baze podataka podržavaju relativno mali broj korisnika (obično manje od 50) ili određeno odeljenje u okviru organizacije, tada se nazivaju baze podataka za radne grupe. Kada bazu podataka koristi celokupna organizaciju i kada podržava mnogo korisnika (više od 50, obično nekoliko stotina) koji rade u različitim odeljenjima, tada se kaže da je to baza podataka preduzeća (eng. *enterprise database*).

Lokacija podataka može da se koristi za klasifikaciju baza. Na primer, baza podataka koja podržava podatke koji se nalazi na jednom mestu (serveru) zove se

centralizovana baza podataka. Baze podataka koja podržavaju podatke raspoređene na nekoliko različitih mesta (servera) nazivaju se distribuirane baze podataka.

Ipak, najpopularniji način klasifikovanja baza podataka danas je na osnovu toga kako će podaci biti korišćeni i koliko je važno vreme za dobijanje informacija iz podataka Na primer, transakcije koje se svakodnevno izvršavaju, kao što su prodaja proizvoda ili usluga, plaćanja, rezervacije, snabdevanje i slično moraju da budu brzo obavljene. Takve transakcije se moraju sprovesti na vreme i bez grešaka. Baza podataka koja je pre svega dizajnirana da podrži svakodnevne operacije jedne organizacije se naziva transakciona baza (*operational, production database*) Nasuprot tome, baze podataka kod kojih se podaci u vremenu skupljaju i skladište (*datawarehouse*) fokusirane su prvenstveno na čuvanje podataka u cilju generisanja informacija potrebnih za donošenje taktičkih i strateških odluka. Takve odluke obično zahtevaju značajnu manipulaciju nad podacima za dobijanje informacija npr. o visini cene proizvoda i usluga, prognozi prodaje, tržišnog pozicioniranja itd. Pored toga, skladište podataka može da sadrži podatke dobijene iz različitih izvora podataka. U cilju lakšeg preuzimanja takvih podataka struktura skladišta podataka može da se značajno razlikuje od strukture transakcionih baza.

1.2.1 Lične baze podataka

Lične baze podataka se prave za korišćenje od strane jednog korisnika i već su dugo prisutne u korišćenju personalnih računara. Pojavom ličnih digitalnih pomoćnika (PDA), lične baze podataka su našle primenu i u nizu mobilnih uređaja koji osim računarskih imaju i neke druge primene npr. mobilni telefoni, Internet čitači. Jednostavne aplikacije sa bazom podataka u kojoj čuvaju informacije i detalje o komunikaciji sa svakim klijentom, mogu da se koriste i sa računara i sa ličnog digitalnog pomoćnika, kao i da se prebacuju sa jednog na drugi uređaj radi izrade sigurnosnih kopija (backup) ili zbog zahteva posla. Uzmimo za primer preduzeće koje ima određeni broj prodavaca koji su u kontaktu sa postojećim i potencijalnim klijentima. Ako svaki prodavac ima još neke aplikacije, npr. grafičke prezentacije, cenovnik sa uslovima prodaje po kojem klijentu može ponuditi najpovoljniju kombinaciju proizvoda i količina za naručivanje, onda bi mu za takav posao prenosni računar, zbog svojih performansi i skladišnog prostora, mogao biti optimalno rešenje. S druge strane, ako prodavac ima samo listu klijenata i kontakata, njemu bi lični digitalni pomoćnik i aplikacija koja koristi malu bazu podataka bili najbolje rešenje.

Lične baze podataka se široko primenjuju jer često mogu bitno unaprediti produktivnost pojedinca. Međutim, one sadrže jedan faktor rizika: podatke ovih baza nije lako deliti sa drugim korisnicima. Na primer, ako bi menadžer prodaje želeo celokupan spisak klijenata i kontakata, to se ne bi moglo ni brzo ni lako uraditi uzimanjem podataka iz ličnih baza svakog od prodavaca. Ovo ilustruje veoma čest

problem: ako su neki podaci od interesa jednom čoveku, onda su verovatno (ili će brzo postati) od interesa i drugim ljudima. Zbog toga, lične baze podataka bi trebalo svesti na korišćenje pod posebnim okolnostima (npr. u veoma malim preduzećima) gde je verovatnoća potreba za deljenjem podataka između korisnika izuzetno mala.

1.2.2 Baze podataka za radne grupe

Radnu grupu čini relativno mali broj ljudi koji sarađuju na jednom projektu ili aplikaciji ili na grupi sličnih projekata ili aplikacija. Radna grupa obično sadrži desetak ljudi. Oni mogu biti uključeni u npr. planiranje, projektovanje ili razvoj novog računarskog programa. Baza podataka za radne grupe služi za podršku zajedničkog rada jedne takve grupe. Uzmimo za primer, radnu grupu koja pravi i standardne i programe po porudžbini, koji se prodaju softverskim kompanijama kao i krajnjim korisnicima. Obično, jedna ili više osoba rade na datom programu, ili dele programe, u isto vreme. Grupi je potrebna baza podataka koja će da prati razvoj svakog dela i koja će da omogući da se podaci što lakše razmenjuju među članovima tima.

Svaki član radne grupe ima svoj računar, a računari su umreženi putem LAN-a. Baza podataka se nalazi na centralnom računaru koji se zove server baze podataka, koji je takođe na mreži. Stoga, svaki član grupe ima pristup podacima. Različiti članovi grupe (u zavisnosti da li je u pitanju rukovodilac projekta ili projektant, programer) mogu imati različita ovlašćenja (privilegije), a time i različite poglede na podatke. Primetićete da je glavna mana ličnih baza podataka, teško ostvariva razmena podataka, ovde prevaziđena (barem je razmena podataka u okviru grupe lako ostvariva). Međutim, razmena podataka otvara nova pitanja i probleme koji nisu postojali kod ličnih baza podataka. Glavni problemi upravljanja podacima su vezani za njihovu bezbednost i integritet, s obzirom da više korisnika može istovremeno da obavlja ažuriranje podataka.

1.2.3 Baze podataka odeljenja

Odeljenje je funkcionalna radna jedinica u okviru organizacije. Tipični primeri odeljenja su: kadrovsko, marketing, proizvodnja, računovodstvo i sl. Odeljenje je obično veće od radne grupe (nekada se sastoji i do 100 osoba) i odgovorno je za veći broj različitih poslova. Baze podataka odeljenja su napravljene da podrže različite oblike poslova i aktivnosti koje obavlja to odeljenje. Uzmimo za primer bazu podataka kadrovskog odeljenja u kojoj se prate podaci vezani za zaposlene, vrste poslova, stručnu spremu i poslovna zaduženja. Kada su svi relevantni podaci sačuvani u bazi podataka, korisnici mogu da pretražuju bazu podataka u cilju dobijanja odgovora na pitanja kao što su sledeća:

- Za određenu vrstu zanimanja (npr. programer) kakve prilike za zaposlenje trenutno postoje u organizaciji?
- Za tu istu vrstu posla koja stručna sprema ili veština je neophodna?
- Koje veštine, znanje poseduje određeni radnik? I obrnuto, koji radnici poseduju određenu veštinu, znanje?
- Koji sve radnici su obavljali određeni posao u organizaciji? I obrnuto, koje sve poslove je određeni radnik obavljao u organizaciji?
- Koje sve zaposlene nadgleda određeni menadžer?

Tipična pitanja na koja se treba odgovoriti pri pravljenju baze podataka odeljenja:

- Kako baza podataka i njeno okruženje trebaju da budu organizovani da bi se ostvarile zadovoljavajuće performanse, uzimajući u obzir veliki broj korisnika i veliki broj transakcija?
- Kako na odgovarajući način obezbediti podatke od nedozvoljenog pristupa i/ili distribucije?
- Koje alate za razvoj baze podataka i aplikacija treba koristiti u slučaju ovako kompleksnog okruženja?
- Da li i druga odeljenja koriste iste vrste podataka, i ako je to slučaj, kako se najbolje može upravljati podacima po pitanju njihove redundantnosti i konzistentnosti i metapodacima po pitanju njihove konzistentnosti?
- Da li su korisnici baze podataka geografski jedni od drugih udaljeni ili je veličina baze podataka tolika da se podaci moraju čuvati na više računara, tako stvarajući distribuirane baze podataka?
- Da li se bazi podataka može pristupati preko Interneta i da li ona treba da bude uključena u intranet organizacije?

1.2.4 Baze podataka organizacija

Baza podataka organizacije obuhvata čitavu organizaciju ili više njenih odeljenja. Ova vrsta baza podataka je namenjena da podrži sve procese organizacije i proces donošenja odluka. Važno je istaći da jedna organizacija može imati više baza podataka, tako da jedna takva baza podataka ne sadrži sve podatke jedne organizacije. Jedna baza podataka za celu organizaciju srednjih do velikih dimenzija ne bi bila praktična iz mnogo razloga. Kao prvo zbog različitih potreba različitih korisnika, kompleksnosti stvaranja jedinstvenih metapodataka za sve korisnike baze podataka je ogromna. Baza podataka organizacije pruža podršku za jedan određeni broj (skup) odeljenja. Tokom poslednje decenije, razvoj baza podataka organizacije je doveo do dva najvažnija oblika:

1. *Enterprise resource planning* (ERP) sistem
2. Implementacija skladišta podataka (*data werehouses*)

ERP sistemi rade sa tekućim podacima organizacije, dok skladišta podataka sakupljaju podatke iz raznih operativnih baza podataka, uključujući i lične, radnih grupa, odeljenja i ERP baze podataka. Skladišta podataka pružaju mogućnost korisnicima da rade sa prethodnim podacima kako bi pronašli obrasce i trendove događaja i kako bi odgovorili na pitanja koja su vezana za strategiju poslovanja.

Uzmimo za primer veliku zdravstvenu organizaciju koja upravlja grupom medicinskih centara, u šta spadaju domovi zdravlja, bolnice, klinike i starački domovi. Svaki od ovih medicinskih centara ima svoju bazu podataka (ili baze podataka) koja pruža podršku u obavljanju raznih poslova. Ove baze podataka imaju podatke o pacijentima, doktorima, medicinskim uslugama, poslovanju i drugim bitnim entitetima. Baza podataka pruža adekvatnu podršku za većinu poslova u svakom pojedinačnom medicinskom centru. Međutim, postoji potreba za jedinstvenim pogledom na celu organizaciju; na primer, da bi se videla sva poslovanja sa jednim dobavljačem ili pacijentom. Povećanje produktivnosti se može postići uvođenjem, na primer, centralnog sistema za naručivanje materijala za sve zdravstvene centre i raspoređivanjem osoblja i usluga koje vrše na sve zdravstvenim centre. ERP sistem omogućava uvođenje prethodnih promena. Donošenje odluka na nivou cele organizacije, u vezi sa poslovanjem sa dobavljačima, i podnošenje izveštaja raznim agencijama zahteva sakupljene svih prethodnih podataka i informacija. Da bi se zadovoljile ove potrebe, organizacija koristi skladište podataka koje se održava i nalazi u sedištu organizacije. Podaci koji se nalaze u skladištu podataka su preuzeti (i potom sumirani) iz pojedinačnih baza podataka svakog medicinskog centra. Ovaj proces preuzimanja podataka se odvija periodično putem telekomunikaciono- računarske mreže.

1.2.5 Internet, Intranet i Extranet baze podataka

Internet tehnologije služe za olakšavanje deljenja podataka i informacija. Na primer, u okviru fabrike može se koristiti lokalna mreža (LAN) koja povezuje radne stanice zaposlenih iz raznih odeljenja sa serverom na kome se nalazi baza podataka. LAN unapređuje komunikaciju i proces donošenja odluka u okviru same kompanije. Ako se uvede Intranet koji se zasniva na Web tehnologiji, njemu se može pristupati samo u okvirima kompanije. Radna stanica svakog zaposlenog se može koristiti kao Web *browser*, i na taj način se dobija brz pristup informacijama kompanije, uključujući i telefonski adresar, specifikacije proizvoda, elektronsku poštu i tome slično. Takođe se radne stanice mogu koristiti i kao personalni računari koji povezani preko LAN-a pristupaju serveru na kome se nalazi baza podataka. Moguće je dodati i Web interfejse nekim poslovnim aplikacijama, kao što su unošenje porudžbina, da bi na taj način više internih poslovnih aktivnosti moglo biti obavljano od strane zaposlenih preko intraneta.

U cilju efikasnijeg ukupnog poslovanja intranet sistem se može otvoriti ka kupcima preko Interneta. Ovo omogućava maloprodajama da pretražuju katalog proizvoda (uključujući slike i specifikacije proizvoda) i utvrde da li željenog proizvoda ima u skladištu. Tada radnici u maloprodajnim objektima mogu da obaveste svoje kupce i da poruče željeni komad proizvoda preko Interneta. Internet konekcija je konfigurisana kao extranet što znači da samo odobrene maloprodaje mogu da pristupe intranet-u fabrike.

Sve veće korišćenje Interneta, svetske mreže koja povezuje korisnike, nebitno koje platforme je dovelo i do promena u okruženju baza podataka. Prihvatanje Interneta od strane poslovnog sveta je dovelo do bitnih promena u davno utvrđenim modelima poslovanja. Veoma uspešne kompanije su bile ugrožene zbog novih kompanija koje su prihvatile Internet pomoću koga su unapredile informisanje i usluge koje su pružale svojim klijentima, i koje su zaobišle tipične tokove marketinga i distribucije proizvoda. Na primer, kupci konfigurišu i poručuju svoj PC računar direktno od proizvođača računara. Informacije o upražnjenim mestima i aktivnostima organizacije se mogu dobiti preko Interneta za većinu organizacija.

Svaka od ovih radnji zahteva podršku baze podataka. Lak pristup Internetu svih vrsta platformi omogućava kompanijama da reorganizuju svoje poslove i razviju brže aplikacije i to po manjim troškovima. Standardni interfejsi omogućavaju veću produktivnost korisnika, uz manje provedenog vremena na obuci i manju potrebnu podršku. Osnova razvoja korisnikove aplikacije je priključivanje baze podataka iz koje se mogu dobiti sveže informacije. Kada je baza podataka povezana sa nekom Internet lokacijom, korisnici preko Web browser-a mogu da postavljaju određena pitanja na koja će dobiti odgovore bazirane na svežim informacijama. Odgovaranje na pitanja je automatsko; nema potrebe da se putem telefona prolazi kroz niz opcija da bi se postavilo pitanje nekoj osobi i da bi se zatim od nje čekao odgovor. Internet baze podataka su nezamenljive u razvoju sajtova za kupovinu preko Interneta. Kompanije prate sva dešavanja na sajtu kako bi došli do što više informacija o svojim klijentima (obrasci pri kupovini, navigacija sajtom, dužina zadržavanja na svakoj stranici i tome slično) kako bi što više unapredili svoj odnos prema kupcima.

Većina primera koji su navedeni prikazuju *Business-to-Customer* (B2C) veze. Kada su kupci kod nekih firmi druge firme, takav odnos se obično naziva B2B (*Business-to-Busines*s) odnos. Internet se koristi da olakša B2C odnos, zato što su kupci obavezno spoljni faktor u odnosu na firmu, i mogućnost kupca da pristupi poslovnim podacima i informacijama je od velike važnosti za uspešan odnos. Dozvoljavanjem pristupa poslovnim bazama podataka, od strane osoba koje nisu deo organizacije, javljaju se nova pitanja za rukovođenje informacionim sistemima vezana za sigurnost i integritet podataka.

Kompanije su godinama vršile razmenu informacija putem elektronske razmene podataka (EDI- *eletronic data interchange*). Mnoge kompanije i dalje koriste EDI

sistem za obavljanje B2B poslovanja. Neke kompanije, uglavnom nove ili one koje nisu imale EDI sistem, koriste extranet za obavljanje B2B razmena podataka i informacija. Extranet koristi Internet tehnologiju, međutim, pristup extranetu je, za razliku od Interneta kome mogu svi pristupiti, ograničen. U stvari, pristup je ograničen na kompanije koje su u ulozi dobavljača i kupca i koje imaju međusobni dogovor o pristupu podacima i informacijama jednih drugima. Obično, prethodno pomenuti akteri imaju pristup delu intraneta drugog aktera. Ovaj način pristupa olakšava poslovne odnose tako što pruža brži i efikasniji način obrade i pristupanja podacima.

Kao što je prethodno pomenuto mnoge organizacije su koristile Internet tehnologiju za stvaranje privatne mreže namenjene za upravljanje informacijama u okviru organizacije. Po izgledu ne postoji razlika između intranet i Internet stranica, ali pristup intranet stranici je ograničen samo na korisnike u okviru te organizacije. Stoga je i pristup bazi podataka organizacije ograničen. Intranet može da ostvari Internet konekciju ali ta konekcija će biti zaštićena firewall-om koji sprečava neovlašćeni pristup intranetu.

2 OSNOVNI KONCEPTI I DEFINICIJE

Dobre odluke u poslovanju zahtevaju dobre informacije koje se dobijaju na osnovu prethodnih činjenica. Te sirove činjenice su poznate pod nazivom podaci. Podacima se najlakše upravlja kada su oni organizovani i smešteni u bazi podataka. U ovom poglavlju naučićete šta su baze podataka, njihove osnovne funkcionalnosti i zašto daju najbolje rezultate u odnosu na druge sisteme za upravljanje podacima. Takođe ćete naučiti o različitim tipovima baza podataka i o tome zašto je dizajn baza podataka tako važan. Baze podataka su nastale iz klasičnih računarskih sistema datoteka. Iako je upravljanje podacima u datotekama danas zastareo način, on je veoma važan za razumevanje ograničenja koja postoje kod klasičnih fajl sistema. Takođe ćete naučiti kako se primenom baza podataka eliminiše većina nedostataka kod klasičnih fajl sistema za upravljanje podacima.

Baza podataka se može definisati kao organizovani skup logički povezanih podataka. Ona može biti bilo koje veličine i kompleksnosti. Na primer, prodavac može da ima malu bazu podataka vezanu za kupce na svom *notebook* računaru koja se sastoji od nekoliko megabajta podataka. Preduzeće koje zapošljava hiljadu i više ljudi može da ima veoma veliku bazu podataka od nekoliko terabajta podataka (jedan terabajt = 10^{12} bajtova) na *mainframe* kompjuteru na kome se nalazi aplikacija za podršku odlučivanju. Veoma velika skladišta podataka imaju više od petabajta podataka (1 petabajt = 10^{15} bajtova). U širem smislu, bazu podataka možemo posmatrati kao integrisani skup podataka o nekom sistemu i skup postupaka za njihovo održavanje i korišćenje, organizovan prema potrebama korisnika. To je dobro struktuirana kolekcija podataka, koja postoji jedno određeno vreme, koja se održava i koju koristi više korisnika ili programa.

2.1 PODATAK

Istorijski, pod terminom podatak se podrazumeva činjenica o nekom predmetu i/ili događaju koja se može zabeležiti i sačuvati na računaru. Na primer, u bazi podataka nekog prodavca podaci bi bile činjenice kao što su ime, adresa i broj telefona kupca. Ovakav tip podatka se zove struktuirani podatak. Najvažniji struktuirani podaci su brojevi, karakteri i datumi (vreme). Današnje baze podataka pored struktuiranih podataka sadrže i druge vrste podataka kao što su razna dokumenta, mape, fotografije, zvuk, čak i video zapise. Na primer, u bazi podataka nekog prodavca mogla bi se naći i slika kupca. Takođe bi se mogao naći zvučni ili video zapis poslednjeg razgovora sa kupcem. Ova vrsta podatka se naziva nestruktuirani podatak ili multimedijalni podatak. Multimedijalni podaci se najčešće mogu naći na Web serverima i u Internet bazama podataka.

Danas se podatak definiše kao sačuvana reprezentacija predmeta i/ili događaja koja ima smisla i važnosti za korisnika baze podataka. Ova definicija uključuje i struktuirane i nestruktuirane podatke. Često se u okviru jedne baze podataka mogu naći kombinovani struktuirani i nestruktuirani podaci kako bi se stvorilo multimedijalno okruženje. Na primer, automehaničarska radnja može kombinovati struktuirane podatke (koji opisuju klijenta i njegova kola) sa multimedijalnim podacima (slika automobila i skenirana kopija osiguranja).

Pod podatkom se podrazumeva činjenica prihvaćena kao takva tj. kakva jeste. Podatak sam po sebi nema značenje, tek kada se interpretira nekom vrstom sistema za obradu podataka poprima značenje i postaje informacija. Tipično, termin "podatak" se odnosi na ono što je u bazi podatak. Dakle, podatak je sirova činjenica – još uvek nije obrađena i ne zna se njeno značenje. Računar vrši obradu podataka, prema zadatom programu, te se na osnovu saznanja sadržanih u podacima, a kao rezultat njihove obrade, stiču nova saznanja - informacije.

2.2 INFORMACIJA

Termini podatak i informacija su usko povezani i često se koriste kao sinonimi. Međutim, korisno je razlikovati termine podatak i informacija. Informaciju definišemo kao podatak koji je bio obrađen na takav način da se znanje osobe koja koristi podatak povećalo. Obrada podataka može biti jednostavna, kao što je tabelarni prikaz podataka, ili može biti kompleksna, kao što je primena složenih kriterijumskih funkcija ili statističkih modela nad sirovim podacima. Na primer, razmotrimo sledeći spisak činjenica:

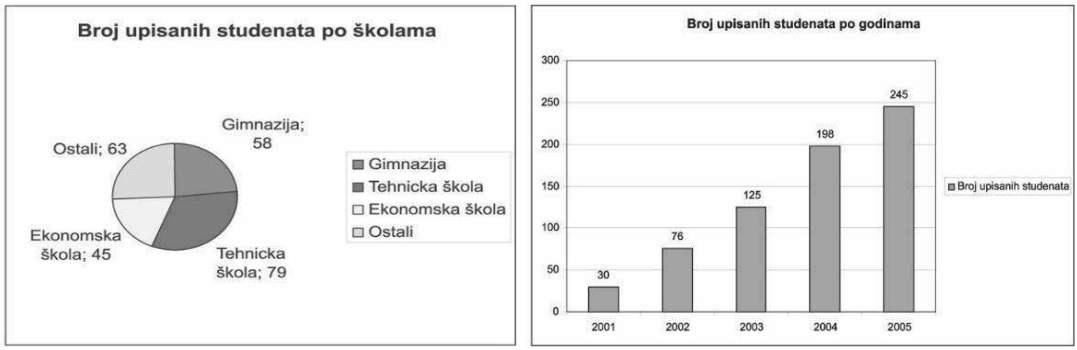
Petar Petrović	1506988710325
Marko Marković	2011999050123
Janko Janković	1112998930456
- - - - - - - - - -	- - - - - - - - - -

Prikazane činjenice po definiciji predstavljaju podatke, ali bi se većina složila da su ovi podaci u sadašnjoj formi beskorisni. Čak iako pretpostavljamo da se radi o imenima osoba i njihovim matičnim brojevima, podaci ostaju beskorisni jer ne znamo čemu služe. Pogledajte šta se događa kada stavimo ove iste podatke u kontekst, kao što je pokazano na slici Slika 2.1. Dodavanjem još nekoliko dodatnih podataka i njihovim uređivanjem, prepoznajemo spisak upisanih studenata. Na ovaj način se dolazi do informacije koja je korisna npr. upravi fakulteta, profesorima, studentskoj službi i sl.

Ime i prezime	JMBG	St. program	Godina upisa
Petar Petrović	1506988710325	PE	2016
Marko Marković	2011999050123	IT	2017
Janko Janković	1112998930456	IR	2017
- - - - - - - - - -	- - - - - - - - - -	SII	2015

Slika 2.1: Uređeni prikaz podataka iz BP - informacija o upisu

Drugi način da se iz podataka dobiju informacije je da se podaci sumiraju ili na neki drugi način obrade i prezentuju. Na primer, sumirani podaci o upisu studenata mogu biti prezentovani u vidu grafičke informacije (Slika 2.2). Ova informacija se može iskoristiti kao osnova za odlučivanje o dodavanju novih predavanja ili o zapošljavanju novog nastavnog kadra. Moderne baze podataka vrlo često sadrže i podatke i informacije. Podaci se često obrađuju i čuvaju u obrađenoj formi i služe za pomoć pri donošenju odluka, a takvim podacima (informacijama) se najbrže pristupa.



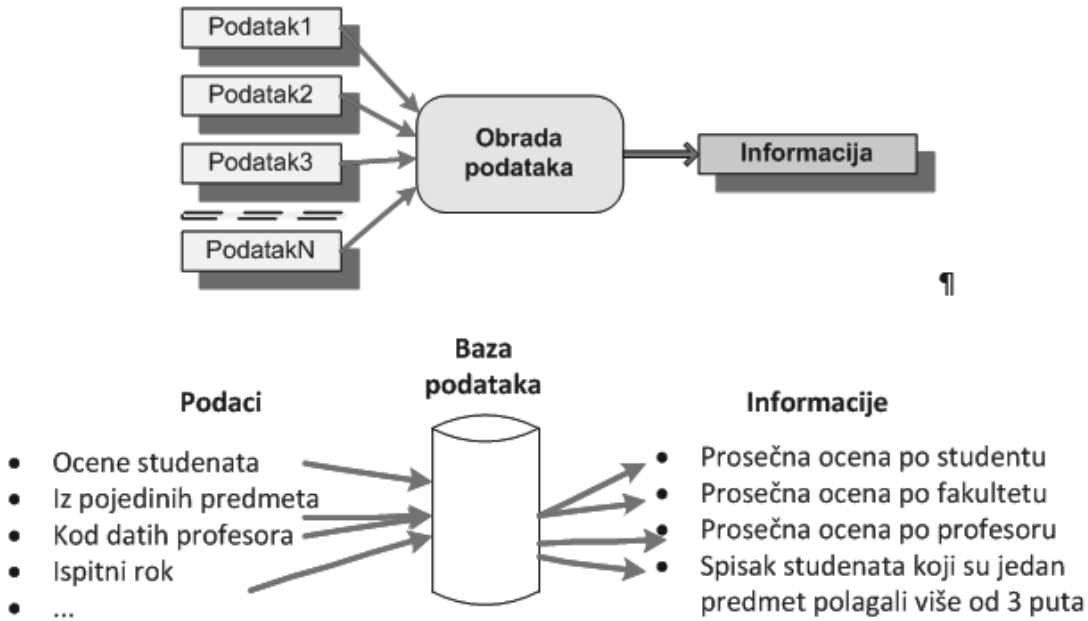
Slika 2.2: Grafički prikaz podataka iz BP - informacija o upisu

Podaci obrađeni tako da dobijaju značenje čine informaciju. Na osnovu dosadašnjih razmatranja proizilazi sledeće:

- Podaci su gradivni elementi informacija
- Informacije nastaju obradom podataka
- Informacije se koriste za otkrivanje značenja podataka
- Informacija koja je precizna, relevantna i dobijena na vreme je ključ za donošenje dobrih odluka
- Dobre odluke su ključ za opstanak organizacija na tržištu.

Relevantne i korisne informacije zahtevaju tačne podatke. Takvi podaci se moraju prikupljati bez grešaka i moraju se čuvati u formatu koji obezbeđuje lak pristup i

obradu podataka. Kao i sa svakim važnim resursom, podacima se mora pažljivo upravljati. Upravljanje podacima je disciplina koja se bavi prikupljanjem i skladištenjem podataka i njihovom obradom u cilju efikasnog dobijanja relevantnih informacija.



Slika 2.3: Obradom prikupljenih podataka nastaje informacija

2.3 METAPODACI - PODACI O PODACIMA (METADATA)

Podaci koji se prikupljaju i čuvaju u bazi podataka često se nazivaju i podaci krajnjih korisnika (*end user data*). Metapodaci su podaci koji opisuju svojstva ili karakteristike podataka krajnjih korisnika i kontekst tih podataka. Neka tipična svojstva podataka su naziv (ime) podatka, definicija (tip), dužina (veličina), i dozvoljene vrednosti (ograničenja). Kontekst podataka, koji opisuju metapodaci, podrazumeva izvor podataka, gde se čuvaju podaci, vlasništvo i korišćenje.

Field	Type	Null	Key	Default	Extra
Sifra	int(11)	NO	PRI	NULL	auto_increment
Naziv	varchar(45)	YES		NULL	
Cena	float	NO		NULL	
JedinicaMere	varchar(10)	NO		KG	
ZemljaPorekla	varchar(45)	YES		NULL	
Kategorija_Rb	int(11)	NO	MUL	NULL	

Slika 2.4: Primer metapodataka

Metapodaci opisuju svojstva podataka i nalaze se odvojeno od tih podatka. Metapodaci ne prikazuju ni jedan podatak. Slika 2.4 prikazuje metapodatke za tabelu *Proizvod(Sifra, Naziv, Cena, JedinicnaMera, ZemljaPorekla, Kategorija_RB)* iz MySQL softvera. Oni omogućavaju dizajnerima i korisnicima baza podataka da razumeju koji podaci postoje u bazi, šta oni znače i koja je razlika između podataka koji na prvi pogled izgledaju isto. Upravljanje metapodacima je veoma bitno jer podaci bez jasnog značenja mogu biti zbunjujući, pogrešno protumačeni ili puni grešaka.

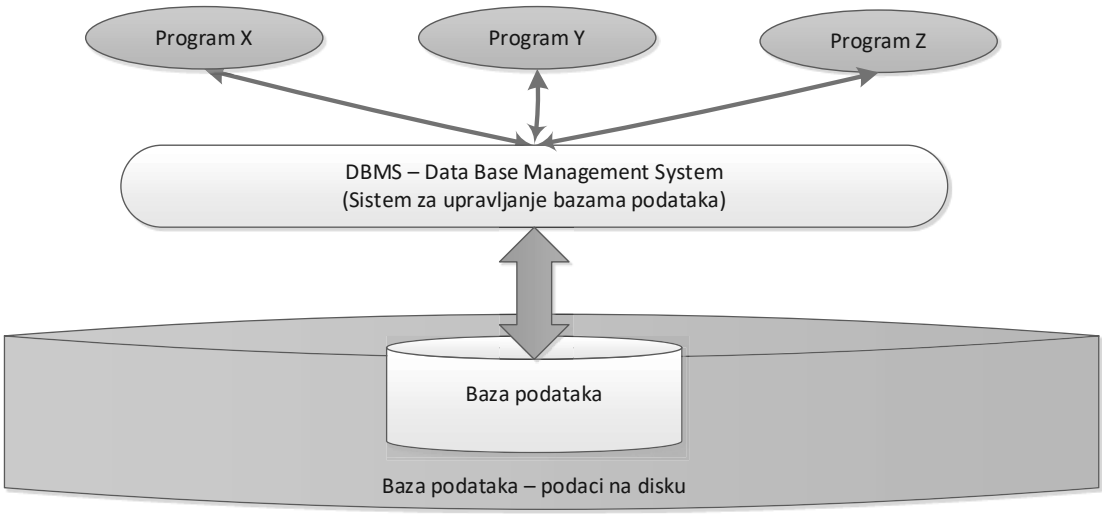
2.4 SISTEM ZA UPRAVLJANJE BAZAMA PODATAKA

Sistem za upravljanje bazama podataka (DBMS - *Data Base Management System*) je softverski sistem koji se koristi za kreiranje, održavanje i manipulisanje podacima, kao i za kontrolu prava pristupa bazi podataka. Osnovni cilj primene DBMS je da se obezbedi način za smeštanje podataka i dobijanje informacija iz baze podataka na najpogodniji i najefikasniji način. Primenjuje se za upravljanje velikim količinama podataka. DBMS omogućava krajnjim korisnicima i programerima da dele podatke, tj. omogućava da se podaci koriste od strane više aplikacija, a ne da svaka aplikacija ima svoju kopiju podatka sačuvanu u posebnim datotekama. DBMS takođe pruža mogućnost kontrole pristupa podacima, osigurava integritet podataka, uspostavlja kontrolu konkurentnosti i vrši oporavak baze podataka. Programeri aplikacija za rad sa bazama podataka ne moraju da poznaju detalje o načinu zapisa baze podataka na disku, ne moraju da formulišu algoritme za efikasan pristup podacima, niti su opterećeni bilo kakvim aspektima oko upravljanja podacima u bazi podataka.

Danas je veoma bitan i značajan koncept baze podataka po kome je to, u stvari, zajednički resurs koga istovremeno (konkurentno) koristi veći broj programa, jer se pravi efekti baze podataka ispoljavaju kada se radi u mrežnom okruženju. Posmatrajmo bazu podataka jedne banke u kojoj se nalaze računi građana. Moguće je da se u istom trenutku na šalteru u jednoj ekspozituri podiže novac sa jednog računa i uplaćuje na

drugi račun, a da se istovremeno u sasvim drugoj ekspozituri uplaćuje novac na isti taj račun. Pomenuti DBMS je upravo tu da upravlja konkurentnim radom više korisnika i da obezbeđuje sinhronizaciju njihovog rada.

Takođe, DBMS ima funkciju da spreči štetne posledice (narušen integritet baze, nekonzistentno stanje baze...) pri promenama (transakcijama) koje se vrše nad bazom podataka u višekorisničkom okruženju. U tu svrhu postoje razne tehnike kao što su tehnika zaključavanja podataka, tehnika vremenskog markiranja itd. Posebno je značajno upravljanje istovremenim (konkurentnim) transakcijama. Tačnost, zaštita i dostupnost baza podataka, kao i korektnost i performanse transakcija koje pristupaju tim bazama su bitni parametri za uspeh svakog poslovnog sistema. Termini baza podataka i upravljanje bazom podataka se ponekad mešaju. Stručno govoreći, baza podataka je uvek skup podataka, a ne računarski program.



Slika 2.5: DBMS je interfejs između aplikacija (korisnika) i zapisa baze podataka na disku

DBMS je uveden kao interfejs između korisnika (korisničkih programa, aplikacija) i zapisa baze podataka na disku (Slika 2.5). Korisnički programi ne pristupaju podacima direktno, već komuniciraju sa ovim softverom (programom). Na neki način DBMS skriva od korisnika kompleksnu internu strukturu samih podataka u bazi podataka. DBMS upravlja strukturom baze podataka: definiše objekte baze, njihova svojstva (atribute), dozvoljene vrednosti atributa, veze između objekata, ograničenja nad objektima i međusobnim vezama. Omogućava manipulaciju podacima u bazi: unošenje, brisanje i izmene, tj. omogućava njeno održavanje. Kontroliše pristup podacima: ko može da pristupi podacima, kojim podacima i šta može sa njima da radi. DBMS dozvoljava deljenje BP između više aplikacija/korisnika i čini upravljanje

podacima uspešnijim i delotvornijim Uobičajeno je da kada se govori o softveru za baze podataka, onda se misli upravo na DBMS. DBMS upravlja interakcijom između krajnjih korisnika (aplikacija) i baze podataka. Krajnji korisnici imaju bolji pristup većem broju bolje organizovanih podataka

2.5 PRIMENA BAZA PODATAKA

Baze podataka imaju široku primenu. Karakteristični primeri su dati u nastavku.

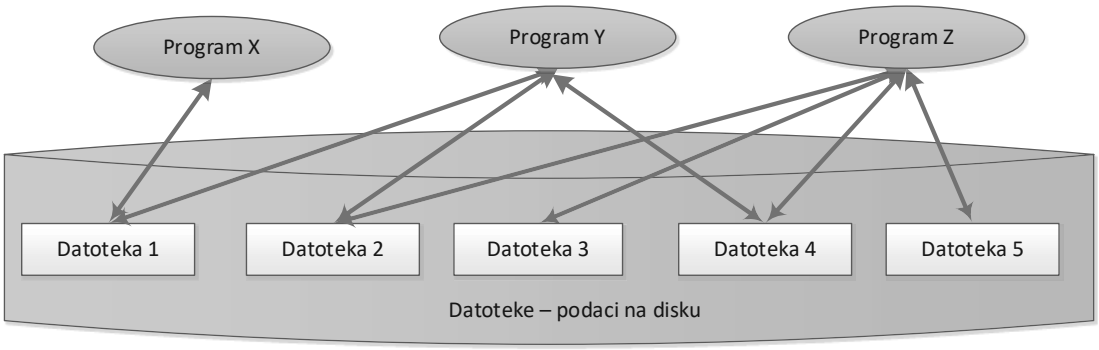
- *Bankarstvo*: Informacije o klijentima, računi, krediti, bankarske transakcije i sl.
- *Avio-saobraćaj*: Informacije o letovima, rezervacija karata i sl. Avio-kompanije su među prvima počele da koriste baze podataka kroz geografski distribuiran način – putem terminala i računara koji su geografski široko rasprostranjeni i koji pristupaju centralizovanim podacima putem telekomunikacionih kanala i Interneta.
- *Univerziteti*: Informacije o studentima, upis, ocene, kursevi itd.
- *Transakcije putem kreditnih kartica*: Plaćanje i generisanje mesečnih izveštaja.
- *Telekomunikacije*: Evidentiranje obavljenih razgovora, generisanje mesečnih računa, podrška za "pripejd" kartice, čuvanje informacija o komunikacionoj mreži i sl.
- *Finansije*: Prodaja i kupovina finansijskih instrumenata kao što su akcije i obveznice.
- *Trgovina*: Podaci o kupcima, proizvodima i različiti katalozi sa cenama.
- *Proizvodnja*: Upravljanje lancima snabdevanja, praćenje proizvodnje, stavki o zalihama u magacinima ili prodavnicama, generisanje otpremnica, faktura i sl.
- *Ljudski resursi*: Informacije o zaposlenima, plate, porezi na plate i beneficije i sl.

Iz navedene liste jasno je da su baze podataka danas suštinski deo skoro svih organizacija. U ranom periodu primene baza podataka mali broj ljudi je imao direktan kontakt sa bazama. Pristup se odnosio na posedovanje štampanih izveštaja ili se pristup ostvarivao posredno npr. preko službenika kod bakarskih transakcija ili rezervacije karata. Prvi masovni direktni pristupi bazama podataka su ostvarivani putem servisa preko telefonske linije. Razvoj Internet servisa krajem devedesetih godina povećao je direktan pristup korisnika bazama podataka. Web interfejsi su danas omogućili niz usluga koje se mogu realizovati u realnom vremenu (*on-line*). Korisnički interfejsi su sakrili detalje direktnog pristupa bazama, tako da većina ljudi danas nije ni svesna da radi sa bazama podataka koje su postale bitan deo njihovog života. Značaj baza podataka ogleda se i u tome što su softverske kompanije, kao što je Oracle postale jedne od najvećih na svetu, a softverske kompanije kao što su Microsoft i IBM poklanjaju veliku pažnju razvoju sopstvenih sistema za upravljanje bazama podataka.

3 KLASIČNI SISTEMI SA DATOTEKAMA I BAZE PODATAKA

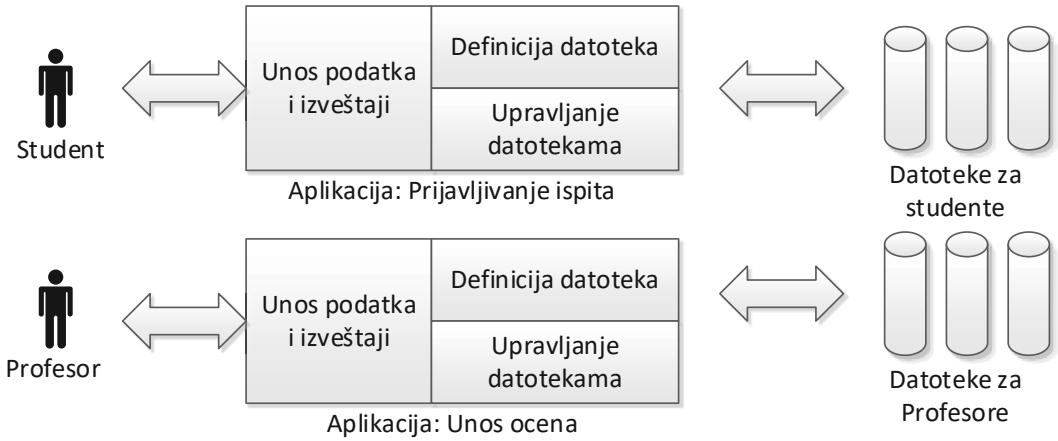
Kada su se računari počeli koristiti za obradu podataka, nisu postojale baze podataka. Računari su u to vreme bili znatno slabiji nego današnji personalni računari, zauzimali su čitavu prostoriju i koristili su se skoro isključivo za naučna izračunavanja. Postepeno su računari uvođeni u poslovni svet. Da bi bili od koristi za poslovne aplikacije, računari moraju da skladište, manipulišu, i preuzimaju velike datoteke podataka. Kako su poslovne aplikacije postajale sve kompleksnije, postalo je očigledno da klasični sistemi zasnovani na datotekama imaju veliki broj nedostataka i ograničenja. U većini bitnih poslovnih aplikacija danas se umesto klasičnog sistema zasnovanog na datotekama koriste baze podataka.

Klasičan sistem obrade podataka zasnovan na datotekama i programskim jezicima prikazan je blok šemom na sledećoj slici. Programi su direktno povezani sa datotekama, svaki program mora da poznaje detaljan zapis podataka na disku (Slika 3.1).



Slika 3.1: Klasičan sistem obrade podataka zasnovan na programskim jezicima i datotekama

Da bi objasnili osnovne karakteristike sistema zasnovanog na datotekama, posmatrajmo informacioni sistem jednog univerziteta. Standardna funkcionalnost takvog sistema je da studenti preko interneta prijavljuju ispite, a profesori nakon ispita unose ocene. U klasičnoj obradi podataka sa datotekama postojao bi veliki broj datoteka za studente i za profesore. Neke od datoteka bi bile zajedničke.



Slika 3.2: Klasična obrada podataka zasnovana na sistemu datoteka

Aplikacije za prijavu ispita i za unos ocena bi bile opterećene velikim brojem preciznih definicija datoteka. Ove definicije (format i način pristupa datoteci) bi neminovno proširivale osnovni kod aplikacije, tako da bi sama funkcionalnost oko prijave ispita i unosa ocena bila praktično u drugom planu. Kako se u nekoj aplikaciji povećava broj linija koda, sama aplikacija postaje nepregledna i sklona greškama, od kojih su najteže tzv. logičke greške. Testiranje takvih aplikacija i ispravljanje logičkih grešaka je veoma složeno. Pošto postoji veliki broj datoteka, neminovno postoji i velika redudansa (ponavljanje) podataka. Isti podaci se mogu naći u različitim datoteka, a održavanje takvih podataka je teško i greške u radu su neminovne.

3.1 NEDOSTACI SISTEMA ZASNOVANOG NA DATOTEKAMA

Postoji više mana koje su tipične za sistem koji je zasnovan na datotekama i klasičnim programskim jezicima. Na prvom mestu to je **zavisnost između programa i podataka**. Opisi datoteka se čuvaju u okviru svakog programa koji pristupa toj datoteci. Kao posledica ovoga, svaka promena koja se napravi u datoteci, a odnosi se na strukturu, momentalno podrazumeva da se mora menjati i opis datoteka u svakom programu koji pristupa tim podacima. Pretpostavimo da se veličina polja "Adresa studenta" menja sa 20 karaktera na 30 karaktera. Zbog toga se opis datoteke u svakom programu mora ažurirati. Često je teško i samo lociranje svih programa na koje je uticala ovakva promena. Što je još gore pri ažuriranju se često prave greške.

Redudansa podataka: Kako se u prikazanom sistemu procesi odvijaju nezavisno jedni od drugih, ponavljanje podataka nije izuzetak već je pravilo. Zbog nepotrebnih duplikata potreban je veći prostor za njihovo čuvanje kao i više truda i rada pri njihovom ažuriranju. Neplanirana redudansa podataka može da dovede do gubitka podataka. Na

primer, isti podaci mogu se voditi pod različitim imenima atributa u različitim dokumentima, ili obrnuto, isto ime se može koristiti za različite vrste podataka.

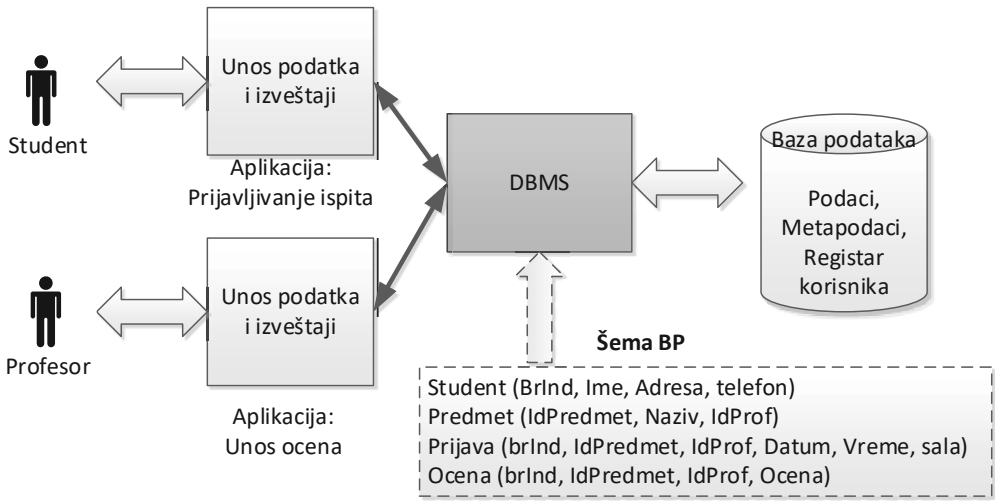
Ograničenost deljenja podataka: Korišćenjem klasičnog sistema zasnovanog na datotekama, svaki proces ima svoje datoteke i korisnici nemaju šansu da međusobno dele podatke sa korisnicima iz drugih procesa.

Dugo vreme za razvoj: Sa klasičnim sistemom zasnovanom na datotekama postoji mala šansa za korišćenje prethodnih razvojnih dostignuća. Svaka nova aplikacija zahteva od projektanta da krene od nule. Svaki put je neophodno definisati nove formate i opise podataka i pisati kod za pristup podacima za svaki program. Ovako veliko vreme za razvoj nije u skladu sa današnjim poslovnim potrebama, gde je svaki minut bitan da bi se postigao uspeh.

Teško održavanje programa: Skup svih prethodno navedenih nedostataka dovodi do preterane potrebe za održavanjem programa. Čak 80% budžeta predviđenog za razvoj sistema zasnovanog na datotekama odlazi na njegovo održavanje. Zbog toga, naravno, ostaje jako malo prostora za razvoj novih aplikacija.

3.2 PRISTUP ZASNOVAN NA BAZAMA PODATAKA

Pristup zasnovan na BP potencira integraciju i deljenje podataka između svih odeljenja jedne organizacije. Ovaj pristup zahteva potpunu promenu u načinu razmišljanja, počevši od najvišeg nivoa upravljanja. Takva promena načina razmišljanja za većinu organizacija je veoma teška.



Slika 3.3: Blok šema informacionog sistema zasnovanog na bazama podataka

Da bi objasnili pristup zasnovan na BP posmatrajmo prethodno razmatrani zastareli informacioni sistem koji se klasično zasnivao na datotekama. Koncept sistema zasnovanog na BP prikazan je na Slici 3.3. Odmah se može uočiti da podaci koji su prethodno čuvani u više različitih datoteka, sada su integrisani u jedinstvenu BP. Metapodaci, koji opisuju ove podatke, nalaze se zajedno sa njima u BP. Sistem za upravljanje BP pruža mogućnost stvaranja različitih pogleda na istu (ili više) BP za različite korisnike u organizaciji. DBMS dozvoljava korisnicima da dele, pretražuju, pristupaju i ažuriraju integrisane podatke.

3.3 PREDNOSTI PRISTUPA ZASNOVANOG NA BAZAMA PODATAKA

Pristup zasnovan na bazama podataka ima mnogo potencijalnih prednosti u odnosu na pristup zasnovan na datotekama. Te prednosti su sledeće:

- **Nezavisnost između programa i podataka**

  Odvajanje metapodataka od aplikacija koje koriste podatke naziva se nezavisnost podataka. Ova osobina kod baza podataka dozvoljava promenu i prenos podataka organizacije na druge računarske sisteme bez potrebe za promenom programa koji obrađuje ove podatke.

- **Minimalna redudansa podataka**

  Cilj pristupa zasnovanog na bazama podataka je da se podaci koji su se u prethodnom pristupu čuvali odvojeno (i više puta su zbog toga ponavljani) sada integrišu u jedinstvenu logičku strukturu. Svaki podatak se nalazi samo na jednom mestu u bazi podataka. Pristup zasnovan na bazama podataka ne uklanja redudansu u potpunosti, ali omogućava projektantu baze podataka da pažljivo isplanira vrstu i količinu redudanse. U nekim slučajevima je poželjno napraviti ograničenu redudansu kako bi se performanse baze podataka poboljšale (npr. brža pretraga).

- **Poboljšana konzistentnost podataka**

  Eliminisanjem (ili kontrolisanjem) redudanse podataka, u velikoj meri se smanjuju šanse za nekonzistentnošću podataka. Na primer, ukoliko je adresa kupca zapisana na samo jednom mestu ne može da postoji ne podudaranje u podacima u bazi podataka. Takođe, ažuriranje podataka je u velikoj meri uprošćeno, kada je svaka vrednost zapisana na samo jednom mestu. Na kraju, uklanjanjem redudanse podataka dolazi do uštede memorije.

- **Poboljšana razmena podataka**

  Baza podataka je dizajnirana kao resurs organizacije koji koriste svi njeni zaposleni (kojima je ona neophodna u opisu posla). Određenim internim i eksternim korisnicima je dozvoljeno korišćenje baze podataka, i svaki od njih (bio

u pitanju jedan korisnik ili grupa) ima jedan ili više pogleda koji mu olakšavaju korišćenje baze podataka. Korisnički pogled je logički opis jednog dela baze podataka koji je neophodan korisniku da obavi neki zadatak.

- **Veća bezbednost podataka**

  Kada veliki broj korisnika ima pristup istim podacima postoji visok nivo rizika da se podaci mogu zloupotrebiti ili da se čak mogu neovlašćeno menjati. Kompanije ulažu značajna sredstva, vreme i sopstvene resurse u cilju obezbeđenja svojih podataka. DBMS je okruženje preko koga se kroz funkcije administracije bolje upravljati sa bezbednosno osetljivim podacima i privilegijama krajnjih korisnika.

- **Povećana produktivnost u razvoju aplikacija**

  Velika prednost pristupa zasnovanog na bazama podataka je ta što se u znatnoj meri smanjuju troškovi i vreme potrebno za razvoj novih poslovnih aplikacija. Postoje dva važna razloga zašto se aplikacije baza podataka razvijaju znatno brže nego kod klasičnih sistema sa datotekama:

  - Pretpostavljajući da su baza podataka i sve propratne aplikacije već napravljene i implementirane, programer se može koncentrisati na određenu funkciju koja je neophodna za novu aplikaciju, a ne mora da razmišlja o definisanju podataka ili o detaljima vezanim za implementaciju.
  - DBMS pruža veliki broj alata za izveštavanje, kao što su generatori formi i izveštaja, i jezike uz pomoć kojih se automatizuju neke od aktivnosti kao što su dizajn i implementacija baza podataka.

- **Smanjena potreba za održavanjem programa**

  Sačuvani podaci se moraju često menjati iz velikog broja razloga: nove vrste podataka se dodaju, formati podataka se menjaju, i tako dalje. Poznat primer ovoga problema je ulazak u 2000-tu godinu, kada se sa uobičajenog sistema prikazivanja godina sa dve cifre moralo preći na četiri cifre. U sistemu obrade datoteka, metapodaci i logika pristupanju podacima se nalaze u individualnim aplikacionim programima (ovo je zavisnost između programa i podataka o kojoj je ranije bilo reči). Kao rezultat ovoga, promena formata podataka i metoda pristupanja momentalno dovodi do potrebe menjanja aplikativnih programa. Kod baza podataka, podaci su znatno više nezavisni od aplikativnih programa koji ih koriste. U okviru određenih granica, možemo da promenimo jednu od stavki, format podataka ili aplikativni program, a da ne moramo da promenimo drugu stavku. Kao rezultat ovoga, javlja se smanjenje potreba za održavanjem programa.

3.4 TROŠKOVI I RIZICI PRISTUPA ZASNOVANOG NA BAZAMA PODATAKA

U prethodnom delu teksta navedeno je nekoliko glavnih potencijalnih prednosti pristupa zasnovanog na bazama podataka. Međutim, kod velikog broja organizacija bilo je različitih problema kod ostvarenja i iskorišćenja tih prednosti. Na primer, postizanje nezavisnosti podataka (i stoga, smanjene potrebe za održavanjem programa) se pokazalo kao teško ostvarivo zbog ograničenja starijih modela baza podataka i softvera za upravljanje bazama podataka. Na sreću, relacioni modeli (kao i noviji objektno-orjentisani modeli) nemaju ovih problema. Drugi razlog za neuspeh da se iskoriste ove prednosti, je loše planiranje i implementacija baza podataka – čak ni najbolji softver za upravljanje bazama podataka ne može da prevaziđe ovakve manjkavosti. Pristup zasnovan na bazama podataka sadrži neke dodatne troškove i rizike koji se moraju rešavati kada se sistem počne primenjivati.

- **Novo, obučeno osoblje**

  Često se dešava da preduzeće, koje se odluči za pristup zasnovan na bazama podataka, mora da angažuje ili obuči ljude za projektovanje, implementiranje i održavanje baza podataka, kao i da te ljude uključi u postojeću radnu organizaciju. Dalje, zbog čestih izmena i brzine razvoja tehnologije, znanje ovog novog osoblja zahteva stalnu nadgradnju i unapređivanje. Jedino obučeno osoblje može da izvuče maksimum korisnosti iz novih tehnologija.

- **Troškovi i složenost instaliranja, upravljanja i rada sistema sa bazama podataka**

  Višekorisnički sistem za upravljanje bazama podataka je veliki i složen softver koji u startu mnogo košta, zahteva obučeno osoblje za instaliranje i rad i ima značajne godišnje troškove za održavanje i tehničku podršku. Instaliranje ovakvog sistema može zahtevati nadogradnju hardvera. Stalne obuke se podrazumevaju, da bi se mogle pratiti nove verzije i nadogradnje softvera. Takođe se može pojaviti potreba za dodatnim i skupljim softverom za baze podataka radi veće sigurnosti podataka.

- **Troškovi prilagođavanja (konvertovanja) podataka**

  Termin nasleđeni sistemi se uglavnom koristi kada se govori o starijim aplikacijama u preduzeću koje su bazirane na datotekama ili starijim bazama podataka. Troškovi prilagođavanja ovakvih starijih sistema za rad sa modernim bazama podataka (mereni u novcu, vremenu i zahtevnosti posla) često deluju kao velika prepreka za preduzeće.

- **Potreba za izradom sigurnosnih kopija i oporavkom podataka (backup)**

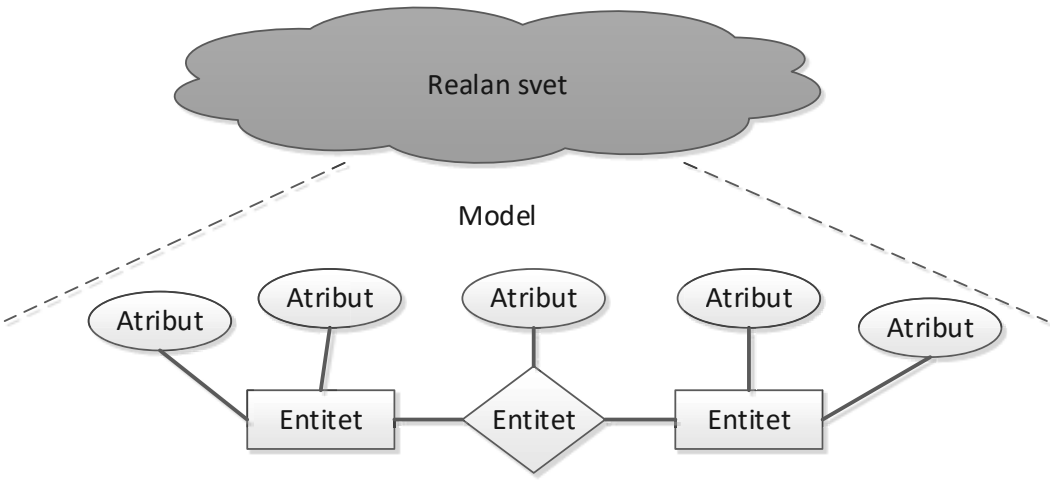
  Deljena baza podataka preduzeća uvek mora biti tačna i dostupna. To zahteva razvijanje i korišćenje jasnih procedura izrade sigurnosnih kopija kao i oporavak baze podataka kada neko oštećenje nastane. U današnjem okruženju, gde postoje raznovrsni bezbednosni rizici, rešavanje ovog problema je od izuzetne važnosti. Moderan sistem za upravljanje bazama podataka obično sam obavlja izradu sigurnosnih kopija i oporavak podataka u slučaju havarija.

- **Konflikti u organizaciji**

  Deljena baza podataka zahteva saglasnost u vezi sa definicijama i vlasništvom podataka, kao i utvrđenu osobu ili osobe odgovorne za održavanje podataka. Iskustvo je pokazalo da nesuglasice u pogledu definicija podataka, formata i kodiranja podataka, prava na ažuriranje deljenih podataka i sl. su česta i vrlo teška tema za rešavanje. Da bi se ovi problemi rešili potrebno je da je organizacija u potpunosti posvećena uvođenju/korišćenju pristupa zasnovanog na bazama podataka. Zatim je potreban sposoban administrator baze podataka kao i smislen pristup razvoju baza podataka. Ukoliko podrška i posvećenost glavnih menadžera za pristup okrenut bazama podataka izostane, velika je šansa da će krajnji korisnici razviti veći broj samostalnih baza podataka. Ove baze podataka će teško pružiti prednosti koje smo prethodno opisali. U krajnosti, one mogu da dovedu do donošenja loših odluka što naravno ugrožava celu organizaciju.

4 MODELOVANJE

Informacioni sistemi pojedinih firmi omogućavaju upravljanje podacima koji su bitni za njeno poslovanje. Međutim, broj internih podataka i podataka iz okruženja je ogroman te je nemoguće sve podatke i sve uočene detalje opisati i sačuvati unutar informacionog sistema. Postupkom selekcije identifikuju se i čuvaju samo relevantni podaci. Time se dolazi do pojma modela podataka. On je izraz i posledica zahteva za obradom podataka relevantnih za određeno područje primene. Modeli su čovekovo sredstvo pojednostavljivanja problema i njegovo posmatranje samo sa stanovišta bitnih za ciljeve analize. Objekt posmatranja (npr. automobil) ima uvek više osobina (atributa) od kojih u datom trenutku analize može biti dovoljan samo njihov manji broj (npr. samo *registarski broj, tip automobila, ime i prezime vlasnika*). To su najvažniji atributi potrebni u postupku pretraživanja i pronalaženja vlasnika vozila na osnovu registarskog broja vozila unutar jednog informacionog sistema. Ostali atributi kao što su *boja, godina proizvodnje, broj sedišta* i sl. nisu bitni (mogu se zanemariti) za takav postupak. Čovek, obdaren sposobnostima apstraktnog načina mišljenja, stvara jedan apstraktni model realnog sveta. Takav model realnog sveta (objekta posmatranja) zasniva se na simbolima i zove se konceptualni model podataka.



Slika 4.1: Realan svet i njegov model

Modelovanje podataka se radi paralelno sa analizom potreba. Kako se informacije prikupljaju, objekti se identifikuju, dodeljuju im se imena koristeći termine bliske krajnjim korisnicima. Objekti se onda modeluju i analiziraju korišćenjem dijagrama objekti-veze (ER dijagrami). Dijagram se može pregledati od strane dizajnera i krajnjeg korisnika da bi

se osigurala njegova kompletnost i tačnost. Ako model nije tačan, modifikuje se, što ponekad zahteva da se prikupe dodatne informacije. Ciklus pregledanja i modifikovanja se nastavlja sve dok se ne dobije potvrda da je model korektan.

4.1 RAZVOJ KONCEPTUALNIH MODELA

Objekti iz realnog sveta se u računarskoj primeni opisuju pomoću podataka. Podaci su zato apstrakcija realnosti, tj. sredstva za kodiranje osobina objekata iz realnog sveta. Modelovane, kao postupak kojim se realni svet svodi na određeni broj podataka, predstavlja kompleksan posao i sastoji se iz više koraka:

- **Izbor (selekcija).** U prvom koraku se mnoštvo objekata iz realnog sveta redukuje na manji skup objekata, koji će činiti objekte modela. Npr. objekti mogu biti student, predmet, profesor, studentska služba, polaganje ispita i sl. U procesu selekcije ovaj broj objekata se može redukovati na manji broj, ako je cilj praćenje uspešnosti studiranja na fakultetu. Time se složenost realnog sistema smanjuje. Selekcija se ne odnosi samo na objekte nego i na njihove osobine, kao i na međusobne veze (relacije) između objekata.
- **Imenovanje.** Svakom objektu u realnom svetu, svakoj vezi između uočenih objekata, kao i svakom atributu (svojstvu) uočenog objekta ili veze dodeljuje se ime.
- **Klasifikacija.** Nehomogeni skup objekata i odnosa se svrstava u homogene klase i tipove objekata. Klasifikacija uvek zavisi od područja primene.

Rezultat navedenih koraka modelovanja zove se konceptualni model. On sadrži, za posmatrani problem iz realnog sveta, sve relevantne tipove objekata, njihove osobine i međusobne veze. Rezultati proučavanja modela podataka doveli su do saznanja da svaki model podataka ima tri neodvojive komponente:

- strukturu podataka,
- operacije nad podacima,
- ograničenja (*constraints*).

Struktura i ograničenja, za razliku od operacija, opisuju stanje realnog sistema, tj. predstavljaju statički opis stanja sistema. Strukturu modela čine objekti, njihova svojstva, veze između objekata i njihovih svojstava. Operacije nad podacima u modelu su, u stvari, operacije nad strukturom modela kojima se izražava dinamika realnog sistema. Operacije izražavaju kretanje i promene tj. dinamiku realnog sistema. Ograničenja su pravila koja razdvajaju dopuštena od nedopuštenih stanja realnog sistema i u svojoj prirodi deo su strukture modela podataka. Ponekad se ne posmatraju kao odvojene komponenta, nego kao deo strukture modela podataka.

4.1.1 Entiteti

Modelima podataka nastoji se preslikati realan sistem. Realan sistem sastoji se od objekata iz realnog sveta i njihovih veza između kojih se uspostavljaju različiti odnosi. Pod entitetom se podrazumeva sve što se može jednoznačno odrediti, identifikovati i razlikovati. Tako široko postavljena definicija pokazuje da entitet može biti svaki "realan" ili "apstraktan" objekt o kojem u određenom trenutku razmišljamo. Entitet je realan ako fizički, stvarno postoji. Najopštije se može tvrditi da su granice entiteta u modelu podataka određene ljudskim pogledom i načinom razmišljanja.

Svaki entitet uočen u realnom sistemu ima svoje osobine koje ga čine složenim i njihove vrednosti omogućavaju razlikovanje entiteta. Svojstvo entiteta uključuje dva elementa - atribut i vrednost atributa (npr. entitet *Student* ima atribute: *Ime, Prezime, Broj indeksa, Adresu, Telefon* i sl. i vrednosti *Marko, Marković, 123/03, Danijelova, 15, 011/376-543* respektivno). Svaki put kada se promeni vrednost atributa, potrebno je promenu evidentirati, tj. ažurirati tu vrednost atributa za dati entitet.

Precizno govoreći, objekti koji se označe pojmom entiteta mogu se zvati klase entiteta. Svaki objekt ima osobine (atribute) klase entiteta kojoj pripada. Npr. klasu entiteta *Student* čine pojedinačni entiteti od kojih svaki ima zajedničke atribute: *Ime, Prezime, Broj indeksa, Adresa, Telefon* i sl. Svaki pojedinačni entitet ima sve navedene atribute, ali će se razlikovati od drugih entiteta po vrednostima pojedinih atributa.

Atribut opisuje entitet. Jedno konkretno pojavljivanje atributa naziva se vrednost. Ako je atribut dovoljno složen, tako da ima svoje dodatne atribute, može se posmatrati kao novi entitet.

Domen atributa je skup svih mogućih vrednosti koje atribut može poprimiti.

Primarni ključ je jedan ili više atributa čija vrednost jednoznačno određuje primerak entiteta. Na primer, za entitet *Auto*, primarni ključ je atribut registarski broj. Dva različita člana ili primerka entiteta ne mogu imati isti primarni ključ. Primarni ključ je jedinstven za svakog člana entiteta. Na primer, za entitet *Student* primarni ključ bi mogao biti broj indeksa.

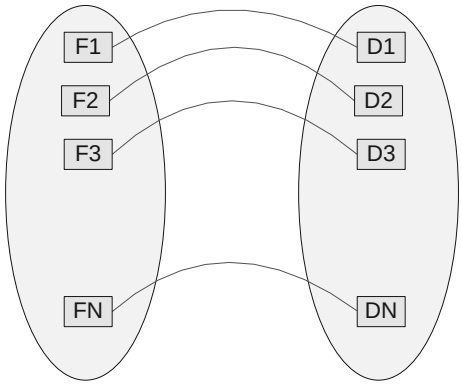
4.1.2 Veze između entiteta

Baza podataka se ne odnosi samo na pojedinačne objekte nego i na odnose između objekata. U realnom sistemu objekti nisu međusobno izolovani, nego se nalaze u međusobnoj interakciji. Student se upisuje na fakultet, sluša predavanja iz pojedinih predmeta, prijavljuje polaganje ispita, polaže ispit itd. To su primeri logičkih i realnih veza između objekata, koje slede iz realnih odnosa u posmatranom sistemu studiranja na jednom fakultetu. Istražimo jedan skup odnosa između studenata koji slušaju

predavanja kod određenog profesora. Postavlja se pitanje šta su u takvim odnosima objekti, koje su njihove osobine (atributi) i kako prikazati njihove odnose.

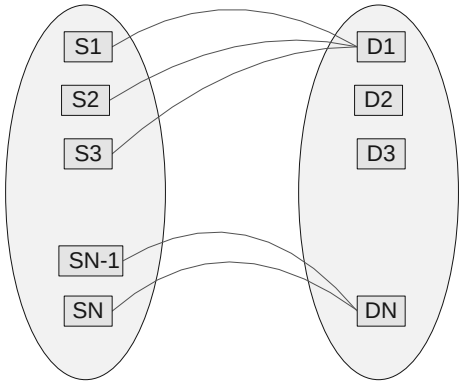
Identifikovati objekte, njihove osobine i odnose znači praktično izgraditi model podataka. U modelu podataka ne postoje samo atributi objekta, nego i veze između objekata. Prvo se selektuju objekti, imenuju se, a zatim se analiziraju tipovi odnosa koji se uspostavljaju između objekata. Odnosi između objekata posmatranja prikazuju se najčešće primenom logike skupova i preslikavanja njihovih elemenata.

Najjednostavniji odnos između ta dva tipa objekata naziva se preslikavanje 1:1. Kod takvog preslikavanja svaki se element skupa X može preslikati na najviše jedan element skupa Y. Istovremeno, i svaki element skupa Y može biti preslikan na najviše jedan element skupa X. Karakterističan primer bi bio sa entitetima *Fakultet* i *Dekan*. Na jednom fakultetu može biti samo jedan dekan, a jedan dekan može biti dekan na samo jednom fakultetu. Takvi odnosi između entiteta su retki, a mogu se predstaviti sledećom slikom:



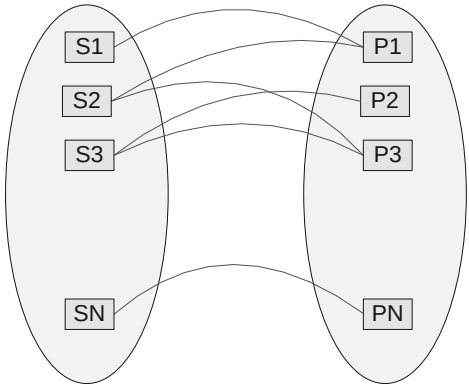
Slika 4.2: Preslikavanje entiteta 1:1

Druga vrsta odnosa naziva se preslikavanje N:1 (ili 1:N). Više elementa skupa X može se preslikati na najviše jedan element skupa Y. Istovremeno jedan element skupa Y može se preslikati na više elemenata skupa X. Pogodan primer za ovu vrstu odnosa između entiteta je odnos između entiteta *Student* i *Dekan*. Više studenata na jednom fakultetu ima samo jednog dekana, a jedan dekan je dekan za više studenata na svom fakultetu.



Slika 4.3: Preslikavanje entiteta N:1

Najsloženije preslikavanje je tipa M:N. Svaki element prvog skupa može se preslikati na više elemenata drugog skupa, ali se i svaki element drugog skupa može preslikati na više elemenata prvog skupa. Karakterističan primer ovakvih veza postoji ako se uoče entiteti *Student* i *Profesor*. Jednom studentu predaje više profesora, a ujedno jedan profesor predaje za više studenata.



Slika 4.4: Preslikavanje tipa M:N

4.1.3 Troslojna arhitektura za apstrakciju podataka

Osnovni koncept baze podataka je ideja o skupu činjenica ili delova znanja. Činjenice mogu da budu struktuirane na različite načine koji se nazivaju modeli podataka. Model podataka nije statična struktura nego se menja kako bi odražavao promene koje se dešavaju i u realnom sistemu. Na primeru informacionog sistema

jednog fakulteta, studenti polažu pojedine ispite, neke poništavaju i dobijaju drugačije ocene, upisuju se novi studenti, drugi diplomiraju, neki asistenti postaju profesori itd.

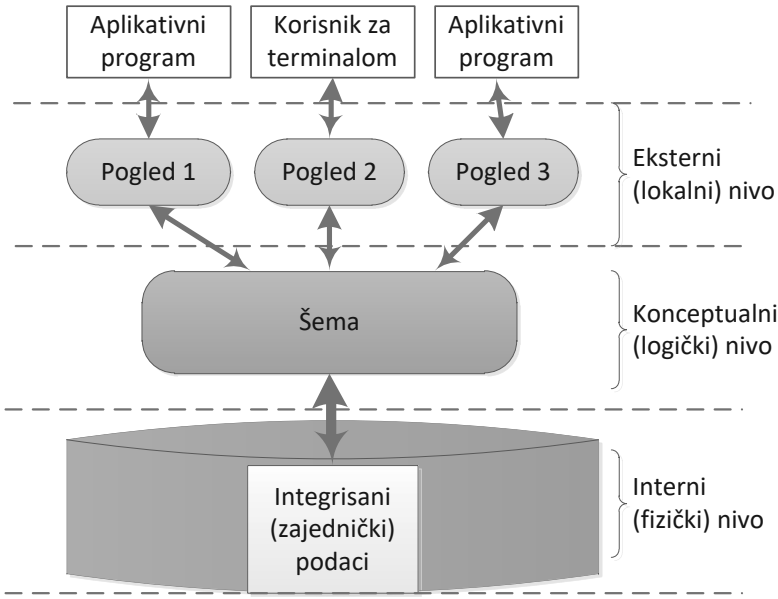
Za jednostavne slučajeve, kao i mali broj promena relacija i entiteta, moguće je ažuriranje podataka vršiti ručno. Za kompleksnije sisteme (sa nekoliko stotina ili hiljada entiteta ili relacija) ažuriranje podataka postaje ogroman problem. Jedino se uz pomoć računara može održavati ažurnost podataka u velikim informacionim sistemima. Obrada podataka postaje ne samo pitanje produktivnosti neke firme ili organizacije, nego i opstanka, rasta i razvoja u okruženju s intenzivnom konkurencijom. Obrada podataka je deo svakog poslovnog procesa, stoga je poznavanje baza podataka bitno ne samo za projektante informacionih sistema i programere, nego i za krajnje korisnike rezultata takvih obrada. Oni nisu samo skup povremenih korisnika baza podataka, kao što se to može reći za programere ili projektante informacionih sistema. Danas veliki broj zaposlenih, koji nisu upoznati sa konceptualnom šemom BP, kreiraju, unose, ažuriraju ili jednostavno koriste baze podataka na različitim nivoima organizovanosti poslovnih sistema.

U suštini sistemi baza podataka su skupovi međusobno povezanih datoteka sa podacima i skupovi programa koji omogućavaju korisnicima da pristupaju i menjaju ove podatke na efikasan način. Glavni cilj baza podataka je da se korisnicima pruži apstraktan pogled na podatke. Na taj način skrivaju se detalji o tome kako se podaci skladište i održavaju. Potreba za efikasnošću dovela je do toga da dizajneri sistema moraju da koriste kompleksne strukture za predstavljanje podataka u bazama podataka. Budući da mnogi korisnici baza podataka nisu obučeni za rad sa računarom, programeri skrivaju složenost podataka kroz nekoliko nivoa apstrakcije. Cilj je da se pojednostavi interakcija korisnika sa bazom podataka.

Model baze podataka danas je poznat kao ANSI/X3/SPARC model (Slika 4.5). Na bazi tog modela razvijeni su sistemi za upravljanje bazama podataka koji imaju troslojnu arhitekturu ili varijantu te arhitekture. Aplikativni programi komuniciraju s bazom podataka preko odgovarajućeg eksternog modela. Zahtev za učitavanje određenih podataka aplikativni sloj upućuje na eksterni sloj, odnosno odgovarajući korisnički model. DBMS preslikava eksterni model na konceptualni i konceptualni na interni model.

Konceptualni (logički) nivo je najbliži stvarnosti. Taj se nivo definiše u procesu kreiranja modela podataka. Jedan od ciljeva modela podataka je oblikovanje podataka za sadašnje i buduće aplikacije. Može se reći da konceptualni nivo čine sve relacione šeme modela podataka, sve relacije i ograničenja. Najjednostavnije rečeno, logički sloj opisuje šta je smešteno u bazi i koji odnosi postoje između tih podataka. Obično se logički sloj sastoji od određenog broja jednostavnih međusobno povezanih struktura. Ove jednostavne strukture često zahtevaju kompleksne strukture na fizičkom sloju. Cilj baza podataka je da korisnik logičkog nivoa ne treba da bude upoznat sa složenošću fizičkog sloja. Na primer, administratori baza podataka su najčešće upoznati sa detaljima logičkog sloja.

Spoljašnji nivoi (modeli A, B i C) formiraju se na temelju konceptualnog nivoa i predstavljaju samo pogled (VIEW) prema potrebama pojedinih korisnika. Uobičajeno je da su to samo delovi ukupne baze podataka. Sistem sa bazama podataka obično ima više pogleda na jedinstvenu bazu podataka.



Slika 4.5: Troslojna arhitektura apstrakcije podataka

Interni (fizički) sloj baze odnosi se na zapisivanje konceptualnog sloja na nekom medijumu za čuvanje (najčešće disku). Radi se o slogovima zapisanim u datotekama. Niži sloj, uslovno rečeno, ili nivo bliži disku od internog sloja BP, je operativni sistem, koji na osnovu logičkih adresa slogova čita sadržaj diska.

4.2 MODELI BAZA PODATAKA

Za modelovanje strukture podataka koriste se različite tehnike. Određeni modeli se lakše koriste za neke tipove sistema upravljanja bazama podataka nego drugi modeli. Model čini osnovu za osmišljavanje, definisanje i implementaciju baze podataka.

Istorijski gledano sistemi za upravljanje bazama podataka mogu se podeliti u sledeće osnovne modele:

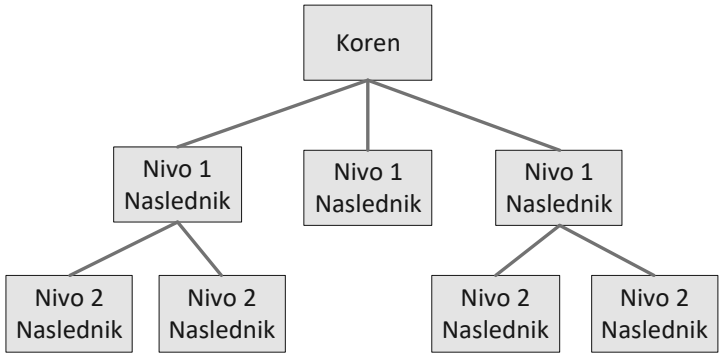
- **Hijerarhijski model** – čine ga podaci složeni u hijerarhijsku strukturu;
- **Mrežni model** – može se predstaviti usmerenim grafom u kojem su čvorišta podaci, a lukovi među čvorištima definišu veze među podacima;

- **Relacioni model** – zasnovan na matematičkom pojmu relacije. Podaci i veze među podacima se prikazuju preko dvodimenzionalnih tabela.
- **Objektni model** – bazira se na konceptu objekata, koji predstavljaju skup podataka i operacija koje se na njima mogu izvršavati.

4.2.1 Hijerarhijski model

Hijerarhijski model je najstariji od svih modela baza podataka, i za razliku od mrežnog, relacionog ili objektno orjentisanog, nema dobro dokumentovanu istoriju svoje koncepcije i početne verzije ovakvog modela. Ovaj model se razvio iz informacionog sistema za upravljanje u 50-tim i 60-tim godinama prošlog veka. Usvojen je u mnogim bankama i osiguravajućim društvima koji ga, kao nasleđe, i danas koriste.

U hijerarhijskom modelu podaci su smešteni u seriju slogova (zapisa) Da bi se uspostavila veza između slogova, hijerarhijski model uspostavlja relaciju roditelj – naslednik. Ovo je takozvano 1:N mapiranje između slogova koje se radi korišćenjem stabla. U ovom modelu, relacije su takve da jedan naslednik može imati samo jednog roditelja, ali roditelj može imati više naslednika. Roditelji i naslednici su povezani vezama koje se nazivaju pokazivači (u fizičkoj realizaciji to je adresa u memoriji gde se slog nalazi). Roditelj ima listu pokazivača za svakog od svojih naslednika. Hijerarhijski model je dobro uređena struktura, koja podseća na hijerarhijsku strukturu u npr. državi, vojsci ili nekoj velikoj organizaciji.



Slika 4.6: Šematski prikaz jednog hijerarhijskog modela

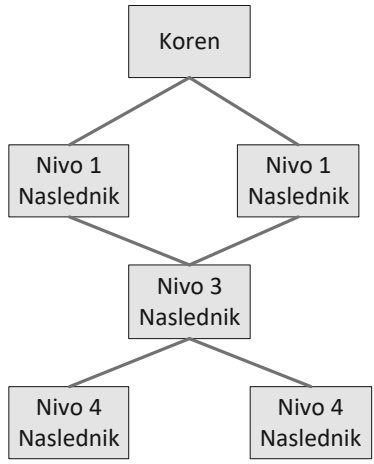
Pravilo roditelj – naslednik omogućava pristup podacima. Da bi se došlo do tabele na nižem nivou, kreće se od korena i ide prema dole kroz stablo dok se ne dođe do cilja. Naravno, očigledan problem sa ovim modelom je da korisnik mora da zna kako je stablo organizovano da bi pronašao bilo šta.

Hijerarhijski model ima ozbiljnih nedostataka. Na primer, ne može se dodati slog u tabelu naslednika dok se ne uključi u roditeljsku tabelu. Hijerarhijski model je sposoban da radi jedino sa jednostrukim stablima, ali ne može da se nosi sa povezivanjem ogranaka ili stvaranjem višestrukih veza. Zbog toga se stvara redudansa (višestruko pojavljivanje) podataka i mogućnost netačnog ažuriranja. Na primeru hijerarhijske organizacije nekog fakulteta koji ima katedre, profesore, studente itd. mogu se lako uočiti navedene slabosti. Lako je predstaviti da na jednoj katedri ima više profesora, ali se ne može predstaviti da jedan profesor radi na više katedri. Da bi se ovo uradilo, mora postojati dva pojavljivanja istog profesora. To može dovesti do netačnosti kod ažuriranja podataka, npr. moguće je da informacije budu različite u dva zapisa, što vodi do konfuzije.

Hijerarhijski model se više ne koristi kao osnova za trenutne komercijalne sisteme, ali još uvek postoji mnogo nasleđenih sistema baziranih na ovom modelu. Zbog svih nedostataka koji postoje u hijerarhijskom modelu, razvijen je mrežni model.

4.2.2 Mrežni model

Mrežni model je prvi put predstavljen 1971. godine. Može se smatrati savremenikom relacionog modela, gledajući starost i prva istraživanja učinjena u 60-tim godinama prošlog veka. Omogućava da se višestruki skupovi podataka koriste zajedno putem pokazivača (ili pointera). Neke kolone sadrže pokazivače na druge tabele umesto samih podataka. Na taj način, tabele su povezane pokazivačima i mogu se posmatrati kao mrežna struktura. Dok u hijerarhijskom modelu svaki slog ima jedan „roditeljski“ slog i neograničeno „naslednika“, mrežni model omogućava svakom zapisu da ima višestruke roditelje i naslednike, kreirajući mrežastu strukturu.



Slika 4.7: Šema mrežnog modela

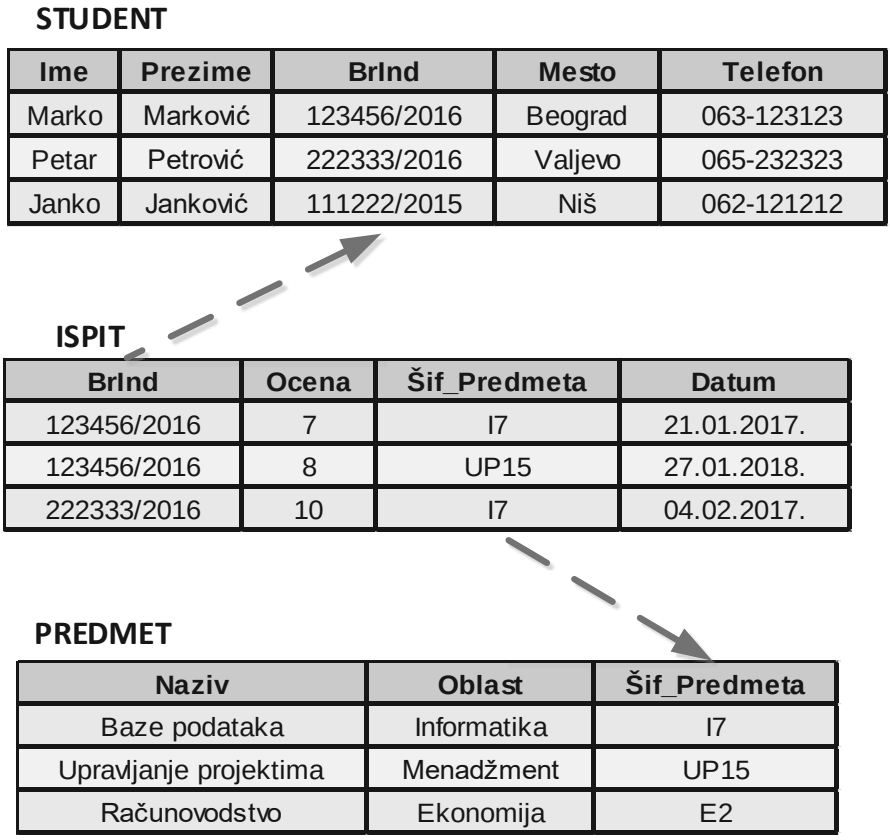
Mrežni model se danas uglavnom ne upotrebljava za dizajniranje baza podataka, ali ipak ima slučajeva gde se kao deo nasleđa koristi u nekim kompanijama. Predstavlja unapređenje hijerarhijskog modela, ali je kompleksan i težak za upotrebu. Pored toga, teško ga je podržati matematičkim aparatom, što onemogućava kasnije efikasno programiranje.

4.2.3 Relacioni model

Kao i mnoge druge tehnologije u računarskoj industriji, koreni relacionih BP potiču iz IBM-a i njihovog istraživanja automatizovanja kancelarijskih operacija u 60-tim i 70-tim godinama XX veka. U srcu relacionog modela nalazi se koncept tabele (koja se naziva relacija) u kojoj su smešteni svi podaci. Svaka tabela je načinjena od slogova (redova u tabeli), a svaki slog ima svoja polja (atribute). Osnovne karakteristike relacionog modela podataka su sledeće:

- Sve se predstavlja relacijama (tabelama);
- Zasniva se na strogoj matematičkoj teoriji;
- Minimalna redundansa podataka;
- Jednostavno ažuriranje podataka;
- Izbegnute su anomalije ažuriranja;
- Redosled kolona i redova ne utiče na informacioni sadržaj tabele;
- Ne mogu da egzistiraju dva identična reda (zapisa) u jednoj tabeli;
- Svaki red se može jednoznačno odrediti (postoji primarni ključ);
- ...

U relacionom modelu podataka klase objekata se predstavljaju tabelama. Na primer klasa *Student* se može opisati atributima *Ime, Prezime, Broj indeksa, Mesto i Telefon*. Klasa *Predmet* sa atributima *Naziv predmeta, Oblast i Šifra predmeta*. Neki od studenata su polagali neke predmete i dobili ocene na ispitu, što je prikazano u veznoj tabeli *Ispit*, koja se može opisati atributima *Broj indeksa, Ocena, Šifra predmeta i Datum*. Trenutno stanje studenata, predmeta i ocena sa ispita je uneseno u tabele i prikazano na slici 4.8.



Slika 4.8: Relacioni model podataka

Suština relacionog modela je da se i klase objekata i klase veza između objekata predstavljaju na jedinstven način, tj. preko tabela. U našem primeru postoje tri tabele: STUDENT, PREDMET i ISPIT. U relacionom modelu podataka tabela se definiše kao relacija, koja mora da ispuni odgovarajuće uslove. Svaka relacija mora da ima primarni ključ – jedan ili više atributa koji na jedinstven način opisuju svaki zapis u jednoj tabeli. Primarni ključ se pažljivo bira. Na primer u klasi studenata loš izbor primarnog ključa bi bio atribut *Ime*, zato što se mogu pojaviti dva studenta sa istim imenom. Dobar izbor primarnog ključa je atribut *Broj indeksa*, zato što ne postoje dva studenta sa istim brojem indeksa. Za klase objekata *Student* i *Predmet* vrši se prevođenje u relacioni model na sledeći način (podvlačenjem su označeni atributi koji čine primarni ključ):

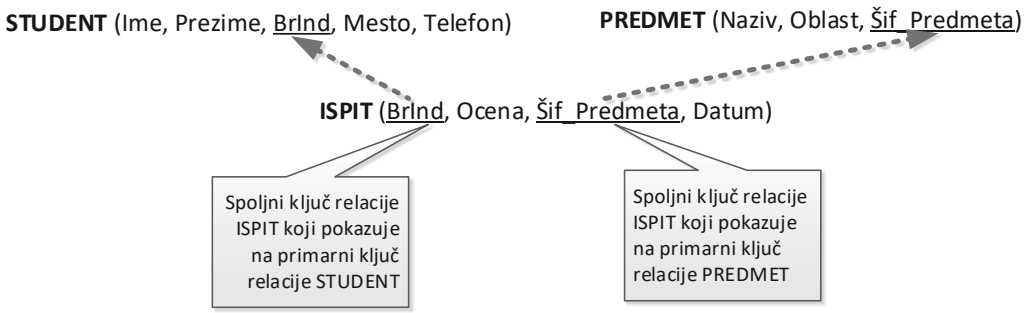
STUDENT (Ime, Prezime, <u>BrInd</u>, Mesto, Telefon),

PREDMET (Naziv, Oblast, <u>Šif Predmeta</u>)

Za klasu veza ISPIT, može se definisati prirodan primarni ključ u odnosu na objekte koje povezuje. U našem primeru relacija ISPIT bi glasila:

ISPIT(BrInd, Ocena, Šif_Predmeta, Datum)

Dakle, za posmatrani realan slučaj gde su studenti polagali neke predmete i dobili ocene, izvršeno je modelovanje preko tri tabele tj. relacije. Sve tabele su povezane. Povezivanje se vrši preko vrednosti atributa u relacijama. na sledeći način:



Slika 4.9: Relacije se povezuju vrednostima spoljnih i primarnih ključeva

Svaka relacija (tabela) se identifikuje jedinstvenim imenom. Redosled atributa u okviru jedne relacije nije od značaja. Nazivi atributa u okviru jedne relacije moraju da budu unikatni. Nema potrebe da se vodi računa o tome kako su podaci smešteni na disku. Ovo je različito od hijerarhijskog i mrežnog modela u kojima korisnik mora da razume kako su podaci struktuirani unutar BP da bi mogao da ih pretražuje, unosi nove, ažurira ili briše postojeće slogove.

Snaga relacionog modela podataka dolazi do izražaja kod održavanja, tj. ažuriranja podataka u relacijama. Relacije STUDENT i PREDMET su nezavisne relacije i zapisi u takve relacije mogu da se unose nezavisno. DBMS (sistem za upravljanje bazama podataka) vodi računa da u jednu relaciju nije moguće uneti dva identična zapisa (reda). Ova osobina se naziva identifikacioni integritet. Relacija ISPIT je zavisna, što znači da se u nju mogu uneti zapisi samo ukoliko vrednost atributa BrInd odgovara nekoj vrednosti atributa BrInd u relaciji STUDENT. Pored toga, vrednost atributa Šif_Predmeta mora da odgovara bar jednoj vrednosti u relaciji PREDMET. Ova osobina se naziva referencijalni integritet i o njemu takođe brine DBMS. Na kraju, nakon unosa podataka, u nezavisnim relacijama STUDENT i PREDMET nije moguće brisanje zapisa ukoliko jedna od vrednosti atributa BrInd i Šif_Predmeta odgovara bar jednoj od vrednosti tih atributa u relaciji ISPIT, zato što bi se brisanjem narušio tzv. referencijalni integritet. Navedena pravila se mogu ublažiti ukoliko se kod operacija ažuriranja navedu drugačije specifikacije referencijalnog integriteta.

4.2.4 Objektni model

Objektno orjentisani model je jedan od novijih modela baza podataka. Istraživači su za njega postali zainteresovani krajem 70-tih i početkom 80-tih godina prošlog veka, kada se koncept objektno orjentisanih sistema počeo pojavljivati. Bazira se na konceptu objekata, koji predstavljaju skup podataka i operacija koje se na njima mogu izvršavati.

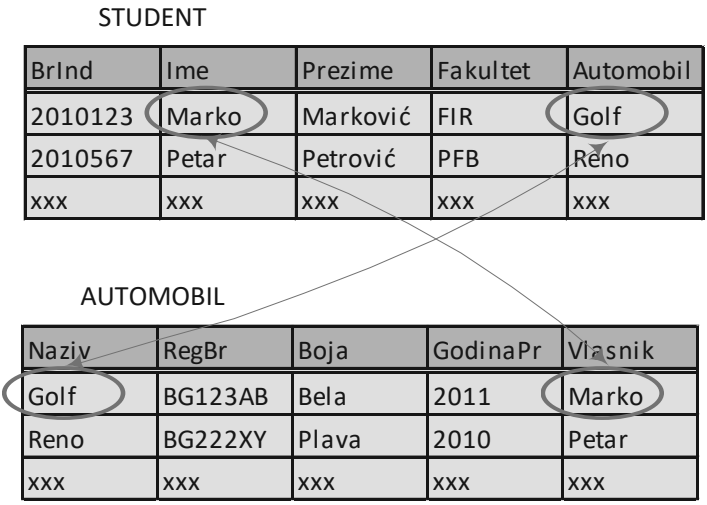
Istraživanje se radilo i da bi se prevazišla mnoga ograničenja u relacionom modelu na određenim tipovima podataka. Tipovi podataka koji se mogu koristiti u relacionim bazama su veoma ograničeni. Svaki atribut (polje) može da poprimi samo jednu vrednost. U objektno orijentisanom modelu podataka entitet se predstavlja klasom. Klasa obuhvata i atribute i ponašanje entiteta (moguće operacije nad podacima). Npr. Klasa: **student**

- Atributi: BrInd, Ime, Prezime, Fakultet
- Procedura: **polaganjeIspita()**

Objekti su samo jedno pojavljivanje u odgovarajućoj klasi. Objektno orijentisan model karakteriše bogatstvo tipova podataka – tip može biti i drugi objekat. Direktna veza između objekata u aplikaciji i objekata u BP rezultuje boljim performansama baze podataka.

Posmatrajmo primer u kome se vodi evidencija o Studentima sa svojstvima: Broj indeksa, Ime, Prezime, Fakultet i Tip automobila koji student poseduje. Dalje vodi se evidencija o Automobilima sa svojstvima Naziv automobila, Registarski broj, Boja, Godište i Vlasnik. Prethodni primer se može prikazati na sledeći način

Objektno orjentisani DBMS-ovi omogućavaju čuvanje objekata direktno, bez mapiranja za različite strukture podataka. Relacioni DBMS zahteva mapiranje iz objekata u tabele. U objektno orijentisanom modelu, informacija je sačuvana kao stalni objekt, a ne kao red u tabeli. Ovo sistem čini efikasnijim u smislu prostora potrebnog za smeštanje i čuvanje podataka i osigurava da korisnik manipuliše podacima samo na onaj način koji je programer odredio.



Slika 4.10: U objektno orijentisanim BP tip podatka može biti drugi objekat

S druge strane, obzirom da se kontrola vrši na veoma niskom nivou, mnogo je teže za treću stranu da napravi neki dodatak. Dok kod relacionih baza podataka možemo imati korist od softvera izrađenog od strane trećeg dobavljača, korisnici objektno orjentisanih sistema za upravljanje bazama podataka ili moraju da naruče dodatni softver od originalnog programera ili da ga razviju u saradnji sa drugim firmama koje koriste isti sistem.

4.3 MODEL OBJEKTI VEZE

Početi kreiranje baze podataka nekog informacionog sistema, u suštini znači doći do kompleta CREATE naredbi kojima se definiše šema baze podataka - tabele, relevantni atributi, domeni, ograničenja, itd. U osnovi projektovanja baze podataka je utvrđivanje preciznog modela organizacije za koju se radi informacioni sistem. Kao i u ostalim inženjerskim disciplinama, složenost ovakvog posla zahteva da proces kreiranja bude izveden dobro definisanom metodologijom i da bude testiran u skladu sa objektivnim kriterijumima. Jedan od najvećih problema u procesu razvoja BP je činjenica da projektanti, programeri i krajnji korisnici na potpuno različite načine shvataju podatke i načine njihove upotrebe, kao i procese iz posmatranog okruženja koje treba modelirati. Da bi se obezbedio precizan opis prirode podataka i načina na koji se oni koriste, potrebno je proizvesti jasan model koji nije striktno tehničke prirode. Najčešće korišćeni model u praksi je model objekti-veze.

U ovom poglavlju data je metodologija kreiranja relacionih baza podataka na bazi standardnog Modela Objekti-Veze (Chen 1976). Kreiranje baza podataka je praktično

proces u dva koraka. Početna faza je bazirana na MOV modelu, a druga faza je implementacija. Rezultat MOV faze se redefiniše primenom teorije normalizacije posle koje se garantuje kvalitet šema relacija u skladu sa odgovarajućim kriterijumima. Tipičan dizajn baze podataka obuhvata definisanje skupova relacija, na stotine atributa, i mnogo ograničenja koja proističu iz ograničenja u realnom svetu. Dizajniranje baze podataka zahteva dobar nivo kreativnosti, iskustva, tehničke ekspertize i razumevanje osnovnih pravila. Glavna komponenta MOV pristupa je koncept entiteta (objekata i veza) - opšti pojam za nešto što postoji i može se jednoznačno prepoznati. Entiteti obuhvataju objekte koji se nalaze u jednoj organizaciji, npr. studenti, profesori i predavanja na univerzitetu. Dalje, entiteti obuhvataju i veze među objektima jedne organizacije, na primer profesor_predaje_predmet. Ograničenja integriteta entiteta i veza čine važan deo MOV opisa odnosno specifikacije. Na primer profesor može da predaje jedno predavanje u određenom vremenu u jednoj sali na fakultetu.

MOV modelovanje obuhvata:

- Skup entiteta (objekti i veze)
- Uočavanje ograničenja
- Definisanje ključeva
- Grafička predstava (ER dijagram)
- Definisanje atributa
- Dizajn globalne šeme
- Svođenje globalne šeme na tabele (relacije)

4.3.1 Dijagrami objekti-veze (DOV)

Dijagram objekti-veze (DOV[1])je grafička prezentacija povezanih entiteta i ograničenja koja čine dati dizajn odnosno projekat. Kao i kod ostalih vizuelno orijentisanih dizajn metodologija, on pruža grafički sažetak strukture baze podataka koji je veoma koristan dizajneru - ne samo u procenjivanju tačnosti, odnosno pravilnosti dizajna, nego i za savetovanje sa kolegama i za objašnjavanje programerima koji će je koristiti. Na žalost, ne postoji standardan plan za MOV dijagrame. Kada je jednom određena organizacija predstavljena setom DOV dijagrama, postoje klasični načini koji dijagrame pretvaraju u skupove CREATE TABLE naredbi. Važna prednost ove metodologije je da dizajner može da se usredsredi na celokupno i tačno modelovanje organizacije, a da efikasnost izvršavanja zadatih upita i ažuriranja u odnosu na krajnju bazu podataka stavi u drugi plan. Kasnije, kada se MOV dijagrami pretvore u CREATE naredbe, dizajner može dodati efikasnost koristeći teoriju normalizacije polaznih šema relacija.

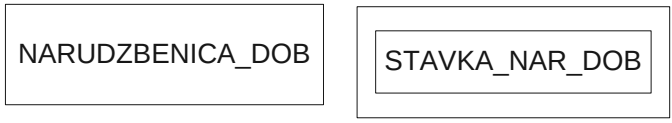
[1] U originalu: ERD – *entity relationships diagrams*

U DTP (dijagramima tokova podataka) koji su analizirani u prethodnom poglavlju nisu prikazane nikakve veze između tokova podatka, odnosno između skladišta podataka. U stvarnosti te veze postoje, a očigledan dokaz njihovog postojanja su rečnici podataka. Na primer struktuirani tip *NARUDZBENICA* sadrži podatke dobavljača, podatke artikala koji se naručuju i podatke naručioca. Sva tri navedena entiteta predstavljaju strukture za sebe. Dijagrami objekti veze (DOV) otklanjaju ovaj nedostatak. Oni se sastoje od tri osnovne komponente:

- Objekti (entiteti)
- Atributi
- Veze (relacije)

4.3.2 Objekti

Objekti grupišu srodne podatke. Mogu predstavljati entitete iz realnog sveta, interfejse iz DTP, strukture iz rečnika podataka, ali mogu biti i čisti fabrikati, koji samo treba da istaknu povezanost različitih podataka pri procesiranju u sistemu. Entitet objekat egzistira nezavisno i može predstavljati fizički, realni objekat (npr. Sredstvo) ili konceptualni, apstraktni objekat (npr. Radno iskustvo). Objekat se identifikuje nazivom i listom svojstava, a grafički se predstavlja kao pravougaonik u kome se ispisuje naziv entiteta, koji je najčešće imenica. U DOV se razlikuju takozvani *jaki* i *slabi* objekti.

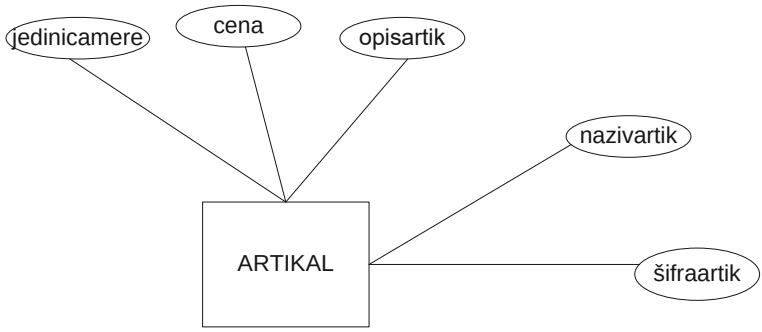


Slika 4.11: Primer označavanja objekata

Na primeru označavanja objekata (Slika 4.11), *narudžbenica* je prikazana kao jak a *stavka narudžbenice* kao slab objekat. Između jakog i slabog objekta postoji identifikaciona i egzistencijalna zavisnost što znači da stavka narudžbenice ne može postojati u skladištu podataka ako ne postoji narudžbenica koja ju identifikuje.

4.3.3 Atributi

Atributi su osobine (svojstva) entiteta. Atribut podrazumeva ime i vrednost svojstva (npr. atribut “boja” i njegova vrednost “plavo”). Entitet se opisuje pomoću jednog ili više svojstava (atributa). Atributi su podaci osnovnog tipa, ili predefinisani domeni. Označavaju se elipsoidima i povezani su pravolinijskim konektorima sa objektima (Slika 4.12).



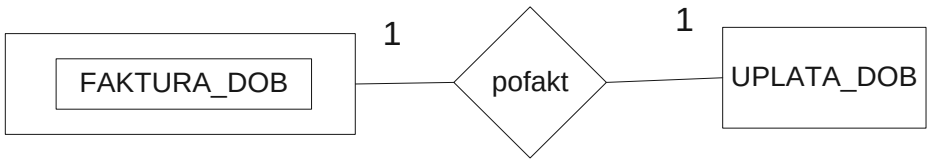
Slika 4.12: Primer označavanja atributa objekata

Broj atributa objekata nije ograničen, kao ni pozicija na kojoj će se atributi uneti u dijagram.

4.3.4 Veze

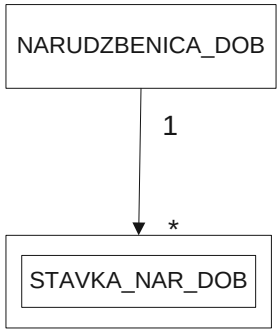
Veze su najvažniji deo DOV, jer definišu načine na kojima su objekti uzajamno povezani. Veze se imenuju i njihovi nazivi oslikavaju sematniku povezanosti između objekata. Pored imena, vezu potpuno definiše njena kardinalnost. Kardinalnost predstavlja odnos broja objekata koji se povezuju. Određivanje kardinalnosti se uglavnom vrši proučavanjem veza i odnosa između posmatranih objekata. Kardinalnosti može biti:

- Jedan prema jedan (1:1) - na primer jedna *uplata dobavljaču* se vrši po tačno jednoj *fakturi dobavljača*.
- Jedan prema više (1:*) - na primer jedna *narudžbenica* sadrži više *stavki narudžbenice* (Slika 4).
- Više prema više (*:*) - više *dobavljača* ima ugovore sa više *špeditera.*



Slika 4.13: Primer imenovane veze između entiteta

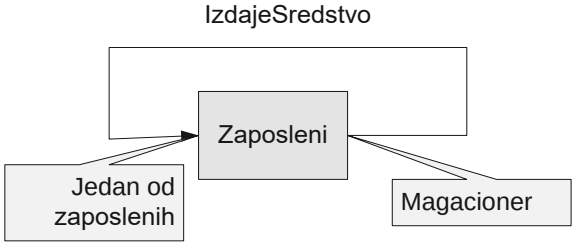
U situacijama kada su veze između objekata implicitno jasne, radi uštede u prostoru na dijagramu, veze se ne moraju imenovati. Veza uglavnom ima samo jednosmerni smisao, pa je uobičajeno da se iscrta i strelica koja označava pravilan smer.



Slika 4.14: Primer neimenovane veze

Na primeru narudžbenice, implicitno je jasno da se ona sastoji od stavki narudžbenice (Slika 4.14). Kardinalnost veze od jakog prema slabom objektu je *uvek* jedan (*jedan* jak objekat egzistencijalno određuje *više* slabih objekata).

Broj entiteta koji učestvuju u vezi se naziva stepen veze. Veza u kojoj učestvuju dva entiteta se naziva binarna, veza trećeg stepena (učestvuju tri entiteta) se naziva ternarna, itd. Veza u kojoj jedan entitet učestvuje više puta u različitim ulogama naziva se rekurzivna ili unarna veza. Na primer, ako je uočen entitet Zaposleni, jedan od zaposlenih je i magacioner. Magacioner izdaje sredstva zaposlenima pa se uočava veza IzdajeSredstvo. Po nekada magacioner mora i sebi da izda sredstvo. Drugim rečima, enetitet Zaposleni učestvuje dva puta u vezi IzdajeSredstvo: prvi put kao magacioner koji izdaje sredstva drugima, a drugi put kao jedan od zaposlenih kome takođe može da se izda sredstvo.

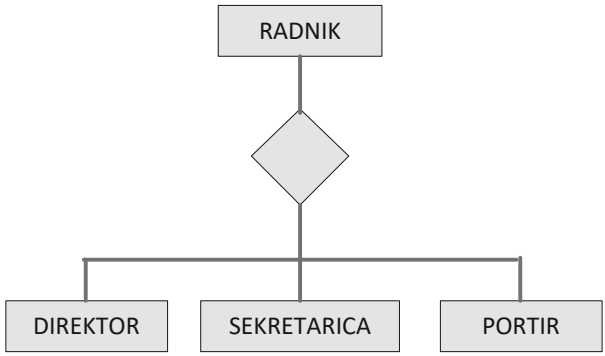


Slika 4.15: Primer rekurzivne veze

4.3.5 Prošireni model objekti veze (PMOV)

Pored osnovnog, postoji i prošireni model objekti veze, koji omogućava detaljnije definisanje veza između objekata. Pored asocijativnih veza predstavljenih na prethodnom primeru, koje treba da oslikaju semantiku udruživanja objekata u sistemu, postoje i specifične veze kojima se izražava hijerarhija i komponovanje objekata. Postoje dve reprezentativne vrste ovakvih veza:

Generalizacija/specijalizacija – Generalizacija je apstrakcija u kojoj se skup sličnih tipova objekata tretira kao generički tip objekta (nadtip). Slični tipovi objekata su oni tipovi koji imaju neke zajedničke atribute, veze i/ili operacije. Specijalizacija je inverzni postupak u kome se za neki tip objekta, definišu njegovi podtipovi, koji imaju neke njima specifične atribute, veze i/ili operacije. Na primer, (Slika 4.16) skup tipova Direktor, Sekretarica, Portir može se predstaviti generičkim tipom Radnik.



Slika 4.16: Generalizacija/specijalizacija

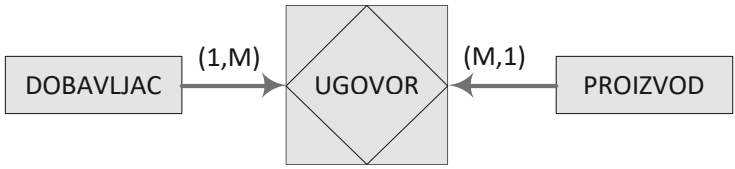
U generalizovanoj hijerarhiji objekata važi pravilo nasledjivanja osobina i pravilo nasledjivanja operacija:

- Podtipovi nasledjuju sve atribute i veze svoga nadtipa.
- Podtipovi nasledjuju sve operacije svoga nadtipa.

Definisanjem podtipova nekog tipa razrešava se problem atributa na koji nisu primenljive osobine svih objekata u skupu objekata jednog tipa, na taj način se definiše podtip kao skup pojavljivanja na koje je dato svojstvo primenljivo.

Agregacija/dekompozicija – Agregacija je apstrakcija u kojoj se veza između dva ili više tipova objekata tretira kao objekat na višem nivou apstrakcije. Zbog toga što istovremeno predstavlja i objekat i vezu agregacija se često naziva i mešoviti tip

objekta-veza. Objekti koji čine agregaciju se nazivaju komponentama agregacije. Postupak inverzan aregaciji se naziva dekompozicija.



Slika 4.17: Agregacija/dekompozicija

Agregirani objekat se označava simbolom upisanog romba u pravougaonik, čime se izražava njegova dvojaka priroda. Agregati su veoma povoljni za preciznije definisanje veza koje imaju kardinalnost *više-više*. Pošto su ovi tipovi veza teški za održavanje referencijalnog integriteta, onda se veza pretvara u objekat, koji ima kardinalnost *jedan-više* prema susednim objektima.

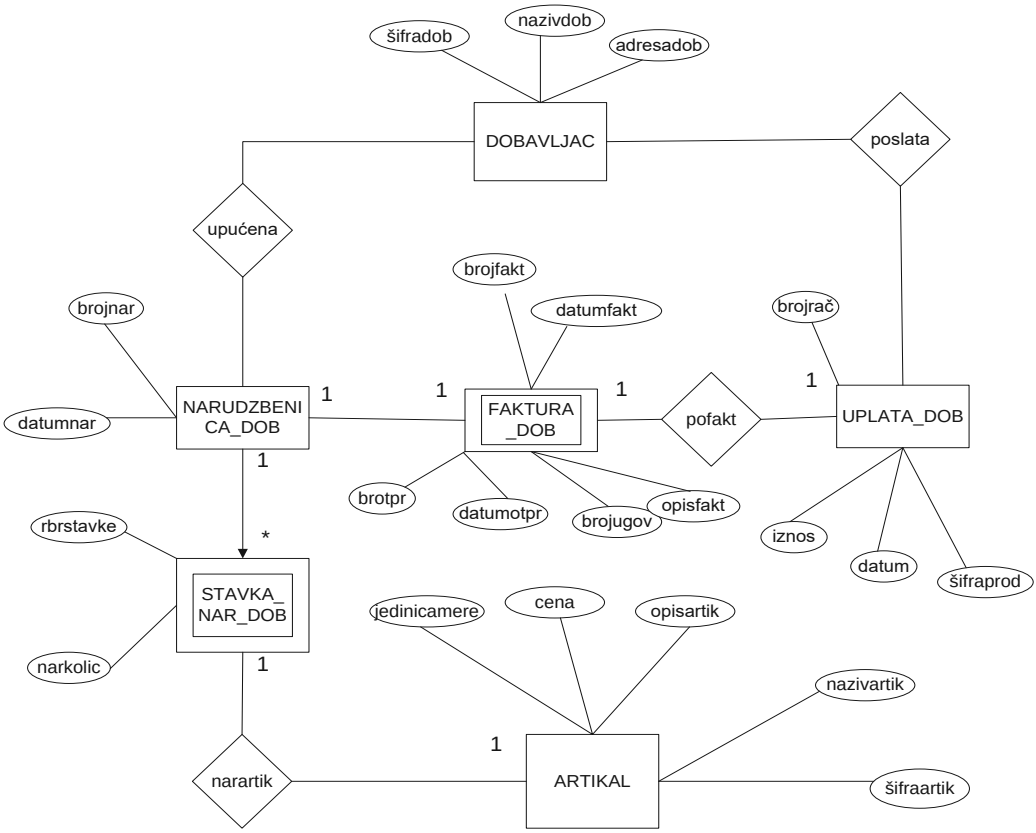
Zbog svoje dvojake prirode, agregati su uglavnom slabi objekti jer egzistencijalno zavise od dva ili više drugih objekata koji ih jednoznačno određuju (izuzetak je agregacija *na sebe*; na primer neki *proizvod* može da se sastoji od više objekata koji su takođe tipa *proizvod*; u toj situaciji kompozit ne mora da bude slab objekat, jer postoje *proizvodi* koji mogu da budu, ali nisu kompoziti). Agregati imaju svoje atribute, mogu da budu u vezama drugog tipa (generalizacije/specijalizacije) sa nekim drugim objektima (moguće agregiranim, takođe).

4.3.6 Primer

Na narednoj slici predstavljen je primer DOV (Slika 4.18). Pri modelovanju DOV, polazi se od DTP i rečnika podataka, kojima se opisuje određeni poslovni proces. Na osnovu interfejsa i tokova podataka (TP) iz DTP, definišu se objekti. TP su dekomponovani u rečniku podataka kao strukture.

Elementi strukture (polja) su atributi objekata. Ako je element ugnježdena struktura (struktura u strukturi) ona se predstavlja kao drugi objekti koji se nalaze u vezi (na primer narudžbenica i stavaka narudžbenice, ispitni spisak i pojedinačni rezultat).

Ove veze su obično implicitne i predstavljena im je samo kardinalnosti i usmerenost. Zatim sledi definisanje preostalih veza između objekata. Veze se imenuju i određuje im se kardinalnost.



Slika 4.18: Primer dijagrama objekti veze za proces nabavke

Opis DOV (Slika 4.18): u procesu nabavke, formira se narudžbenica (objekat *NARUDZBENICA_DOB*), kojom se potražuje roba od određenog dobavljača (objekat *DOBAVLJAC*). Za svaki artikal koji se nabavlja (objekat *ARTIKAL*), formira se jedna stavka narudžbenice (slabi objekat *STAVKA_NAR_DOB*). Pored podataka artikla, stavka sadrži i količinu koja se nabavlja (*nar_kol*). Kreirana narudžbenica se šalje dobavljaču i on na osnovu nje isporučuje robu, uz koju šalje fakturu - račun za naplatu (objekat *FAKTURA_DOB*). Kupac po fakturi vrši uplatu dobavljaču (objekat *UPLATA_DOB*), a potvrdu (kopiju) o uplati šalje dobavljaču, kao dokaz izmirenja svojih obaveza.

Model objekti veze omogućava potpunije shvatanje funkcionisanja sistema semantičkim opisom objekata i njihovih uzajamnih veza. Korišćenjem DOV pojednostavljuje se prevođenje logičkog u fizički model podatka.

Model objekti veze je kompatibilan sa UML[2] dijagramom klasa, što znači da pored modelovanja podataka (*data layer*), omogućava i objektno orjentisano modelovanje aplikacione logike (*business logic layer*).

[2] UML *Unified modeling language* – standard za objektno-orjentisano modelovanje IS kroz različite tipove dijagrama

5 STRUKTURNA SISTEMSKA ANALIZA (SSA)

Strukturna sistemska analiza (SSA) predstavlja savremen pristup procesu razvoja poslovnih informacionih sistema. Analiza tokova podataka u sistemu, određivanje ključnih entiteta u sistemu i njihovih atributa i entiteta van sistema s kojima on komunicira predstavlja samo najvažnije u nizu zadataka SSA. Osnovne karakteristike SSA su:

- Razvijanje sistema se vrši od *vrha-na dole*;
- Analiza i dizajn podrazumevaju korišćenje različitih alata, tehnika i modela u cilju što preciznijeg snimanja aktuelnog sistema i korisničkih zahteva;
- Osnovni alati SSA su: funkcionalni dijagrami, dijagrami tokova podataka, rečnici podataka, specifikacija procesa, dijagrami *objekti-veze*;
- Razdvajanje fizičkog i logičkog modela - . fizički model je najčešće fokusiran na *preživljavanje* postojećeg sistema ili dizajn novog, dok je logički model više usmeren na analizu zahteva kojima sistem treba da odgovori;
- Uključivanje korisničkih uloga u različitim fazama razvoja;
- SSA omogućava istovremeno izvršavanje pojedinih faza analize i dizajna;
- SSA je podržana naprednim tehnologijama, što olakšava razvoj sistema;

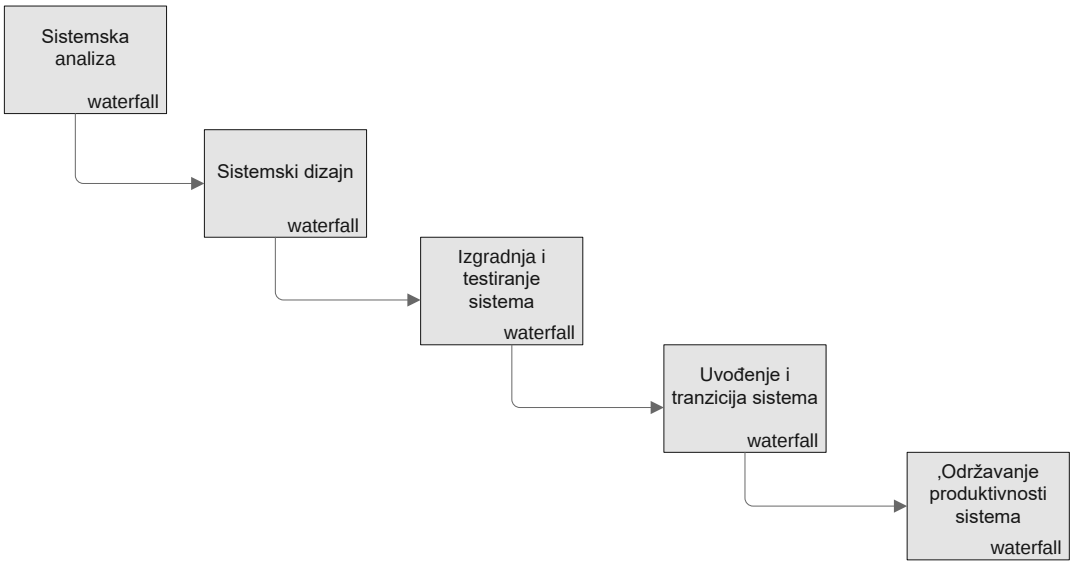
SSA predstavlja ključni proces u projektu razvoja poslovnih informacionih sistema. Sistemska analiza je preduslov dobrog dizajna informacionog sistema i uključuje tehnike analize informacionog sistema, modelovanja podataka i tehnike normalizacije dobijenog modela.

U metodologiji 70-ih godina preovlađujući pristup je bio *waterfall* pristup koji se sastojao od 5 sekvencijalnih faza (odvijale su se jedna za drugom po tačno utvrđenom redosledu):

- Sistemska analiza;
- Sistemski dizajn;
- Izgradnja i testiranje sistema;
- Uvođenje i tranzicija sistema;
- Održavanje produktivnosti sistema.

Model *vodopada* (Slika 5.1) zamenio je *spiralni* model, koji se zasniva na iterativnosti procesa razvoja informacionih sistema (Slika 5.2). Prednost ovakve metodologije je u tome što se sistem brzo uvodi u korišćenje (postizanjem samo

osnovnih funkcionalnosti), a zatim se dograđuje i prilagođava potrebama konkretnih korisnika. Na taj način proces razvoja nije završen kada se informacioni sistem uvede u upotrebu, već se nastavlja dodavanjem novih softverskih modula, osavremenjavanjem postojećih funkcionalnosti. To znači da razvoj sistema traje *dok se sistem koristi* (dok je *živ*).



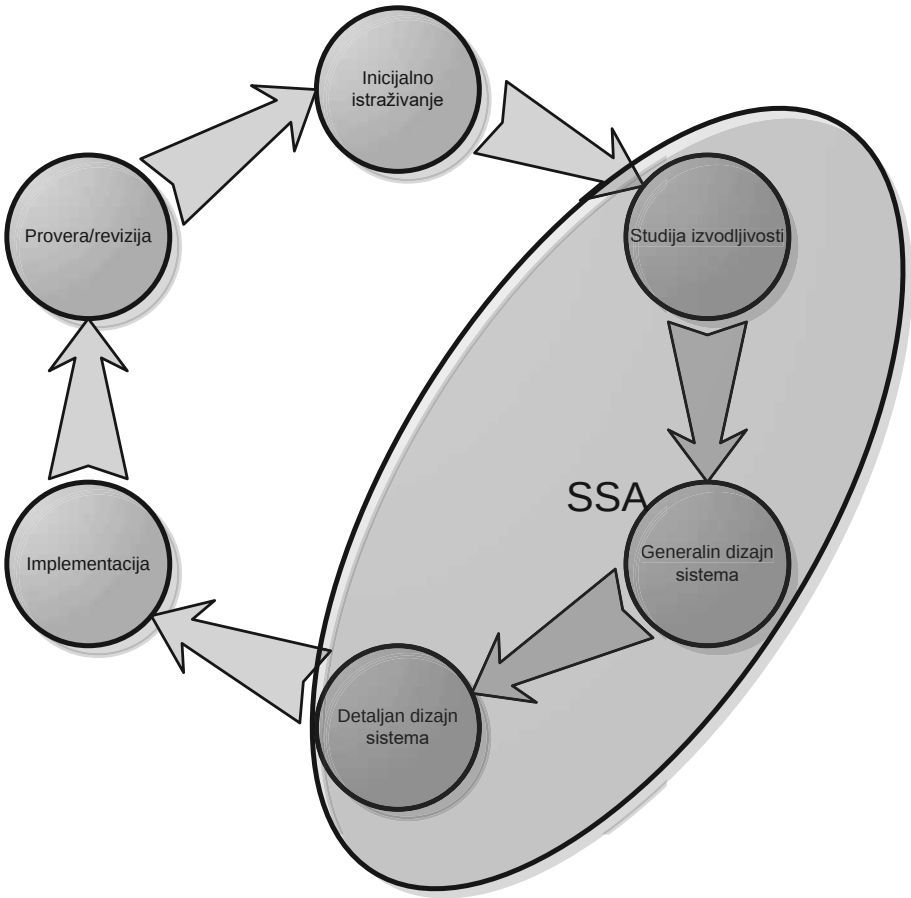
Slika 5.1: Waterfall metodologija

Razvoj poslovnih sistema predstavlja cikličan (*iterativno inkrementalan*) proces, koji se odvija po fazama. Zbog svoje stadijumske prirode, celokupan proces se često naziva *životni ciklus* razvoja sistema (Slika 5.2).

U obe predstavljene metodologije SSA ima značajno mesto i predstavlja preduslov procesu dizajna sistema. Metodologija životnog ciklusa mnogo fleksibilnije uključuje SSA u proces razvoja poslovno informacionog sistema.

SSA se realizuje po fazama, i dobro definisanoj metodologiji. Suštinski, sistem se analizira na različite načine. To može biti sa akcentom na:

- poslovne funkcije (funkcionalna dekompozicija),
- tokove podataka (dekompozicija dijagrama tokova podataka),
- definisanje rečnika podataka,
- definisanje veza između entiteta u sistemu (izrada dijagrama objekti veze).



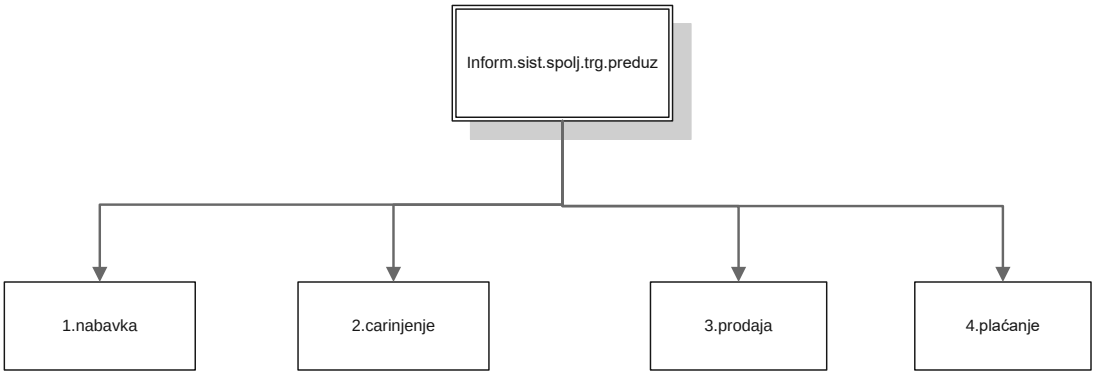
Slika 5.2: Faze životnog ciklusa razvoja sistema

5.1 FUNKCIONALNA DEKOMPOZICIJA

Dijagrami funkcionalne dekompozicije se koriste u određivanju osnovnih sistemskih funkcija i njihovom dekomponovanju. Ove funkcije obično odgovaraju u potpunosti predstavljenim procesima u dijagramima tokova podataka.

5.1.1 Jacksonovi dijagrami

Dijagrami funkcionalne dekompozicije se još nazivaju, prema svom tvorcu, *Jackson*-ovim dijagramima. U nastavku se na jednom primeru razmatraju detalji ovih dijagrama (Slika 5.3).

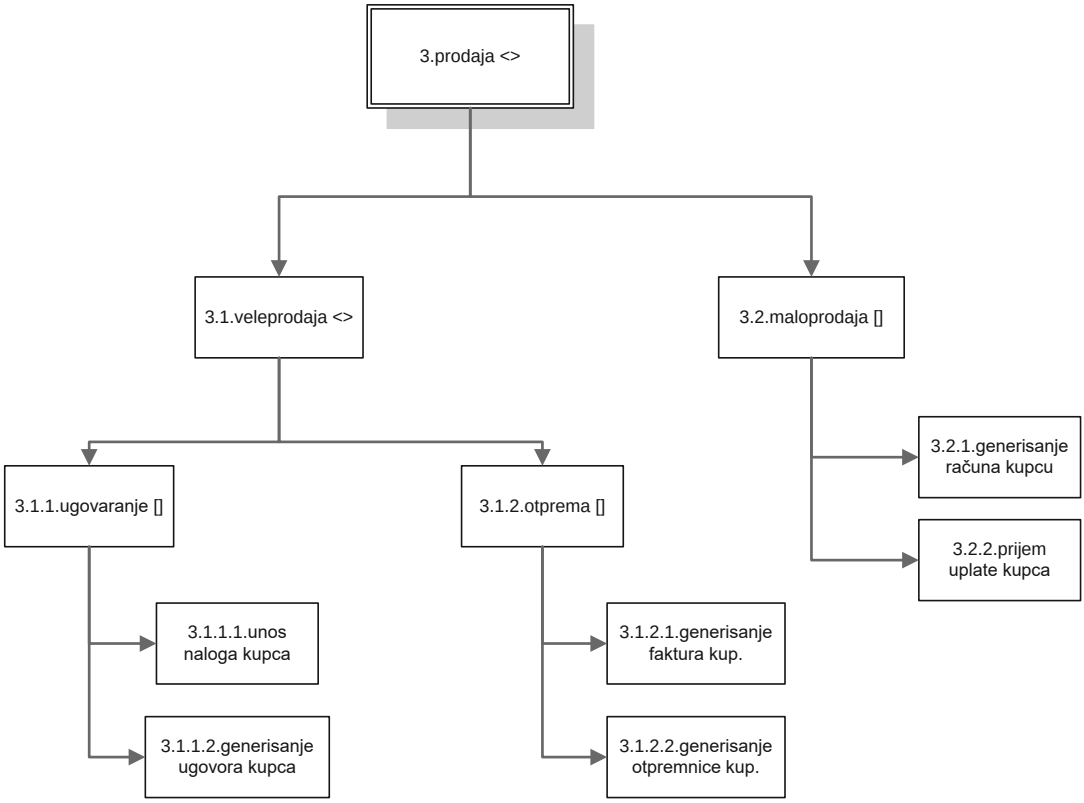


Slika 5.3: Jackson-ov dijagram osnovnih poslovnih funkcija

Funkcionalni dijagram najvišeg nivoa predstavlja osnovne poslovne funkcije organizacije. Na prethodnoj slici predstavljen je dijagram osnovnih poslovnih funkcija jedne spoljno-trgovinske organizacije. U vrhu dijagrama obavezno se predstavlja naziv IS koji se analizira. Slično hijerarhijskim organizacionim dijagramima predstavljaju se osnovne funkcije preduzeća. Ove funkcije su selektivne, što znači da se odvijaju bez uzajamnih zavisnosti.

5.1.2 Pravila u kreiranju Jacksonovih dijagrama

Osnovne funkcije najčešće imaju selektivnu prirodu tako da se dekomponuju. Završetak funkcionalne dekompozicije je kada se dobiju elementarne sekvencijalne funkcije (primitive). Primitivne funkcije su one koje se dalje mogu samo dekomponovati na konkretne naredbe u ciljnom programskom jeziku, ili u *pseudo kodu*. Na slici Slika 5.4 predstavljen je primer dekomponovanja poslovne funkcije *prodaja*. Pošto se razlikuje proces *veleprodaje* (rad sa pravnim licima, ugovaranje, veleprodajne cene, rabat, naručivanje, dostavljanje, transakcije isključivo preko žiro-računa u bankama) od *maloprodaje* (rad sa fizičkim licima, gotovinsko plaćanje, manje količine robe, plaćanje na prodajnom mestu), tako je osnovna *prodaja* funkcija dekomponovana na navedene dve.

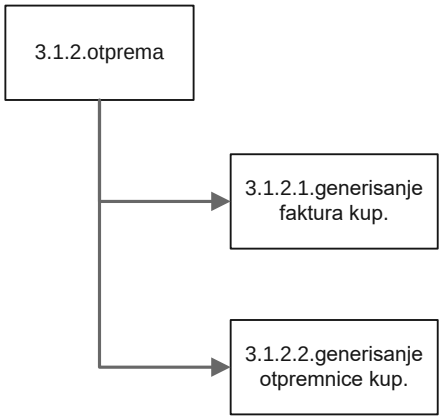


Slika 5.4: Dekompozicija poslovne funkcije Prodaja

Praćenje dekompozicije je olakšano numeracijom funkcija. Tako se npr. funkcija *maloprodaja* (3.2) dekomponuje na dve manje funkcije: *generisanje racuna kupcu*(3.2.1) i *prijem uplate kupca* (3.2.2). Može se uočiti da su funkcije *generisanje racuna* i *prijem uplate* – sekvencijalne; kupac ne plaća robu dok ne dobije račun od prodavca. Tu je kraj dekomponovanja funkcije *maloprodaja*, sledi opis logike primitivne funkcije pseudo kodom.

Selektivni procesi na dijagramima funkcionalne dekompozicije označavaju se znacima <>, dok se sekvencijalni procesi označavaju znacima [].

Funkcionalni dijagrami se ne bave podacima koji postoje u sistemu, već samo treba da istaknu frekventnost (važnost) i kompleksnost pojedinačnih poslovnih funkcija. Kroz funkcionalnu dekompoziciju mogu se identifikovati sličnosti u dekompoziciji različitih poslovnih funkcija. Kao posledica ove činjenice mogu se identifikovati nove funkcije, i već u fazi analize učiniti koraci koji će omogućiti opitimizaciju rešenja IS.



Slika 5.5: Primer identifikovanja istih funkcija u funkcionalnoj dekompoziciji

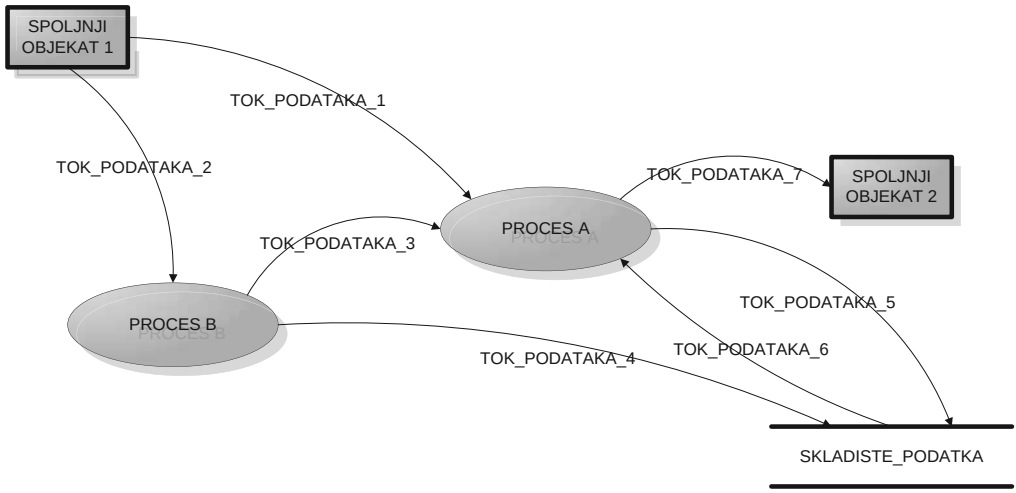
Na primer (Slika 5.5), organizacije (trgovine) koje se bave prodajom robusne robe (nameštaj, vozila, industrijske mašine) koriste istu funkciju *otprema* u dekompoziciji *veleprodaje* i *maloprodaje*. Roba se i u jednom i u drugom slučaju mora dostaviti kupcu. Na taj način, funkcija *otprema* se dizajnira i implementira umesto 2 puta – samo jedanput i ona treba da bude dostupna iz obe opcije (nad-funkcije) aplikacije (*veleprodaje* i *maloprodaje*).

Funkcionalna analiza uz pomoć alata za modelovanje obezbeđuje važne detalje koji se mogu koristiti u kasnijim fazama analize. Međutim, funkcionalni pristup nije sveobuhvatan – bavi se pitanjem šta sistem radi, ne i kako radi.

5.2 DIJAGRAMI TOKOVA PODATAKA (DTP)

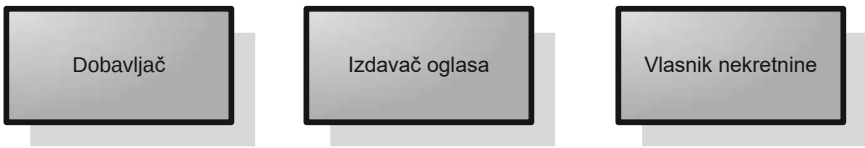
Dijagrami tokova podataka (DTP) predstavljaju jednu od najpopularnijih tehnika modelovanja poslovnih procesa [2]. DTP opisuju protok informacija u sistemu. Oni predstavljaju prirodan nastavak razvoja IS nakon funkcionalne analize. DTP se sastoje od četiri vrste elemenata (Slika 5.6):

- Interfejsi
- Procesi
- Tokovi podataka
- Skladišta podataka



Slika 5.6: Struktura DTP

Interfejsi predstavljaju entitete (objekte) iz realnog sveta koji okružuje sistem. To mogu biti osobe koje se nalaze u određenoj ulozi u odnosu na sistem (npr. pacijent, nastavnik, student, kupac, dobavljač...), organizacije (banka, hotel, škola, carina...) ili sistem koji nudi neki servis za posmatrani IS (SMS servis, e-banking servis, čitač smart kartica, tržište nekretnina...). Interfejsi se prikazuju pravougaonikom i nazivom (Slika 5.7).



Slika 5.7: Primeri interfejsa

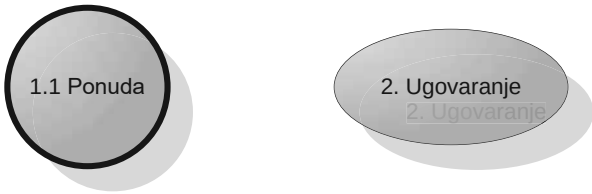
Zajedničko za sve interfejse je da su to objekti *izvan* analiziranog sistema, koji interaguju (razmenjuju podatke) sa sistemom. Kao što se može videti iz prethodnog primera, interfejsi nisu konkretna fizička lica, niti konkretne organizacije; *dobavljač* može biti bilo koje fizičko, ili pravno lice. Isti je slučaj sa *vlasnikom nekretnine*. *Izdavač oglasa* može biti bilo koja izdavačka, ili novinarska kuća. Bitna je uloga koju interfejs obavlja, odnosno način na koji sistem komunicira s interfejsom. U toku dekomponovanja DTP, interfejsi moraju ostati konzistentni – oni se ne menjaju, niti dekomponuju. Drugim rečima, broj i nazivi interfejsa na početku dekompozicije mora da odgovara broju i nazivima interfejsa na kraju tog procesa.

U toku procesa dekomponovanja, radi preglednosti dijagrama, jedan interfejs se može ponoviti na istom dijagramu, s tim da je kopija označena zvezdicom (Slika 5.8).



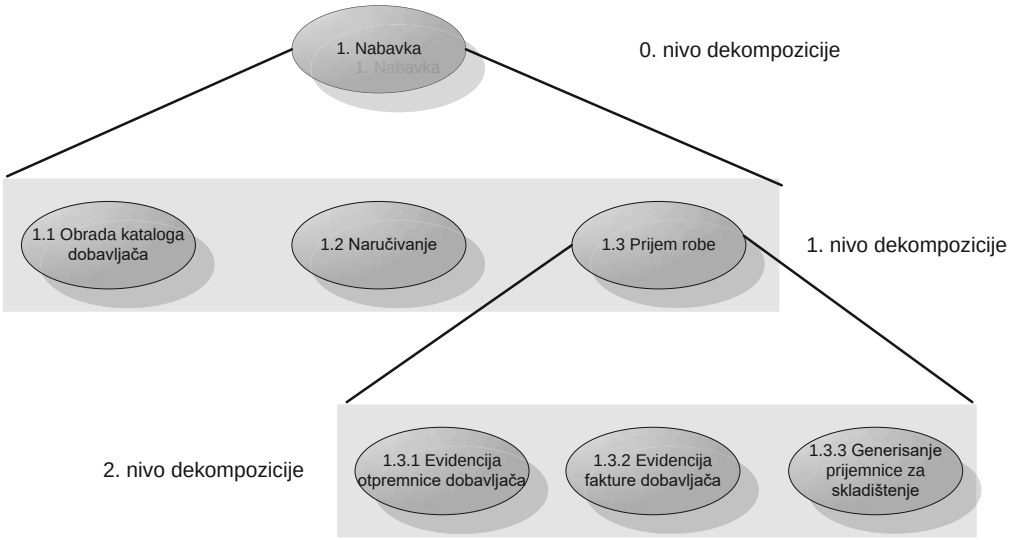
Slika 5.8: Original i kopija interfejsa

Procesi odgovaraju funkcijama iz prethodne faze SSA (funkcionalna dekompozicija). Interfejsi interaguju sa sistemom posredstvom procesa. Svaki proces predstavlja neku specifičnu poslovnu aktivnost. Proces može predstavlja neku automatsku obradu podataka (generisanje izveštaja, računa, autentifikacija korisnika...), ali isto tako neku manuelnu aktivnost (nabavka, isporuka, proizvodnja, učlanjivanje, izdavanje knjige...).



Slika 5.9: Primer označavanja procesa

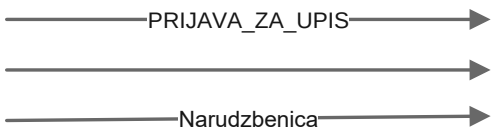
Procesi se označavaju krugom ili elipsom u koji se unosi naziv procesa (Slika 5.9). Za razliku od interfejsa, procesi se numerišu. Brojčano označavanje je identično kao kod funkcionalnih dijagrama. Nije dozvoljeno praviti kopije procesa na istom DTP. Procesi se mogu dekomponovati. Suštinski, dekompozicija DTP se svodi na dekomponovanje poslovnih procesa. Na sledećem primeru (Slika 5.10) istaknut je primer dekomponovanja procesa *nabavka*. Na nultom nivou dekomponovanja predstavljeni su osnovni poslovni procesi. Ovi procesi se dekomponuju od 1. nivoa dekompozicije. Proces *nabavka* se dekomponuje na 3 podprocesa: *obrada kataloga dobavljača, naručivanje* i *prijem robe*. To znači da se osnovni proces *nabavka* više ne pojavljuje od 1. nivoa dekompozicije, već samo njegovi podprocesi. Na drugom nivou demonstrirana je dekompozicija podprocesa *prijem robe*. Ovaj podproces se dekomponuje na tri nova, koja opisuju šta sistem radi po prijemu robe. To znači da se *prijem robe* od 2. nivoa više ne pojavljuje kao celovit proces, već njegovi podprocesi.



Slika 5.10: Primer dekompozicije poslovnih procesa

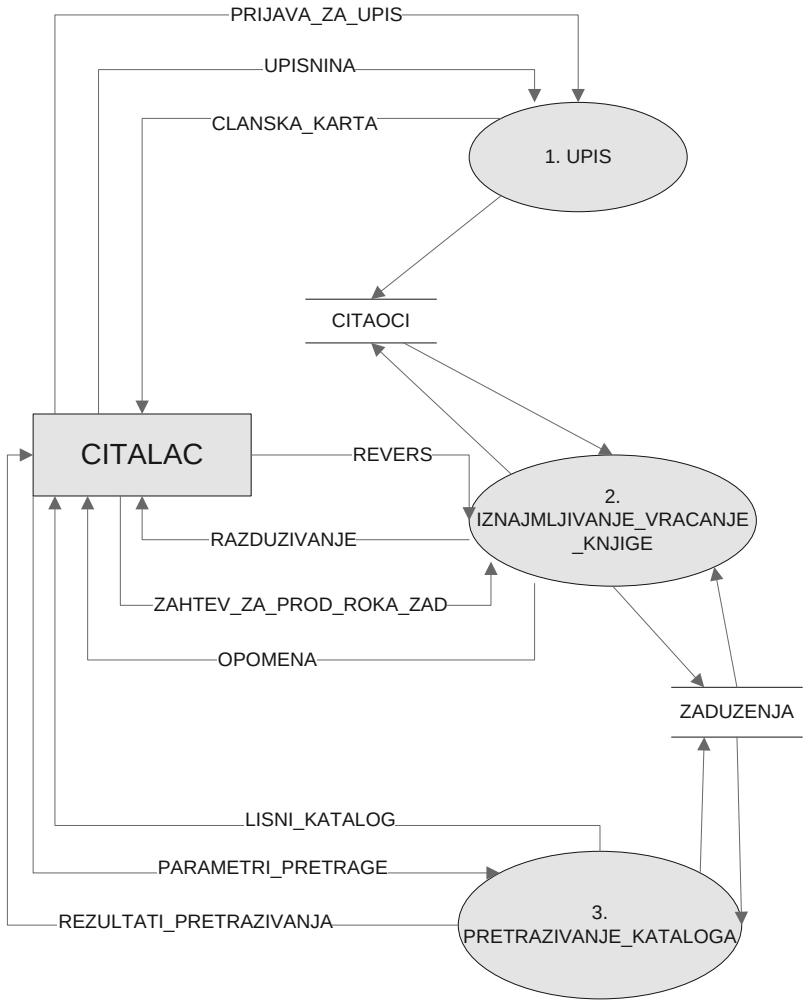
Nije precizirano do kog nivoa se vrši dekomponovanje poslovnih procesa. Ne sme se zaboraviti da DTP treba da u potpunosti odgovaraju dijagramima funkcionalne dekompozicije. Analogno funkcionalnoj dekompoziciji, poslovni procesi se dekomponuju do nivoa pre dobijanja elementarnih sekvencijalnih procesa. Gornji primer predstavlja završenu dekompoziciju podprocesa *prijem robe* procesa *nabavke*. Ako bi se nastavilo sa dekomponovanjem bilo kog od procesa označenih 1.3.1 do 1.3.3., dobili bi se potpuno sekvencijalni podprocesi koji se izvršavaju po unapred utvrđenom redosledu, što je više stvar implementacionih detalja, nego aktivnosti analize.

Tokovi podataka (TP) predstavljaju interakciju između interfejsa i procesa u sistemu. Oni obavezno moraju biti imenovani (Slika 5.11). TP imaju statičku prirodu, jer ne odslikavaju redosled izmene podataka između sistema i okoline, niti redosled akcija koje izazivaju unutar sistema. TP mogu sadržati osnovne tipove – celi brojevi, realni brojevi, karakteri i izvedene tipove - struktuirane podatke, kao što su *adresa, narudzbenica, katalog proizvoda* (sadrže podatke različitih osnovnih tipova ili ugnježdene strukture). Važna karakteristika TP je njihova obavezna usmerenost, koja opisuje smer toka podataka u sistemu.



Slika 5.11: Primer označavanja TP

I unutar sistema postoje TP (Slika 5.12): između procesa i skladišta podataka i između procesa uzajamno. Ti tzv. interni tokovi podataka nisu imenovani, jer se podrazumeva da je njihova struktura definisana kroz spoljnje tokove (sistem - interfejs).



Slika 5.12: Primer tokova podataka na fragmentu DTP za IS biblioteke

Namena internih TP između procesa i skladišta je da istaknu da li se i gde ulazni podaci pamte u sistemu, odnosno iz kojih izvora (skladišta) se generišu izlazni podaci prema interfejsima.

TP između procesa u sistemu odslikavaju njihovu uzajamnu povezanost. Nije preporučljivo da se ovakvi TP pojavljuju u DTP 0. nivoa. Oni se pojavljuju kao proizvod dekomponovanja osnovnih procesa, nisu imenovani i samo treba da predstave podprocese koji se koriste od više drugih podprocesa na istom nivou dekompozicije (odgovaraju procesima u stereotipskoj *uses* vezi u dijagramima slučajeva korišćenja UML).

U dekomponovanju DTP, tokovi podataka moraju biti konzistentni počevši od kontekstualnih dijagrama do najkonkretnijih DTP. Konvencija je da se ne smeju pojaviti novi TP u procesu dekompozicije. Ako je to nužno, potrebno je ažurirati dijagrame na višim nivoima opštosti.

Skladišta podataka predstavljaju elemente sistema u kojima se podaci čuvaju (Slika 5.13). Skladište podataka ne predstavlja bazu podataka, niti tabelu u bazi. U ovoj fazi analize sistema skladišta samo treba da istaknu grupisanje podataka.

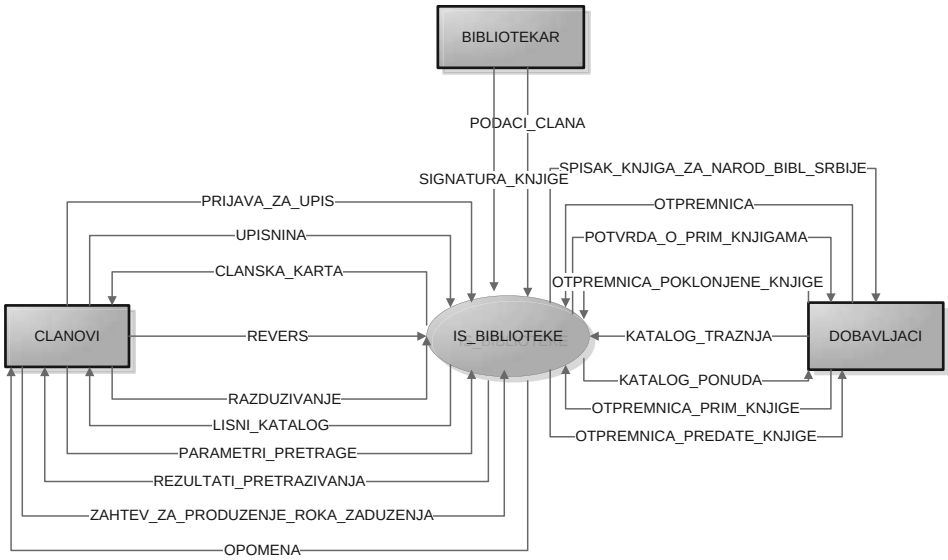


Slika 5.13: Primer skladišta podataka

Za razliku od interfejsa, simbol skladišta podataka je pravougaonik koji nema bočne stranice. Imena skladišta su obično množine imenica – entiteta koji grupišu srodne podatke. Na primer, skladište *STUDENTI* može da obuhvata ne samo osnovne podatke studenata (ime i prezime, jmbg, broj indeksa...), već i detaljnije podatke (godina studija, rezultati ispita, smer-nastavna grupa...).Često se nazivima skladišta dodaje prefiks *SK* koji još više ističe razliku u odnosu na interfejse. Isticanje razlike između skladišta i interfejsa nije neopravdano: na prethodnom primeru (Slika 5.12) postoji interfejs *CITALAC* i skladište *CITAOCI*. U praksi je čest slučaj da interfejsi i skladišta imaju slične nazive, tako da je poželjno svako nastojanje isticanja razlike na dijagramima (naravno, u okvirima konvencije označavanja).

5.2.1 Kontekstualni dijagrami

Modelovanje poslovnih procesa započinje kontekstualnim dijagramom. Kontekstualni dijagram je takav dijagram tokova podataka na kome je celokupan sistem prikazan kao jedan proces - *crna kutija* (Slika 5.14).



Slika 5.14: Primer dijagrama konteksta

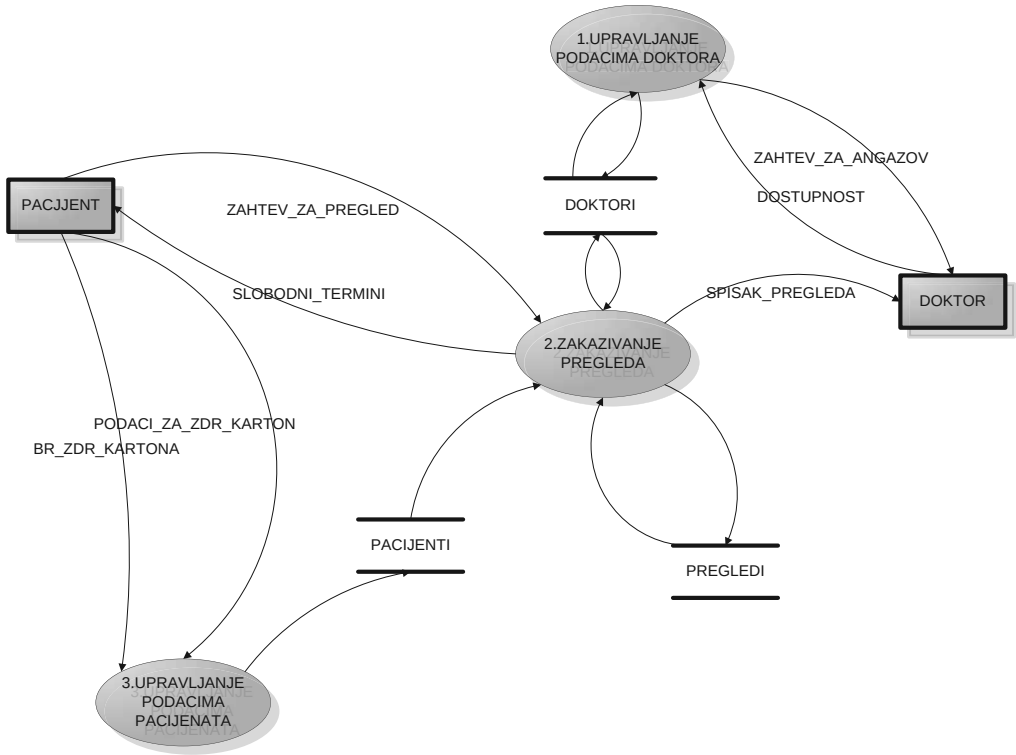
Težišni zadatak kontekstualnog dijagrama je definisanje svih interfejsa sa kojima sistem komunicira, kao i tokova podataka koji čine tu komunikaciju. U početku je najčešće nemoguće definisati sve interfejse i tokove podataka. Inicijalni sadržaj kontekstualnog dijagrama treba da obuhvati osnovne tokove i spoljnje entitete. U kasnijim fazama dekompozicije DTP je moguće naknadno dodati nove interfejse i tokove podataka, s napomenom da mora biti zadržana konzistentnost dekomponovanih DTP u odnosu na kontekstualni dijagram – svi interfejsi i tokovi podataka koji se koriste u dekomponovanju, moraju biti na dijagramu konteksta.

Na prethodnom primeru (Slika 5.14) prikazan je kompletan kontekstualan dijagram za informacioni sistem biblioteke. Može se uočiti da postoji mali broj interfejsa i veliki broj tokova podataka. Najčešće greške u ovoj fazi dekomponovanja je dodavanje interfejsa koji predstavljaju deo sistema, a ne spoljni objekat s kojim sistem komunicira. Ovaj kontekstualni dijagram sadrži interfejs *bibliotekar* koji se naizgled može razmatrati kao deo sistema. Obično zabuna dolazi zbog toga što su entiteti kao bibliotekar zaista deo stvarnog sistema. Međutim, bibliotekara nikako ne možemo smatrati delom informacionog sistema biblioteke, već ga razmatramo kao spoljni objekat koji, na primer zahteva da se prijavi prilikom počinjanja korišćenja IS, odnosno odjavi prilikom završetka korišćenja sistema. *Bibliotekar* nije subjekat koji vrši registrovanje novih članova i koji izdaje knjige na korišćenje članovima – te aktivnosti su odgovornost sistema.

5.2.2 Dijagram toka podataka 0. nivoa

Nakon kreiranja kontekstualnog dijagrama sledi njegova dekompozicija kroz dijagrame tokova podataka (Slika 5.15). Težište DTP 0. nivoa je identifikacija *osnovnih* poslovnih procesa koji se dešavaju u sistemu i distribuiranje TP između interfejsa i procesa. Pri kreiranju DTP 0. nivoa treba imati u vidu 3 važne činjenice:

- Moraju biti prikazani svi TP koji će se pojavljivati na detaljnijim DTP koji su proizvodi dekomponovanja (DTP 1, 2 nivoa),
- Moraju biti prikazani svi interfejsi koji će se pojavljivati na detaljnijim DTP i
- Ne moraju biti prikazana sva skladišta i poslovni procesi, jer se mogu dekomponovati



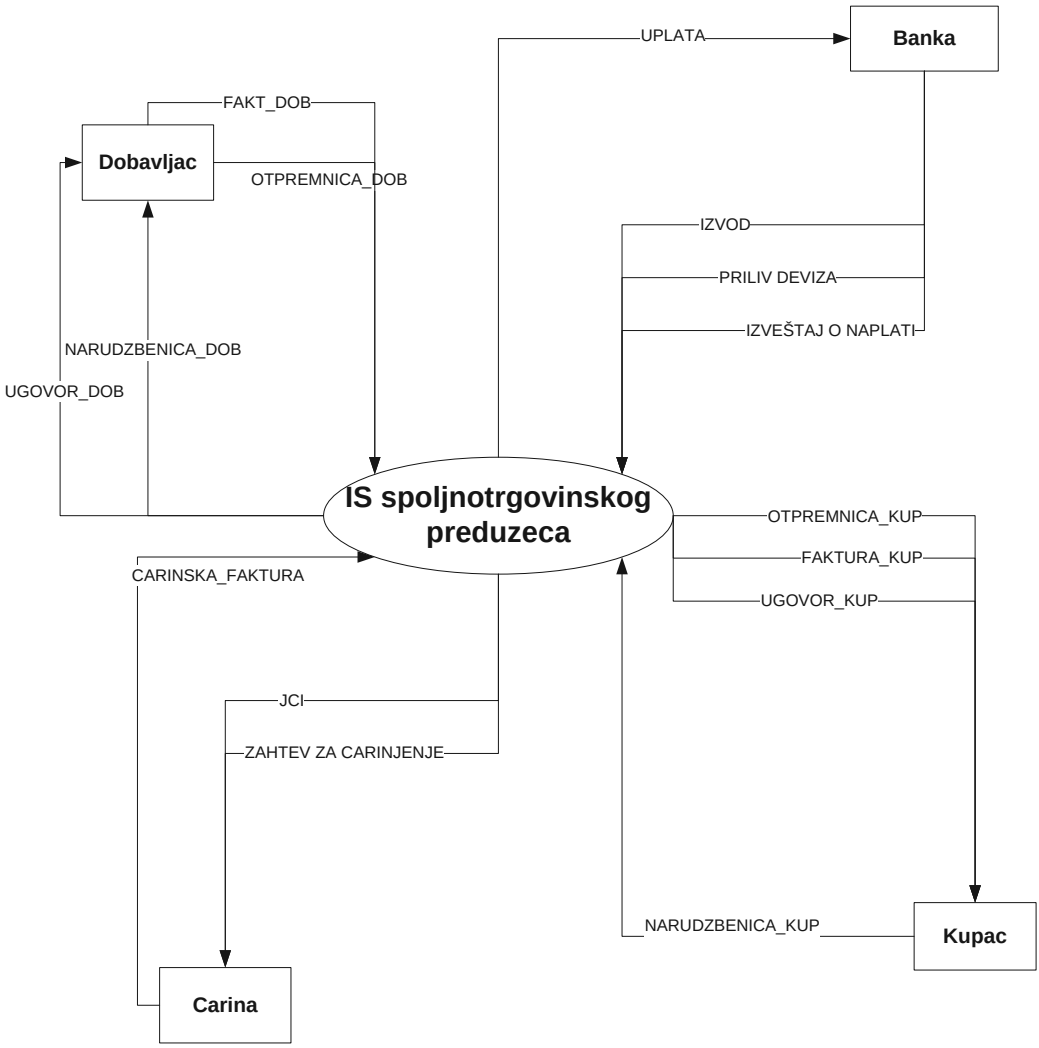
Slika 5.15: Primer DTP 0. nivoa za IS medicinske klinike

Problem su dakle tokovi podataka, jer u slučaju velikog broja DTP 0. nivoa postaje vrlo nepregledan. To je indikator da je sistem koji se modeluje metodama SSA - preglomazan. U tom slučaju rešenje je da se informacioni sistem u samom početku deli na više komponenti – nezavisnih softverskih modula. Tako će, na primer IS velikog

međunarodnog hotelskog sistema biti podeljen na IS za rezervacije, IS održavanja i nabavke, IS vođenja kadrovskog sektora, IS za održavanje bezbednosti. Tek onda je moguća SSA takvih sistema (kroz modelovanje pojedinačnih komponenti).

5.2.3 Kompletan primer dekompozicije kroz DTP

U ovom poglavlju biće opisana dekompozicija DTP jednog spoljnotrgovinskog preduzeća. Započinje se kontekstualnim dijagramom. Kontekstualni dijagram predstavlja sistem kao *crnu kutiju* koja interaguje sa okruženjem (interfejsima) posredstvom tokova podataka (Slika 5.16).



Slika 5.16: Kontekstualni dijagram za IS spoljnotrgovinskog preduzeća

Težište modelovanja na kontekstualnom dijagramu je definisanje interfejsa i tokova podataka. Postoje četiri osnovna entiteta s kojima preduzeće koje se bavi spoljnom trgovinom interaguje: *dobavljač, kupac, carina* i *banka.* Dobavljači i kupci mogu biti iz zemlje, ili inostranstva. Nabavci robe prethodi ugovaranje sa dobavljačem, kako bi se pravno zaštitile obe strane i kako bi nabavka iz inostranstva u procesu carinjenja bila razmatrana kao legalna. Sama nabavka se odvija kroz dobro poznate TP: dobavljač dobija narudžbenicu (dokument za naručivanje robe) na osnovu koje isporučuje robu uz fakturu (novčana vrednost narudžbe koji modelovano preduzeće mora da plati) i otpremnicu (dokument koji prati porudžbinu i na osnovu koga se vrši provera kompletnosti i ispravnosti pristigle robe; ovaj dokument se dalje može koristiti za regulisanje skladištenja robe).

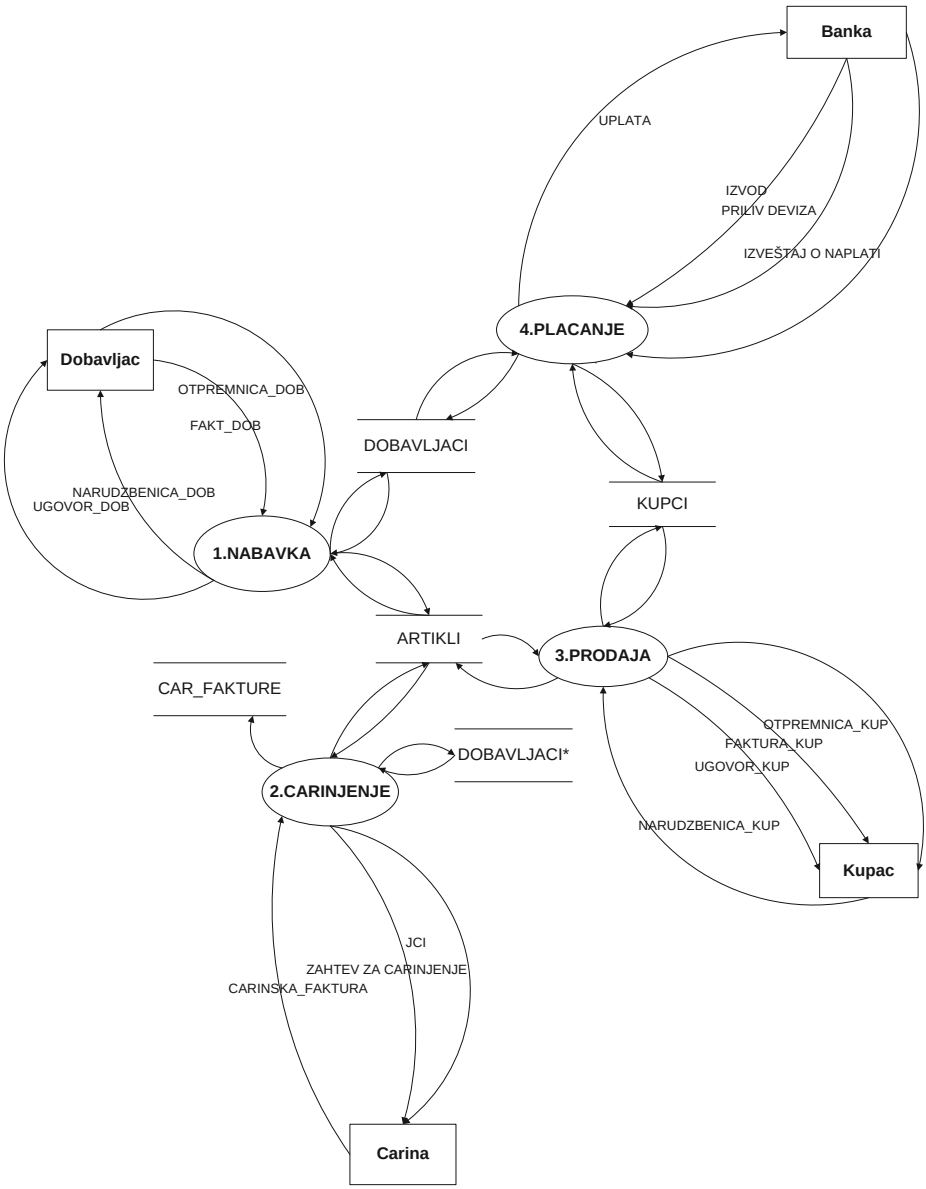
U slučaju da roba pristigne iz inostranstva ona se najpre carini. Preduzeće koje uvozi robu podnosi carini *zahtev za carinjanje*. Takođe podnosi se dokument *JCI* (jedinstvena carinska isprava) koji predstavlja deklaraciju sa podacima o poreklu, nameni, sastavu količini i vrednosti narudžbine koja se carini. Interfejs *carina* ispostavlja preduzeću fakturu za uplatu iznosa carine na uvezenu robu.

Nakon nabavke i carinjenja, roba se može prodavati. Procesi nabavke i prodaje su veoma slični – samo je uloga preduzeća promenjena: prema dobavljačima, preduzeće je kupac, a prema kupcima ono prodaje robu. Pošto se preduzeće bavi veleprodajom, kupovina se ugovara kao i prilikom nabavke. Nakon ugovaranja kupovina započinje naručivanjem, zatim sledi isporuka robe uz fakturu i otpremnicu. Česta greška koja se javlja u ovom slučaju da su tokovi podataka prema dobavljačima i kupcima identično imenovani. Mora se naznačiti razlika, npr *faktura_dob* i *faktura_kup*. Ovo je neophodno jer, iako su strukture ovih dokumenata gotovo identične, modelovano preduzeće se nalazi u različitim ulogama u odnosu na svoje klijente (dobavljače i kupce).

U procesu modelovanja ne samo da treba uzeti stvarno stanje i funkcionisanje sistema. Poželjno je već u fazi SSA, omogućiti proširenje i poboljšanje kvaliteta delatnosti kojom se preduzeće bavi. TP nabavke i prodaje mogu biti prošireni u smislu povećanja kvaliteta usluga. Na primer TP – *katalog_dobavljaca* omogućava bolji kvalitet procesa nabavke. Ako postoje katalozi dobavljača koji sadrže ažurne podatke, IS može na osnovu zadatih kriterijuma dati predlog za najoptimalniju nabavku (npr. ukrštanje kriterijuma cena, isporuka, garancijski period, postgarancijsko održavanje...). Isti tako katalog prema kupcima će sigurno omogućiti kupcima da odaberu robu koja im najviše odgovara (*najbolja reklama je preporuka kupca drugima*).

U poslovanju fizičkih i pravnih lica, sva plaćanja odvijaju se preko poslovnih banaka. Tako da je ovaj interfejs gotovo nezaobilazan u modelovanju poslovnih sistema. Česta greška u modelovanju je da plaćanja i potraživanja postoje, ali se isključuje uloga banke, već se umesto nje pojavljuju konkretni potražioci (u našem slučaju to su dobavljači, carina, poreska uprava...). Banka predstavlja interfejs koji

posreduje između strana koje učestvuju u novčanim transakcijama. Preduzeće u banci izmiruje sva novčana davanja preko TP *uplata*. Banka preduzeću dostavlja periodično, ili po promeni različite vrste izveštaja, koji mu omogućavaju upravljanje vlastitim novčanim sredstvima (*izvod, priliv deviza, izveštaj o naplati*).

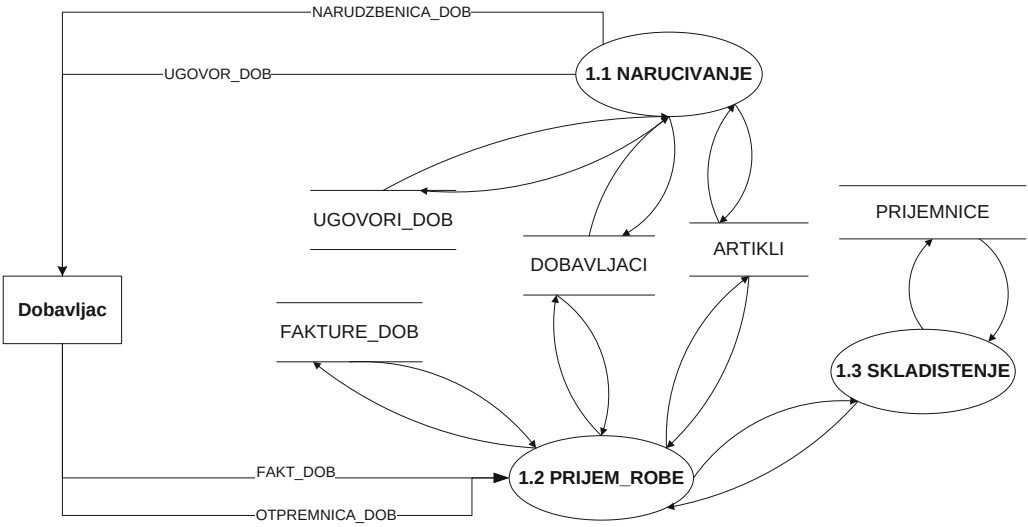


Slika 5.17: DTP 0. nivoa za IS spoljnotrgovinskog preduzeća

Težište kod modelovanja DTP 0. nivoa (Slika 5.17) je na definisanju osnovnih poslovnih procesa i pridruživanju tokova podataka tim procesima. Na ovom nivou dekomponovanja se pojavljuju i skladišta podataka, koja samo treba da istaknu grupisanje podataka. IS spoljnotrgovinskog preduzeća sadrži četiri osnovna poslovna procesa: *nabavka, carinjenje, prodaja* i *plaćanje*.

Proces *nabavke* obuhvata interakcije sistema sa dobavljačima, proces *carinjenja* – s carinom, proces *prodaje* obuhvata TP od i ka kupcima i proces *plaćanje* obuhvata interakcije prema banci. Mogu se uočiti skladišta čiji nazivi impliciraju koje vrste podataka sadrže. Proces carinjenja interaguje sa skladištem *dobavljaci,* koje je 2 put prikazano na istom dijagramu. Kopija je označena zvezdicom, i na taj način su izbegnuta presecanja TP na dijagramu. Skladišta su sa procesima povezana internim, neimenovanim TP. Podrazumeva se da su ti tokovi u potpunosti opisani spoljnim TP (između procesa i interfejsa).

Zatim se vrši dekompozicija osnovnih poslovnih procesa. Proces nabavke se dekomponuje na 3 podprocesa (Slika 5.18): *narucivanje, prijem_robe* i *skladistenje*.



Slika 5.18: DTP 1. nivoa za proces nabavke

Takođe, pojavila su se 3 nova skladišta, koja omogućavaju odvajanje osnovnih podataka dobavljača od tekućih podatka procesa nabavke. Svi tokovi podataka su zadržani i konzistentni sa TP na DTP 0. nivoa. Novina na ovom dijagramu je direktna veza dva procesa (*prijem robe i skladistenje*). Proces *skladistenje* nema ni jednu direktnu vezu sa nekim spoljnim interfejsom, ali zato omogućava detaljnije snimanje procesa koji se odvijaju unutar sistema pri nabavci robe. Skladištenje na DTP ne znači

fizičko smeštanje robe u skladišni prostor, već njeno evidentiranje, označavanje i određivanje mesta gde će se roba čuvati. Na osnovu stanja zaliha IS može da odredi kada i koliko je robe potrebno za neometano poslovanje preduzeća.

Iz dijagrama se može uočiti da za predstavljanje TP nije obavezno koristiti isključivo krive ili izlomljene linije. Moguće ih je kombinovati, u cilju postizanja što bolje preglednosti i jasnoće dijagrama.

5.3 REČNIK PODATAKA

Nakon završetka dekomponovanja DTP, postoje svi potrebni uslovi za definisanje rečnika podataka. Rečnik podataka predstavlja skup *podataka o podacima* koji se koriste u analiziranom sistemu. To su generalizovani podaci, često u literaturi zvani *metapodacima* (metadata). Rečnik podataka obuhvata tri celine:

- Opis struktura podataka koje se koriste u tokovima podataka,
- Opis polja definisanih nad podacima i
- Opis domena.

5.3.1 Definisanje struktura

Podaci koji se pojavljuju u interakcijama DTP između interfejsa i procesa su najčešće struktuirani. Uzrok tome je činjenica da u sebi sadrže elementarne podatke različitih osnovnih tipova (tekstualne, celobrojne, realni brojevi), a koji su nerazdvojno vezani – izolovani nemaju nikakvo značenje. Na primer pojedinačni podaci: smer, ocene položenih ispita nemaju konkretan informacioni značaj ako nisu objedinjeni u jedinstvenoj strukturi – *student,* kada dobijaju sasvim drugu semantiku – postaju podaci konkretnog studenta.

ISPITNA_PRIJAVA
<<STUDENT>, <ISPIT>, <ISPITIVAC>, <PREDSEDNIK_ KOMISIJE>, BROJ_POKUSAJA>

ISPITNI_SPISAK
<<ISPIT>, <ISPITIVAC>, <PREDSEDNIK_ KOMISIJE>, {<STUDENT>}>

REZULTATI_ISPITA
< <ISPIT> ,{<JEDINACNI_REZULTAT>} >

ISPIT
<DATUM_ISP, PREDMET >

JEDINACNI_REZULTAT
<<STUDENT>,OCENA>

STUDENT
<<OSOBA>,<INDEKS>>

OSOBA
<IME,PREZIME,JMBG>

INDEKS
<GODINA_UPISA, PIN,DODATNO_OBELEZJE, <SEMESTAR>>

SEMESTAR
<RBR_SEMESTRA,DATUM_UPISA>

PREDSEDNIK_ KOMISIJE
<<ISPITIVAC>>

ISPITIVAC
<<OSOBA>,NAUCNO_ZVANJE, NASTAVNO_ZVANJE, [ID_ZAPOSLENOG, BR_UGOVORA]>

Slika 5.19: Primer definisanja struktura u rečniku podataka

Na prethodnom primeru (Slika 5.19) strukture su imenovane (boldirane), a njihov sadržaj predstavljen je uglastim zagradama < >. Može se uočiti da struktura u sebi sadrži drugu ugnježdenu strukturu. Na primer, *Student* je struktura koja se sastoji od 2 strukture: *Osoba* i *Indeks*. Strukture mogu sadržati i nabrojive elemente (nabrajanja - *enumerations*). Tako na primer, struktura *Ispitni_spisak* sadrži listu studenata, što je notirano korišćenjem vitičastih zagrada **{ }**. Strukture mogu sadržati i opcione elemente. To su elementi koji se mogu alternativno koristiti. Na primer struktura *Ispitivac* sadrži dva opciona elementa – *id_zaposlenog* i *broj_ugovora*. Semantika ovih elementa je da ako je ispitivač stalno zaposlen u prosvetnoj ustanovi, onda on ima identifikacioni broj. Ako isti nije stalno zaposlen, da bi bio u statusu ispitivača, mora da bude ovlašćen, što se reguliše ugovorom između u fakulteta i ispitivača. Notacija opcionih elemenata vrši se pravougaonim zagradama **[]**.

Na primeru (Slika 5.19) predstavljen je završen rečnik podataka. Osnovno načelo pri definisanju struktura je da *svi podaci koji se vezano višestruko pojavljuju na više mesta u rečniku treba da se grupišu u imenovanu strukturu*. Na taj način olakšava se modifikacija sistema jer, ako izmenimo podatak u strukturi, ta promena će se odraziti na sve izvedene strukture koji tu strukturu koriste. Na primer ako se promeni način

davanja broja indeksa na fakultetu, to ćemo uraditi *samo* na jednom mestu – u strukturi *indeks*. Ta mala promena će se odraziti međutim na sve strukture koji u sebi sadrže *indeks*. Naizgled u gornjem primeru to je samo struktura *Student*. Ali promena je dakle i u strukturama koje koriste strukturu *STUDENT: ISPITNA_PRIJAVA, ISPITNI_SPISAK, REZULTATI_ISPITA, JEDINACNI_REZULTAT* !

Da broj indeksa nije definisan kao struktura, administrator podataka sistema bi morao da istu izmenu izvrši na svim mestima gde se pojavljuju podaci indeksa, što za kompleksnije sisteme može da predstavlja problem.

5.3.2 Definisanje polja

Polja su zapravo osnovni podaci iz kojih su sačinjene strukture koje su opisane u prethodnom paragrafu. Polja se definišu tako što im se dodeljuje naziv, domen nad kojim su definisana i ograničenja.

NAZIV POLJA	DOMEN	OGRANICENJE
DATUM_ISP	DATE	
DATUM_UPISA	DATE	
GODINA_UPISA	INT(4)	
IME	NAZIV	
PREZIME	NAZIV	
PREDMET	NAZIV	
OCENA	INT(2)	IN(5,6,7,8,9,10)
RBR_SEMESTRA	INT(1)	IN(1,2,3,4,5,6,7,8)
PIN	IDENTIFIKACIJA	
ID_ZAPOSLENOG	IDENTIFIKACIJA	
BR_UGOVORA	IDENTIFIKACIJA	
DODATNO_OBELEZJE	CHAR(3)	IN("II","III","IV")
NAUCNO_ZVANJE	NAZIV	IN ("SPECIJALISTA", "DOKTOR", "MAGISTAR", "DIPL.ING")
NASTAVNICKO_ZVANJE	NAZIV	IN ("REDOVNI PROFESOR, "VANREDNI PROFESOR ", "DOCENT", "ASISTENT","ASISTENT PRIPRAVNIK")

Slika 5.20: Primer definisanja polja u rečniku podataka

Na gornjem primeru (Slika 5.20) u prvoj koloni mogu se uočiti nazivi polja koji se podudaraju sa nazivima osnovnih podataka u strukturama istog rečnika podataka (Slika 5.19).

Domeni su predstavljeni u istoimenoj koloni. To su skupovi definisani nad osnovnim ili izvedenim tipovima podatka. Nad domenima su definisana polja. Ova indirekcija (polje > domen > tip) omogućava da se podaci precizno definišu. Domeni mogu biti:

- Predefinisani – definisani nad osnovnim, ili izvedenim sistemskim tipovima (npr. *INT*- celi broj, osnovni tip, *DATE* – datumski tip, izveden, koji je implementiran u svim savremenim sistemima za upravljanje BP, *CHAR(30)* – karakterski niz, izveden tip...), ili
- Korisnički definisan domen (vidi sledeći paragraf)

U trećoj koloni su ograničenja koja precizno definišu opsege (skupove ili intervale) u kojima se vrednosti podataka mogu kretati. U primeru (Slika 5.20) može se uočiti da nije moguće definisati ograničenja nad svim poljima. Nemoguće je ograničiti spisak imena, prezimena studenata, ili unapred definisati konačan skup predmeta koji će se realizovati na fakultetu. S druge strane nužno je ograničiti skup mogućih ocena koje student može da dobije. Čest slučaj iz srednjoškolske prakse je da nastavnici pored ocena, u dnevnike upisuju tačke, pluseve, minuse. Zamislite IS koji dozvoljava takve unose i kakve posledice bi to imalo u proračunima pojedinačnih i zbirnih uspeha učenika/studenata. Isto tako uvođenjem ograničenja na naučna i nastavnička zvanja onemogućava neovlašćeno definisanje i dodeljivanje novih, nepostojećih i nepropisnih titula i zvanja. Suština ograničenja je da ograniči korisnički unos na zadate vrednosti/intervale i time sačuvaju konzistentnost podataka.

5.3.3 Definisanje domena

Domeni mogu biti predefinisani ili korisnički definisani. Korisnički definisani domeni su *uvek* izvedeni (korisnik je u ovom slučaju analitičar/dizajner sistema). Za ovu vrstu domena može se naći naziv *semantički* što znači da se njihovim definisanjem se bliže određuje smisao podataka (polja koja ga koriste u svojoj definiciji). U primeru su navedena 2 izvedena domena: *IDENTIFIKACIJA* i *NAZIV* (Slika **5.21**). Naziv domena je semantički, jer ukazuje na razlog definisanja: svi nazivi i imena u sistemu koriste u definiciji domen *naziv*; domen *identifikacija* namenjen je jednoznačnom određivanju entiteta u sistemu.

NAZIV DOMENA	PREDEFINISANI DOMEN	OGRANICENJE
NAZIV	CHAR(25)	
IDENTIFIKACIJA	CHAR(10)	IS_UNIQUE_CODE

Slika 5.21: Primer definisanja domena u rečniku podataka

Korisnički domeni se definišu nad osnovnim tipovima podataka u situaciji kada se isti osnovni tipovi sa istim ograničenjima višestruko koriste pri definisanju polja. Na gornjem primeru (Slika 5.20) - IME, PREZIME, PREDMET, NAUCNO_ZVANJE, NASTAVNICKO_ZVANJE, su polja definisana nad istim tipom podatka i sa istom dimenzijom CHAR(25). Iz razloga lakšeg ažuriranja, definisan je domen *NAZIV* . Kada 25 karaktera postane malo za unos podataka navedenih polja, promenom definicije domena *NAZIV*, automatski se menjaju i definicije polja koja su definisana nad tim domenom. Na primer, ako sistem pređe sa ASCII kodiranja podataka na korišćenje UTF-8 – *Unicode* slovčanog formata, kako bi se podržala slova nacionalnih alfabeta, svaki ASCII 8-bitni karakter postaje niz od 4 do 7 karaktera (32 do 56 bita), što znači da je potreban 4 do 7 puta veći broj karaktera (naziv koji je sadržao 20 karaktera ASCII koda imaće 80 do 140 karaktera u UTF-8 formatu). Drugi definisani domen je IDENTIFIKACIJA. Ovaj domen se koristi za jednoznačno označavanje objekata (instanci struktura) koji ga poseduju. Zbog toga on poseduje ograničenje koje ističe da svaki generisani identifikator mora biti jedinstven u sistemu.

6 RELACIONI MODEL PODATAKA

Relacioni model podataka predstavlja teorijsku osnovu za baze podataka relacionog tipa. Njegovi principi i struktura predstavljeni su 1970. godine od strane Edgara T. Koda[3]. Kasniji razvoj baza podataka bio je pod velikim uticajem ideja prezentovanih u Kodovim originalnim delima. Do danas, većina komercijalnih DBMS-ova se zasniva na relacionom modelu, iako postoje objektno-orjentisane opcije, kao i NoSQL baze čiji je nastanak uslovljen zahtevima da se efikasno pristupa ogromnim količinama podataka koji se uglavnom odnose na Internet.

Relacioni model podataka ima matematičku podlogu i omogućava efikasnu podršku za računar. U osnovi relacionog modela je pojam relacije (tabele koja ispunjava određene uslove). Nad relacijama se primenjuju operacije - niz operatora srodnih prirodnom jeziku, a programski jezici za manipulaciju relacionim podacima su zasnovani na matematičkoj teoriji – relacionoj algebri. Ova čvrsta matematička osnova znači da relacioni izrazi (najčešće upiti) mogu da se analiziraju. Kroz proces optimizacije upita, sistem za upravljanje bazama podataka (DBMS) može transformisati početni upit u oblik koji je efikasniji za izvršavanje. Programeri aplikacija ne moraju da poznaju unutrašnju organizaciju baze podataka, niti moraju biti svesni na koji način funkcioniše izvršavanje upita. Ključan koncept kod relacionih baza podataka je tzv. troslojna arhitektura, gde su aplikativni programi potpuno odvojeni od fizičke reprezentacije podataka u bazi. Osnova za ovakvu predstavu je softverska komponenta - DBMS koji se pojavljuje kao srednji sloj (međusloj) između aplikacija i podataka na disku. Aplikativni programer formuliše upit na prirodan način (kroz tzv. jezik visokog nivoa, kao što je SQL), a odgovarajući DBMS će ga izvršiti na najoptimalniji način.

6.1 ISTORIJA I RAZVOJ

Kao i mnoge druge tehnologije u računarskoj industriji, nastanak relacionih baza podataka se vezuje za IBM. Klasični problemi u radu 60-tih i 70-tih godina XX veka odnosili su se na kancelarijsko poslovanje. U tom periodu velike organizacije su uvidele da postaje preskupo da zapošljavaju ogroman broj ljudi za poslove kao što su smeštanje i indeksiranje dokumenata i da vredi ulagati u razvoj jeftinijih i efikasnijih rešenja. Mnoga istraživanja su izvedena u toku ovog perioda. Razvijeni su hijerarhijski, mrežni i relacioni model, a usavršavanjem hardvera, pojavom mikroprocesora, kao i razvojem memorijskih komponenata, računarska tehnologija je počela naglo da se razvija. Performanse novih računara su neprestano povećavane uz smanjenje fizičkih dimenzija, što je praćeno padom cena računarskih komponenata.

[3] *"A Relational Model of Data for Large Shared Data Banks"*, Kodov rad objavljen u mesečnom magazinu *Communications of ACM*, 1970 godine.

Rad *Edgar Ted Codd*-a je dao osnove za korišćenje relacionog računa i algebre. Zbog same tehničke prirode rada i oslanjanja na matematički aparat, njegova važnost nije odmah shvaćena. Ipak, doveo je do formiranja IBM-ove istraživačke grupe System R. Od projekta System R se očekivalo da stvori sistem relacione baze podataka koji bi mogao postati proizvod. Prvi prototip, prezentovan 1974/75. godine je eksperimentalno korišćen u nekoliko organizacija. Kasnije su radne višekorisničke verzije testirane u proizvodnji i knjigovodstvu u nekoliko aeronautičkih kompanija. System R je evoluirao u SQL/DS koji je kasnije postao DB2 sistem za upravljane bazama podataka. Jezik kreiran u System-u R SQL (*Structured Query Language*) je postao standard za relacione baze podataka i danas je ISO standard. Iako je IBM uveo originalni koncept relacionih baza podataka i SQL standard, prvi komercijalni proizvod realizovao je Honeywell u junu 1976. godine. Bio je baziran na istim principima kao i IBM-ov sistem, ali je dizajniran i implementiran odvojeno od IBM-ovog rada.

Softver za relacione baze podataka se kontinuirano usavršavao tokom 80-tih. Delom putem povratnih informacija od kupaca, a delom zbog razvoja računarskih sistema i povećane upotrebe personalnih računara i distribuiranih sistema. Prve baze podataka su imale mogućnost rada sa podacima veličine do 8 MB (Sistem R). Današnje baze podataka mogu biti i veličine terabajta ili petabajta podataka kada sadrže podatke za mailing liste, informacije o kupcima za veleprodajne lance itd.

SQL standard je prešao iz IBM-a u ANSI (*American National Standards Institution*) i ISO (*International Standards Organization*), koji je formirao i radnu grupu za nastavak njegovog razvoja. Ovaj razvoj je još uvek u toku.

6.2 KLJUČNI KONCEPTI

U srcu relacionog modela nalazi se koncept tabele (koja se pod određenim uslovima naziva i relacija) u kojoj su smešteni svi podaci. Svaka tabela je načinjena od slogova (redova u tabeli) i polja (kolona u tabeli, atributa).

Veoma je važno zapaziti da kako i gde su tabele smeštene ne pravi nikakvu razliku. Svaka tabela se identifikuje jedinstvenim imenom koje baza podataka koristi da bi pronašla tabelu. U okviru jedne baze podataka nazivi tabela moraju biti unikatni, kako bi DBMS znao kojoj tabeli se obraća. Korisniku je potrebno samo da zna ime tabele. Nema potrebe da se vodi računa o tome kako su podaci smešteni na disku. Ovo je različito od hijerarhijskog i mrežnog modela u kojima korisnik mora da razume kako su podaci struktuirani unutar baze podataka da bi mogao da ih pretražuje, umeće nove, ažurira ili briše slogove.

Relaciona baza podataka standardno se sastoji iz više tabela. Ipak, za razliku od mrežne baze podataka, tabele nisu povezane pokazivačima. Umesto toga koriste se „ključevi“ da upare redove podataka u različitim tabelama. Ključ može biti samo jedna

ili više kolona u tabeli, koja odgovara kolonama u drugim tabelama. Preciznije, vrednosti koje se unose u takve kolone nisu slobodne, pošto su kolone povezane. Bilo koja od kolona u tabeli može biti ključ (ako ispunjava uslov unikatnosti atributa koji mogu biti uneseni) ili se više kolona može grupisati u jedan ključ. Za razliku od pokazivača, nije potrebno da se definišu svi ključevi unapred. Kolona se može koristiti kao ključ čak iako originalno nije bila namenjena za to.

Kada ključ sadrži podatke koji imaju eksterno, stvarno značenje (kao što je ID ime osobe, ISBN kod knjige ili serijski broj automobila), takav ključ se naziva „prirodni" ključ. Ako prirodni ključ nije pogodan, može se dodeliti ključ (npr. davanje identifikacionog broja zaposlenim ili redni broj unosa podataka). U praksi, većina baza podataka ima oba – i generisane (veštačke) i prirodne ključeve. Generisani ključevi imaju prednost nad prirodnim ključevima zato što se kod prirodnih ključeva (pogotovo kod onih koji se sastoje iz više atributa) mogu javiti posebne funkcijske zavisnosti koje mogu da smetaju pri održavanju baze podataka ili pri postavljanju upita. Prirodni ključevi jesu manje pouzdani, ali se koriste za pretrage i integraciju sa drugim bazama podataka. Na primer, zapisi u dve nezavisno kreirane baze podataka mogu da se upare po jedinstvenom matičnom broju, osim u slučajevima kada je JMBG pogrešan, nedostaje ili je promenjen.

Zahtev za podatkom iz relacione baze podataka se dobija postavljanjem SQL upita. Iako je SQL originalno namenjen za krajnje korisnike, mnogo češće se SQL upiti ugrađuju u softver koji omogućava lakši korisnički interfejs. Kao odgovor na upit, baza podataka vraća skup podataka, koji je u stvari lista redova koji sadrže odgovor. Najjednostavniji upit je da se dobiju svi redovi iz tabele, ali češće, redovi se filtriraju u skladu sa željenim kriterijumom da bi se dobio traženi odgovor. Često se podaci iz više tabela kombinuju u jednu, kombinovanjem zapisa ili čak kombinovanjem kolona i njihovim spajanjem kada je to moguće.

Koncept relacionih baza podataka je fleksibilan. Kada je baza podataka definisana i realizovana, programeri naknadno mogu da pišu upite koji nisu bili predviđeni od strane dizajnera baze podataka. Preko pogleda (WIEW) u odnosu na stvarne fizičke podatke u bazi podataka na disku, stvara se privid da aplikacije rade sa svojim bazama. Više pogleda mogu da potiču iz iste baze podataka. Kao rezultat, relacione baze podataka mogu da se koriste u više aplikacija koje originalni dizajneri nisu predvideli, što je posebno važno za baze podataka koje se mogu koristiti decenijama. Ovo je model relacionih baza podataka učinilo veoma popularnim u poslovnoj primeni.

Sa stanovišta pouzdanosti i stabilnosti podataka, relacione baze podataka su uvele ključne koncepte. Kada je model baze dobro definisan, veoma je teško uneti pogrešan podataka, izvršiti pogrešno brisanje ili menjanje podataka. Sve što je suprotno definiciji baze podataka, DBMS će odbiti da izvrši, ukoliko to narušava egzistencijalni integritet. Zbog toga je model, tj. definisanje baze podataka je od velikog značaja posebno u realnom radu aplikacija.

6.3 KOMPONENTE U RELACIONOM MODELU PODATAKA

Relacioni model podataka razmatra sledeće komponente:

- Strukturna komponenta (objekti, entiteti);
- Integritetska komponenta (ograničenja, zaštita);
- Manipulativna komponenta (ažuriranje i manipulisanje).

Već smo ranije objasnili da relacioni model podataka na realni svet gleda putem tabela. Tabele u relacionom modelu nazivamo relacijama. Redosled kolona u relaciji nije bitan. U jednoj relaciji ne postoje dva identična reda. Tabelama se predstavljaju klase entiteta iz realnog sveta.

Svaki entitet poseduje svojstva. Svojstva entiteta se nazivaju atributi. Atributima dajemo značenje pojedinačnim podacima. Relacioni model razlikuje proste (jednostavne) i složene atribute.

Pojedinačan podatak (prost atribut) je najmanja nedeljiva jedinica u relacionom modelu. Pojedinačni podaci su "Beograd", 12,56, "Petar" itd. Ako bi podelili bilo koji od pojedinačnih podataka on bi izgubio prvobitni smisao. Na primer podatak "Petar" možemo deliti na slogove ali dobijeni slogovi nemaju značenje koje je imao podatak pre podele.

Skup svih mogućih vrednosti nekog atributa nazivamo domenom atributa. Skup svih mogućih gradova je domen atributa *Grad*. Skup svih mogućih boja je domen atributa *Boja* itd. Svaki atribut mora imati samo jedan pridruženi domen. Više različitih atributa može se zadati nad istim domenom. Na primer, atributi *Mesto_boravka* i *Mesto_rodjenja* imaju isti domen. Takođe atributi *Ime_studenta* i *Ime_profesora* imaju isti domen. Postoje klasični domeni (kao što su numerički, karakteri, stringovi, datumi i sl.). Pojedini domeni mogu biti predefinisani (na primer, ocena iz nekog predmeta na fakultetu je celobrojni podataka, ali je ograničen na vrednosti od 5 do 10). Korisnički definisani domeni nastaju kada korisnik precizno opiše vrednosti koje se mogu uneti za željeni atribut.

Relacioni model podataka počiva na nekoliko formalnih pojmova.

6.3.1 Šema relacije

Šema relacije **R**, u oznaci $R(A_1,A_2,...,A_N)$, je konačan skup atributa $\{A_1,A_2,...,A_N\}$ i konačan skup ograničenja nad vrednostima tih atributa. Kako je šema relacije definisana preko skupa, redosled atributa nije bitan. Svaki naziv atributa je jedinstven u okviru šeme relacije, tj. ne može se ponoviti isto ime atributa u jednoj šemi relacije.

Šemom relacije se predstavljaju svojstva klase objekata ili veza nekog sistema. Šema relacije može da se tumači i kao definicija strukture neke datoteke. Nazivi atributa u šemi relacije moraju da budu različiti, redosled kolona u šemi relacije nije bitan.

6.3.2 Relacija

Relacija **r** zadata nad šemom relacije je konačan skup n-torki šeme relacije **R**. Relacija ne sadrži dve jednake n-torke. Redosled n-torki nije bitan. Kako je relacija skup n-torki i sve n-torke su iste dužine, u relacionom modelu n-torke ne mogu imati promenljivu dužinu.

Jednostavnije rečeno, šema relacije je opis strukture jedne tabele, a sama relacija je sadržaj te tabele. Naravno, da bi takva tabela bila relacija poštuju se gore navedena ograničenja. Relaciji u praksi odgovara jedna datoteka, a svakoj n-torki odgovara jedan slog te datoteke. Slogovi u datoteci su zapisani u određenom redosledu, najčešće po redosledu unošenja

Primer: Šema relacije kojom se opisuje klasa studenata, gde su relevantni atributi Broj indeksa i Ime studenta, može da bude:

STUDENT (BrInd Ime),

a relacija nad ovakvom šemom u jednom trenutku može da bude:

student	(BrInd	Ime)
	2017123	J.Jankovic
	2018456	P.Petrovic
	2016222	J.Jovanovic
	2016555	M.Markovic

6.3.3 Relaciona baza podataka

Relaciona baza podataka je konačan skup relacija koje su definisane odgovarajućim šemama relacija $\{R_i\}$ i konačan skup ograničenja koja važe između datih šema. Pošto u jednoj bazi postoji više šema relacija koje su međusobno povezane, neminovno je da postoje nova, posebna ograničenja koja se odnose na ove veze. Suština relacionih baza je upravo u ovim vezama. Kod definicije šeme relacije bila su razmatrana samo ograničenja nad vrednostima pojedinih atributa. Relaciona baza podataka predstavlja stanje nekog sistema u vremenu. Ona je promenljiva, a promene se odnose na dodavanje (INSERT), brisanje (DELETE) i izmenu (UPDATE) n-torki.

6.3.4 Super ključ

Jedna tabela se može posmatrati kao relacija ukoliko ispunjava odgovarajuće uslove. Najvažniji uslov je da se u jednoj relaciji nikada ne mogu pojaviti dve identične n-torke (zapisa, reda). Ovo je tzv. osobina unikatnosti n-torki. Na osnovu ove osobine, sigurno postoji skup atributa (jedan, više ili svi atributi jedne šeme relacije) koji na jedinstven način određuju svaku n-torku. Zaključuje se da super ključ uvek postoji. Dalje, na osnovu date šeme relacije, može se zaključiti da se ne moraju upotrebiti svi atributi za identifikaciju željene n-torke, tj. postoje i minimalniji identifikatori (po broju atributa). Na taj način dolazimo do pojma – kandidat ključ.

6.3.5 Kandidat ključ

Kandidat ključ predstavlja jedan ili više atributa koji na jedinstven način određuju svaki zapis jedne relacije. To je jedan od prvih pojmova u relacionom modelu koji se koristi za pouzdan (siguran pristup) željenim podacima. U suštini, baze podataka postoje da bi se u njih smeštali podaci, ali pristup željenim podacima mora biti nepogrešiv. Na osnovu kandidat ključa mogu se pouzdano razlikovati dve n-torke u jednoj relaciji.

Kandidat ključ ima iste osobine kao i super ključ, ali je minimalan. Jedna šema relacije može imati više kandidat ključeva. Na primer u šemi relacije

Ispit (BrInd, SifP, Ocena)

Super ključ je sigurno {BrInd, SifP, Ocena} zato što se na osnovu ovih atributa nedvosmisleno može pristupiti željenom zapisu. Međutim, istu osobinu ima i {BrInd, SifP}, pa se to može smatrati za kandidat ključ za šemu relacije ISPIT. Pošto jedan student ima ocene iz više predmeta, a iz jednog predmeta ocene ima više studenata, ni jedan pojedinačni atribut nema osobinu da može na jedinstven način da ukaže na svaku n-torku.

Dakle, kandidat ključ ima osobinu minimalnosti. Kandidat ključevi su važni zato što omogućavaju direktan pristup podacima.

6.3.6 Primarni ključ

Izborom jednog kandidat ključa koji nam služi za identifikaciju n-torki u odgovarajućoj relaciji, biramo njen *primarni ključ*. U relacionom modelu podataka primarni ključevi se obeležavaju. Ostale kandidat ključeve nazivamo alternativnim ključevima. Primarni ključ se može sastojati iz jednog ili više atributa. Može biti prirodni ili generisani (veštački). Prirodni ključevi uvek posledica nekog prirodnog

(suštinskog) ograničenja iz realnog sveta koje se nikada ne može promeniti. Kada ima više ravnopravnih kandidata za primarni ključ, bira se onaj koji zauzima manje memorije. Svaka šema relacije ima jedan i samo jedan primarni ključ.

6.3.7 Spoljni ključ

Da bi se uveo pojam spoljašnjeg ključa, neophodno je da postoje dve relacije. Uloga spoljnih ključeva je u uspostavljanju veza između relacija. Ukoliko neki atribut u relaciji pokazuje (zadat je nad istim domenom) na primarni ključ iste ili druge relacije, taj atribut nazivamo *spoljnim ključem* relacije. Spoljni ključ u šemi relacije R je svaki njen podskup atributa za koji važi ograničenje vrednosti u relaciji **r** na sledeće dve vrednosti:

- vrednost primarnog ključa u nekoj relaciji (tzv. ciljnoj relaciji)
- vrednost NULL (ukoliko je to dozvoljeno pri definisanju)

Jedna šema relacije može da sadrži više spoljnih ključeva. Spoljni ključ može biti u sastavu primarnog ključa. Spoljni ključ jedne relacije može istovremeno biti i primarni ključ date relacije u celini. Spoljni ključ se može ili se ne mora obeležavati - obeležavanje doprinosi preglednosti modela.

6.3.8 NULL vrednost

Posebnu ulogu u definisanju opštih pravila integriteta ima vrednost "NULL". To je univerzalna vrednost koja se može uneti za atribute koji su definisani nad različitim domenima. Često nismo u stanju da znamo vrednost za neki podatak. Razlozi mogu biti različiti. Mogu biti posledica strukture šeme relacije. U trenutku unosa podataka moguće je da se ne zna vrednost nekog atributa. Na primer, ako se kod unosa podataka o studentu koji se upisuje zahteva i unos željenog smera, a smer se određuje od III godine studija, takav podatak se sigurno ne može uneti. Ili podatak postoji, ali se do njega teško može doći. Ako pri definiciji šeme relacije nije drugačije rečeno na tom mestu se može naći NULL podatak.

Kada se u jednoj relaciji pojavljuje veliki broj podataka sa NULL vrednostima to ukazuje na to da je modelovanje loše urađeno. Ili su dobijene relacije loših struktura, pa se trebaju popraviti tj., normalizovati. Rad sa zapisima u kojima postoje NULL vrednosti je posebno osetljiv, pošto je logika komparacije atributa sa NULL vrednostima specifična.

6.4 INTEGRITET PODATAKA U RELACIONOM MODELU

Kroz istoriju razvoja relacionog modela podataka najveće izmene je pretrpeo integritet podatka. Jedan od razloga je i to što ova oblast nije precizno utemeljena kao što su precizno zasnovani objekti relacionog modela i operacije nad objektima.

Integritet (konzistentnost) baze podataka odnosi se na dozvoljene vrednosti podataka sadržanih u bazi. Neispravni podaci mogu biti posledica ažuriranja (unošenja i brisanja podataka) bilo namernog, bilo nenamernog od strane korisnika u slučajevima kada je relacioni model loše definisan. Integritet podataka podrazumeva sve mere kojima je cilj da spreči unos neispravnih podataka. Mere za sprečavanje slučajnih pogrešaka se značajno razlikuju od mera za sprečavanje namernih aktivnosti. Iz integriteta baza podataka izuzete su sve mere kojima je cilj sprečavanje ilegalnih operacija. One su predmet izučavanje posebne oblasti koja se odnosi na zaštitu podataka.

Pravila integriteta su ograničenja sadržaja baze podataka na neka dozvoljena stanja. Cilj je sprečavanje unosa pogrešnih podataka u bazu. Sva ažuriranja, brisanja i unošenja n-torki moraju biti u skladu sa tim ograničenjima. Ako se ta pravila ispoštuju, ne mora da znači da je uneti podatak ispravan, ali ukoliko se ne ispoštuju, onda je podatak koji se unosi sigurno neispravan.

Pravila integriteta delimo na:

- Korisnička pravila integriteta;
- Identifikacioni integritet;
- Referencijalni integritet.

6.4.1 Korisnička pravila integriteta

Ova pravila zavise od konkretne baze. Ona proističu iz ograničenja koja važe i u realnom svetu. To su pravila koja se odnose na ograničenja koja važe za pojedine atribute. Na primer za ocene iz pojedinih predmeta na fakultetu – od 5 do 10. Ili, broj indeksa mora da bude manji od maksimalne kvote koja je odobrena kod akreditacije. Ili, studentu mora da se unesu sve ocene, da bi pristupio odbrani diplomskog rada itd. Kod loših šema relacija, gde se javljaju međusobno zavisni atributi, mora se voditi računa šta se unosi za vrednosti tih atributa.

6.4.2 Identifikacioni integritet

Odnosi se na opšta ograničenja za primarni ključ relacije. Već smo u definisanju primarnog ključa zahtevali minimalnost i jednoznačnost. Vrednost primarnog ključa

jednoznačno određuju n-torke i ne može se iz njega izbaciti ni jedan atribut, a da on i dalje poseduje jednoznačnost. Ovim dobijamo uslov za integritet primarnog ključa. S obzirom na definiciju primarnog ključa, identifikacioni integritet se može definisati preko njega na sledeći način: ni jedna komponenta primarnog ključa relacije ne sme imati NULL vrednost.

6.4.3 Referencijalni integritet

Referencijalni integritet se odnosi na dozvoljena stanja spoljnih ključeva. Uslov referencijalnog integriteta se može definisati na sledeći način: Baza podataka ne sme da sadrži ni jednu nepovezanu vrednost za spoljni ključ. Znači, vrednost spoljnog ključa može biti vrednost primarnog ključa odgovarajuće ciljne n-torke ili NULL vrednost (ukoliko je to pri kreiranju šeme relacije omogućeno).

Suština referencijalnog integriteta je u ograničavanju vrednosti spoljnog ključa. Sa stanovišta izmena (ažuriranja) u relaciji koja sadrži spoljni ključ (pozivajuća relacija) to podrazumeva da važe sledeća ograničenja:

- Ne može se uneti n-torka sa vrednošću stranog ključa koja nije jednaka nekoj vrednosti primarnog ključa u ciljnoj relaciji ili NULL vrednosti;
- Ne može se izmeniti n-torka tako da vrednost stranog ključa ne bude jednaka nekoj vrednosti primarnog ključa u ciljnoj relaciji ili NULL vrednosti.

Sa stanovišta ciljne relacije važi sledeće:

- Dodavanje nove n-torke (u ciljnoj relaciji) ne narušava referencijalni integritet - nastaje samo nova vrednost primarnog ključa
- Uklanjanjem n-torke (a izmena ponekad) dovodi do nestanka jedne vrednosti primarnog ključa. Ako bi se ta operacija izvršavala bezuslovno to bi narušilo referencijalni integritet

Postavlja se pitanje kako postupiti kada se vrši ažuriranje u ciljnoj relaciji, a da se ne naruši referencijalni integritet. Takva specifikacija se naziva: dinamička specifikacija referencijalnog integriteta i odnosi se samo na kritične operacije, a to su operacija uklanjanja (DELETE) i operacija izmene (UPDATE). Uz ove akcije neophodno je navesti jednu od sledeće tri klauzule: RESTRICTED, CASCADES, NULLS

- RESTRICTED: operacija se ne izvršava ako u pozivajućoj relaciji postoji vrednost stranog ključa koja odgovara vrednosti primarnog ključa u ciljnoj relaciji

- CASCADES: operacija se uvek izvršava. Ako je uklonjena n-torka iz ciljne relacije, u pozivajućoj relaciji se uklanjaju sve n-torke sa datim ključem. Ako je došlo do izmene, menjaju se sve vrednosti n-torki u pozivajućoj relaciji sa novim spoljnim ključem
- NULLS: operacija se uvek izvršava. U pozivajućoj relaciji se u svim n-torkama koje imaju dati promenjeni spoljni ključ, menja njegova vrednost u NULL. NULLS klauzula se ne može sprovesti ako je spoljni ključ u sastavu primarnog ključa, ili ga čini u celini – to bi bilo u suprotnosti sa identifikacionim integritetom.

7 RELACIONA ALGEBRA

Relaciona algebra pripada kategoriji formalnih upitnih jezika proceduralnog karaktera. Čini je skup operatora za rad sa relacijama, a rezultati operacija su takođe relacije. Relaciona algebra je osnova za upitne jezike koje koriste ljudi. Svaki od algebarskih izraza je jedan upit ili pretraživanje. Upitni jezik je jezik kojim korisnici zahtevaju informacije iz baze podataka

Operacija je primena nekog operatora na jednu ili više izvornih relacija kako bi se formirala nova relacija. Relacionu algebru čini skup od 8 operacija koje se nazivaju osnovnim (5 elementarnih i 3 izvedene)

- **Elementarne:** restrikcija (selekcija), projekcija, unija, razlika, Dekartov proizvod
- **Izvedene:** presek, spajanje, deljenje

Operacije se mogu klasifikovati i prema broju operanada. Pojedine operacije se izvršavaju nad jednim, a pojedine nad dve relacije.

- **Unarne** (1 operand)
- **Binarne** (2 operanda)

Simbol	Naziv	Složenost	Br. operanada
σ	restrikcija	elementarna	unarna
π	projekcija	elementarna	unarna
$\cup$	unija	elementarna	binarna
-	razlika	elementarna	binarna
$\cap$	presek	izvedena	binarna
$\times$	Dekartov proizvod	elementarna	binarna
$\rtimes$	spajanje	izvedena	binarna
$/$	deljenje	izvedena	binarna

Tabela 7.1: Klasifikacija osam osnovnih operacija

7.1 RESTRIKCIJA (σ)

Restrikcija (selekcija, ograničenje, eng: *restrict*) – je operacija relacione algebre koja iz polazne relacije kao rezultat daje tačno određene n-torke tj. samo one n-torke koje zadovoljavaju zadati kriterijum (izdvajanje redova u tabeli). Kriterijum je neki logički izraz koji je izračunljiv nad svakom n-torkom. Dobijena relacija ima istu strukturu kao i polazna.

Ako je **r** relacija nad šemom R(X), a P(X) uslov restrikcije, formalna definicija restrikcije je:

$\sigma_{P(X)}(r) = t(X) = \{x \mid x \in r \wedge P(X)\}$

Operacija restrikcije kao rezultat može da dâ prazan skup ∅ n-torki.

Uslov restrikcije P se sastoji iz članova koji su povezani sa:

∧ (and), ∨ (or), ¬ (not),

a svaki član je u formi:

<atribut> *op* <atribut> ili <konstanta>,

gde je *op* jedan od: =, ≠, >, ≥, <, ≤.

Primer 1.

Iz relacije r(A;B;C;D):

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

nakon operacije $\sigma_{A=B \wedge D>5}(r)$ dobija se:

A	B	C	D
α	α	1	7
β	β	23	10

Primer 2.

Iz relacije Predmet (SifPredmet, Naziv, ECTS):

SifPredmet	Naziv	ECTS
SPRI01	Informatika	6
SPRI02	Matematika	8
SPRI03	Programiranje 1	8
SPRI04	Engleski jezik 1	6
--------	--------	---
SPRI08	Strukture podataka i algoritmi	8

izdvojiti sve one predmete koji se vrednuju sa 6 ECTS poena.

Nakon operacije restrikcije: $\sigma_{\text{ECTS}=6}(\text{Predmet})$ dobija se:

SifPredmet	Naziv	ECTS
SPRI01	Informatika	6
SPRI04	Engleski jezik 1	6

Primer 3.

Iz relacije Predmet (SifPredmet, Naziv, ECTS) izdvojiti sve one predmete koji se vrednuju sa 15 ECTS poena.

Nakon operacije restrikcije: $\sigma_{\text{ECTS}=15}(\text{Predmet})$ dobija se $\varnothing$ (prazan skup), zato što ni jedan predmet u relaciji Predmet ne nosi 15 ECTS poena.

7.2 PROJEKCIJA (π)

Projekcija (eng: *project*) kao rezultat daje relaciju koja se sastoji samo od određenih atributa zadate relacije (izdvajanje kolona u tabeli). Iz polazne relacije po zadatom skupu atributa formira se nova relacija kao skup n-torki nad tim atributima. Zadati skup

atributa mora biti podskup skupa atributa polazne relacije. Vrednosti atributa u n-torkama nastale relacije odgovaraju onima u polaznoj relaciji.

Ako je **r** relacija nad šemom R(X), Y zadati skup atributa, a *x* i *y* n-torke nad X i Y, formalna definicija restrikcije je: $\pi_Y(r) = t(Y) = \{t \mid Y \subseteq X \wedge y \in x\}$

Primenom operacije projekcije moguće je da više n-torki polazne relacije daje iste vrednosti. Pošto rezultat operacije mora biti relacija, uzima se samo jedna n-torka.

Primer 1.

Iz relacije r(A;B;C):

A	**B**	**C**
α	10	1
α	20	1
β	30	1
β	40	2

Nakon primene operacije projekcije $\pi_{A,C}$ (r) dobija se t_1(A,C)

A	**C**
α	1
α	1
β	1
β	2

a zbog uslova identifikacionog integriteta krajnji rezultat je t_2 (A,C)

A	**C**
α	1
β	1
β	2

Za razliku od restrikcije, rezultirajuće n-torke nemaju sve atribute koje ima originalna relacija, već samo one po kojima se izvodi projekcija.

Primer 2.

Projekcija relacije Student (BrInd, Ime, Prezime, Adresa, telefon) po atributima BrInd, Ime i Prezime:

$\pi_{\text{BrInd, Ime, Prezime}}(\text{Student})=t(\text{BrInd, Ime, Prezime})$

kao rezultat daje relaciju koja sadrži samo podatke BrInd, Ime i Prezime. Kako je BrInd primarni ključ relacije Student ne smanjuje se broj n-torki u novonastaloj relaciji. Projekcija samo po Imenu ili Prezimenu može da dovede do smanjenja broja n-torki u novonastaloj relaciji, kao i do gubitka podataka za studente sa istim imenom ili prezimenom.

Primer 3.

Iz relacije Predmet (SifPredmet, Naziv, ECTS) pronaći koliko različitih ECTS se koristi za vrednovanje predmeta.

Nakon operacije restrikcije: $\pi_{\text{ECTS}}(\text{Predmet})$ dobija se relacija:

ECTS
8
6

zato što se svi predmeti vrednuju ili sa 6 ili sa 8 ECTS poena. Pošto se rezultat posmatra kao relacija, svi duplikati su eliminisani (ima više predmeta koji se vrednuju sa 6 ECTS i više predmeta koji se vrednuju sa 8 ECTS. Takođe, ni jedan predmet se ne vrednuje sa nekim drugim brojem ECTS.

Primer 4.

Operacije relacione algebre se mogu kombinovati. Na primer, nad relacijom:

Ispit(Brind#,SifPredmeta#,IdProf#,Sala,Vreme)

koja sadrži podatke o studentima, predmetima koje polažu, profesorima kod kojih polažu, sali gde se polaže i vreme polaganja. Ako se primeni kombinovana (složena) operacija relacione algebre:

$\pi_{\text{BrInd}}(\sigma_{\text{SifPredmeta='BP'}}(\text{Ispit}))\rightarrow \text{bp\_ispit(BrInd)}$

kao rezultat se dobija relacija u kojoj se nalaze podaci o brojevima indeksa studenata koji su polagali predmet Baze podataka.

Pogodnim kombinovanjem i postavljanjem uslova mogu se dobiti željene informacije. Neka je BP šifra za predmet Baze podataka, PJ šifra predmeta programski jezici, MV je identifikacija profesora Mladena Veinovića, A1 označava amfiteatar. Tada su moguće složene operacije (postavljeni upiti):

- $\pi_{\text{SifPredmeta}}(\sigma_{\text{Sala='A1'}}(\text{Ispit})) \rightarrow ?$
- $\pi_{\text{BrInd}}(\sigma_{\text{Sala='A1'} \wedge \text{Vreme=10:00}}(\text{Ispit})) \rightarrow ?$
- $\pi_{\text{BrInd}}(\sigma_{\text{IdProf='MV'}}(\text{Ispit})) \rightarrow ?$
- $\pi_{\text{BrInd}}(\sigma_{\text{ IdProf='MV'} \wedge \text{ SifPredmeta='BP'}}(\text{Ispit})) \rightarrow ?$
- $\pi_{\text{IdProf}}(\sigma_{\text{ SifPredmeta='PJ'} \vee \text{ SifPredmeta='BP'}}(\text{Ispit})) \rightarrow ?$

7.3 UNIJA ($\cup$)

Rezultat unije dve relacije je relacija koja se sastoji od svih n-torki koje se nalaze i u jednoj i u drugoj relaciji. Unija je operacija relacione algebre kojom se iz dve polazne relacije formira novu koja sadrži sve n-torke iz obe relacije. Ova operacija nije moguća između bilo koje dve relacije, tj. mora biti zadovoljeno:

- Šeme relacija moraju imati isti broj atributa;
- Atributi šema relacija redom odgovaraju po značenju i tipu (ne mora po nazivu).

Navedeni uslovi se nazivaju unijska kompatibilnost

Ako su **r** i **s** relacije nad šemom R(X) i S(X), gde X označava unijski kompatibilan skup atributa u obe relacije, formalna definicija unije je:

$r \cup s = t(X) = \{x \mid x \in r \vee x \in s\}$.

Svaka n-torka koja je prisutna u obe relacije pojavljuje se samo jednom u rezultatu (novodobijenoj relaciji).

Primer 1.

Za unijski kompatibilne relacije r i s

A	B
α	1
α	2
β	1

A	B
α	2
β	3

nakon operacije r ∪ s dobija se sledeća relacija

A	B
α	1
α	2
β	1
β	3

Primer 2.

Unijom relacija A i B

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
1772	Maksimović	Ilija	015 723 543

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
2345	Petrović	Petar	023 47 833

Dobija se relacija C = A ∪ B

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
1772	Maksimović	Ilija	015 723 543
2345	Petrović	Petar	023 47 833

Primer 3.

Neka postoje dve unijski kompatibilne relacije koje sadrže podatke o studentima koji su položili predmete Baze podataka i Mreže i neka je njihov trenutni sadržaj:

PoložioBaze

BrInd	Ime
2017100	Petar Petrović
2016200	Marko Marković

PoložioMreže

BrInd	Ime
2018300	Janko Janković
2017100	PetarPetrović

Nakon operacije unije nad ove dve relacije (PoložioBaze ∪ PoložioMreže) novodobijena relacija može da se nazove PoložioBarJedan, ima istu strukturu kao polazne relacije i njen trenutni sadržaj bi bio:

PoložioBarJedan

BrInd	Ime
2017100	Petar Petrović
2016200	Marko Marković
2018300	Janko Janković

U njoj su svi studenti koji su položili ILI Baze, ILI mreže ILI oba predmeta.

7.4 RAZLIKA (-)

Rezultat razlike (eng: *difference*) - dve relacije je relacija koja se sastoji od svih n-torki koje se nalaze u prvoj i ne nalaze u drugoj relaciji, tj. iz prve relacije se isključuju sve n-torke zajedničke s drugom relacijom. Iz dve polazne relacije formira se nova koja sadrži sve n-torke prve relacije koje se ne nalaze u drugoj. Ova operacija je moguća samo između unijski kompatibilnih relacija.

Ako su **r** i **s** relacije nad šemom R(X) i S(X), X označava unijski kompatibilan skup atributa u obe relacije, formalna definicija razlike je:

$$\mathbf{r} - \mathbf{s} = \mathbf{t}(X) = \{x \mid x \in \mathbf{r} \wedge x \notin \mathbf{s}\}$$

Primer 1.

Za relacije A i B

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
1772	Maksimović	Ilija	015 723 543

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
2345	Petrović	Petar	023 47 833

primenom operacije razlike formira se relacija C = A - B

Sifra	Prezime	Ime	Tel_broj
1772	Maksimović	Ilija	15 723 543

ili relacija C= B- A

Sifra	Prezime	Ime	Tel_broj
2345	Petrović	Petar	023 47 833

Primer 2.

Neka postoje dve unijski kompatibilne relacije koje sadrže podatke o studentima koji su položili predmete Baze podataka i Mreže i neka je njihov trenutni sadržaj:

PoložioBaze

BrInd	Ime
2017100	Petar Petrović
2016200	Marko Marković

’

PoložioMreže

BrInd	Ime
2018300	Janko Janković
2017100	Petar Petrović

Nakon operacije razlike nad ove dve relacije (PoložioBaze - PoložioMreže) novodobijena relacija može da se nazove PoložioSamoBaze, ima istu strukturu kao polazne relacije i njen trenutni sadržaj bi bio:

PoložioSamoBaze

BrInd	Ime
2016200	Marko Marković

U njoj su svi studenti koji su položili samo Baze, a nisu položili Mreže.

7.5 PRESEK - ∩

Presek (eng. *intersect*) - rezultat preseka dve relacije je relacija koja se sastoji od n-torki koje su zajedničke za obe relacije, odnosno koja sadrži sve n-torke koje se nalaze i u jednoj i u drugoj relaciji. Ova operacija je moguća samo između unijski kompatibilnih relacija.

Ako su **r** i **s** relacije nad šemom R(X) i S(X), X označava unijski kompatibilan skup atributa u obe relacije, formalna definicija preseka je:

$\mathbf{r} \cap \mathbf{s} = \mathbf{t}(X) = \{x \mid x \in \mathbf{r} \wedge x \in \mathbf{s}\}$.

Presek je izvedena operacija, može se izvesti iz

$\mathbf{r} \cap \mathbf{s} = \mathbf{r} - (\mathbf{r}\text{-}\mathbf{s})$

Primer 1.

Za relacije A i B

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
1772	Maksimović	Ilija	015 723 543

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952
2345	Petrović	Petar	023 47 833

primenom operacije preseka formira se relacija C = A ∩ B

Sifra	Prezime	Ime	Tel_broj
3244	Aksentijević	Petar	011 334 952

Primer 2.

Neka postoje dve unijski kompatibilne relacije koje sadrže podatke o studentima koji su položili predmete Baze podataka i Mreže i neka je njihov trenutni sadržaj:

PoložioBaze

BrInd	Ime
2017100	Petar Petrović
2016200	Marko Marković

PoložioMreže

BrInd	Ime
2018300	Janko Janković
2017100	Petar Petrović

Nakon operacije preseka ove dve relacije (PoložioBaze ∩ PoložioMreže) novodobijena relacija može da se nazove PoložioObaPredmeta, ima istu strukturu kao polazne relacije i njen trenutni sadržaj bi bio:

PoložioObaPredmeta

BrInd	Ime
2017100	Petar Petrović

U njoj su svi studenti koji su položili I Baze podataka I Mreže.

7.6 DEKARTOV PROIZVOD (×)

Dekartov proizvod dve relacije je relacija koja se sastoji od svih mogućih kombinacija parova n-torki, pri čemu je prva n-torka iz prve, a druga iz druge relacije. Iz dve polazne relacije formira novu sa n-torkama dobijenim tako što se svaka n-torka prve relacije spaja sa svakom iz druge. Šema nastale relacije sadrži sve atribute polaznih relacija.

Ako su **r** i **s** relacije nad šemom R(X) i S(Y), a X i Y su skupovi atributa u šemama relacija, formalna definicija Dekartovog proizvoda je:
$\mathbf{r} \times \mathbf{s} = \mathbf{t}(XY) = \{xy \mid x \in \mathbf{r} \wedge y \in \mathbf{s}\}$

Kod označavanja za puni naziv atributa se može koristiti relacija.atribut (ako je $X \cap Y \neq \emptyset$), da bi se mogli razlikovati atributi iz jedne i druge relacije.

Primer 1.

Za polazne relacije r i s

A	B
α	1
β	2

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

kao rezultat dekartovog proizvoda r×s dobija se

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Nakon primene dekartovog proizvoda, u rezultujućoj relaciji samo pojedine n-torke imaju smisla.

Primer 2.

Nad relacijama

Student(Br_Ind, Ime, Prezime),

Ocena(Br_Ind, Sif_Pred, Ocena)

izdvojiti sve studente koji su položili predmete kao i ocene iz datih predmeta.

Nakon primene Dekartovog proizvoda, samo neke od n-torki sadrže tražene podatke.

Student × **Ocena** → **t**(S.Br_Ind, Ime, Prezime, O.Br_Ind, Sif_Pred, Ocena)

S

BR_Ind	Ime	Prezime
2010234	Marko	Marković
2011345	Petar	Petrović
2010456	Marko	Petrović

O

BR_Ind	Sif_Pred	Ocena
2010234	BP	8
2010234	RM	9
2010456	BP	10

Student × **Ocena**

S×O

S.BR_Ind	IME	PREZIME	O.BR_Ind	SIF_PRED	OCENA
2010234	Marko	Marković	2010234	BP	8
2010234	Marko	Marković	2010234	RM	9
2010234	Marko	Marković	2010456	BP	10
2011345	Petar	Petrović	2010234	BP	8
2011345	Petar	Petrović	2010234	RM	9
2011345	Petar	Petrović	2010456	BP	10
2010456	Marko	Petrović	2010234	BP	8
2010456	Marko	Petrović	2010234	RM	9
2010456	Marko	Petrović	2010456	BP	10

7.7 SPAJANJE (><)

To je operacija relacione algebre koja iz dve polazne relacije formira novu sa n-torkama dobijenim u dva koraka:

- Svaka n-torka iz prve relacije redom se spaja sa svim n-torkama iz druge relacije
- Iz tako dobijenih n-torki izdvajaju se one koje zadovoljavaju zadati uslov P

Ako su **r** i **s** relacije nad šemom R(X) i S(Y), X i Y su skupovi atributa u šemama relacija, formalna definicija operacije spajanja je: $\mathbf{r} >_{P(XY)}< \mathbf{s} = \sigma_{P(XY)}(r \times s) = \{xy \mid x \in \mathbf{r} \wedge y \in \mathbf{s} \wedge P(xy)\}$

7.7.1 θ spajanje

Prethodna definicija dozvoljava proizvoljni uslov P, pod uslovom da je izračunljiv za svaku n-torku nakon Dekartovog proizvoda

Neka su **r** i **s** relacije nad šemom R(X) i S(Y). Neka su Xi i Yk atributi za koje važi da je $Xi \in X$ i $Yi \in Y$. Pod θ spajanjem $r >_{Xi\ \theta\ Yi}< s$ podrazumeva se spajanje kod koga operator θ označava bilo koji operator poređenja: (=,≤,<,>,≥,≠)

7.7.2 Ekvi spajanje

Prethodno θ spajanje ograničava formu uslova spajanja, međutim i dalje dobijeni rezultat nema praktičnu primenu. Specijalni slučaj gde θ predstavlja jednakost (=) je čest slučaj u praksi.

Ekvi spajanje po jednom atributu: $r >_{Xi = Yi}< s$

Ekvi spajanje po više atributa označava se sa: $r >_{(X1,X2,...,Xn) = (Y1,Y2,...,Yn)} < s$

7.7.3 Prirodno spajanje

Ekvi spajanje daje jedan suvišan atribut, zato što su vrednosti atributa po kojima se vrši spajanje uvek iste. Nepotrebni atribut se eliminiše dodatnom operacijom projekcije. Navedeni slučaj je čest u praksi, pa je uvedena specijalna operacija prirodnog spajanja. Podrazumeva sekvencu tri elementarne operacije

- Dekartov proizvod relacija
- Restrikciju po uslovu jednakosti atributa
- Projekcija po razlici unije svih atributa i skupa spojnih atributa iz bilo koje od relacija

Prirodno spajanje dve relacije po jednom atributu označava se sa:

$r > Xi,*,Yk < s$

Specijalni slučaj označavanja: r > * < s podrazumeva prirodno spajanje po svim atributima koji imaju jednake nazive u obe relacije.

Formalna definicija prirodnog spajanja: Ako su **r** i **s** relacije nad šemom R(X) i S(Y), a A⊆X i B⊆Y su uređeni podskupovi jednakog broja atributa relacija **r** i **s**, respektivno, prirodno spajanje definišemo kao:

$r > (A)*(B) < s = \pi_{XY-B}(\sigma_{(A)=(B)}(r \times s)) = t(XY-B)$

Jednakost uređenih podskupova A i B podrazumeva jednakost korespondentnih elemenata. Umesto XY-B sa desne strane može se navesti XY-A.

Primer 1.

Za polazne relacije r i s

A	B
α	1
β	2

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

kao rezultat prirodnog spajanja $r >*< s = \pi_{XY-B}(\sigma_{A=C}(r \times s))$ dobija se

A	B	D	E
α	1	10	a
β	2	10	a
β	2	20	b

Kao rezultat prirodnog spajanja za relacije **Student** i **Ocena** dobio bi se sledeći rezultat:

S×O	BR_Ind	IME	PREZIME	SIF_PRED	OCENA
	2010234	Marko	Marković	BP	8
	2010234	Marko	Marković	RM	9
	2010456	Marko	Petrović	BP	10

7.8 DELJENJE (/)

Deljenje je najsloženija operacija relacione algebre.

Deljenje se ne može izvesti sa proizvoljnim tabelama. Za A/B potrebno je da se svi atributi relacije B nalaze u relaciji A.

Npr: Moguće je deljenje za:

a (X1,X2,...,Xn,Y1,Y2,...,Ym)
b (Y1,Y2,...,Ym)

Primer: Za dve relacije r i s, date sa:

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

D	E
a	1
b	1

nakon operacije deljenja r/s dobija se:

A	B	C
α	a	γ
γ	a	γ

Rezultat je ovakav pošto jedino n-torke u relaciji **r** nad atributima A, B i C: (α, a, α) i (γ, a, γ) jesu u kombinaciji sa obe n-torke nad atributima D i E u relaciji **s**: (a, 1) i (b, 1).

8 SQL

U ovom poglavlju je predstavljen način upotrebe strukturnog upitnog jezika SQL (eng. *Structured Query Language*) za upravljanje relacionim bazama podataka, kroz konkretan i praktičan primer, postepenim uvođenjem novih jezičkih konstrukcija. SQL je deklarativan jezik. Postoje različite implementacije sistema za upravljanje bazama podataka (eng. *Database Management System*) kojima je moguće davati naredbe jednim standardizovanim SQL jezikom, dok svaka od implementacija za sebe definiše način na koji su interno implementirana izvršavanja pojedinačnih naredbi.

Iako je SQL standardizovan 1986. godine pod imenom X3.135, zvanično se smatra da je 1987. godine standard usvojen kada ga je prihvatila Međunarodna organizacija za standarde (ISO). Pre toga je postojalo više implementacija SQL jezika. Nakon što je SQL standardizovan, nastalo je nekoliko izvedenih i nadograđenih verzija, ali one sve implementiraju minimum zajedničkih karakteristika koje su propisane standardom. Prema tome, za potrebe svakodnevnog praktičnog rada, razlike između pojedinih implementacija su često zanemarljive i odnose se na vrlo konkretne elemente, mehanizme i komponente koje te različite implementacije sistema za upravljanje relacionim bazama podataka podržavaju i uvode, izvan standardom propisanih.

U ovom poglavlju je predstavljen način korišćenja SQL jezika na primeru MySQL/MariaDB sistema za upravljanje relacionim bazama podataka. MySQL i MariaDB su rešenja sistema za upravljanje relacionim bazama podataka koja su besplatna i dostupna su u obliku otvorenog koda softvera.

SQL jezik se sastoji iz tri osnovne grupe naredbi. To su:

1. Grupa naredbi za upravljanje podacima (eng. *Data Manipulation Language*),
2. Grupa naredbi za definisanje struktura (eng. *Data Definition Language*) i
3. Grupa naredbi za upravljanje sistemom baze (eng. *Data Control Language*).

Grupe naredbi za upravljanje podacima omogućavaju dopremanje i menjanje podatke koji se čuvaju u struktuiranom obliku.

Grupe naredbi za definisanje podataka omogućavaju pravljenje i menjanje strukture skladišta podataka u bazi.

Grupa naredbi za upravljanje sistemom baze omogućavaju upravljanje privilegijama korisnika sistema, kao i upravljanje podešavanjima samog sistema.

8.1 MYSQL SERVER

MySQL je sistem za upravljanje relacionim bazama podataka otvorenog koda. Objavljen je pod GNU Opštom Javnom licencom (eng. *GNU General Public License*). Za realizaciju praktičnog primera na kojem se zasniva uvod u SQL jezik u ovom poglavlju, koristi se MySQL sistem za upravljanje relacionim bazama podataka.

8.1.1 Postupak instalacije MySQL RDBMS

Postupak instalacije MySQL serverskog i klijentskog softvera može da se razlikuje u trenutku čitanja ove knjige u odnosu na postupak objašnjen u ovom poglavlju.

Zavisno od toga koji operativni sistem se koristi, potrebno je preuzeti adekvatnu instalacionu datoteku.

8.1.1.1 Instalacija MySQL na Windows operativnom sistemu

Potrebno je preuzeti instalacionu datoteku za MySQL Community Server. U trenutku pisanja ove knjige, Internet adresa na kojoj je bilo moguće preuzeti instalacionu datoteku je bila dev.mysql.com/downloads/mysql.

Nakon preuzimanja instalacione datoteke za MySQL Community Server, potrebno je pokrenuti instalaciju i ispratiti sve korake prikazane u prozorima instalacionog programa. Svi koraci treba da budu izvršeni primenom podešavanja i opcija koje su podrazumevano postavljene u prozorima svakog koraka instalacije. U koracima u kojima je potrebno da se upiše lozinka za root korisnika sistema, neophodno je uneti lozinku koja će biti korišćena od strane administratora nakon završetka instalacije.

8.1.1.2 Instalacija MySQL na GNU/Linux operativnom sistemu

S obzirom na to da postoji veliki broj distribucija GNU/Linux operativnih sistema, u ovom poglavlju je objašnjen postupak instalacije na distribucijama koje su popularne u trenutku pisanja knjige.

Instalacija na GNU/Linux sistemima baziranim na Debian distribuciji

Među ovim distribucijama su Debian, Ubuntu, Linux Mint operativni sistemi. Za uspešno instaliranje MySQL servera, potrebno je imati pristup korisniku koji ima pravo da izvrši naredbe sa administratorskim privilegijama na sistemu, tj. da poseduje sudoer prava. Podrazumeva se da je dostupna Internet konekcija.

U terminalu (konzoli sistema) potrebno je izvršiti sledeće naredbe:

```
sudo apt-get update
```

Prva naredba ažurira repozitorijum dostupnog softvera.

```
apt-get install mysql-server -y
```

Druga naredba vrši instalaciju MySQL servera preuzimanjem i instalacijom potrebnih datoteka sa repozitorijuma softvera.

```
sudo mysql_secure_installation
```

Treća naredba pokreće alat za automatsko podešavanje MySQL servera. Potrebno je ispratiti sve korake u interaktivnoj konzolnoj aplikaciji. Potrebno je posebno obratiti pažnju na deo za unos lozinke root korisnika MySQL servera. Neophodno je uneti lozinku koja će biti korišćena od strane administratora nakon završetka instalacije. Prilikom unosa lozinke, na terminalu se ne vide karakteri, niti zvezdice kao maska lozinke. Potrebno je precizno uneti lozinku, a zatim istu tu potvrditi. U slučaju greške, program će zatražiti da se korak ponovi.

```
systemctl enable mysql
```

Četvrta naredba aktivira MySQL servis i omogućava da se isti automatski pokrene prilikom sledećeg pokretanja operativnog sistema.

```
systemctl start mysql
```

Peta naredba vrši aktiviranje MySQL servera bez potrebe za ponovnim pokretanjem računara i operativnog sistema.

Instalacija na GNU/Linux sistemima baziranim na RedHat distribuciji

Među ovim distribucijama su RedHat, Fedora, CentOS operativni sistemi. Za uspešno instaliranje MySQL servera, potrebno je imati pristup korisniku koji ima pravo da izvrši naredbe sa administratorskim privilegijama na sistemu, tj. da poseduje sudoer prava. Podrazumeva se da je dostupna Internet konekcija.

U trenutku pisanja ove knjige, Internet adresa sa koje je bilo moguće preuzeti datoteku za instalaciju paketa softvera za YUM alat za upravljanje paketima je bila dev.mysql.com/downloads/repo/yum.

Sa te adrese je potrebno preuzeti adekvatnu datoteku za operativni sistem na kojem treba da bude instaliran softver.

Kada RPM je datoteka za instalaciju paketa softvera preuzeta, u terminalu (konzoli sistema) potrebno je izvršiti sledeće naredbe:

```
sudo rpm -ivh IME_PREUZETE_DATOTEKE.rpm
```

Prva naredba dodaje YUM repozitorijum sa kojeg se vrši setup MySQL servera.

```
sudo yum install mysql-server -y
```

Druga naredba vrši instalaciju MySQL servera preuzimanjem i instalacijom potrebnih datoteka sa repozitorijuma softvera.

```
systemctl enable mysql
```

Treća naredba aktivira MySQL servis i omogućava da se isti automatski pokrene prilikom sledećeg pokretanja operativnog sistema.

```
systemctl start mysql
```

Četvrta naredba vrši aktiviranje MySQL servera bez potrebe za ponovnim pokretanjem računara i operativnog sistema.

```
sudo mysql_secure_installation
```

Peta naredba pokreće alat za automatsko podešavanje MySQL servera. Potrebno je ispratiti sve korake u interaktivnoj konzolnoj aplikaciji. Potrebno je posebno obratiti pažnju na deo za unos lozinke root korisnika MySQL servera.

Neophodno je uneti lozinku koja će biti korišćena od strane administratora nakon završetka instalacije. Prilikom unosa lozinke, na terminalu se ne vide karakteri, niti zvezdice kao maska lozinke. Potrebno je precizno uneti lozinku, a zatim istu tu potvrditi.

U slučaju greške, program će zatražiti da se korak ponovi. Treba ispratiti korake prikazane u konzoli. U slučaju nepredviđene greške, pokušati ponovnu instalaciju ili praćenje dodatnih uputstava iz zvanične dokumentacije.

8.1.2 Interakcija sa serverom

Interakcija sa MySQL serverom je moguća na više načina. Postoje programi koji interakciju sa MySQL serverom zasnivaju na grafičkom korisničkom interfejsu i oni koji se oslanjaju na konzolnu tekstualnu interakciju.

Nakon uspešne instalacije MySQL servera na operativnom sistemu, potrebno je odabrati način na koji će se upravljati bazama podataka na serveru.

U ovom poglavlju se upravljanje MySQL serverom, zadavanje naredbi i dopremanje odgovora vrši korišćenjem alata za konzolnu interakciju sa MySQL serverom.

Glavna prednost korišćenja konzolnog interfejsa je ta što podstiče učenje SQL naredbi koje su najčešće u grafičkim alatima sakrivene za aktivnosti i operacija do kojih se dolazi korišćenjem dugmadi, tabelarnih prikaza, zaglavlja tabela, oznaka itd. Osim toga, postoji veliki broj alata koji se zasnivaju na upravljanju kroz grafički korisnički interfejs koji se u velikoj meri razlikuju i čiji interfejsi se često menjaju. To je drugi razlog zbog kojeg je odabran konzolni pristup, jer je SQL jezik standardizovan i ne menja se toliko često, što garantuje da će primeri iz ovog poglavlja knjige biti trajnije upotrebljivi za učenje SQL jezika i rada sa bazama podataka.

8.1.2.1 Konekcija na server na Windows operativnom sistemu

Nakon uspešne instalacije MySQL servera korišćenjem podrazumevanih opcija u svim koracima instalacionog procesa, u Start meniju treba pronaći prečicu "MySQL 8.0 Command Line Client - Unicode" i pokrenuti je. U prvom koraku je potrebno uneti lozinku root korisnika. Nakon unosa ispravne lozinke root korisnika MySQL servera otvara se MySQL konzola. Početak komandne linije počinje sa:

```
mysql>
```

8.1.2.2 Konekcija na server na GNU/Linux operativnom sistemu

Nakon uspešne instalacije MySQL servera, potrebno je otvoriti konzolu operativnog sistema i ukucati sledeću naredbu:

```
mysql -u root -p
```

Nakon izvršavanja ove narede, potrebno je ispratiti uneti lozinku root korisnika MySQL servera.

8.1.3 Konzola MySQL servera

U konzoli MySQL servera se unose SQL izrazi koji mogu da budu naredbe ili upiti. Naredbe su SQL izrazi koji ne vraćaju rezultat u vidu seta podataka, dok su upiti SQL izrazi koji vraćaju rezultate u vidu seta podataka.

Konzola obezbeđuje interfejs za interakciju sa MySQL serverom. U jednoj MySQL konzoli u jednom trenutku može da se izvršava samo jedna naredba. Dok se jedna naredba ne završi, sledeća ne može da počne. SQL izrazi se u konzolu unose tekstualno i svaki SQL izraz mora da se završi sa simbolom ; (tačka zarez).

Na kraju rada, potrebno je zatvoriti konzolu MySQL servera koja se vrši naredbom:

```
exit
```

8.2 SINTAKSA SQL JEZIKA

SQL jezik je standarizovan jezik. Iako postoje implementacije koje u određenim elementima odstupaju od standarda, većina naredbi za upravljanje podacima, postavljanje upita i za uređivanje struktura su zajedničke za sve implementacije.

8.2.1 Neosetljivost na velika i mala slova

SQL jezik nije osetljiv na velika i mala slova. Sledeća dva SQL upita su identični:

```
SELECT * FROM location WHERE name LIKE "%grad%";
select * from location where name like "%grad%";
```

Neosetljivost na velika i mala slova se odnosi na rezervisane reči SQL jezika. Zavisno od operativnog sistema i podešavanja MySQL servera, on može da bude osetljiv na velika i mala slova kada su u pitanju imena tabela, polja, indeksa itd. Na operativnim sistemima kao što su GNU/Linux distribucije, sledeće dve naredbe neće biti tretirane kao identične, dok na Windows operativnom sistemu najčešće hoće:

```
SELECT * from LOCATION where NAME like "%grad%";
select * from LOCATION WHERE NAME LIKE "%grad%";
```

Napomena: *Zbog čitljivosti SQL izraza, preporučeno je da se rezervisane reči SQL jezika pišu velikim, a imena baza, tabela, polja i indeksa malim slovima.*

8.2.2 Beline

U SQL jeziku, beline nemaju specijalno značenje. Jedan SQL izraz može da se prostire u više redova, a svaki red može da bude pomeren od leve margine sa jednom ili više različitih tipova belina. Beline između delova SQL izraza takođe nemaju uticaj na značenje, osim između rezervisanih reči, imena tabela, polja in indeksa, gde je neophodno da postoji barem jedna belina. Beline mogu da budu razmaci, tabulatori ili prelasci u novi red.

MySQL server će sledeća dva SQL izraza da tretira kao identične:

```
SELECT *
FROM location
WHERE name LIKE "%grad%";

SELECT * FROM location WHERE name LIKE "%grad%";
```

Prvi SQL izraz je pregledniji, jer su delovi SQL izraza jasno podeljeni u tri celine:

- Šta se selektuje?
- Odakle se selektuje?
- Koji je uslov za filtriranje?

Drugi SQL izraz je manje pregledan, ali je kompaktan, ali je svakako ispravan.

Za prikazane SQL izraze, MySQL server će vratiti identičan rezultat, tj. podatke o svim lokacijama čije ime sadrži tekst "grad" u sebi.

8.2.3 Komentari

Komentari se najčešće koriste kada se SQL izraz čuva u datoteci i kada je potrebno da se dodatno pojasni neki izraz.

SQL jezik podržava dva tipa komentara:

1. Komentari u jednom redu i
2. Komentari u više redova.

Komentari u jednom redu

Kada je potrebno napisati kraći komentar za koji je dovoljan samo jedan red teksta, može da se koristi tip komentara u jednom redu. Komentari u jednom redu se pišu tako što se otkucaju dva uzastopna simbola - (minus) praćena razmakom iza kojeg sledi tekst komentara. Sav tekst do kraja tog reda se smatra za komentar, a sav tekst pre dva uzastopna simbola - (minus) nije pod komentarom.

Primer SQL izraza opisanog komentarom u jednom redu je:

```
-- Ovaj upit vraća sve lokacije koje sadrže u imenu tekst "grad"
SELECT *
FROM location
WHERE
  name LIKE "%grad%";
```

Komentari u više redova

Kada je potrebno napisati detaljniji komentar za koji nije dovoljan samo jedan red teksta, može da se koristi tip komentara u više redova. Komentari u više redova se pišu tako što se blok komentara otvori kombinacijom karaktera /* (sleš i asterisk), nakon kojeg sledi tekst komentara koji može da se prostire u više redova, ali na kraju treba da se zatvori sa kombinacijom karaktera */ (asterisk i sleš). Sav tekst pre /* i posle */ nije deo komentara i tretira se kao naredba SQL izraza.

Primer SQL izraza opisanog komentarom u više redova je:

```
/* Ovaj upit vraća podatke iz tabele lokacija.
   Biće vraćeni podaci samo za zapise koji u polju
   pod imenom name sadrže tekst "grad". */
SELECT *
FROM location
WHERE
  name LIKE "%grad%";
```

Kombinovanje komentara

SQL jezik podržava kombinovanje komentara. Komentari mogu da se nalaze unutar SQL izraza. Oni ne remete izvršavanje SQL izraza.

Primer SQL izraza čiji delovi su opisani komentarima kombinovanjem komentara a jedan red i komentara za više redova je:

```
-- Sve lokacije koje sadrže "grad"
SELECT * -- Uzeti sve podatke o lokaciji
FROM location
WHERE
  name LIKE /* LIKE umesto RLIKE, jer se koriste regularni izrazi */ "%grad%";
```

8.3 TIPOVI PODATAKA

Svi podaci u relacionim bazama podataka imaju svoj tip, ime i vrednost. Standardni SQL jezik propisuje nekoliko osnovnih tipova podataka, dok pojedinačne implementacije uvode sopstvene izvedene, ali i nove osnovne tipove koji nisu propisani standardnim SQL jezikom.

Osnovni tipovi podataka koje se koriste u MySQL implementaciji SQL jezika su:

- Tekstualni,
- Numerički i
- Vremenski.

8.3.1 Tekstualni tipovi podataka

Tekstilni tipovi podataka se koriste za čuvanje tekstualnih vrednosti, kao što su imena, opisi, oznake, brojevi telefona, adrese elektronske pošte itd. MySQL podržava nekoliko specifičnih tekstualnih tipova podataka koji se razlikuju prema mehanizmu čuvanja vrednosti, kodiranju tekstualnog zapisa i prema najvećoj dužini tekstualne vrednosti koja može da se čuva u poljima određenog tipa. Ovi tipovi podataka se dele na kodirane i nekodirane.

Kodirani tekstualni tipovi podataka se koriste za čuvanje tekstualnih vrednosti, gde tekst nije kodiran ASCII kodnom mapom, već Unicode kodnom mapom promenljive širine. Najčešće se dužina vrednosti ovih tipova podataka izražava u broju karaktera.

Nekodirani ili **binarni** tipovi podataka se koriste za čuvanje binarnih zapisa, kao što su sadržaji datoteka ili binarnih serijalizacija objekata iz programa. Najčešće se dužina vrednosti ovih tipova podataka izražava u broju bajtova.

Specifični kodirani tekstualni tipovi podataka su:

- CHAR — Dužina je ograničena na 255 karaktera.
- VARCHAR — Dužina je ograničena na 65.535 karaktera.
- TEXT — Dužina je ograničena na 65.535 karaktera.
- TINYTEXT — Dužina je ograničena na 256 karaktera.
- MEDIUMTEXT — Dužina je ograničena na 16.777.215 karaktera.
- LONGTEXT — Dužina je ograničena na 4.294.967.295 karaktera.

Specifični binarni tipovi podataka su:

- BINARY — Dužina zapisa je ograničena na 255 bajtova.
- VARBINARY — Dužina zapisa je ograničena na 65.535 bajtova.
- BLOB — Dužina zapisa je ograničena na 65.535 bajtova.
- MEDIUMBLOB — Dužina zapisa je ograničena na 16.777.215 bajtova.
- LONGBLOB — Dužina zapisa je ograničena na 4.294.967.295 bajtova.

Pored kodiranih tekstualnih i binarnih tipova podataka, postoje tipovi podataka ENUM i SET. Oni omogućavaju da se unapred definisane tekstualne vrednosti čuvaju u poljima tih tipova uz manju potrošnju prostora.

- ENUM — Najviše 65.535 jedinstvenih vrednosti. Polje može da sadrži samo jednu od vrednosti.
- SET — Najviše 64 jedinstvene vrednosti. Polje može da sadrži najviše 64 jedinstvene vrednosti.

8.3.2 Numerički tipovi vrednosti

Numerički tipovi podataka se koriste za čuvanje numeričkih vrednosti, kao što su količina, cena, visina itd. MySQL podržava nekoliko specifičnih numeričkih tipova podataka koji se razlikuju prema tome da li čuvaju cele ili realne vrednosti. Numerički tipovi podataka se dele na celobrojne, realne. Realni se dodatno dele na realne sa fiksnom i na realne sa promenljivom pozicijom decimalne tačke.

Numerički tipovi podataka mogu da budu označeni i neoznačeni. Označeni numerički tipovi podržavaju negativne vrednosti, dok neoznačeni podržavaju samo pozitivne vrednosti i nulu.

Označeni celobrojni numerički tipovi podataka su:

- INT — Opseg od -2147483648 do 2147483647.
- TINYINT — Opseg od -128 do 127.
- SMALLINT — Opseg od -32768 do 32767.
- MEDIUMINT — Opseg od -8388608 do 8388607.
- BIGINT — Od -9223372036854775808 do 9223372036854775807.

Neoznačeni celobrojni numerički tipovi podataka su:

- INT — Opseg od 0 do 4.294.967.295.
- TINYINT — Opseg od 0 do 255.
- SMALLINT — Opseg od 0 do 65.535.
- MEDIUMINT — Opseg od 0 do 16.777.215.
- BIGINT — Opseg od 0 do 18.446.744.073.709.551.615.

Označeni realni numerički tipovi sa fiksnom decimalnom tačkom su:

- DECIMAL(C, D) — Opseg od $-10^{C-D-1} - \frac{1}{10^D}$ do $10^{C-D} - \frac{1}{10^D}$.
- NUMERIC — Identična implementacija kao DECIMAL tip.

Neoznačeni realni numerički tipovi sa fiksnom decimalnom tačkom su:

- DECIMAL(C, D) Opseg od 0 do $10^{C-D} - \frac{1}{10^D}$.
- NUMERIC Identična implementacija kao DECIMAL tip.

Pored navedenih tipova podataka, postoje realni numerički tipovi sa promenljivom decimalnom FLOAT i DOUBLE čije vrednosti su u opsegu od $-10^{C-D-1} - \frac{1}{10^D}$ do $10^{C-D} - \frac{1}{10^D}$, s tim da se u memoriji čuva približan broj, jer je memorijski prostor određen za čuvanje FLOAT i DOUBLE vrednosti ograničen, a granične vrednosti su:

- FLOAT Opseg od $-3{,}4 * 10^{38}$ do $3{,}4 * 10^{38}$.
- DOUBLE Opseg od $-1{,}8 * 10^{308}$ do $1{,}8 * 10^{308}$.

8.3.3 Vremenski tipovi podataka

Vremenski tipovi se koriste za čuvanje datuma, vremena, kombinacije datuma i vremena, intervala itd. MySQL podržava nekoliko specifičnih vremenskih tipova podataka, koji se razlikuju prema mehanizmu čuvanja u memoriji i prema nameni.

Vremenski tipovi podataka se dele na:

- YEAR Opseg od 1901 do 2155. godine. Može da bude i 0000.
- DATETIME Opseg od 1. 1. 1000. u 00.00.01 do 19. 1. 2038. u 03.14.08.
- DATE Opseg datuma od 1. 1. 1000. do 31. 12. 9999. godine.
- TIME Opseg vremena od 00.00.00 do 23.59.59.
- TIMESTAMP Opseg od 1. 1. 1000. u 00.00.01 do 19. 1. 2038. u 03.14.08.

Tipovi podataka DATETIME i TIMESTAMP podržavaju zapis do milionitog delova sekunde, od 0,000001 do 0,999999 sekundi.

8.3.4 Specijalni tipovi podataka

MySQL podržava specijalne tipove podataka pored standardnih koji su opisani. Među njima su prostorni tipovi, koji nalaze primenu u određenim oblastima, kao što su geografski informacioni sistemi i drugi sistemi koji rade sa prostornim podacima. Osim prostornih tipova podataka, MySQL podržava JSON tipove podataka. JSON tip je specifičan, jer u određenoj meri narušava principe relacionih baza podataka, jer obezbeđuje mehanizam za čuvanje složenih podataka proizvoljne strukture u poljima tabele. JSON podaci se u MySQL bazama podataka beleže kao tekstualni, ali posebne jezičke konstrukcije omogućavaju korišćenje i referenciranje struktura podataka smeštenih unutar polja koje sadrži JSON vrednost.

U ovoj knjizi ove dve grupe tipova podataka nisu posebno obrađene zbog njihove specifične namene, kao i zbog toga što nisu u pitanju standardni tipovi podataka predviđeni SQL jezikom.

8.3.5 Odabir tipa podatka

U poglavlju posvećenom modelovanju podataka, objašnjeno je preslikavanje entiteta. U tom postupku, jedan od koraka je određivanje spiska polja kojima je opisan entitet. Prilikom formiranja tabela koje u relacionom modelu predstavljaju te entitete, pored imenâ njihovih polja, potrebno je odrediti njihove tipove podataka.

Odabir adekvatnog tipa podatka je važan, jer u slučaju prvobitnog pogrešnog odabira tipa, u slučaju kasnije potrebe za promenom tipa, neophodno je izvršiti proces migracije.

Migracija ima za cilj da se struktura podataka transformiše na taj način da se podaci ne izgube, a da se otvori novi prostor za proširenje opsega vrednosti koje to polje u budućnosti može da čuva.

Primeri odabira ispravnog tipa podatka:

- Ime osobe može da bude VARCHAR tip podatka ograničene dužine, ali ne može da bude DECIMAL(10,2) ili INT;
- Cena artikla može da bude DECIMAL(10,4) ako je potrebno evidentirati cene koje su realne vrednosti, ali ako su cene uvek celobrojne, dovoljno je koristiti tip podatka INT;
- ID broj kartice može da bude INT UNSIGNED ako se obim vrednosti broja ID kartice uklapa u okvire najvećeg podržanog broja neoznačene celobrojne numeričke vrednosti tipa INT. Međutim, ako ID broj kartice sadrži specijalne karaktere ili, eventualno, slovne karaktere, onda je bolje koristiti CHAR ili VARCHAR potrebne dužine;
- Kada je potrebno evidentirati status paketa ili pošiljke, moguće je koristiti tip podatka TINYINT UNSIGNED dužine 1 u kojem je pomoću vrednosti 1 ili 0 moguće ukazati na to da li je pošiljka poslata ili ne. Međutim, bolja opcija je definisati ENUM tip sa podržanim vrednostima koje mogu da ukažu na više koraka u postupku dostave paketa ili pošiljke, npr. pending, sent, lost, returned, delivered itd.

8.4 SQL KROZ PRAKTIČAN PRIMER

U ovoj knjizi je upoznavanje za SQL jezikom zasnovano na učenju kroz praktičan primer. Primer koji se koristi obuhvata sve korake, počev od rastavljanja projektnog teksta sa poslovnom logikom, preko određivanja entiteta i njihovih atributa i prevođenja istih u model baze podataka, do definisanja struktura podataka korišćenjem SQL naredbi. Nakon pravljenja tabela, sa svim neophodnim relacijama, indeksima i ograničenjima, korišćenjem SQL naredbi je pokazano kako se upravlja podacima korišćenjem SQL jezika.

8.4.1 Analiza i dekompozicija poslovnog procesa

Sastavni deo rada sa bazama podataka je njihovo modelovanje. Prilikom definisanja modela baze podataka, najčešće se oslanjamo na postojeću poslovnu logiku za koju treba implementirati bazu podataka koja treba da čuva poslovne podatke i podatke o poslovnim transakcijama. U takvim situacijama, ne postoji uvek potpuna sloboda za samostalno formiranje modela prema sopstvenim zahtevima, već je neophodno obaviti dekompoziciju teksta projektnog zahteva i na osnovu njega formirati model baze podataka. Tekst projektnog zahteva sadrži sažeti pregled poslovne logike za koju treba formirati adekvatan model baze podataka. Projektni zahtev na kojem se zasnivaju primeri kroz koje se uvode jezičke konstrukcije SQL jezika je dat u nastavku.

Tekst projektnog zahteva

```
Kompanija ima veći broj lokacija na različitim adresama sa kojih se vrši prodaja
artikala. Svaki artikal poseduje jedinstveno ime, kao i kraći i duži opis artikla.
Jedan artikal može da bude razvrstan u više od jedne kategorije. Postoji više
kategorija koje imaju jedinstveno ime. Svakom artiklu može da bude pridružena jedna
ili više osobina sa pripadajućom vrednošću. Osobine se razlikuju po tipu osobine i
imaju jedinstveno ime. Iznos cena artikala može da se menja. Potrebno je da se vodi
evidencija o istorijskim promenama cena artikala. Kompanija nabavlja artikle od
različitih dobavljača. Svaki dobavljač ima jedinstveni naziv, adresu, jedinstveni
telefon i jedinstvenu adresu elektronske pošte. Prilikom evidencije nabavke, beleži
se količina nabavljenih artikala, identifikacioni broj transakcije koji mora da bude
jedinstven, kao i lokacija na koju se artikli dostavljaju. Klijentima kompanije se
izdaju računi za prodate artikle. Za svakog klijenta se beleži ime, prezime,
jedinstvena adresa elektronske pošte, kao i adresa za dostavu artikala.
Prilikom evidencije prodaje klijentu, formira se račun za koji se vezuje jedan ili
više artikala, kao i količina prodatih artikala. Za jedan račun može da se evidentira
više plaćanja različitih iznosa. Kompanija podržava više vrsta plaćanja. Svaka vrsta
plaćanja ima svoj jedinstveni naziv. Kada je račun u potpunosti isplaćen, kompanija
može klijentu da isporuči artikle vezane za određeni račun. Svaka isporuka se
evidentira i ima svoj jedinstveni broj pošiljke i status pošiljke. Status isporuke
može da ukazuje na to da se čeka na isporuku, da je pošiljka poslata, da je vraćena,
da je izgubljena ili da je dostavljena. Kompanija može da vrši transfer artikala sa
jedne lokacije na drugu. Prilikom transfera, beleži se količina prenetih artikala.
```

Postupak dekompozicije

Koraci u dekompoziciji teksta projektnog zahteva su:

- Identifikovanje entiteta
- Identifikovanje izričitih atributa entitetâ
- Identifikovanje podrazumevanih atributa entitetâ
- Identifikovanje relacija između entiteta

8.4.2 Identifikovanje entiteta

Iz teksta projektnog zahteva, moguće je identifikovati entitete sa kojima aplikacija za koju treba modelovati bazu podataka treba da radi.

U nastavku je ponovljen tekst projektnog zahteva sa imenima entiteta istaknutim velikim slovima.

Kompanija ima veći broj **LOKACIJA** na različitim adresama sa kojih se vrši prodaja **ARTIKALA**. Svaki **ARTIKAL** poseduje jedinstveno ime, kao i kraći i duži opis artikla. Jedan **ARTIKAL** može da bude razvrstan u više od jedne **KATEGORIJE**. Postoji više **KATEGORIJA** koje imaju jedinstveno ime. Svakom **ARTILKU** može da bude pridružena jedna ili više **OSOBINA** sa pripadajućom vrednošću. **OSOBINE** se razlikuju po **TIPU OSOBINE** i imaju jedinstveno ime. Iznos **CENE ARTIKLA** može da se menja. Potrebno je da se vodi evidencija o istorijskim promenama **CENA ARTIKALA**. Kompanija nabavlja **ARTIKLE** od različitih **DOBAVLJAČA**. Svaki **DOBALJAČ** ima jedinstveni naziv, adresu, jedinstveni telefon i jedinstvenu adresu elektronske pošte. Prilikom evidencije **NABAVKE**, beleži se količina nabavljenih **ARTIKALA**, identifikacioni broj transakcije koji mora da bude jedinstven, kao i **LOKACIJA** na koju se **ARTIKLI** dostavljaju. **KLIJENTIMA** kompanije se izdaju **RAČUNI** za prodate **ARTIKLE**. Za svakog **KLIJENTA** se beleži ime, prezime, jedinstvena adresa elektronske pošte, kao i adresa za dostavu artikala.
Prilikom evidencije prodaje **KLIJENTU**, formira se **RAČUN** za koji se vezuje jedan ili više **ARTIKALA**, kao i količina prodatih artikala. Za jedan **RAČUN** može da se evidentira više **PLAĆANJA** različitih iznosa. Kompanija podržava više **VRSTA PLAĆANJA**. Svaka **VRSTA PLAĆANJA** ima svoj jedinstveni naziv. Kada je **RAČUN** u potpunosti isplaćen, kompanija može **KLIJENTU** da isporuči **ARTIKLE** vezane za određeni **RAČUN**. Svaka **ISPORUKA** se evidentira i ima svoj jedinstveni broj pošiljke i status pošiljke. Status isporuke može da ukazuje na to da se čeka na isporuku, da je pošiljka poslata, da je vraćena, da je izgubljena ili da je dostavljena. Kompanija može da vrši **TRANSFER ARTIKALA** sa jedne **LOKACIJE** na drugu. Prilikom **TRANSFERA**, beleži se količina prenetih **ARTIKALA**.

Kada se svi obeleženi entiteti svedu u nominativ jednine imenice i prikažu u vidu liste, vidi se da je identifikovano 14 izričito navedenih entiteta u tekstu projektnog zahteva. Ti entiteti su:

- LOKACIJA
- ARTIKAL
- KATEGORIJA
- OSOBINA
- TIP OSOBINE
- CENA
- DOBAVLJAČ
- NABAVKA
- KLIJENT
- RAČUN
- PLAĆANJE
- VRSTA PLAĆANJA
- ISPORUKA
- TRANSFER

Ovi entiteti će u modelu baze podataka biti predstavljeni u vidu tabela, između kojih će biti napravljene relacije koja u jednom od narednih koraka treba da budu identifikovanje.

8.4.3 Identifikovanje izričitih atributa entitetâ

Identifikovanje izričitih atributa entitetâ se vrši isključivo na osnovu teksta projektnog zahteva i predstavlja postupak proširenja informacija o postojećim entitetima. U ovom postupku se navode samo oni atributi koji su izričiti navedeni u projektnoj dokumentaciji i koji mogu da se izvedu iz teksta projektnog zahteva.

U nastavku je ponovljen tekst projektnog zahteva iz prethodnog koraka, sa imenima atributa istaknutim podvučenim slovima.

```
Kompanija ima veći broj LOKACIJA na različitim adresama sa kojih se vrši prodaja
ARTIKALA. Svaki ARTIKAL poseduje jedinstveno ime, kao i kraći i duži opis artikla.
Jedan ARTIKAL može da bude razvrstan u više od jedne KATEGORIJE. Postoji više
KATEGORIJA koje imaju jedinstveno ime. Svakom ARTILKU može da bude pridružena jedna
ili više OSOBINA sa pripadajućom vrednošću. OSOBINE se razlikuju po TIPU OSOBINE i
imaju jedinstveno ime. Iznos CENA ARTIKLA može da se menja. Potrebno je da se vodi
evidencija o istorijskim promenama CENA ARTIKALA. Kompanija nabavlja ARTIKLE od
različitih DOBAVLJAČA. Svaki DOBALJAČ ima jedinstveni naziv, adresu, jedinstveni
telefon i jedinstvenu adresu elektronske pošte. Prilikom evidencije NABAVKE, beleži
se količina nabavljenih ARTIKALA, identifikacioni broj transakcije koji mora da bude
jedinstven, kao i LOKACIJA na koju se ARTIKLI dostavljaju. KLIJENTIMA kompanije se
izdaju RAČUNI za prodate ARTIKLE. Za svakog KLIJENTA se beleži ime, prezime,
jedinstvena adresa elektronske pošte, kao i adresa za dostavu artikala. Prilikom
evidencije prodaje KLIJENTU, formira se RAČUN za koji se vezuje jedan ili više
ARTIKALA, kao i količina prodatih artikala. Za jedan RAČUN može da se evidentira više
PLAĆANJA različitih iznosa. Kompanija podržava više VRSTA PLAĆANJA. Svaka VRSTA
PLAĆANJA ima svoj jedinstveni naziv. Kada je RAČUN u potpunosti isplaćen, kompanija
može KLIJENTU da isporuči ARTIKLE vezane za određeni RAČUN. Svaka ISPORUKA se
evidentira i ima svoj jedinstveni broj pošiljke i status isporuke. Status isporuke
može da ukazuje na to da se čeka na isporuku, da je pošiljka poslata, da je vraćena,
da je izgubljena ili da je dostavljena. Kompanija može da vrši TRANSFER ARTIKALA sa
jedne LOKACIJE na drugu. Prilikom TRANSFERA, beleži se količina prenetih ARTIKALA.
```

Kada su svi obeleženi atributi svedeni u nominativ jednine i kada su pridruženi listi identifikovanih entiteta, nastaje sledeći spisak:

- LOKACIJA (**adresa**)
- KATEGORIJA (**ime**)
- OSOBINA (**vrednost**)
- TIP OSOBINE (**ime**)
- CENA (**iznos**)
- RAČUN
- PLAĆANJE (**iznos**)
- VRSTA PLAĆANJA (**naziv**)
- TRANSFER (**količina**)
- ARTIKAL (**ime**, **kraći opis**, **duži opis**)
- DOBAVLJAČ (**naziv**, **adresa**, **telefon**, **adresa elektronske pošte**)
- NABAVKA (**količina**, **identifikacioni broj transakcije**)
- KLIJENT (**ime**, **prezime**, **adresa elektronske pošte, adresa**)
- ISPORUKA (**jedinstveni broj pošiljke**, **status**)

Identifikovanje podrazumevanih atributa entitetâ

Identifikovanje podrazumevanih atributa entitetâ se vrši dopunom prethodno sačinjene liste, tako što se dodaju atributi koji su uobičajeno potrebni za čuvanje informacija o zapisima u tabelama baze podataka.

Uobičajeni podrazumevani atributi su:

- Atribut za jedinstveni identifikacioni broj zapisa;
- Atribut za datum i vreme pravljenja zapisa;
- Atribut za indikaciju statusa aktivnosti i vidljivosti zapisa, itd.

Kada se postojećem spisku entiteta i njihovih atributa dodaju uobičajene podrazumevane atribute, nastaje sledeći spisak:

- LOKACIJA (adresa, **vreme pravljenja**, **ID**)
- ARTIKAL (ime, kraći opis, duži opis, **vreme pravljenja**, **ID**)
- KATEGORIJA (ime, **vreme pravljenja**, **ID**)
- OSOBINA (vrednost, **vreme pravljenja**, **ID**)
- TIP OSOBINE (ime, **vreme pravljenja**, **ID**)
- CENA (iznos, **vreme pravljenja**, **ID**)
- DOBAVLJAČ (naziv, adresa, tel, e-pošta, **vreme pravljenja**, **ID, aktivan**)
- NABAVKA (količina, identifikacioni broj transakcije, **vreme pravljenja**, **ID**)
- KLIJENT (ime, prezime, e-pošta, adresa, **vreme pravljenja**, **ID, aktivan**)
- RAČUN (**vreme pravljenja**, **ID**)
- PLAĆANJE (iznos, **vreme pravljenja**, **ID**)
- VRSTA PLAĆANJA (naziv, **vreme pravljenja**, **ID, aktivna**)
- ISPORUKA (jedinstveni broj pošiljke, status, **vreme pravljenja**, **ID**)
- TRANSFER (količina, **vreme pravljenja**, **ID**)

8.4.4 Identifikovanje relacija između entiteta

Tekst projektnog zahteva najčešće otkriva odnos među entitetima. Ovaj odnos može da bude definisan izričito ili implicitno. Izričito navedene relacije mogu da se direktno prevedu u relacije u relacionom modelu baze podataka. Implicitne relacije treba prvo da se utvrde, imenuju, a zatim da se prevedu u relacije u relacionom modelu baze.

U ovom koraku su prikazane identifikovane relacije koje postoje među entitetima i za svaku relaciju su prikazane rečenice iz teksta projektnog zahteva koje ukazuju na relaciju i na vrstu relacije, sa delovima koji ukazuju na relaciju.

Relacije koje su utvrđene implicitno su objašnjene u komentaru ispod relacija.

ARTIKAL i KATEGORIJA - više prema više

Između entiteta ARTIKAL i KATEGORIJA postoji relacija više prema više (M:N) što je zaključeno na osnovu rečenice:

```
Jedan artikal može da bude razvrstan u više od jedne kategorije.
```

Relacija M:N se realizuje pravljenjem vezne tabele sa relacijama jedan prema više ka tabelama koje povezuje. To su ARTIKAL i KATEGORIJA. Sadrži podatke o tome koje zapise vezuje, kada su povezani, kao i jedinstveni identifikacioni broj te veze.

Na osnovu ovoga, nastaje novi entitet sa atributima:

- ARTIKAL_KATEGORIJA (ID artikla, ID kategorije, vreme pravljenja, ID)

ARTIKAL i OSOBINA - više prema više

Između entiteta ARTIKAL i OSOBINA postoji relacija više prema više (M:N) što je zaključeno na osnovu rečenice:

```
Svakom artiklu može da bude pridružena jedna ili više osobina sa pripadajućom vredn…
```

Relacija M:N se realizuje pravljenjem vezne tabele sa relacijama jedan prema više ka tabelama koje povezuje. U ovom slučaju, to su tabele za entitete ARTIKAL i OSOBINA, a tabela, pored informacija o tome koje zapise tih entiteta povezuje, kog datuma i vremena je relacija napravljena, kao i sa kojim jedinstvenim identifikacionim brojem veznog zapisa se identifikuje, sadrži vrednost koja pojašnjava karakteristiku konkretne osobine u odnosu na artikal sa kojim je povezana.

Na osnovu ovoga, nastaje novi entitet sa atributima:

- ARTIKAL_OSOBINA (ID artikla, ID osobine, vrednost, vreme pravljenja, ID)

ARTIKAL i RAČUN - više prema više

Između entiteta ARTIKAL i RAČUN postoji relacija više prema više (M:N) što je zaključeno na osnovu rečenice:

```
Prilikom evidencije prodaje klijentu, formira se račun za koji se vezuje jedan ili
više artikala, kao i količina prodatih artikala.
```

Relacija M:N se realizuje pravljenjem vezne tabele sa relacijama jedan prema više ka tabelama koje povezuje. U ovom slučaju, to su ARTIKAL i RAČUN, a tabela, pored podataka o tome koje zapise entitetâ povezuje, kog datuma i vremena je relacija napravljena, kao i sa kojim jedinstvenim identifikacionim brojem veznog zapisa se identifikuje, sadrži podatak o tome koja količina artikla je povezana sa računom.

Na osnovu ovoga, nastaje novi entitet sa atributima:

- RAČUN_ARTIKAL (ID artikla, ID računa, količina, vreme pravljenja, ID)

OSOBINA i TIP OSOBINE - jedan prema više

Između entiteta OSOBINA i TIP OSOBINE postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Osobine se razlikuju po tipu osobine i imaju jedinstveno ime.
```

Na osnovu ovoga, osobina može biti jednog tipa osobine, a da jedan tip može imati više osobina koje mu pripadaju. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli OSOBINA referenciranjem primarnog ključa tabele TIP OSOBINA.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- OSOBINA (vrednost, vreme pravljenja, ID, **ID tipa osobine**)

CENA i ARTIKAL - jedan prema više

Između entiteta CENA i ARTIKAL postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenica:

```
Iznos cena artikala može da se menja.
Potrebno je da se vodi evidencija o istorijskim promenama cena artikala.
```

Na osnovu ovoga, vidimo da cena može pripadati samo jednom artiklu, ali da jedan artikal može imati više cena koje se menjaju. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli CENA referenciranjem primarnog ključa tabele ARTIKAL.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- CENA (iznos, vreme pravljenja, ID, **ID artikla**)

TRANSFER i LOKACIJA - dvostruka veza jedan prema više

Između entiteta TRANSFER i LOKACIJA postoji dvostruka relacija jedan prema više (1:N) što je zaključeno na osnovu rečenica:

```
Kompanija može da vrši transfer artikala sa jedne lokacije na drugu.
Prilikom transfera, beleži se količina prenetih artikala.
```

Na osnovu ovoga, jasno je da transfer može da se vrši između dve lokacije i da sa jedne lokacije može da bude obavljeno više transfera. S obzirom na to da i izlazna i ulazna lokacija predstavljaju relaciju ka istom entitetu LOKACIJA, potrebno je

razlikovati dve veze. Ovo je moguće ostvariti stvaranjem dve relacije 1:N koje se realizuju pravljenjem dva strana ključa u tabeli TRANSFER i referenciranjem primarnog ključa tabele LOKACIJA u obe relacije. Imenovanjem atributa moguće je definisati koja relacija je za izlaznu, a koja za ulaznu lokaciju transfera. Ovaj postupak predstavlja objedinjeno pravljenje dve relacije jedan prema više u jednom koraku.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- TRANSFER (količina, vreme pravljenja, ID, **ID lokacije ulaza**, **ID lokacije izlaza**)

TRANSFER i ARTIKAL - jedan prema više

Između entiteta TRANSFER i ARTIKAL postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenica:

```
Kompanija može da vrši transfer artikala sa jedne lokacije na drugu.
Prilikom transfera, beleži se količina prenetih artikala.
```

Na osnovu ovoga, transfer može da se obavlja samo za jedan artikal, ali da jedan artikal može da biti prenet više puta. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli TRANSFER referenciranjem primarnog ključa tabele ARTIKAL. Pored stranog ključa, treba dopuniti entitet atributom za evidentiranje količine.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- TRANSFER (količina, vreme pravljenja, ID, ID lokacije ulaza, ID lokacije izlaza, **ID artikla, količina**)

NABAVKA i DOBALJAČ - jedan prema više

Između entiteta NABAVKA i DOBALJAČ postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Kompanija nabavlja artikle od različitih dobavljača.
```

Na osnovu ovoga, jasno je da jedna nabavka može da stigne samo od jednog dobavljača, ali da od jednog dobavljača može stići više nabavki. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli NABAVKA referenciranjem primarnog ključa tabele DOBAVLJAČ.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- NABAVKA (količina, identifikacioni broj transakcije, vreme pravljenja, ID, **ID dobavljača**)

NABAVKA i LOKACIJA - jedan prema više

Između entiteta NABAVKA i LOKACIJA postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Prilikom evidencije nabavke, beleži se količina nabavljenih artikala, identifikacioni
broj transakcije koji mora da bude jedinstven, kao i lokacija na koju se artikli
dostavljaju.
```

Na osnovu ovoga, jasno je da jedna nabavka može da stigne samo na jednu lokaciju, ali da na jednu lokaciju može da stigne više nabavki. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli NABAVKA referenciranjem primarnog ključa tabele LOKACIJA.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- NABAVKA (količina, identifikacioni broj transakcije, vreme pravljenja, ID, ID dobavljača, **ID lokacije**)

NABAVKA i ARTIKAL - jedan prema više

Između entiteta NABAVKA i ARTIKAL postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Prilikom evidencije nabavke, beleži se količina nabavljenih artikala, identifikacioni
broj transakcije koji mora da bude jedinstven, kao i lokacija na koju se artikli
dostavljaju.
```

Na osnovu ovoga, jasno je da jedna nabavka može da obuhvati samo na jedan artikal, ali da jedan artikal može da bude nabavljen kroz više nabavki. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli NABAVKA referenciranjem primarnog ključa tabele ARTIKAL. Pored stranog ključa, potrebno je da se dopuni entitet i atributom za evidentiranje podatka o nabavljenoj količini artikla.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- NABAVKA (**količina**, identifikacioni broj transakcije, vreme pravljenja, ID, ID dobavljača, ID lokacije, **ID artikla**)

RAČUN i KLIJENT - jedan prema više

Između entiteta RAČUN i KLIJENT postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Klijentima kompanije se izdaju računi za prodate artikle.
```

Na osnovu ovoga, jedan račun može da bude izdat samo jednom klijentu, ali za jednog klijenta može da bude izdato više računa. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli RAČUN referenciranjem primarnog ključa tabele KLIJENT.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- RAČUN (vreme pravljenja, ID, **ID klijenta**)

PLAĆANJE i RAČUN - jedan prema više

Između entiteta PLAĆANJE i RAČUN postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Za jedan račun može da se evidentira više plaćanja različitih iznosa.
```

Na osnovu ovoga, jasno je da jedno plaćanje može da bude vezano samo za jedan račun, a za jedan račun može da bude izvršeno više plaćanja. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli PLAĆANJE referenciranjem primarnog ključa tabele RAČUN. Pored stranog ključa, potrebno je da se dopuni entitet i atributom za evidentiranje podatka o iznosu izvršenog plaćanja po računu.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- PLAĆANJE (iznos, vreme pravljenja, ID, **ID računa, iznos**)

PLAĆANJE i VRSTA PLAĆANJA - jedan prema više

Između entiteta PLAĆANJE i VRSTA PLAĆANJA postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Kompanija podržava više vrsta plaćanja.
```

Ova relacija nije izričita na osnovu teksta, ali je implicitna, jer vrste plaćanja koje kompanija podržava moraju da budu povezane na neki način sa konkretnim plaćanjima, te se vidi da je ovde potrebna relacija jedan prema više između plaćanja i vrste plaćanja. Na osnovu ovoga, jedno plaćanje može da pripada jednoj vrsti plaćanja, a jedna vrsta plaćanja može da ima više plaćanja te vrste. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli PLAĆANJE referenciranjem primarnog ključa tabele VRSTA PLAĆANJA.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- PLAĆANJE (iznos, vreme pravljenja, ID, ID računa, iznos, **ID vrste plaćanja**)

ISPORUKA i RAČUN - jedan prema više

Između entiteta ISPORUKA i RAČUN postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Kada je račun u potpunosti isplaćen, kompanija može klijentu da isporuči artikle
vezane za određeni račun.
```

Na osnovu ovoga, jedna isporuka može da pripada samo jednom računu, a za jedan račun može da bude izvršeno više isporuka. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli ISPORUKA referenciranjem primarnog ključa tabele RAČUN.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- ISPORUKA (jedinstveni broj pošiljke, status, vreme pravljenja, ID, ID računa)

ISPORUKA i KLIJENT - jedan prema više

Između entiteta ISPORUKA i KLIJENT postoji relacija jedan prema više (1:N) što je zaključeno na osnovu rečenice:

```
Kada je račun u potpunosti isplaćen, kompanija može klijentu da isporuči artikle
vezane za određeni račun.
```

Na osnovu ovoga, jedna isporuka može da bude izvršena samo za jednog klijenta, a jednom klijentu može da bude izvršeno više isporuka. Relacija 1:N se realizuje pravljenjem stranog ključa u tabeli ISPORUKA referenciranjem primarnog ključa tabele KLIJENT.

Na osnovu ovoga, postojeći entitete i sa atributima se dopunjuje na sledeći način:

- ISPORUKA (jedinstveni broj pošiljke, status, vreme pravljenja, ID, ID računa, ID klijenta)

Nakon dopune modela baze podataka dodavanjem novih identifikovanih relacija i podataka, potrebno je da se izvrše sledeći koraci:

1. Objedinjavanje entiteta i svih njihovih atributa;

2. Da se imena entiteta, atributa i relacija prilagode konvenciji imenovanja.

8.4.5 Objedinjavanje entiteta i atributa

Nakon što je obavljen postupak dekompozicije, koji je uključio identifikovanje entiteta, atributa i relacija, potrebno je sačiniti objedinjeni spisak ažurnih entiteta sa njihovim atributima. Tako objedinjeni spisak, za konkretan primer projektnog zadatka koji se obrađuje u ovom poglavlju je:

- LOKACIJA (adresa, vreme pravljenja, ID)
- ARTIKAL (ime, kraći opis, duži opis, vreme pravljenja, ID)
- KATEGORIJA (ime, vreme pravljenja, ID)
- OSOBINA (vrednost, vreme pravljenja, ID, ID tipa osobine)
- TIP OSOBINE (ime, vreme pravljenja, ID)
- CENA (iznos, vreme pravljenja, ID, ID artikla)
- DOBAVLJAČ (naziv, adresa, telefon, adresa elektronske pošte, vreme pravljenja, ID, aktivan)
- NABAVKA (količina, identifikacioni broj transakcije, vreme pravljenja, ID, ID dobavljača, ID lokacije, ID artikla)
- KLIJENT (ime, prezime, adresa elektronske pošte, adresa, vreme pravljenja, ID, aktivan)
- RAČUN (vreme pravljenja, ID, ID klijenta)
- PLAĆANJE (iznos, vreme pravljenja, ID, ID računa, iznos, ID vrste plaćanja)
- VRSTA PLAĆANJA (naziv, vreme pravljenja, ID, aktivna)
- ISPORUKA (jedinstveni broj pošiljke, status, vreme pravljenja, ID, ID računa, ID klijenta)
- TRANSFER (količina, vreme pravljenja, ID, ID lokacije ulaza, ID lokacije izlaza, ID artikla, količina)
- ARTIKAL_KATEGORIJA (ID artikla, ID kategorije, vreme pravljenja, ID)
- ARTIKAL_OSOBINA (ID artikla, ID osobine, vrednost, vreme pravljenja, ID)
- RAČUN_ARTIKAL (ID artikla, ID računa, količina, vreme pravljenja, ID)

8.4.6 Usklađivanje sa KONVENCIJOM IMENOVANJA

Za razliku od neformalnih jezika na kojem je izražen tekst projektnog zadatka na kojem se zasniva praktični primer obrađen u ovom poglavlju, formalni jezici, kao što su programski jezici, jezici za označavanje i upitni jezici poput SQL jezika, poseduju određena ograničenja u pogledu imenovanja jezičkih elemenata.

Neka od ograničenja mogu da se vide na primeru entitet VRSTA PLAĆANJA:

- Određene implementacije SQL jezika ne dozvoljavaju razmake u imenima,
- Određene implementacije SQL jezika podržavaju samo ASCII u imenima,
- Imena tabela ne mogu da počinju cifrom itd.

Zbog ovakvih ograničenja, neophodno je da se izvrše određena preimenovanja entiteta i atributa prilikom preslikavanja u relacioni model. Ujedno, potrebno je da se izvrši usklađivanje imena sa nekom od konvencija imenovanja koje se u praksi koriste.

U praksi postoji veliki broj konvencija imenovanja koje mogu da budu korišćene i koji je moguće pridržavati se prilikom imenovanja tabela, njihovih polja, indeksâ i drugih elemenata baze podatka. Odabir konvencije imenovanja može da bude stvar ličnog izbora ili može da bude pripisan od strane firme, u vidu tehničkih ograničenja kojih inženjeri softvera treba da se pridržavaju. U svakom slučaju, važno je u potpunosti i na svakom mestu se držati odabrane ili propisane konvencije imenovanja i od nje se ne bi trebalo odstupati.

Određene konvencije imenovanja imaju ime. Međutim, većina ili nema svoje specifično ime ili se na njih ukazuje imenom poznatog projekta u kojem su primenjene. Takođe, neke su u potpunosti specifične za određeni proizvod, preduzeće, grupu ljudi koji ih koristi itd, pa kao takve nisu šire poznate u praksi.

Uz određena poboljšanja, konvencija imenovanja koja se koristi u obradi primera u ovom poglavlju najviše podseća na tzv. Sakila konvenciju. Glavne odlike ove konvencije imenovanja su:

- Koristi se engleski jezik za imenovanje elemenata baze podataka (tabela, polja, indeksa, itd);
- Koriste se isključivo mala slova engleskog alfabeta, cifre i simbol _ (donja crta);
- Imena ne smeju da počinju cifrom;
- Imena ne smeju da počinju ili da se završavaju simbolom _ (donja crta);
- Imena tabela ili polja moraju da budu napisana u jednini;
- Imena tabela ne smeju u imenu da sadrže dva uzastopna simbola _ (donja crta);
- Svaka tabela mora da poseduje polje za jedinstveni identifikacioni broj (čak i vezne tabele za M:N relacije);
- Primarni ključ tabele mora da bude isključivo polje za jedinstveni identifikacioni broj (ne sme biti kombinovani ključ);
- Ime polja za jedinstveni identifikacioni broj, koje se koristi za primarni ključ, mora da se imenuje dodavanjem nastavka _id na ime tabele, npr. ako je ime tabele transaction, primarni ključ mora da bude polje sa imenom transaction_id.
- Polja ne smeju da se završavaju sa _id ako nisu primarni ili strani ključevi;
- Polje koje je strani ključ mora da se zove isto kao polje koje je primarni ključ u tabeli ka kojoj strani ključ ukazuje. Međutim, ako u tabeli postoje dva strana ključa koja ukazuju ka istoj tabeli, ispred imena polja koje je isto kao primarni ključ referencirane tabele se upisuje reč koja pojašnjava namenu primarnog ključa, odvojena dvostrukim simbolom _ (donja crta), npr. from__location_id i to__location_id itd.

- Polje koje je vremenskog tipa (datum, vreme, vremenski žig itd) mora da se završava sufiksom _at, npr. created_at, modified_at, deleted_at itd.
- Polje koje je logičkog tipa ili se koristi kao logička vrednost, mora da počinje prefiksom is_, npr. is_active, is_visible, is_deleted itd.
- Ime indeksa stranog ključa se formira po sledećem pravilu: na prefiks fk_ se dodaje ime tabele, praćeno simbolom _ (donja crta), iza kojeg sledi ime polja koje predstavlja strani ključ, npr. fk_invoice_client_id (tabela je invoice, a strani ključ je client_id).
- Ime indeksa koji garantuje jedinstvenu vrednost polja u tabeli se formira po sledećem pravilu: na prefiks uq_ se dodaje ime tabele, praćeno simbolom _ (donja crta), iza kojeg sledi ime polja koje se indeksira ili imena poljâ koja su deo složenog ključa, razdvojena simbolom _ (donja crta), npr. uq_client_email (tabela je client, a jedinstveno polje se zove email) ili uq_invoice_article_invoice_id_article_id (tabela je invoice_article, a pod indeksom su dva polja: invoice_id i article_id).
- Ime normalnog indeksa se formira po sledećem pravilu: na ime tabele se dodaje simbol _ (donja crta), praćena imenom polja koje se indeksira ili imenima poljâ koja se indeksiraju, razdvojena simbolom _ (donja crta), iza kojih na kraju sledi sufiks _idx, npr. payment_type_is_active_idx (tabela je payment_type, a polje je is_active).
- Ime okidača se formira po sledećem pravilu: na prefiks trigger_ se dodaje ime tabele, praćeno sufiksima: _bi, _bu, _bd, _ai, _au ili _ad kada se okidač izvršava: pre dodavanja, pre izmene, pre brisanja, posle dodavanja, posle izmene ili posle brisanja respektivno, npr. trigger_payment_bi (tabela je payment, a okidač se izvršava pre dodavanja zapisa).
- Ime pogleda treba da počinje sa rečju view praćenom sa dva simbola _ (donja crta), npr. view__procurement_details itd. Ime pogleda iza obaveznog prefiksa ne mora da bude napisano u jednini, ali mora da bude sastavljeno od karaktera koji su dozvoljeni u imenima elemenata baze podataka.

Prilikom preslikavanja entiteta i atributa, nastaće promene u nazivima koji su korišćeni u prethodnim koracima modelovanja.

Zbog toga je potrebno napraviti tabelu preslikavanja u kojoj će biti predstavljeno koji entiteti i njihovi atributi odgovaraju kojim tabelama i njihovim poljima u novom modelu baze podataka koju gradimo.

8.4.7 Preslikavanje entiteta u relacioni model

Entiteti, atributi i relacije koji su nastali prilikom dekompozicije teksta projektnog zahteva treba da budu preslikani u relacioni model pre pristupanja izradi baze podataka prema takvom modelu.

U relacionim bazama podataka, entitete predstavljaju tabele, atribute predstavljaju polja tih tabela, dok se veze između entiteta grade definisanjem adekvatnih indeksâ nad poljima tabela i pravljenjem stranih ključeva, tj. relacija između tabela u modelu.

Entiteti → Tabele u bazi
Atributi → Polja u tabelama
Veze → Relacije između tabela

Tabela preslikavanja entiteta u tabele u bazi podataka

Ime entiteta	Ime tabele usklađeno sa konvencijom imenovanja
LOKACIJA	location
ARTIKAL	article
KATEGORIJA	category
OSOBINA	feature
TIP OSOBINE	feature_type
CENA	article_price
DOBAVLJAČ	supplier
NABAVKA	procurement
KLIJENT	client
RAČUN	invoice
PLAĆANJE	payment
VRSTA PLAĆANJA	payment_type
ISPORUKA	shipment
TRANSFER	transfer
ARTIKAL_KATEGORIJA	article_category
ARTIKAL_OSOBINA	article_feature
RAČUN_ARTIKAL	invoice_article

Prilikom preslikavanja atributa entitetâ u polja tabelâ, potrebno je voditi računa o tipovima tih polja, zbog usklađivanja imena polja sa konvencijom imenovanja, posebno u slučajevima vremenskih i logičkih tipova, kao i u slučaju polja koja su strani ključevi, ali i podrazumevanog polja za jedinstveni identifikacioni broj, koji je prema konvenciji koristi kao primarni ključ tabele.

Prilikom preslikavanja jednog entiteta sa svim njegovim atributima u odgovarajuću tabelu sa poljima i potrebnim indeksima treba voditi računa o svim navedenim pravilima iz konvencije, a to se vidi na primeru entiteta NABAVKA.

NABAVKA	→	**procurement**	
- ID	→	- procurement_id	celobrojnog tipa
- ID dobavljača	→	- supplier_id	celobrojnog tipa
- ID lokacije	→	- location_id	celobrojnog tipa
- ID artikla	→	- article_id	celobrojnog tipa
- količina	→	- quantity	realnog tipa
- vreme pravljenja	→	- created_at	vremenskog tipa
- ID broj transakcije	→	- transaction_number	tekstualnog tipa

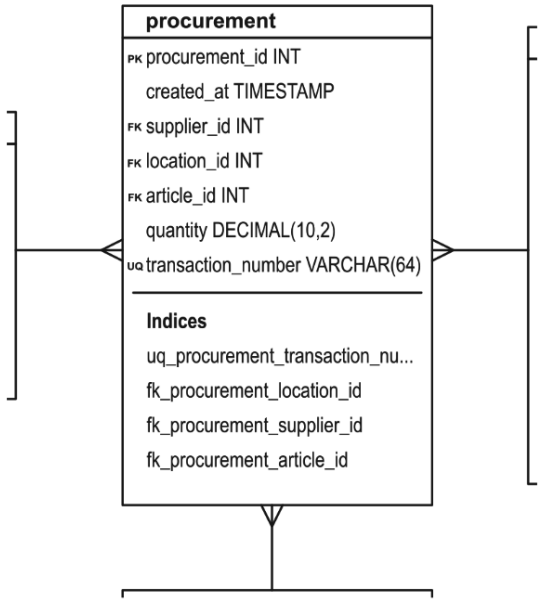
Ono što tokom dekompozicije nije bilo definisano, a neophodno je da se definiše prilikom preslikavanja entiteta u relacioni model su indeksi i relacije. Za gore navedeni primer, potrebno je da se kreiraju relacije između tabele **procurement**, koja odgovara entitetu NABAVKA, i tabela supplier, location i article koje odgovaraju entitetima DOBAVLJAČ, LOKACIJA i ARTIKAL, respektivno. Takođe, neophodno je da se definiše indeks za jedinstvenu vrednost identifikacionog broja transakcije.

Veza ka entitetu DOBAVLJAČ	→	fk_procurement_supplier_id
Veza ka entitetu LOKACIJA	→	fk_procurement_location_id
Veza ka entitetu ARTIKAL	→	fk_procurement_article_id
Jedinstveni id. broja transakcije	→	uq_procurement_transaction_number

8.4.8 Grafički prikaz tabela i relacija

Tabele, polja i relacije između tabela mogu da budu prikazane grafički. Često se u grafičkom prikazu određene informacije o poljima gube, zbog potrebe da se najbitnije odrednice grafički prikazu na ograničenom prostoru. U najvećem broju slučajeva, grafički prikaz tabela obuhvata prikaz imena, spiska polja sa osnovnim indikatorima tipa podatka, bez dodatnih specifikacija koje određeni tipovi možda imaju, kao i uz eventualne slikovne prikaze uloge polja u tabeli, npr. da li je u pitanju primarni ključ, strani ključ, jedinstveni indeks, običan indeks itd. Određeni grafički prikazi mogu da uključe ispis imena indeksa i relacija što često zahteva dodatni prostor i uvećava ukupne dimenzije konačnog dijagrama.

Prethodno preslikani entitet NABAVKA, odnosno tabela **procurement** sa svim poljima, indeksima, oznakama polja i indikatorima relacije usmerenim ka povezanim tabelama može da bude prikazan kao na ilustraciji u produžetku.

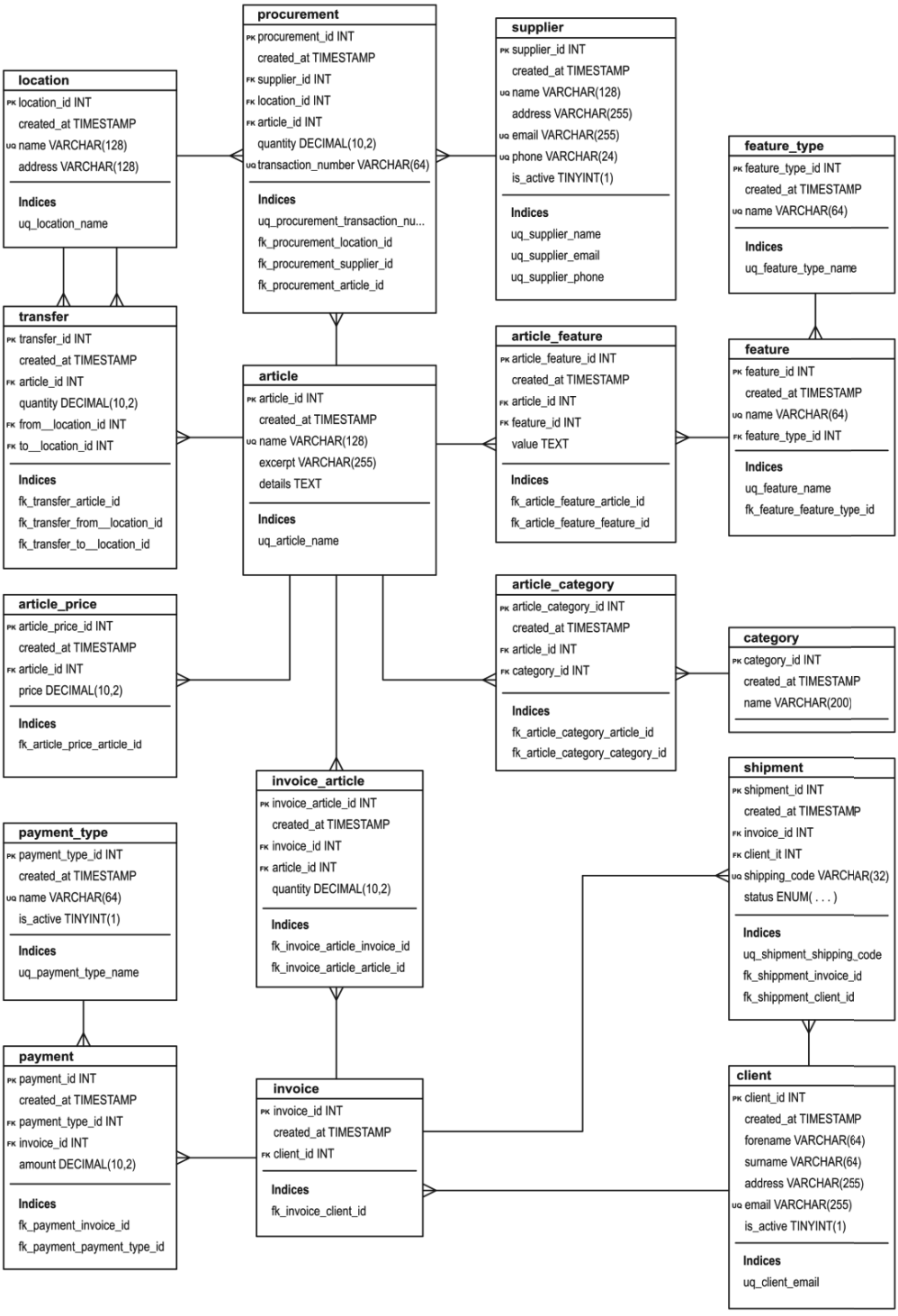


Slika 8.1: Ilustracija tabele procurement sa prikazom polja, njihovih tipova, oznaka i indeksa

8.4.9 Model baze podataka

Na osnovu dekompozicije teksta projektnog zadatka i nakon preslikavanja svih entiteta, atributa i veza u elemente relacionog modela, konačan model baze podataka je prikazan u nastavku.

U ovom poglavlju nije opisano preslikavanje svakog pojedinačnog entiteta, ali su detalji iz preslikavanja prikazani u narednim koracima obrađenim u ovom poglavlju, a koji se odnose na pravljenje samih tabela u bazi podataka korišćenjem SQL jezika.



Slika 8.2: Kompletan model baze podataka na osnovu dekompozicije projektnog zahteva

8.4.10 SQL naredba za pravljenje baze podataka - CREATE TABLE

Sintaksa naredbe za pravljenje baze podataka je prema MySQL uputstvu:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
[DEFAULT] { CHARACTER SET [=] character_set_name | COLLATE [=] collation_name }
```

Rezervisane reči DATABASE ili SCHEMA su alijasi, a IF NOT EXISTS, ukoliko se koristi, određuje da novu bazu podataka server treba da napravi samo ako već ne postoji baza pod tim imenom. Odrednice grupe karaktera koja mogu da se koriste za imenovanje elemenata unutar baze podataka i za vrednosti unutar polja su CHARACTER SET i COLLATE i njihove moguće vrednosti su dostupne u zvaničnoj dokumentaciji, a mogu da zavise od instalacije i načina kako je podešen sâm MySQL.

Primer naredbe za pravljenje baze podataka čije ime je **kompanija**:

```
CREATE DATABASE kompanija CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

U navedenom listingu, za grupu karaktera koju baza podataka koristi odabrana je UTF-8 kodna šema, a za poravnanje vrednosti je odabrana unikodna kolacija UTF-8 kodne šeme koja nije osetljiva na velika i mala slova (_ci znači case insensitive).

8.4.11 SQL naredba za odabir baze podataka - USE

Pre početka pravljenja tabela, indeksa, relacija ili upravljanja podacima u bazi podataka, neophodno je izdati MySQL serveru naredbu za odabir baze podataka na koju se odnose naredbe koje slede.

Sintaksa SQL naredba za odabir baze podataka za upotrebu u daljem radu je:

```
USE db_name;
```

U slučaju konkretnog primera koji je u ovom poglavlju obrađen, ime baze podataka je **kompanija**, tako da je naredba koju koristimo:

```
USE kompanija;
```

Nakon izvršavanja naredbe, ukoliko baza podataka postoji, ispisuje se poruka:

```
Database changed
```

Ukoliko na serveru ne postoji baza podataka sa datim imenom, vidimo poruku:

```
1049 - Unknown database 'kompanija'
```

8.4.12 SQL naredba za pravljenje tabele - CREATE TABLE

Svedena varijanta sintakse naredbe za pravljenje tabele je prema MySQL uputstvu:

```
CREATE TABLE [IF NOT EXISTS] tbl_name
(col_name data_type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
[[PRIMARY] KEY],...);
```

Kompletna sintaksna definicija naredbe za pravljenje tabele je dostupna u zvaničnom uputstvu za MySQL. Na osnovu svedene varijante sintakse, moguće je napisati naredbu za pravljenje tabela sa njihovim poljima. U postupku pravljenja tabelâ, u prvom koraku, biće napravljena samo polja tih tabela. Indeksi i relacije će biti napravljene naknadno.

Tabela location

```
CREATE TABLE location (
  location_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(128) NOT NULL,
  address VARCHAR(128) NOT NULL );
```

Na ovaj način je napravljena tabela location, koja ima polja:

- location_id je neoznačenog celobrojnog tipa (INT UNSIGNED), ne može ostati nepopunjen (NOT NULL), primarni ključ tabele je (PRIMARY KEY) i automatski će mu biti dodeljena vrednost iz sekvence vrednosti tabele (AUTO_INCREMENT) za jedan veća od poslednje dodeljene vrednosti;
- created_at je vremenski žig (TIMESTAMP), ne može ostati nepopunjen (NOT NULL) i podrazumevana vrednost je vreme na serveru u trenutku upisa;
- name je tekstualnog tipa, ograničenog kapaciteta od najviše 128 karaktera i ne može ostati nepopunjeno;
- address je tekstualnog tipa, ograničenog kapaciteta od najviše 128 karaktera i ne može ostati nepopunjeno.

Tabela supplier

```
CREATE TABLE supplier (
  supplier_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(128) NOT NULL,
  address VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  phone VARCHAR(24) NOT NULL,
  is_active TINYINT(1) UNSIGNED NOT NULL DEFAULT 1 );
```

Na ovaj način je napravljena tabela supplier, koja ima polja:

- supplier_id je neoznačenog celobrojnog tipa (INT UNSIGNED), ne može ostati nepopunjen (NOT NULL), primarni ključ tabele je (PRIMARY KEY) i automatski će mu biti dodeljena vrednost iz sekvence vrednosti tabele (AUTO_INCREMENT) za jedan veća od poslednje dodeljene vrednosti;
- created_at je vremenski žig (TIMESTAMP), ne može ostati nepopunjen (NOT NULL) i podrazumevana vrednost je vreme na serveru u upisa;
- name je tekstualnog tipa, ograničenog kapaciteta od najviše 128 karaktera i ne može ostati nepopunjeno;
- address je tekstualnog tipa, ograničenog kapaciteta od najviše 255 karaktera i ne može ostati nepopunjeno;
- email je tekstualnog tipa, ograničenog kapaciteta od najviše 255 karaktera i ne može ostati nepopunjeno;
- phone je tekstualnog tipa, ograničenog kapaciteta od najviše 24 karaktera i ne može ostati nepopunjeno;
- is_active je neoznačenog celobrojnog tipa malog kapaciteta (TINYINT(1) UNSINGNED) koji se često koristi zao zamena za logički tip u MySQL bazama podataka, ne može ostati nepopunjen (NOT NULL) i podrazumevana vrednost je 1 (koja služi kao logička vrednost tačno).

Tabela article

```
CREATE TABLE article (
  article_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(128) NOT NULL,
  excerpt VARCHAR(255) NOT NULL,
  details TEXT NOT NULL
);
```

Tabela procurement

```
CREATE TABLE procurement (
  procurement_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  supplier_id INT UNSIGNED NOT NULL,
  location_id INT UNSIGNED NOT NULL,
  article_id INT UNSIGNED NOT NULL,
  quantity DECIMAL(10,2) UNSIGNED NOT NULL,
  transaction_number VARCHAR(64) NOT NULL
);
```

U tabeli procurement, polja supplier_id, location_id i article_id su polja koja se koriste kao strani ključevi. U narednim koracima će biti prikazano korišćenje naredbe za pravljenje stranih ključeva i spajanje tabela referencama.

Tabela transfer

```
CREATE TABLE transfer (
  transfer_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  article_id INT UNSIGNED NOT NULL,
  quantity DECIMAL(10,2) UNSIGNED NOT NULL,
  from__location_id INT UNSIGNED NOT NULL,
  to__location_id INT UNSIGNED NOT NULL );
```

U tabeli transfer postoje dva polja koja su namenjena da budu strani ključevi ka istoj tabeli location. Prema konvenciji imenovanja, ako postoje dva strana ključa u istoj tabeli koja ukazuju na istu tabelu, potrebo je ispred imena polja dodati odrednicu koja preciznije pojašnjava namenu polja razdvojenu dvostrukim simbolom _ (donja crta), te postoje polja from__location_id i to__location_id koja služe za čuvanje podataka o tome sa koje lokacije ka kojoj lokaciji je preseljena određena količina artikala referenciranih preko stranog ključa article_id. U narednim koracima će biti prikazano korišćenje naredbe za pravljenje stranih ključeva i spajanje tabela referencama.

Tabela article_price

```
CREATE TABLE article_price (
  article_price_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  article_id INT UNSIGNED NOT NULL,
  price DECIMAL(10,2) UNSIGNED NOT NULL );
```

Tabela category

```
CREATE TABLE category (
  category_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(200) NOT NULL );
```

Tabela article_category

```
CREATE TABLE article_category (
  article_category_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  article_id INT UNSIGNED NOT NULL,
  category_id INT UNSIGNED NOT NULL );
```

Tabela feature_type

```
CREATE TABLE feature_type (
  feature_type_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(64) NOT NULL );
```

Tabela feature

```
CREATE TABLE feature (
  feature_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(64) NOT NULL,
  feature_type_id INT UNSIGNED NOT NULL
);
```

Tabela article_feature

```
CREATE TABLE article_feature (
  article_feature_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  article_id INT UNSIGNED NOT NULL,
  feature_id INT UNSIGNED NOT NULL,
  value TEXT NOT NULL );
```

Tabela client

```
CREATE TABLE client (
  client_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  forename VARCHAR(64) NOT NULL,
  surname VARCHAR(64) NOT NULL,
  address VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  is_active TINYINT(1) UNSIGNED NOT NULL DEFAULT 1
 );
```

Tabela invoice

```
CREATE TABLE invoice (
  invoice_id INT UNSIGNED NOT NULL
    AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  client_id INT UNSIGNED NOT NULL
);
```

Tabela invoice_article

```
CREATE TABLE invoice_article (
  invoice_article_id INT UNSIGNED NOT NULL
    AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  invoice_id INT UNSIGNED NOT NULL,
  article_id INT UNSIGNED NOT NULL,
  quantity DECIMAL(10,2) UNSIGNED NOT NULL
);
```

Tabela payment_type

```
CREATE TABLE payment_type (
  payment_type_id INT UNSIGNED NOT NULL
    AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  name VARCHAR(64) NOT NULL,
  is_active TINYINT(1) UNSIGNED NOT NULL DEFAULT 1
);
```

Tabela payment

```
CREATE TABLE payment (
  payment_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  payment_type_id INT UNSIGNED NOT NULL,
  invoice_id INT UNSIGNED NOT NULL,
  amount DECIMAL(10,2) UNSIGNED NOT NULL
);
```

Tabela shipment

```
CREATE TABLE shipment (
  shipment_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  invoice_id INT UNSIGNED NOT NULL,
  client_it INT UNSIGNED NOT NULL,
  shipping_code VARCHAR(32) NOT NULL,
  status ENUM('pending', 'sent', 'returned', 'lost', 'delivered')
    NOT NULL
    DEFAULT 'pending'
);
```

U tabeli shipment, polje status je ENUM tipa, sa mogućim vrednostima pending, sent, returned, lost i delivered. Podrazumevana vrednost tog polja je pending, a polje u zapisu ne može da bude ostavljeno prazno.

S obzirom na to da u modelu baze podataka postoji veza između shipment i client, koja bi inače mogla da bude ustanovljena preko postojeće veze između tabele shipment i tabele invoice, koja je povezana sa client, jasno je da kompanija podržava mogućnost da artikli po računu napravljenom za jednog klijenta mogu da budu poslati drugom klijentu, ukoliko je takav dogovor.

Da ne postoji ova planirana relacija između tabela shipment i client, ovakva pogodnost ne bi bila moguća.

8.4.13 SQL naredba za brisanje tabele - DROP TABLE

Svedena sintaksa naredbe za brisanje tabele je prema MySQL uputstvu:

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name] ...
```

Jednom naredbom je moguće izvršiti brisanje više od jedne tabele, kada se njihova imena navedu razdvojena zarezom. Deo naredbe IF EXISTS će obezbediti da tabela bude obrisana ako postoji, a ako ne postoji, da izvršavanje ove naredbe ne izazove vraćanje greške.

Primer naredbe za brisanje tabele shipment koja je poslednja napravljena u prethodnom koraku obrađenog primera je:

```
DROP TABLE shipment;
```

8.4.14 SQL naredba za pravljenje indeksa - CREATE INDEX

Svedena sintaksa naredbe za pravljenje indeksa je prema MySQL uputstvu:

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
USING {BTREE | HASH}
ON tbl_name (col_name,...);
```

Prilikom pravljenja indeksa, potrebno je da se odredi tip indeksa koji se pravi. Moguće je pravljenje četiri vrste indeksa:

- INDEX **Normalan indeks** - za optimizaciju performansi.
- UNIQUE INDEX **Jedinstveni indeks** - za garanciju jedinstvene vrednosti.
- FULLTEXT INDEX **Indeks za pretragu teksta** - za optimizaciju pretrage.
- SPATIAL INDEX **Prostorni indeks** - za optimizaciju prostornih polja.

Prilikom pravljenja indeksa, potrebno je odrediti mehanizam čuvanja indeksa u bazi podataka. Podržana su dva tipa:

- BTREE Koristi algoritam pretrage **binarnog stabla**.
  Ovo je podrazumevani mehanizam za indekse.
- HASH Koristi pretragu **tabele heš vrednosti**.
  Najčešće služi kod pretrage po primarnim ključevima.

Pod indeks može da se upiše više polja čija imena se navode u zagradi iza imena tabele nad kojom se formira indeks.

Na osnovu svedene sintakse, naredba kojom se pravi jedinstveni indeks za polje name u tabeli location kojim se garantuje jedinstvena vrednost imena lokacije je:

```
CREATE UNIQUE INDEX uq_location_name USING BTREE ON location(name);
```

Istim postupkom mogu da budu napravljeni jedinstveni indeksu za sva ostala polja ili grupe polja prema primeru modela baze podataka sa kojim se radi u primeru koji je obrađen u knjizi. SQL naredbe u nastavku prave sve preostale jedinstvene indekse.

```
CREATE UNIQUE INDEX uq_article_name ON article (name);
CREATE UNIQUE INDEX uq_client_email ON client (email);
CREATE UNIQUE INDEX uq_feature_name ON feature (name);
CREATE UNIQUE INDEX uq_feature_type_name ON feature_type (name);
CREATE UNIQUE INDEX uq_payment_type_name ON payment_type (name);
CREATE UNIQUE INDEX uq_procurement_transaction_number ON procurement
  (transaction_number);
CREATE UNIQUE INDEX uq_shipment_shipping_code ON shipment (shipping_code);
CREATE UNIQUE INDEX uq_supplier_email ON supplier (email);
CREATE UNIQUE INDEX uq_supplier_name ON supplier (name);
CREATE UNIQUE INDEX uq_supplier_phone ON supplier (phone);
```

8.4.15 SQL naredba za brisanje indeksa - DROP INDEX

Sintaksa naredbe za brisanje indeksa je prema MySQL uputstvu:

```
DROP INDEX index_name ON tbl_name;
```

Primer SQL naredbe za brisanje indeksa je prikazan u nastavku.

```
DROP INDEX uq_article_name ON article;
```

8.4.16 SQL naredba za izmenu tabele - ALTER TABLE

Sintaksa naredbe za izmenu tabele je složena i obuhvata veliki broj aktivnosti koje mogu da se obave prilikom zahteva za izmenu.

Izmenom tabele je moguće obaviti, između ostalih, sledećih aktivnosti:

- Promena imena tabele,
- Promena potpisa polja tabele,
- Dodavanje polja u tabelu,
- Brisanje polja iz tabele,
- Dodavanje ograničenja/reference u tabelu itd.

Promena imena tabele

Ukoliko postoji potreba da se tabela preimenuje bez gubitka strukture i podataka u njoj, moguće je korišćenje varijante SQL naredbe za izmenu tabele sa aktivnošću promene imena.

Primer ovakve SQL naredbe je prikazan u nastavku.

```
ALTER TABLE old_table_name RENAME new_table_name;
```

Promena imena i potpisa polja

Ukoliko postoji potreba da se u postojećoj tabeli promeni potpis polja (ime, tip podatka ili druge postavke), moguće je korišćenje varijante SQL naredbe za izmenu tabele sa aktivnošću promene potpisa polja.

Primeri ovakvih SQL naredbi su prikazani u nastavku.

Promena imena polja

Ukoliko je potrebno da se promeni ime polja u tabeli table_name sa old_name u new_name, potrebno je izvršiti naredbu:

```
ALTER TABLE table_name CHANGE COLUMN old_name new_name varchar(128) NOT NULL;
```

Prilikom promene imena polja, neophodno je potvrditi celokupan potpis polja, tj. tip podatka polja sa svim pripadajućim podešavanjima, kao i sve druge postavke polja, kao što je u ovom slučaju NOT NULL.

Promena tipa podatka polja

Ukoliko je potrebno da se promeni tip polja column_name na TEXT tip sa NOT NULL postavkom, potrebno je izvršiti naredbu:

```
ALTER TABLE table_name MODIFY COLUMN column_name TEXT NOT NULL;
```

Ukoliko postoji potreba da se promeni ime polja u tabeli uz promenu potpisa, treba da se koristi aktivnost **CHANGE**.

Ukoliko postoji potreba da se promeni samo potpis polja u tabeli, bez menjanja imena, treba da se koristi aktivnost **MODIFY**.

Dodavanje polja u tabelu

Ukoliko postoji potreba da se u postojeću tabelu doda novo polje, moguće je korišćenje SQL naredbe za izmenu tabele sa aktivnošću dodavanja polja određenog imena i potpisa. Primer ovakve SQL naredbe je prikazan u nastavku.

```
ALTER TABLE article
  ADD COLUMN
    is_visible tinyint(1) UNSIGNED NOT NULL DEFAULT 1;
```

Gore navedena SQL naredba dodaje polje is_visible koje služi kao logički podatak realizovan korišćenjem neobeleženog celobrojnog tipa za polje koje ne može da ostane prazno i čija podrazumevana vrednost je 1. Polje će podrazumevano biti dodato na poslednje mesto u tabeli.

Ako postoji potreba da se novo polje doda na tačno određenu lokaciju u tabeli, moguće je u nastavku SQL naredbe precizirati iza kojeg postojećeg polja novo polje treba da bude dodato.

Primer ovakve SQL naredbe je prikazan u nastavku.

```
ALTER TABLE article
  ADD COLUMN
    is_visible tinyint(1) UNSIGNED NOT NULL DEFAULT 1 AFTER details;
```

Gore navedena SQL naredba dodaje novo polje na poziciju iza postojećeg polja sa imenom details.

Brisanje polja iz tabele

Ukoliko postoji potreba da se iz postojeće tabele izbriše postojeće polje, moguće je korišćenje SQL naredbe za izmenu tabele sa aktivnošću brisanja polja iz tabele. Prilikom brisanja polja, biće trajno izgubljene sve vrednosti u tabeli koje su upisane pod tim poljem. Primer ovakve SQL naredbe je prikazan u nastavku.

```
ALTER TABLE article
  DROP COLUMN is_visible;
```

Gore navedena SQL naredba briše polje is_visible iz tabele article. Ovim postupkom se gube sve vrednosti podatka is_visible iz svih zapisa u tabeli article. U slučaju potrebe da se polje naknadno vrati, ne postoji mogućnost vraćanja prvobitnih vrednosti za pojedinačne redove iz tabele, osim iz eventualne rezervne kopije tabele ili baze podataka napravljene pre postupka brisanja polja.

8.4.17 SQL naredba za pravljenje referenci - FOREIGN KEY

Do ovog koraka, model baze podataka je realizovan tako da poseduje isključivo implicitne relacije koje se zasnivaju na konvenciji imenovanja. Npr. na osnovu imena polja payment_type_id u tabeli payment, jasno je da je polje namenjeno kao strani ključ od tabele payment, u kojoj je definisano, ka tabeli payment_type čije ime primarnog ključa nosi, tj. payment_type_id.

Implicitne relacije ne obezbeđuju očuvanje integriteta podataka, jer ne postoje provere koje bi bile postavljene ograničenjima nad poljima stranih ključeva u tabeli.

Postupak pravljenja reference između tabela se realizuje korišćenjem SQL naredbe za izmenu tabele i njene varijante sa korišćenjem aktivnosti za dodavanje ograničenja.

Primer pravljenja jednog ograničenja, odnosno reference nad tabelom article_category kojim se pravi čvrsta relacija između polja stranog ključa article_id i primarnog ključa tabele article je dat u nastavku:

```
ALTER TABLE article_category
  ADD CONSTRAINT fk_article_category_article_id
    FOREIGN KEY (article_id)
    REFERENCES article (article_id)
      ON DELETE RESTRICT
      ON UPDATE CASCADE;
```

Gore napisana SQL naredba vrši izmenu tabele article_category, dodavanjem novog ograničenja (koje sa sobom povlači automatsko pravljenje indeksa sa imenom fk_article_category_article_id) nad stranim ključem za koji se koristi polje article_id, a referencira se tabela article i njen primarni ključ, tj. polje article_id.

Ovim ograničenjem se još preciziraju situacije prilikom pokušaja:

- **brisanja (ON DELETE) zapisa iz referencirane tabele article:** ako postoji barem jedan zapis u tabeli article_category koji sadrži u polju article_id vrednost primarnog ključa zapisa koji se briše, da ta aktivnost bude sprečena (RESTRICT).
- **izmene (ON UPDATE) zapisa iz referencirane tabele article:** ako postoje zapisi u tabeli article_category koji sadrži u polju article_id vrednost primarnog ključa zapisa kojem se menja vrednost, da se vrednost promeni i u svim zapisima u tabeli article_category u polju article_id, gde se ta vrednost pominje (CASCADE).

Istim postupkom mogu da budu napravljene sve ostale reference predviđene primerom modela baze podataka koji se obrađuje. SQL naredbe u nastavku prave sve preostale reference između tabela, kao što je predviđeno predstavljenim modelom.

```
ALTER TABLE article_category
  ADD CONSTRAINT fk_article_category_category_id
    FOREIGN KEY (category_id)
    REFERENCES category (category_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE feature
  ADD CONSTRAINT
    fk_feature_feature_type_id
    FOREIGN KEY (feature_type_id)
    REFERENCES feature_type (feature_type_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE article_feature
  ADD CONSTRAINT fk_article_feature_article_id
    FOREIGN KEY (article_id)
    REFERENCES article (article_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE article_feature
  ADD CONSTRAINT fk_article_feature_feature_id
    FOREIGN KEY (feature_id)
    REFERENCES feature (feature_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE article_price
  ADD CONSTRAINT
    fk_article_price_article_id
    FOREIGN KEY (article_id)
    REFERENCES article (article_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE procurement
  ADD CONSTRAINT
    fk_procurement_location_id
    FOREIGN KEY (location_id)
    REFERENCES location (location_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE procurement
  ADD CONSTRAINT fk_procurement_supplier_id
    FOREIGN KEY (supplier_id)
    REFERENCES supplier (supplier_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE procurement
  ADD CONSTRAINT fk_procurement_article_id
    FOREIGN KEY (article_id)
    REFERENCES article (article_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE transfer
  ADD CONSTRAINT
    fk_transfer_article_id
    FOREIGN KEY (article_id)
    REFERENCES article (article_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;
```

```
ALTER TABLE transfer
  ADD CONSTRAINT fk_transfer_from__location_id
    FOREIGN KEY (from__location_id)
    REFERENCES location (location_id)
      ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE transfer
  ADD CONSTRAINT fk_transfer_to__location_id
    FOREIGN KEY (to__location_id)
    REFERENCES location (location_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE invoice
  ADD CONSTRAINT fk_invoice_client_id
    FOREIGN KEY (client_id)
    REFERENCES client (client_id)
      ON DELETE RESTRICT  ON UPDATE CASCADE;

ALTER TABLE invoice_article
  ADD CONSTRAINT fk_invoice_article_invoice_id
    FOREIGN KEY (invoice_id)
    REFERENCES invoice (invoice_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE invoice_article
  ADD CONSTRAINT fk_invoice_article_article_id
    FOREIGN KEY (article_id)
    REFERENCES article (article_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE payment
  ADD CONSTRAINT fk_payment_invoice_id
    FOREIGN KEY (invoice_id)
    REFERENCES invoice (invoice_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE payment
  ADD CONSTRAINT fk_payment_payment_type_id
    FOREIGN KEY (payment_type_id)
    REFERENCES payment_type (payment_type_id)
      ON DELETE RESTRICT ON UPDATE CASCADE;

ALTER TABLE shipment
  ADD CONSTRAINT fk_shipment_invoice_id
    FOREIGN KEY (invoice_id)
    REFERENCES invoice (invoice_id)
      ON DELETE RESTRICT
      ON UPDATE CASCADE;

ALTER TABLE shipment
  ADD CONSTRAINT fk_shipment_client_id
    FOREIGN KEY (client_it)
    REFERENCES client (client_id)
      ON DELETE RESTRICT
      ON UPDATE CASCADE;
```

8.4.18 Manipulacija podacima

U prethodnim poglavljima je istaknuto da se SQL jezik se sastoji iz tri osnovne grupe naredbi, među kojima je i grupa naredbi za upravljanje podacima - DML (eng. *Data Manipulation Language*), koja omogućavaju dopremanje i menjanje podatke koji se čuvaju u struktuiranom obliku u bazi podataka.

Upravljanjem podacima se smatra:

- Dodavanje podataka,
- Dopremanje (pregled) podataka,
- Izmena podataka i
- Brisanje podataka.

8.4.19 SQL naredba za dodavanje podataka - INSERT

Postoje dva osnovna oblika INSERT naredbe za dodavanje podataka:

- naredba za dodavanje izričito navedenih vrednosti;
- naredba za dodavanje vrednosti dobijenih kao rezultat upita.

Oblik sa izričito navedenim vrednostima ima dve sintaksne forme:

Forma naredbe za dodavanje podatka sa izričito navedenim vrednostima sa mogućnošću dodavanja više zapisa istovremeno. Odlikuje se po tome što omogućava navođenje spiska polja sa propisanim redosledom pod kojim će vrednosti koje se dodaju biti definisane u VALUES klauzuli.

```
INSERT [IGNORE] [INTO] tbl_name [(col_name [, col_name] ...)]
  VALUES
    (value [, value] ...) [, (value [, value] ...)] ...
```

Forma naredbe za dodavanje podatka sa izričito navedenim vrednostima sa navođenjem pojedinačnih parova polja i vrednosti.

Ovaj oblik naredbe za dodavanje zapisa u tabelu baze podataka podržava dodavanje samo jednog zapisa jednom naredbom.

```
INSERT [IGNORE] [INTO] tbl_name
  SET   col_name = value [, col_name = value] ...
```

Oblik za dodavanje vrednosti dobijenih kao rezultat upita ima sintaksnu formu:

```
INSERT [IGNORE] [INTO] tbl_name [ (col_name [, col_name] ...) ] SELECT ...
```

Rezervisana reč INTO, koja može da se nalazi ispred imena tabele je proizvoljna.

Rezervisana reč IGNORE, koja može da se nalazi nakon rezervisane reči INSERT, određuje serveru da ne objavljuje grešku prilikom pokušaja dodavanja zapisa sa vrednostima koje nisu prihvatljive, npr. već postoji vrednost u polju koje je indeksirano kao jedinstveno ili je referenciranje nepostojećeg primarnog ključa uvezane tabele itd.

Ukoliko postoji potreba da se u bazu podataka dodaju tri nove lokacije na kojima kompanija posluje, to je moguće postići izvršavanjem naredbi prikazanim u nastavku.

Prikazana su dva načina za dodavanje podataka sa izričito navedenim vrednostima.

Unos lokacija

Naredba za istovremeno dodavanje dve lokacije, uz navođenje poljâ za koje su definisane vrednosti je prikazana u nastavku.

Vrednosti za polja moraju da budu poređane onim redosledom kojim su navedena imena polja u zagradi iza imena tabele.

```
INSERT
  location
  (name, address)
VALUES
  ("Radnja R1", "Adresa naše prve radnje"),
  ("Radnja R2", "Adresa naše druge radnje");
```

Naredba za dodavanje jedne lokacije, navođenjem pojedinačnih parova polja i vrednosti koje se dodaju je prikazana u nastavku.

```
INSERT
  location
SET
  name = "Radnja R3",
  address = "Adresa naše treće radnje";
```

Tekstualne vrednosti u SQL jeziku moraju da budu uokvirene navodnicima.

Nakon izvršavanja ove dve SQL naredbe za dodavanje podataka tabela location će sadržati tri zapisa prikazana u prikazu u nastavku.

location

location_id	created_at	name	address
1	2018-03-25 07:24:19	Radnja R1	Adresa naše prve radnje
2	2018-03-25 07:24:19	Radnja R2	Adresa naše druge radnje
3	2018-03-25 07:24:55	Radnja R3	Adresa naše treće radnje

Vrednosti polja location_id i created_at su automatski dodeljene na serveru tako da:

- Polje location_id, koje je definisano kao AUTO_INCREMENT, dobija uvek vrednost za jedan veću od prethodno dodeljene.
- Polje created_at, koje je podrazumevano CURRENT_TIMESTAM, dobija datum i vreme servera u trenutku dodavanja zapisa.

U nastavku je prikazano još primera naredbi za dodavanje zapisa u tabele baze.

Unos načina plaćanja

```
INSERT payment_type (name)
VALUES
  ("Gotovinsko plaćanje"),
  ("Plaćanje platnom karticom"),
  ("Plaćanje čekovima"),
  ("Kupovina preko vaučera");
```

Dodaje u tabelu payment_type četiri zapisa sa izričito navedenim vrednostima za četiri vrste plaćanja koje kompanija podržava. Vrednosti polja created_at i is_active su automatski dodeljena prema definiciji podrazumevane vrednosti za ta polja.

Unos dobavljača

```
INSERT
  supplier (name, address, email, phone)
VALUES
  ("Dobavljač 1", "Dobavljačka bb", "office@dobavljac.com", "+38111100001"),
  ("Dobavljač 2", "Dobavljačka 11", "office@dobavlja.in.rs", "+38111100002");
```

Dodaje u tabelu supplier jednog dobavljača sa izričito navedenim podacima.

Unos kategorije artikala

```
INSERT
  category
SET
  name = "Čaše";
```

Dodaje u tabelu category jednu kategoriju sa izričito navedenim imenom.

Unos tipova osobina artikala

```
INSERT
  feature_type (name)
VALUES
  ("Dimenzije"),
  ("Izrada");
```

Dodaje u tabelu feature_type dve kategorije sa izričito navedenim imenima.

Unos pojedinačnih osobina artikala

```
INSERT feature (name, feature_type_id)
VALUES
  ("Širina",    1),
  ("Visina",    1),
  ("Dubina",    1),
  ("Obim",      1),
  ("Materijal", 2),
  ("Oblik",     2);
```

Dodaje u tabelu feature šest zapisa osobina sa definisanim imenima i vrednostima za strani ključ feature_type_id koji ukazuju na tipove osobina (1) Dimenzije i (2) Izrada. Vrednosti ID brojeva zapisa koji predstavljaju tip osobine (feature_type) su 1 i 2 ako je uspešno izvršeno dodavanje zapisa u skladu sa ovim primerom.

Unos artikla

```
INSERT article SET
  name    = "Čaša - model T30",
  excerpt = "Staklena čaša opšte namene.",
  details = "Detaljan opis ovog artikla.";
```

Dodaje u tabelu article jedan artikal sa izričito navedenim vrednostima za ime, kratak i detaljan opis. Ovaj artikal još uvek nije povezan sa kategorijama i osobinama.

Povezivanje artikla sa kategorijom

```
INSERT
  article_category
SET
  article_id = 1,
  category_id = 1;
```

Dodaje u tabelu article_category zapis koji povezuje artikal (1) "Čaša - model T30" sa kategorijom (1) "Čaše" čime je ostvarena jedna veza iz relacije M:N između tabela article i category. Ovim postupkom je Čaša - model T30 svrstana u kategoriju Čaše.

Unos osobina artikala

```
INSERT article_feature
  (article_id, feature_id, value)
VALUES
  (1, 2, "100 mm"),
  (1, 4, "72 mm"),
  (1, 5, "staklo"),
  (1, 6, "valjkast");
```

Dodaje četiri zapisa u tabelu article_feature čije se artikal (1) "Čaša - model T30" dodatno obeležava sa četiri osobine sa pratećim vrednostima i to:

- (2) Visina sa vrednošću 100 mm;
- (4) Obim sa vrednošću 72 mm;
- (5) Materijal sa vrednošću staklo;
- (6) Oblik sa vrednošću valjkast.

Unos i promena cene artikla

```
INSERT article_price SET
  created_at = "2018-03-25 08:35:39",
  article_id = 1,
  price      = 124;

INSERT article_price SET
  created_at = "2018-03-25 09:35:11",
  article_id = 1,
  price      = 130;
```

Prethodne dve naredbe dodaju dva zapisa, pojedinačno, u tabelu article_price, čime se artiklu (1) "Čaša - model T30" dodeljuju cene 124, pa 130 dinara po jedinici mere. Poslovnom logikom aplikacije koja treba da upotrebljava ovu bazu podataka, prodaja uvek treba da se vrši po najnovijoj ceni, dok se sve prethodne cene čuvaju iz potrebe za očuvanjem integriteta podataka i zbog praćenja promena cena artikala kroz vreme.

Evidencija nabavke artikla

```
INSERT procurement SET
  supplier_id = 1,
  location_id = 2,
  article_id  = 1,
  quantity    = 5,
  transaction_number = "4001-2311-890112";
```

Dodaje jedan zapis u tabelu procurement sa izričito navedenim vrednostima kojima se evidentira nabavka 5 komada artikla (1) "Čaša - model T30" od dobavljača (1) "Dobavljač 1" dostavljenih na lokaciju (2) "Radnja R2".

Evidencija prenosa robe

```
INSERT transfer SET
  article_id       = 1,
  from__location_id = 2,
  to__location_id   = 3,
  quantity         = 4;
```

Dodaje jedan zapis u tabelu transfer sa izričito navedenim vrednostima kojima se evidentira prenos 4 artikla sa lokacije (2) "Radnja R2" na lokaciju (3) "Radnja R3".

Unos klijenta

```
INSERT client SET
  forename = "Milena",
  surname  = "Milić",
  address  = "Bulevar oslobođenja 1",
  email    = "mmilic@milenin-sajt.com";
```

Dodaje jedan zapis u tabelu client sa izričito navedenim vrednostima.

Otvaranje novog računa za kupovinu artikala

```
INSERT invoice (client_id, created_at) VALUES (1, "2018-03-25 09:38:11");
```

Dodaje jedan zapis u tabelu invoice sa izričito navedenim vrednostima za dva naznačena polja. U ovoj naredbi je pokazano da redosled polja ne mora da bude isti kao u definiciji tabele, jer je u zagradi iza imena tabele naveden pravi redosled prema kojem su vrednosti navedene iza klauzule VALUES.

Dodavanje artikla na račun

```
INSERT invoice_article SET
  invoice_id = 1,
  article_id = 1,
  quantity   = 2;
```

Dodaje zapis u tabelu invoice_article kojim se evidentira da je 2 komada artikla (1) "Čaša - model T30" stavljeno na račun (1) koji je u prethodnom koraku otvoren.

Evidencija uplata

Ne postoji obaveza plaćanja računa iz jednog dela, već je moguće napraviti više uplata za isti račun.

Prema tome, moguće je da plaćanja budu izvršena različitim načinima plaćanja za isti račun. Taj scenario je prikazan u dole navedenim naredbama.

```
INSERT payment SET
  created_at      = "2018-03-25 09:45:34",
  payment_type_id = 1, -- Gotovina
  invoice_id      = 1,
  amount          = 60;

INSERT payment SET
  created_at      = "2018-03-25 09:45:34",
  payment_type_id = 4, -- Vaučer
  invoice_id      = 1,
  amount          = 200;
```

Dodaju dva zapisa u tabelu payment, sa izričito navedenim vrednostima, kojima se evidentiraju dve uplate po računu (1).

1. Prva uplata je izvršena gotovinskim plaćanjem (1) u vrednosti od 60 dinara.
2. Druga uplata je izvršena davanjem vaučera na iznos od 200 dinara.

U SQL listingu je prikazano korišćenje komentara u kodu naredbe kojima se dodatno pojašnjava značenje vrednosti stranog ključa payment_type_id (1) i (4).

Evidencija isporuke

```
INSERT shipment SET
  invoice_id    = 1,
  client_it     = 1,
  shipping_code = "MT-2008213514-US";
```

Dodaje jedan zapis u tabelu shipment sa izričito navedenim vrednostima, kojim se evidentira isporuka robe vezane za račun (1) klijentu (1) "Milena Milić" sa navedenim brojem pošiljke.

Vrednost za polje status nije izričito navedeno, jer je podrazumevana vrednost tog polja "pending" i označava da je isporuka planirana, tj. u pripremi.

U jednom od narednih koraka je objašnjeno kako naredbama za izmenu podatka može da se koristi da se status pošiljke promeni.

Dopuna tabela baze podataka za potrebe nastavka obrade ovog primera

```
INSERT category (name)
VALUES
  ("Za kuhinju i trpezariju"), -- biće ID 2, jer već postoji jedna kategorija
  ("Za dnevni boravak"),       -- biće ID 3
  ("Za kuću i baštu"),         -- biće ID 4
  ("Za telefone"),             -- biće ID 5
  ("Osvetljenje");             -- biće ID 6

INSERT feature_type
  SET name = "Opšte informacije"; -- biće ID 3, jer već postoje dve vrste

INSERT feature (name, feature_type_id)
VALUES ("Garancija", 3); -- biće ID 7, jer već postoji šest osobina

INSERT article (name, excerpt, details)
VALUES
  ("Maska za Huawei P20", "Kratak opis maske.", "Detaljan opis..."), -- ID 2
  ("Ogledalo sa svetlom M43", "Kratak opis...", "Detaljan opis..."); -- ID 3

INSERT article_price (created_at, article_id, price)
VALUES
  ("2018-03-25 08:35:12", 2, 250),
  ("2018-03-25 08:35:12", 3, 6750);
```

```
INSERT article_category (article_id, category_id)
VALUES
  (1, 2), -- "Čaša - model T30" u kategoirju "Za kuhinju i trpezariju"
  (2, 5), -- "Maska za Huawei P20" u kategoriju "Za telefone"
  (3, 3), -- "Ogledalo sa svetlom M43" u kategoriju "Za dnevni boravak"
  (3, 4), -- "Ogledalo sa svetlom M43" u kategoriju "Za kuću i baštu"
  (3, 6); -- "Ogledalo sa svetlom M43" u kategoriju "Osvetljenje"

INSERT article_feature (article_id, feature_id, value)
VALUES
  (2, 1, "8 mm"),     -- "Maska za Huawei P20" ima osobinu "Širina" = "8 mm"
  (2, 2, "150 mm"),   -- "Maska za Huawei P20" ima osobinu "Visina" = "150 mm"
  (2, 3, "72 mm"),    -- "Maska za Huawei P20" ima osobinu "Dubina" = "72 mm"
  (2, 5, "silikon"),  -- "Maska za Huawei P20" ima osobinu "Materijal" = "silikon"
  (3, 1, "300 mm"),   -- "Ogledalo sa svetlom M43" ima osobinu "Širina" = "500 mm"
  (3, 2, "500 mm"),     -- "Ogledalo sa svetlom M43" ima osobinu "Visina" = "300 mm"
  (3, 5, "staklo"),   -- "Ogledalo sa svetlom M43" ima osobinu "Materijal" = "staklo"
  (3, 7, "1 godina");   -- "Ogledalo sa svetlom M43" ima osobinu "Garancija" = "1 god."

INSERT procurement (article_id,location_id,supplier_id,quantity,transaction_number)
VALUES
  (1, 1, 1, 10, "4001-2318-125788"),
  (2, 2, 1,  2, "4001-2319-002780"),
  (2, 3, 1,  4, "4001-2318-002781"),
  (3, 3, 1, 12, "4001-2318-002783"),
  (3, 1, 1,  2, "4001-2327-925788");

INSERT transfer (from__location_id, to__location_id, article_id, quantity)
VALUES
  (1, 2, 1, 2), -- Prenos sa lokacije 1 na 2, 2 komada artikla 1
  (2, 3, 2, 1), -- Prenos sa lokacije 2 na 3, 1 komad  artikla 2
  (2, 1, 2, 1), -- Prenos sa lokacije 2 na 1, 1 komad  artikla 2
  (3, 1, 3, 2); -- Prenos sa lokacije 3 na 1, 2 komada artikla 3

INSERT invoice SET -- Otvoriti novi racun (biće ID računa 2)
  created_at = "2018-03-25 09:45:39", -- na zadati datum i vreme
  client_id  = 1;                     -- za klijenta 1

INSERT invoice_article (invoice_id, article_id, quantity)
VALUES -- Dodati na račun artikle u određenoj količini
  (2, 2, 1), -- 1 komad  artikla "Maska za Huawei P20"
  (2, 3, 1), -- 1 komad  artikla "Ogledalo sa svetlom M43"
  (2, 1, 6); -- 6 komada artikla "Čaša - model T30"

INSERT
  payment (created_at, invoice_id, payment_type_id, amount)
VALUES
  ("2018-03-25 11:11:56", 2, 1, 1945), -- Uplata za racun broj 2
  ("2018-04-25 09:33:12", 2, 1, 1945), -- izvrsena kroz cetiri rate
  ("2018-05-26 16:56:10", 2, 1, 1945), -- od po 1945 dinara
  ("2018-06-24 08:45:22", 2, 1, 1945); -- evidentirana u jednom prolazu

-- Kraj unosa podataka.
-- Samostalno možete da izvršite unos dodatnih podataka u tabele ove baze.
```

8.4.20 SQL naredba za dopremanje podataka - SELECT

Svedena sintaksa naredbe za dopremanje podataka je prema MySQL uputstvu:

```
SELECT [DISTINCT] {* | col_name ...}
FROM table_name [[AS] alias_name] [, table_name [[AS] alias_name]] ...
[WHERE condition]
[GROUP BY {col_name | expr} [ASC | DESC], ...]
[HAVING condition]
[ORDER BY {col_name | expr} [ASC | DESC], ...]
[LIMIT [offset,] count];
```

Rezervisana reč DISTINCT koja može da se nalazi nakon rezervisane reči SELECT je proizvoljna i ukoliko je zadata, određuje serveru da dostavi samo jedinstvene redove, tj. ukoliko postoji više redova sa identičnim vrednostima u svim kolonama koje su obeležene za dopremanje, biće zadržana samo po jedna kopija svakog reda.

U sekciji za odabir imena kolona iz tabela iz kojih se podaci dopremaju može da bude napisan simbol * (asterisk/zvezdica) kojim se serveru saopštava da treba da dostavi vrednosti za sve kolone svih tabela korišćenih u upitu. U suprotnom, moguće je navesti tačan popis imena polja (kolona) iz različitih tabela čije vrednosti treba da budu dopremljene u konačnom rezultatu upita.

Ukoliko se upit vrši nad više tabela, ime polja se prefiksuje imenom tabele praćenim simbolom . (tačka), npr. location.name ili supplier.address ako je potrebno dostaviti ime lokacije, a adresu dobavljača. Ako bi se samo navelo ime polje, npr. name i address, nastala bi nejasna situacija da li se misli na name i address iz tabele location ili iz tabele supplier, ukoliko su i jedna i druga tabela korišćene u upitu.

U sekciji FROM klauzule se navodi spisak tabela nad kojima se vrši upit. Prilikom navođenja imena tabela, moguće je određenim tabelama odrediti lokalni alijas. Lokalni alijasi se koriste u situacijama kada treba više puta u upitu da se iskoristi ista tabela sa različitim pravilima za filtriranje, ili kada postoje tabele sa dugačkim imenom, pa umesto korišćenja punog imena invoice_article, moguće je koristiti, npr. alijas ia itd.

U sekciji WHERE klauzule se navodi logički izraz koji služi za filtriranje vrednosti za konačan set rezultata. Logički izrazi u WHERE klauzuli mogu da budu izuzetno složeni. Kompletan pregled svih mogućih varijacija je dostupan u zvaničnoj MySQL dokumentaciji u sekciji WHERE klauzule. U primerima u nastavku je prikazano nekoliko najčešćih varijanti koje u praksi mogu da se koriste.

U sekciji GROUP BY klauzule se određuje spisak kolona po kojima treba obaviti grupisanje. Grupisanje se najčešće koristi uz funkcije agregacije koje mogu da se koriste u SELECT klauzuli i u WHERE klauzuli.

U sekciji HAVING klauzule se navodi logički izraz koji služi za naknadno filtriranje vrednosti za konačan set rezultata nad setom dobijenim nakon grupisanja izvršenog izrazom u GROUP BY klauzuli.

U sekciji ORDER BY klauzule se navodi spisak polja ili izraza na osnovu kojih treba da se obavi konačno sortiranje seta rezultata u određeni poredak, npr. rastući, opadajući ili prema posebno određenom pravilu. Moguće je navesti više kriterijuma za sortiranje rezultata, s tim da prvi navedeni ima prioritet, dok se ostali koriste kada postoje dva reda sa istom vrednošću u polju većeg prioriteta u izrazu sortiranja.

U sekciji LIMIT klauzule se navodi opseg od-do ili ukupan broj zapisa koji treba da budu vraćeni iz konačnog seta rezultata dobijenog nakon svih prethodnih filtriranja, grupisanja, sekundarnog filtriranja i sortiranja seta rezultata.

Dopremanje svih kolona iz svih redova jedne tabele

Često postoji potreba da se dopreme svi podaci iz jedne tabele. Primer upita:

```
SELECT *
FROM payment_type;
```

Upitom je traženo dopremanje svih kolona (*) iz tabele payment_type, bez filtriranja, grupisanja ili promene podrazumevanog poretka upisanih zapisa u tabeli.

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

payment_type_id	created_at	name	is_active
1	2018-03-25 09:14:59	Gotovinsko plaćanje	1
2	2018-03-25 09:14:59	Plaćanje platnom karticom	1
3	2018-03-25 09:14:59	Plaćanje čekovima	1
4	2018-03-25 09:14:59	Kupovina preko vaučera	1

Dopremanje određenih kolona iz svih zapisa jedne tabele

Ukoliko postoji potreba da se dopreme samo određene kolone, tj. vrednosti iz tačno određenih polja jedne tabele, moguće je koristiti upit kao u primeru u nastavku:

```
SELECT
  payment_type_id, name
FROM
  payment_type;
```

Ovakvim upitom zahtevano samo kolone payment_type_id i name iz tabele payment_type, bez dodatnog filtriranja, grupisanja ili promene podrazumevanog poretka upisanih zapisa u tabeli:

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

payment_type_id	name
1	Gotovinsko plaćanje
2	Plaćanje platnom karticom
3	Plaćanje čekovima
4	Kupovina preko vaučera

Dopremanje podataka iz svih zapisa jedne tabele sa preimenovanjem polja

Ukoliko postoji potreba da se dopreme samo određene kolone, tj. vrednosti iz tačno određenih polja jedne tabele, a da uz to određena polja budu preimenovana u setu rezultata u odnosu na to kako se ta polja originalno zovu u tabeli, moguće je koristiti upit kao u primeru u nastavku.

```
SELECT
  payment_type_id AS id,
  name as ime
FROM
  payment_type;
```

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

id	ime
1	Gotovinsko plaćanje
2	Plaćanje platnom karticom
3	Plaćanje čekovima
4	Kupovina preko vaučera

Dopremanje podataka iz samo jedne tabele uz filtriranje rezultata

Kada postoji određeni kriterijum za filtriranje zapisa koje je potrebno dopremiti iz tabele u bazi podataka, potrebno je da se koristi WHERE klauzula u kojoj je naveden logički izraz na osnovu kojeg će biti doneta odluka da li će određeni zapis da se nađe u konačnom setu rezultata ili ne. Ako je rezultat izvršavanja logičkog izraza TRUE, zapis će se naći u konačnom setu rezultata, a ako je FALSE, zapis će biti izuzet iz konačnog seta rezultata nakon prvog koraka filtriranja WHERE klauzulom. Primer upita:

```
SELECT
  feature_id,
  name
FROM
  feature
WHERE
  feature_type_id = 2;
```

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

feature_id	name
5	Materijal
6	Oblik

Prilikom filtriranja, zadržani su samo oni zapisi za koje je važio zadati uslov da vrednost polja feature_type_id bude jednaka 2.

Ako se pogleda spisak zapisa koji postoje u tabeli feature sa svim njenim poljima, visi se da samo za zapise koji su istaknuti važi da je zadati logički uslov ispunjen.

feature_id	created_at	name	feature_type_id
1	2018-03-25 15:34:43	Širina	1
2	2018-03-25 15:34:47	Visina	1
3	2018-03-25 15:34:51	Dubina	1
4	2018-03-25 15:34:54	Obim	1
5	**2018-03-25 15:35:29**	**Materijal**	**2**
6	**2018-03-25 15:35:34**	**Oblik**	**2**
7	2018-03-25 15:32:41	Garancija	3

Dopremanje podataka iz više tabela - spajanje WHERE klauzulom

Kada postoji potreba da budu prikazani podaci iz tabele koja je vezana, npr. relacijom jedan prema više sa određenom tabelom u setu rezultata, moguće je povezati dodatnu tabelu u upit na više načina. Jedan način je korišćenjem WHERE klauzule za izričito spajanje stranog ključa domaće i primarnog ključa strane tabele.

Primer upita:

```
SELECT
  feature.*,
  feature_type.name AS feature_type_name
FROM
  feature, feature_type
WHERE
  feature.feature_type_id = feature_type.feature_type_id;
```

U gore prikazanom upitu, u SELECT sekciji zatraženo je dopremanje svih kolona iz tabele feature (feature.*) i samo polje name iz tabele feature_type, ali uz istovremeno preimenovanje kolone u feature_type_name. U WHERE klauzuli izričito je potvrđena veza između tabela feature i feature_type preko izraza koji je tačan (TRUE) samo u situacijama kada se vrednost stranog ključa u tabeli feature poklapa sa vrednošću primarnog ključa u tabeli feature_type.

Kao rezultat ovakvog upita dobijen je set rezultata kao u tabeli u nastavku.

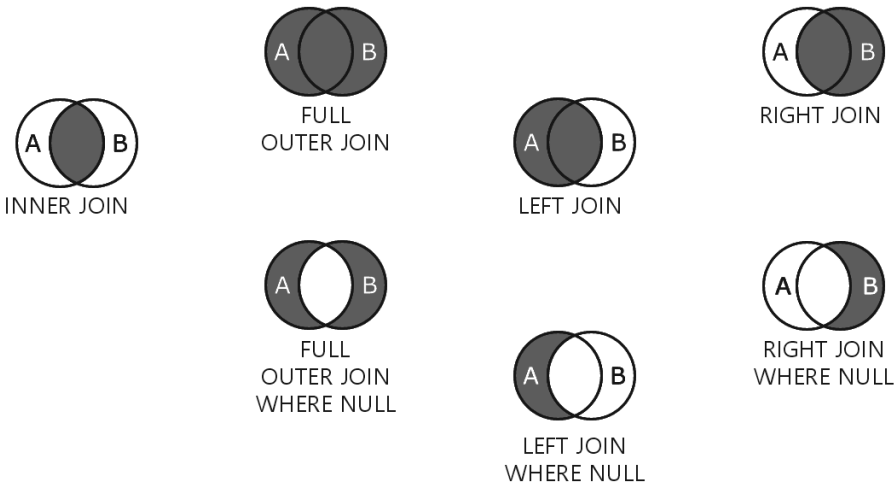
feature_id	created_at	name	feature_type_id	feature_type_name
1	2018-03-25 15:34:43	Širina	1	**Dimenzije**
2	2018-03-25 15:34:47	Visina	1	**Dimenzije**
3	2018-03-25 15:34:51	Dubina	1	**Dimenzije**
4	2018-03-25 15:34:54	Obim	1	**Dimenzije**
5	2018-03-25 15:35:29	Materijal	2	**Izrada**
6	2018-03-25 15:35:34	Oblik	2	**Izrada**
7	2018-03-25 15:32:41	Garancija	3	**Opšte informacije**

Dopremanje podataka iz više tabela - spajanje JOIN klauzulom

SQL jezik predviđa postojanje više vrsta JOIN mehanizama, među kojima su i:

- INNER JOIN
- OUTER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Određeni JOIN mehanizmi nisu podržani u MySQL implementaciji.



Primeri SQL upita za dopremanje podataka koji demonstriraju načine izvršavanja navedenih tipova JOIN mehanizama su prikazani u nastavku. Za ilustraciju primera navedenih u nastavku su korišćene tabele iz modela koji se obrađuje u ovom poglavlju.

INNER JOIN

```
SELECT
  invoice.invoice_id,
  invoice.client_id invoice_client_id,
  shipment.shipment_id,
  shipment.client_it shipment_client_id,
  shipment.shipping_code,
  shipment.status
FROM invoice
INNER JOIN shipment ON invoice.invoice_id = shipment.invoice_id;
```

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

invoice_id	invoice_client_id	shipment_id	shipment_client_id	shipping_code	status
1	1	1	1	MT-2008213514-US	sent

LEFT JOIN

```
SELECT
  invoice.invoice_id,
  invoice.client_id invoice_client_id,
  shipment.shipment_id,
  shipment.client_it shipment_client_id,
  shipment.shipping_code,
  shipment.status
FROM invoice
LEFT JOIN
  shipment ON invoice.invoice_id = shipment.invoice_id;
```

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

invoice_id	invoice_client_id	shipment_id	shipment_client_id	shipping_code	status
1	1	1	1	MT-2008213514-US	sent
2	1	**NULL**	**NULL**	**NULL**	**NULL**

Kod LEFT JOIN mehanizma, biće prikazani svi zapisi iz "leve" tabele, sa kojom se vrši spajanje druge, a iz "desne", koja se spaja, biće prikazane vrednosti zapisa samo ako takvi zapisi postoje, dok će u suprotnom biti zamenjeni sa NULL.

RIGHT JOIN

```
SELECT
  procurement.procurement_id,
  procurement.supplier_id,
  procurement.location_id,
  procurement.article_id,
  procurement.quantity,
  supplier.name
FROM procurement
RIGHT JOIN
  supplier ON procurement.supplier_id = supplier.supplier_id;
```

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

procurement_id	supplier_id	location_id	article_id	quantity	name
1	1	2	1	5	Dobavljač 1
2	1	1	1	10	Dobavljač 1
3	1	2	2	2	Dobavljač 1
4	1	3	2	4	Dobavljač 1
5	1	3	3	12	Dobavljač 1
6	1	1	3	2	Dobavljač 1
NULL	**NULL**	**NULL**	**NULL**	**NULL**	Dobavljač 2

Kod RIGHT JOIN mehanizma, biće prikazani svi zapisi iz "desne" tabele koja se spaja sa "levom" iz koje će biti prikazane vrednosti polja zapisa samo ako takva veza postoji, dok će u suprotnom za polja iz leve tabele biti prikazane vrednosti NULL.

FULL OUTER JOIN

S obzirom da MySQL ne podržava FULL OUTER JOIN mehanizam, potrebno je izvršiti uniju LEFT i RIGHT JOIN rezultata kako bi se postigao željeni rezultata.

Napomena: *Modeli baze podataka najčešće mogu da budu rešeni na taj način da ne postoji potreba za FULL OUTER JOIN mehanizmom spajanja. Model baze podataka na kojem se zasnivaju primeri u ovoj knjizi nije napravljen tako da može najjasnije da se demonstrira primer ovog mehanizma spajanja, jer je u njemu garantovana sveza između tabela postojanjem NOT NULL postavke za strane ključeve.*

```
SELECT
  invoice.invoice_id,
  invoice.client_id invoice_client_id,
  shipment.shipment_id,
  shipment.client_it shipment_client_id,
  shipment.shipping_code,
  shipment.status
FROM invoice
LEFT JOIN shipment ON invoice.invoice_id = shipment.invoice_id

UNION

SELECT
  invoice.invoice_id,
  invoice.client_id invoice_client_id,
  shipment.shipment_id,
  shipment.client_it shipment_client_id,
  shipment.shipping_code,
  shipment.status
FROM invoice
RIGHT JOIN shipment ON invoice.invoice_id = shipment.invoice_id;
```

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

invoice_id	invoice_client_id	shipment_id	shipment_client_id	shipping_code	status
1	1	1	1	MT-2008213514-US	sent
2	1	**NULL**	**NULL**	**NULL**	**NULL**

Primer korišćenja JOIN mehanizma za spajanje tabela koje su u relaciji, a koji daje isti set rezultata kao poslednji primer sa WHERE klauzulom, u situaciji kada je polje stranog ključa definisano kao NOT NULL i postoji garancija da ima vrednost je ilustrovan naredbom u nastavku.

```
SELECT feature.*, feature_type.name AS feature_type_name
FROM feature
JOIN feature_type ON feature.feature_type_id = feature_type.feature_type_id;
```

U gore prikazanom upitu, iskorišćen je JOIN mehanizam koji je naveden odmah iza imena tabele sa kojom se vrši spajanje.U ovom slučaju, to je tabela feature. Odmah iza imena tabele koja se sa njom spaja, iza rezervisane reči ON, naveden je logički izraz, poput onog u WHERE klauzuli iz prethodnog primera, kojim se utvrđuje kriterijum po kojem se sparuju polja iz dve tabele.

8.4.21 Spajanje setova rezultata - UNION

Moguće je spajanje setova rezultata dve ili više naredbi za dopremanje podatka korišćenjem rezervisane reči UNION. Prilikom spajanja setova rezultata, neophodno je voditi računa o tome da broj kolona setova rezultata svih upita koji se spajaju bude isti.

Primer SQL upita sa korišćenjem UNION mehanizma je prikazan u primeru **FULL OUTER JOIN** u prethodnoj sekciji ovog poglavlja.

8.4.22 Filtriranje rezultata - WHERE

Osim za povezivanje tabela navedenih u FROM klauzuli, WHERE rezervisana reč ima osnovnu primenu da obezbedi mehanizam za filtriranje rezultata upita.

WHERE se koristi za primarno filtriranje prilikom dopremanja podataka upitima.

Dopremanje podataka iz više tabela uz filtriranje rezultata

Prilikom spajanja više tabela, kao u prethodnom primeru, uz potrebu da se izvrši filtriranje na osnovu polja iz tabele koja je pridružena upitu spajanjem JOIN ili na FROM+WHERE način, to je moguće izvršiti upitom prikazanim u nastavku.

```
SELECT
  feature.feature_id,
  feature.created_at,
  feature.name,
  feature.feature_type_id
FROM feature
JOIN feature_type ON feature.feature_type_id = feature_type.feature_type_id
WHERE feature_type.name = "Dimenzije";
```

U prethodno prikazanom upitu, u setu rezultata se vraćaju samo podaci iz tabele feature, ali se za filtriranje koristi tekst naziva vrste osobine (feature_type.name) koji je izričito definisan kao vrednost "Dimenzije".

Taj podatak u setu rezultata neće biti prikazan, ali je korišćen umesto ID broja vrste osobine, kada korisniku taj broj nije poznat, dok naziv vrste osobine jeste.

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

feature_id	created_at	name	feature_type_id
1	2018-03-25 15:34:43	Širina	1
2	2018-03-25 15:34:47	Visina	1
3	2018-03-25 15:34:51	Dubina	1
4	2018-03-25 15:34:54	Obim	1

Dopremanje podataka iz više tabela uz filtriranje rezultata složenim izrazom

Kada postoji složeniji logički izraz na osnovu kojeg treba da budu filtrirani zapisi iz jedne tabele ili više povezanih tabela, moguće je koristiti logičke operatore unutar WHERE klauzule. Primer ovakvog upita je dat u nastavku.

```
SELECT
  feature.*,
  feature_type.name AS feature_type_name
FROM feature
JOIN feature_type ON feature.feature_type_id = feature_type.feature_type_id
WHERE feature_type.name = "Dimenzije" AND LENGTH(feature.name) < 5;
```

U prikazanom upitu, u WHERE klauzuli je napisan logički izraz koji je ispunjen samo u situaciji kada za određeni zapis seta rezultata važi da je ime vrste osobine jednako tekstu "Dimenzije" i da je dužina imena same osobine manje od 5 karaktera.

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

feature_id	created_at	name	feature_type_id	feature_type_name
4	2018-03-25 15:34:54	Obim	1	Dimenzije

Neki od operatora poređenja koji su dostupni za upotrebu u logičkim izrazima su prikazani u nastavku.

Operator	Značenje	Primer izraza
=	jednako	forename = "Milan"
<=>	jednako (toleriše NULL)	index <=> "2008/213514"
!= ili <>	nejednako	price != 0 ili price <> 0
<	manje	result < 1203
<=	manje ili jednako	result <= 13
>	veće	score > 51
>=	veće ili jednako	score >= 51
IS	je (logička vrednost)	is_visible IS TRUE
IS NOT	nije (logička vrednost)	is_visible IS NOT TRUE
IS NULL	je NULL	activation_code IS NULL
IS NOT NULL	nije NULL	activation_code IS NOT NULL
BETWEEN	u opsegu	age BETWEEN 60 AND 90
IN	u spisku	gender IN ("male", "female")
NOT IN	nije u spisku	banned_id NOT IN (3, 45, 112)
LIKE	je po šablonu	title LIKE "%test%"
NOT LIKE	nije po šablonu	details NOT LIKE "%banned%"
RLIKE	odgovara regularnom izrazu	index RLIKE "^20[0-9]{8}$"
NOT RLIKE	ne odgovara regularnom izrazu	value NOT RLIKE "^[0-9]+\.$"

Operator RLIKE koristi regularne izraze za proveru vrednosti i vraća logičku vrednost TRUE ili FALSE. Operator LIKE koristi posebne šablonske izraze definisane u SQL jeziku, a kao rezultat takođe vraća logičku vrednost TRUE ili FALSE.

Operator BETWEEN može da se koristi, kako za utvrđivanje pripadanja vrednosti opsegu numeričkih vrednosti, tako i za vrednosti vremenskih tipova podataka, kao što su datumi, vreme, vremenski žigovi itd.

Prilikom pisanja složenog logičkog izraza, dozvoljeno je korišćenje zagrada za izričito određivanje prioriteta operacija. Takođe, dozvoljeno je korišćenje logičkih operatora AND, OR, XOR i NOT koji imaju tautološka značenja logičkih operatora i, ili, ekskluzivno ili i ne, respektivno.

Primer SQL upita sa složenim logičkim izrazom u WHERE klauzuli

```
SELECT
  payment_id, payment_type_id, amount
FROM
  payment
WHERE
  (amount <  200 AND payment_type_id  = 1) OR
  (amount >= 200 AND payment_type_id != 1);
```

U prikazanom upitu, u WHERE klauzuli je napisan logički izraz koji je ispunjen samo u situaciji kada za određenu uplatu važi da je iznos manji od 200 i da je u pitanju uplata gotovinom ili da je iznos veći ili jednak od 200, ali da je u pitanju bilo koja druga vrsta plaćanja osim gotovinsko.

Kao rezultat ovakvog upita nastaje set rezultata kao u tabeli u nastavku.

payment_id	payment_type_at	amount
1	1	60
2	4	200

8.4.23 Grupisanje rezultata i funkcije agregacije - GROUP BY

Funkcije agregacije vraćaju jednu vrednost nakon obrade seta vrednosti. Najčešće se koriste unutar klauzula SELECT i GROUP BY.

Između ostalih, MySQL podržava funkcije agregacije prikazane u nastavku.

Funkcija	Opis funkcije
AVG	Vraća prosek obrađenih vrednosti
MIN	Vraća najmanju od obrađenih vrednosti
MAX	Vraća najveću od obrađenih vrednosti
SUM	Vraća sumu obrađenih vrednosti
COUNT	Vraća broj zapisa u vraćenom rezultatu
COUNT(DISTINCT)	Vraća broj jedinstvenih zapisa u vraćenom rezultatu
STDDEV	Vraća standardnu devijaciju obrađenih vrednosti
STDDEV_SAMP	Vraća uzorak standardne devijacije obrađenih vrednosti
VAR_POP	Vraća standardnu varijansu obrađenih vrednosti
VAR_SAMP	Vraća uzorak standardne varijanse obrađenih vrednosti
GROUP_CONCAT	Vraća tekstualni zapis dobijen spajanjem vrednosti
BIT_AND	Vraća logičko "i" (AND) bitova obrađenih vrednosti
BIT_OR	Vraća logičko "ili" (OR) bitova obrađenih vrednosti
BIT_XOR	Vraća ekskluzivno ili (XOR) bitova obrađenih vrednosti

U nastavku su prikazani primeri SQL upita sa funkcijama agregacije.

Suma evidentiranih uplata, grupisanih po računu

```
SELECT
  invoice_id,
  SUM(amount) amount_paid
FROM payment
GROUP BY
  invoice_id;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

invoice_id	amount_paid
1	260.00
2	7780.00

Suma evidentiranih uplata, grupisanih po godini i mesecu uplate

S obzirom na to da se u bazi podataka, u tabeli payment beleži datum i vreme kao jedan podatak vremenskog tipa, neophodno je izdvojiti informaciju o godini i mesecu za svaki zapis i te podatke koristiti kao osnov za grupisanje, ali i kao informacije koje se u setu rezultata dopremaju.

Funkcije za izdvajanje informacija o podatku vremenskog tipa su date u nastavku.

Ime funkcije	Opis	Primer i rezultat
DATE	Datum iz žiga	DATE("2018-03-25 11:22:55") = 2018-03-25
TIME	Vreme iz žiga	TIME("2018-03-25 11:22:55") = 11:22:55
YEAR	Godina iz datuma	YEAR("2018-03-25") = 2018
MONTH	Mesec iz datuma	MONTH("2018-03-25") = 3
DAY	Dan iz datum	DAY("2018-03-25") = 25
MONTHNAME	Ime meseca iz datuma	MONTHNAME("2018-03-25") = March
WEEKDAY	Dan u nedelji datuma	WEEKDAY("2018-03-25") = 6 (nedelja)
DAYNAME	Ime dana iz datum	DAYNAME("2018-03-25") = Sunday
DAYOFYEAR	Dan u godini datuma	DAYOFYEAR("2018-03-25") = 84
WEEKOFYEAR	Nedelja godine datuma	WEEKOFYEAR("2018-03-25") = 12
HOUR	Sat iz vremena	HOUR("21:09:22") = 21
MINUTE	Minut iz vremena	MINUTE("21:09:22") = 9
SECOND	Sekund iz vremena	SECOND("21:09:22") = 22

```
SELECT
  YEAR(created_at) AS year,
  MONTH(created_at) AS month,
  SUM(amount) amount_paid
FROM payment
GROUP BY year, month;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

year	month	amount_paid
2018	3	2205.00
2018	4	1945.00
2018	5	1945.00
2018	6	1945.00

Najmanji, najveći i ukupan iznos uplata, grupisan po načinu plaćanja

```
SELECT
  payment_type.name payment_type_name,
  MIN(payment.amount) AS minimum_amount,
  MAX(payment.amount) AS maximum_amount,
  SUM(payment.amount) AS total_amount
FROM
  payment
JOIN
  payment_type
    ON payment.payment_type_id = payment_type.payment_type_id
GROUP BY
  payment.payment_type_id;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

payment_type_name	minimum_amount	maximum_amount	total_amount
Gotovinsko plaćanje	60	1945	7840.00
Kupovina preko vaučera	200	200	200.00

Napomena: *Funkcije agregacije mogu da vrate NULL kao svoj rezultat. U takvim situacijama, ukoliko je ipak potrebno da rezultat umesto NULL bude neka druga, podrazumevana vrednost, moguće je korišćenje funkcije IFNULL koja obezbeđuje način da se umesto vrednosti NULL dopremi vrednost koja je izričito navedena ili koja je rezultat nekog izraza.*

Prikazati za sve artikle broj nabavljenih primeraka u evidenciji

```
SELECT
  article.article_id,
  article.name,
  IFNULL( SUM(procurement.quantity), 0 ) AS quantity_procured
FROM
  article
LEFT JOIN
  procurement
    ON article.article_id = procurement.article_id
GROUP BY
  article.article_id;
```

U prikazanom primeru, u situacijama kada nema evidentiranih nabavki, vrednost u koloni quantity_procured će umesto NULL biti 0, kao što je definisano drugim argumentom funkcije ISNULL. Prednost ovog primera je očigledna tek kada se doda još jedan artikal u tabelu article za koji se ne evidentira ni jedna nabavka niti prodaja.

8.4.24 Drugi stepen filtriranja - HAVING

HAVING klauzula ima primenu kada je potrebno da se izvrši naknadno filtriranje rezultata upita nakon obavljenog grupisanja korišćenjem GROUP BY mehanizma. HAVING klauzula se ponaša kao WHERE, s tim da može da se odnosi i na kolone iz set rezultata nakon GROUP BY, a ne samo na originalne vrednosti iz tabelâ nad kojima se upit vrši. Dodavanjem jednog HAVING uslova u nastavku poslednjeg primera koji je obrađen za GROUP BY klauzulu, nastaje upit kao u produžetku.

Najmanji, najveći i ukupan iznos uplata, grupisan po načinu plaćanja, gde je najmanji iznos uplate veći od 100 dinara

```
SELECT
  payment_type.name payment_type_name,
  MIN(payment.amount) AS minimum_amount,
  MAX(payment.amount) AS maximum_amount,
  SUM(payment.amount) AS total_amount
FROM payment
JOIN payment_type ON payment.payment_type_id = payment_type.payment_type_id
GROUP BY payment.payment_type_id
HAVING minimum_amount > 100;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

payment_type_name	minimum_amount	maximum_amount	total_amount
Kupovina preko vaučera	200	200	200.00

U odnosu na prvobitni primer, u kojem je set rezultata imao dva reda, u ovom primeru je zadržan samo jedan red rezultata i to onaj u kojem je vrednost za minimalni uplaćeni iznos veća od vrednosti 100.

Ukupan broja prodaja i ukupno prodatih artikala, grupisano po artiklu, za one artikle kojih je prodato više od 5

```
SELECT
  invoice_article.article_id,
  article.name,
  COUNT(invoice_article.article_id) AS number_of_sales,
  SUM(invoice_article.quantity) AS number_of_items_sold
FROM invoice_article
JOIN article ON article.article_id = invoice_article.article_id
GROUP BY invoice_article.article_id
HAVING number_of_items_sold > 5;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

article_id	name	number_of_sales	number_of_items_sold
1	Čaša - model T30	2	8.00

8.4.25 Mehanizam sortiranja rezultata - ORDER BY

Klauzula ORDER BY se koristi da se rezultat upita sortira u poredak definisan jednim ili sa više pravila. ORDER BY se koristi u SELECT upitima navođenjem seta kolona ili izraza na osnovu kojih je potrebno obaviti sortiranje, s tim da je moguće istaći da li da sortiranje bude u rastućem ili opadajućem poretku. Kada poredak sortiranja nije izričito naveden, podrazumeva se da bude rastući. Oznake za rastući i opadajući poredak su ASC i DESC, respektivno.

Izvod iz svedene sintakse SELECT naredbe koji se odnose na ORDER BY je:

```
[ORDER BY {col_name | expr} [ASC | DESC], ...]
```

U nastavku su primeri korišćenja ORDER BY mehanizma za sortiranje rezultata.

Spisak artikala i tekstualni zapis liste kategorija kojoj artikal pripada, poređanih po broju kategorija kojima artikal pripada, u opadajućem poretku

```
SELECT
  article.name,
  GROUP_CONCAT(category.name SEPARATOR ", ") AS categories
FROM
  article_category
JOIN article
  ON article.article_id = article_category.article_id
JOIN category
  ON category.category_id = article_category.category_id
GROUP BY
  article_category.article_id
ORDER BY
  COUNT(article_category.article_id) DESC;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

name	categories
Ogledalo sa svetlom M43	Osvetljenje, Za kuću i baštu, Za dnevni boravak
Čaša - model T30	Za kuhinju i trpezariju, Čaše
Maska za Huawei P20	Za telefone

U SQL upitu koji je prethodno prikazan, korišćena je funkcija GROUP_CONCAT za grupisanje vrednosti u tekstualni oblik razdvajanjem definisanim razdelnikom.

U istom upitu, funkcija agregacije COUNT je korišćena u ORDER BY klauzuli kao podatak na osnovu kojeg je potrebno obaviti sortiranje, uz isticanje da je potrebno da se koristi opadajući poredak (DESC).

Spisak vrsta osobina i osobina, sortiranih po imenu vrste, pa po imenu osobine

```
SELECT
  feature_type.name AS type,
  feature.name as name
FROM
  feature
JOIN feature_type ON feature.feature_type_id = feature_type.feature_type_id
ORDER BY
  feature_type.name,
  feature.name
```

U prethodno prikazanom SQL upitu, u ORDER BY klauzuli su navedena dva polja na osnovu kojih se vrši sortiranje. Poredak je podrazumevano rastući. Prvo se vrši sortiranje redova po vrednosti prvog navedenog polja, a zatim po vrednosti drugog navedenog polja. Sortiranje se leksikografsko prema engleskom alfabetu.

8.4.26 Ograničavanje eta rezultata - LIMIT

Klauzula LIMIT se koristi kada postoji potreba da iz seta rezultata budu zadržani samo određeni redovi. To je moguće obaviti na dva načina, primenom jednog od dva moda mehanizma za isecanje seta rezultata. Ti mehanizmi su:

- LIMIT sa ofsetom i ograničenjem i
- LIMIT sa ograničenjem, bez ofseta.

Primer upita bez ograničavanja je prikazan u nastavku.

```
SELECT
  transfer_id,
  from__location_id,
  to__location_id,
  article_id,
  quantity
FROM transfer;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

transfer_id	from__location_id	to__location_id	article_id	quantity
1	2	3	1	4
2	1	2	1	2
3	2	3	2	1
4	2	1	2	1
5	3	1	3	2

LIMIT sa ofsetom i ograničenjem je mod LIMIT mehanizma koji podrazumeva definisanje dva argumenta u LIMIT klauzuli. Prvi argument je pozicija reda od kojeg treba iseći set rezultata, a drugi je broj redova koje treba zadržati od pozicije ofseta.

```
SELECT
  transfer_id,
  from__location_id,
  to__location_id,
  article_id,
  quantity
FROM transfer
LIMIT
  1, 3;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

transfer_id	from__location_id	to__location_id	article_id	quantity
2	1	2	1	2
3	2	3	2	1
4	2	1	2	1

LIMIT sa ograničenjem, bez ofseta je mod LIMIT mehanizma koji podrazumeva definisanje jednog argumenta u LIMIT klauzuli koji predstavlja broj redova koje treba zadržati od početka seta rezultata. Ovaj mod se ponaša isto kao mod sa ofsetom i ograničenjem, gde je ofset postavljen na vrednost 0.

```
SELECT
  transfer_id,
  from__location_id,
  to__location_id,
  article_id,
  quantity
FROM
  transfer
LIMIT
  3;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

transfer_id	from__location_id	to__location_id	article_id	quantity
1	2	3	1	4
2	1	2	1	2
3	2	3	2	1

8.4.27 Ugnježđeni upiti

SQL jezik podržava pisanje ugnježđenih upita. Ugnježđeni upiti se koriste kada postoji potreba da se na mestu nekog polja, izraza ili vrednosti dopremi podatak ili informacija do koje se dolazi posebnim upitom. Ugnježđene upite je moguće koristiti u svim klauzulama SQL naredbi, npr. u SELECT klauzuli, WHERE klauzuli itd.

Spisak artikala sa prikazanom ažurnom (najnovijom) cenom artikla

```
SELECT
  article.article_id,
  article.name,
  (
    SELECT
      article_price.price
    FROM
      article_price
    WHERE
      article_price.article_id = article.article_id
    ORDER BY
      article_price.created_at DESC
    LIMIT 1
  ) AS the_latest_price
FROM
  article;
```

Rezultat navedenog upita nad popunjenom bazom podataka je prikazan u nastavku.

article_id	name	the_latest_price
1	Čaša - model T30	130
2	Maska za Huawei P20	250
3	Ogledalo sa svetlom M43	6750

U SQL upitu koji je prethodno prikazan, u SELECT klauzuli su vrednosti za treću kolonu seta rezultata, imenovanu alijasom the_latest_price, dobijena izvršavanjem ugnježđenog upita prikazanom u nastavku.

```
SELECT article_price.price
FROM
  article_price
WHERE
  article_price.article_id = article.article_id
ORDER BY
  article_price.created_at DESC
LIMIT 1
```

Unutar ovog ugnježđenog upita, nalazi se referenca ka tabeli article i njenom polju article_id čija upotreba nije deklarisana unutar tog upita, već izvan ugnježđenog upita.

Kada se ugnježđeni upit koristi za dopremanje jedne vrednosti za jedno polje u jednom redu seta rezultata, neophodno je voditi računa da dimenzije seta rezultata ugnježđenog upita budu 1x1, tj. da set rezultata ima samo jednu kolonu i jedan red.

Da set rezultata ugnježđenog upita ima samo jednu kolonu, može da se osigura izričitim definisanjem samo jednog polja u SELECT klauzuli upita, a da set rezultata ima samo jedan red, može da se osigura korišćenjem LIMIT klauzule sa propisivanjem da je neophodno zadržati samo 1 zapis u setu, tj. samo jedan red.

8.4.28 Rad sa pogledima - VIEW

Prilikom modelovanja relacione baze podataka, zbog raznih nivoa normalizacije, često nastaje potreba da se u upitima spaja veći broj tabela kako bi bili dopremljeni svi potrebni podaci koji opisuju određeni entitet. Pogledi mogu u svojoj definiciji da pozivaju druge poglede kao izvore podataka.

Na primer, ako je potrebno da se dopremi spisak svih evidentiranih nabavki sa prikazom imena dobavljača, lokacije nabavke i imena artikala, neophodno je spojiti tri dodatne tabele sa tabelom nabavki kako bi sve navedene informacije bile dopremljene. Ako postoji potreba da se u raznim drugim upitima koriste ovi povezani podaci, umesto da se svaki put piše upit koji spaja četiri tabele pored onih koje su za taj upit specifične, moguće je da se napravi pogled kojim će sve informacije biti dostupne drugim upitima kroz poziv jednog entiteta, odnosno pogleda na originalni upit.

Za potrebe objašnjena, treba razmotriti upit prikazan u nastavku.

```
SELECT
  p.procurement_id,
  p.created_at AS procurement_created_at,
  p.quantity,
  p.transaction_number,
  l.location_id,
  l.name AS location_name,
  s.supplier_id,
  s.name AS supplier_name,
  s.email AS supplier_email,
  s.address AS supplier_address,
  s.phone AS supplier_phone,
  s.is_active AS supplier_is_active,
  a.article_id,
  a.name AS article_name,
  a.excerpt AS article_excerpt,
  a.details AS article_details
FROM
  procurement AS p
JOIN location AS l ON l.location_id = p.location_id
JOIN supplier AS s ON s.supplier_id = p.supplier_id
JOIN article  AS a ON a.article_id  = p.article_id;
```

Prikazani upit vraća spisak svih nabavki sa pridruženim dodatnim informacijama dopremljenim iz povezanih tabela. Određene kolone rezultata su imenovane alijasima zbog lakšeg utvrđivanja na koji podatak se kolona odnosi.

Ukoliko postoji potreba da se napravi pogled, ispred upita se dodaje deo naredbe:

```
CREATE VIEW view_name AS ...
```

Ime pogleda treba da poštuje konvenciju imenovanja koje se pridržavano, koja nalaže da se imenu pogleda dodaje prefiks view__.

Ukoliko postoji potreba da se obriše pogled, koristi se sledeća naredba:

```
DROP VIEW view_name;
```

Primeri naredbi za pravljenje pogleda su prikazani u nastavku.

Napraviti pogled koji pruža pregled detalja svih nabavki

```
CREATE VIEW view__procurement_details AS
SELECT
  p.procurement_id,
  p.created_at AS procurement_created_at,
  p.quantity,
  p.transaction_number,
  l.location_id,
  l.name AS location_name,
  s.supplier_id,
  s.name AS supplier_name,
  s.email AS supplier_email,
  s.address AS supplier_address,
  s.phone AS supplier_phone,
  s.is_active AS supplier_is_active,
  a.article_id,
  a.name AS article_name,
  a.excerpt AS article_excerpt,
  a.details AS article_details
FROM procurement AS p
JOIN location AS l ON l.location_id = p.location_id
JOIN supplier AS s ON s.supplier_id = p.supplier_id
JOIN article  AS a ON a.article_id  = p.article_id;
```

Nakon što je napravljen pogled na ovaj način, njega je moguće koristiti kao i bilo koji drugu tabelu u bazi podataka u upitima za dopremanje podataka. Prilikom izmene vrednosti u tabelama koje su korišćene u upitu od kojeg je napravljen pogled, podaci dobijenu upitom nad tim pogledom će uvek biti ažurni. Pogledi ne sadrže podatke koji su bili dostupni u tabelama u onom trenutku kada je pogled napravljen, već sadrže definiciju upita koji se pokreće svaki put kada se zahteva dopremanje podataka pogledom. Poglede treba koristiti oprezno, jer mogu izazvati loše performanse upita.

Napraviti pogled koji prikazuje ukupan broj nabavki za sve artikle

```
CREATE VIEW view__article_procurements AS
SELECT
  article.article_id,
  article.name,
  IFNULL(SUM(procurement.quantity), 0) AS quantity_procured
FROM article
LEFT JOIN
  procurement ON article.article_id = procurement.article_id
GROUP BY
  article.article_id;
```

Napraviti pogled koji prikazuje ukupan broj prodaja za sve artikle

```
CREATE VIEW view__article_sales AS
SELECT
  article.article_id,
  article.name,
  IFNULL(SUM(invoice_article.quantity), 0) AS quantity_sold
FROM article
LEFT JOIN
  invoice_article ON article.article_id = invoice_article.article_id
GROUP BY
  article.article_id;
```

Napraviti pogled koji prikazuje ukupan broj nabavki, prodaja, kao i broj artikala na stanju za sve artikle, korišćenjem referenci ka postojećim pogledima

```
CREATE VIEW view__article_stock AS
SELECT
  p.article_id,
  p.name,
  p.quantity_procured,
  s.quantity_sold,
  p.quantity_procured - s.quantity_sold AS quantity_on_stock
FROM
  view__article_procurements AS p
JOIN view__article_sales AS s
  ON s.article_id = p.article_id;
```

Primeri korišćenja pogleda su prikazani u nastavku.

Dopremiti sve informacije o nabavkama artikala od dobavljača koji u svojoj adresi sadrže slovo "o" i u broju transakcije niz cifra "318"

```
SELECT *
FROM
  view__procurement_details
WHERE
  supplier_address LIKE '%o%' AND
  transaction_number LIKE '%318%';
```

8.4.29 SQL naredba za izmenu podataka - UPDATE

Naredba za izmenu podataka se koristi kada postoji potreba da se promene vrednosti polja jednog ili više zapisa u tabeli baze podataka. Jednom naredbom može da se utiče na stanje više od jednog zapisa u tabeli.

Svedena sintaksa naredbe za izmenu podataka je prikazana u nastavku.

```
UPDATE [IGNORE] table_name
  SET col_name = {expr | DEFAULT} [, col_name = {expr | DEFAULT}] ...
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count];
```

Kada u **WHERE** klauzuli nije naveden uslov za filtriranje zapisa u kojima treba da se izvrše navedene izmene vrednosti polja, podrazumeva se da izmene nastanu u svim zapisima u tabeli. Prema tome, veoma je važno pažljivo koristiti UPDATE naredbu, jer u situacijama kada nije napravljena rezervna kopija iz koje podaci mogu da se vrate, postoji mogućnost trajnog uništavanja sadržaja polja tabele u slučaju izvršavanja upita bez filtera ili u slučaju pisanja pogrešnog uslova u WHERE klauzuli.

Unutar WHERE klauzule može da se koristi i ugnježdeni upit za dopremanje podataka potrebnih za preciznije identifikovanje zapisa nad kojima treba da bude primenjen proces promene vrednosti polja zapisa tabele.

ORDER BY klauzulom se može definisati redosled po kojem treba da usledi pravljenje izmena vrednosti polja zapisa identifikovanih za izmenu. U određenim situacijama, kada je vrednost koja se dodeljuje poljima dobijena kao rezultat izraza kod kojeg je redosled pozivanja važan, ORDER BY klauzulom može da se obezbedi sortiranje zapisa u adekvatan poredak pre primenjivanja izmene.

LIMIT klauzula se u naredbi UPDAE koristi kada je potrebno ograničiti izmenu na određeni broj zapisa, bez obzira na to što (filtriranjem WHERE klauzulom ili bez nje) postoji veći broj zapisa nad kojima bi inače bila primenjena izmena vrednosti polja.

Ukoliko postoji potreba da se za vrednost nekog polja postavi ona koja je definisana kao podrazumevana prilikom pravljenja tabele, moguće je na mesto izričito navedene vrednosti upisati rezervisanu reč DEFAULT.

Ukoliko polje nema definisanu podrazumevanu vrednost, vrednost će biti postavljena na NULL ukoliko nije polje definisano kao NOT NULL. Ako polje jeste definisano kao NOT NULL, biće postavljena podrazumevana inicijalna vrednost za top podatka tog polja, npr. prazan string za tekstualni tip, 0 za numerički itd.

Primeri korišćenja naredbe za izmenu podataka su prikazani u nastavku.

Obeležiti statusom pošiljku MT-2008213514-US kao poslatu

```
UPDATE shipment
SET status = 'sent'
WHERE shipping_code = 'MT-2008213514-US';
```

U prikazanoj SQL naredbi, izričito je navedeno u SET klauzuli da se vrednost sent upiše u polje status za sve one zapise tabele shipment, gde je vrednost polja shipping_code jednaka "MT-2008213514-US".

S obzirom da je shipping_code polje obeleženo indeksom koji garantuje jedinstvenu vrednost, nije potrebno korišćenje LIMIT klauzule, jer UNIQUE indeks garantuje da postoji najviše jedno polje koje odgovara zadatom kriterijumu filtriranja.

Izdvajanjem samo ugnježdenog upita, ostaje SQL upit u nastavku.

```
SELECT
  article_id
FROM
  invoice_article
GROUP BY
  article_id
HAVING
  COUNT(DISTINCT invoice_id) < 2
```

Rezultat izvršavanja samo izdvojenog ugnježdenog upita je prikazan u nastavku.

article_id
2
3

Prikazani primer je analogan sledećoj SQL naredbi u situaciji koja je opisana:

```
UPDATE
  article
SET
  details = CONCAT(details, " ** NIJE POPULARNO **")
WHERE
  article_id IN (2, 3);
```

Na mestu zagrada iza operatora IN u uslovu WHERE klauzule može se smatrati da su upisane vrednosti iz jedine kolone svih redova seta rezultata ugnježdenog upita, tj. vrednosti 2 i 3 u konkretnom primeru.

8.4.30 SQL naredba za brisanje podataka - DELETE

Naredbu za brisanje podataka se koristi kada postoji potreba da se obriše jedan ili više zapisa u tabeli. Jednom naredbom može da se obriše više od jednog zapisa.

SQL naredba za brisanje podataka podržava dva moda:

- Brisanje iz jedne tabele i
- Brisanje iz više tabela.

Prilikom **brisanja iz jedne tabele**, brišu sve svi zapisi te tabele koji odgovaraju određenom uslovu u WHERE klauzuli, ako je definisana ili svi zapisi, ako nije definisan uslov. Za brisanje može da se odredi redosled korišćenjem ORDER BY klauzule, kao i da se ograniči koliko zapisa obrisati, korišćenjem LIMIT klauzule.

Svedena sintaksa naredbe za **brisanje iz jedne tabele** je prikazana u nastavnu.

```
DELETE [IGNORE] FROM table_name
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

Prilikom **brisanja iz više tabela**, brišu se svi zapisi iz više tabela koji su upareni uslovom iz WHERE klauzule.

Kod brisanja iz više tabela, nije podržano sortiranje, tj. određivanje redosleda za brisanje, kao ni ograničavanje broja zapisa koji se brišu.

Podržane su dve varijante naredbe za brisanja iz više tabela:

- Brisanje zapisa iz tabela navedenih pre FROM klauzule i
- Brisanje zapisa iz tabela navedenih u FROM klauzuli.

Svedena sintaksa naredbe za brisanje zapisa iz više tabela navedenih pre FROM klauzule je prikazana u nastavku.

```
DELETE [IGNORE] table_name [, table_name] ...
FROM table_references
[WHERE where_condition]
```

Svedena sintaksa naredbe za brisanje zapisa iz više tabela navedenih u FROM klauzuli je prikazana u nastavku.

```
DELETE [IGNORE]
FROM table_name [, table_name] ...
USING table_references
[WHERE where_condition]
```

Napomena: *Zbog očuvanja integriteta podataka, preporučuje se da se podaci iz baze podataka nikada ne brišu. Umesto toga, preporučljivo je da se u modelu baze podataka u sve tabele doda logičko polje is_deleted sa podrazumevanom vrednošću 0 ili vremensko polje deleted_at koje je podrazumevano NULL. Brisanje vrednosti bi bilo simulirano postavljanjem vrednosti za ova dva polja na 1 ili na datum brisanja, respektivno. U takvim situacijama, potrebno je da se u svim upitima koji koriste tabele uvek vrši filtriranje zapisa po indeksiranom polju koje služi kao indikator da li je zapis obrisan ili ne, kako bi se postigao isti rezultat kao da je zapis zaista uklonjen iz baze, s tim da se na ovaj način garantuje integritet podataka i uvek postoji mogućnost da administrativni korisnik sa najvišim nivoom privilegija nad sistemom uvek može da stekne uvid u istorijske vrednosti koje regularnim korisnicima sistema nisu dostupne. Prilikom implementacije ovog mehanizma treba imati na umu eventualnu zakonsku regulativu koja zabranjuje trajno čuvanje podataka i davanje lažnih iskaza o njihovom uklanjanju. U takvim situacijama, treba propisati adekvatan način na koji je moguće zameniti osetljive podatke o korisnicima korišćenjem UPDATE naredbe, kako bi veza sa fizičkim ili pravnim licima bila trajno raskinuta, bez daljeg ugrožavanja njihovim prava, ali i bez štetnog uticaja na očuvanje integriteta podataka.*

Primeri korišćenja naredbe za brisanje podataka su prikazani u nastavku.

Obrisati sve prenose robe kod kojih je količina bila manja od 2

```
DELETE FROM
  transfer
WHERE
  quantity < 2;
```

Obrisati sve prenose robe koja pripada kategoriji koja sadrži u imenu telefon

```
DELETE transfer
FROM transfer, article
JOIN article_category ON article_category.article_id = article.article_id
JOIN category ON category.category_id = article_category.category_id
WHERE
  transfer.article_id = article.article_id AND
  category.name LIKE '%telefon%';
```

U prikazanoj SQL naredbi, korišćen je pristup gde se u FROM klauzuli navodi više tabela, koje su prvenstveno korišćene u WHERE klauzuli za precizno filtriranje zapisa koje treba obrisati, ali su obrisani samo zapisi iz tabele koja je navedena pre FROM klauzule. U ovom slučaju, obrisani su zapisi iz tabele transfer, ali ne i iz tabele article, jer ona nije navedena kao tabela iz koje je potrebno da zapisi budu obrisani.

8.4.31 Rad sa promenljivama

SQL jezik podržava dinamičku alokaciju memorije za privremene vrednosti koje je moguće čuvati u sesiji pod kojom se izvršava upit u vidu promenljivih.

Promenljive je moguće napraviti na dva načina:

- Izričito, SQL naredbom ili
- U naredbi za dopremanje podataka.

Ime promenljive se u određenim implementacijama prefiksuje drugim simbolima. Ime korisničke promenljive se u MySQL implementaciji prefiksuje simbolom @ (at).

Za pravljenja nove izričito definisane promenljive, koristi se sintaksa u nastavku.

```
SET @name = expr [, @name = expr] ...
```

Primeri pravljenja promenljive na ovaj način su prikazani u nastavku.

```
SET @clientId  = 1;
SET @articleId = 2;
SET @quantity  = 1;
SET @discount  = 0.75;
```

Za smeštanje rezultata u promenljivu iz kolone koja se obeležava za dopremanje u SELECT klauzulu naredbe za dopremanje podataka, ispred imena kolone ili izraza za dopremanje se dodaje prefiks određen po šablonu prikazanom u nastavku.

```
@name:={col_name | expr}
```

Primer pravljenja promenljive na ovaj način je prikazan u nastavku.

```
SELECT @price:=price
FROM article_price
WHERE article_id = 2
ORDER BY created_at DESC
LIMIT 1;
```

Prikazana SQL naredba doprema najnoviju cenu artikla sa ID brojem 2, ali ujedno smešta tu vrednost u promenljivu sa imenom @price.

Ukoliko je potrebno naknadno dopremiti sačuvanu vrednost promenljive, moguće je koristiti SELECT naredbu za dopremanje podataka sledeći način prikazan u nastavku.

```
SEELCT @price;
```

8.4.32 Rad sa transakcijama - TRANSACTION

Transakcije su mehanizmi kojima se obezbeđuje da se grupa SQL naredbi izvrši kao jedna celina. Transakcije najčešće obuhvataju dve ili više SQL naredbi koje treba da budu izvršene.

Tri osnovna cilja transakcija su:

- Da omoguće razdvajanje između većeg broja aplikacija koje istovremeno pristupaju istoj bazi podataka. Ovo se garantuje tako što se grupišu naredbe koje izvršava jedna aplikacija i izvršavaju se u celosti. Na taj način ne dolazi do situacije da druga aplikacija vidi delimično izmenjene podatke.
- Da obezbedi način za vraćanje baze podataka u prvobitno stanje u slučaju da dođe do greške prilikom izvršavanja bilo koje od naredbi u transakciji.
- Da osigura da u slučaju prekida rada usled hardverskog, softverskog ili drugih problema, baze podataka ne ostane u delimično izmenjenog stanja, što može da izazove probleme sa integritetom podataka i radom aplikacije.

Tri osnovne naredbe koje se koriste u radu sa transakcijama su:

- START TRANSACTION,
- COMMIT i
- ROLLBACK.

Transakcija mora da počne sa START TRANSACTION i da se završi ili sa COMMIT kada želimo da sačuvamo promene ili sa ROLLBACK kada želimo da vratimo stanje baze podataka na ono pre početka transakcije.

Primer korišćenja transakcija je prikazan u nastavku.

Klijentu sa ID brojem 1 je potrebno napraviti račun za kupovinu jednog komada artikla sa ID brojem 2 po ceni 75% cene u trenutku pravljenja računa

Da bi bilo moguće izvršiti ovakav zahtev bez menjanja modela baze podataka i dodavanja, npr. polja discount u tabelu invoice_article kojom bi se ukazalo na to da cenu artikla treba obračunati sa popustom, neophodno je obaviti sledeće korake:

- Dodati novu cenu za artikal 2, koja je 75% trenutne cene;
- Napraviti račun za klijenta 1;
- Dodati pod novi račun evidenciju kupovine jednog komada artikla 2;
- Dodati novu cenu za artikal 2, koja je jednaka staroj ceni, pre smanjenja.

Za izvršavanje ovakve procedure je neophodno osigurati da se u trenutku njenog izvršavanja ne može dodati novi račun za drugog klijenta i pod njim evidentirati artikal koji bi u tom slučaju mogao da bude dodat pod smanjenom cenom.

Garancija da će se svi koraci ove procedure završiti, zaključno sa vraćanjem cene na prvobitnu, sa početka izvršavanja procedure, je moguća korišćenjem transakcije.

```
/* Komentari se koriste samo da pojasne delove koda u listingu */

-- Započinje novu transakciju
START TRANSACTION;

-- Definise potrebne promenljive
SET @clientId  = 1;
SET @articleId = 2;
SET @quantity  = 1;
SET @discount  = 0.75;

-- Smešta iznos trenutne cene artikla 2 u promenljivu @price
SELECT @price:=price
FROM article_price
WHERE article_id = @articleId
ORDER BY created_at DESC
LIMIT 1;

-- Definise novu promenljivu cija vrednost je izrazunata izrazom
SET @newPrice = @price * @discount;
-- Dodaje novu cenu za artikal 2, koja je jednaka 75% stare
INSERT article_price SET article_id = @articleId, price = @newPrice;

-- Dodaje novi račun za klijenta 1
INSERT invoice SET client_id = @clientId;

-- Uzima podatak o ID broju novog računa korišćenjem LAST_INSERT_ID()
SELECT @lastInvoiceId:=LAST_INSERT_ID();

-- Pod ID broj novog računa dodaje 1 komad artikla 2, po trenutno važećoj ceni
INSERT invoice_article SET invoice_id=@lastInvoiceId,
article_id=@articleId, quantity=@quantity;

-- Dodaje novu cenu za artikal 2, koja je jednaka početnoj iz promenljive @price
INSERT article_price SET article_id = @articleId, price = @price;

-- Ukoliko nije došlo ni do jedne greške u prethodnim naredbama, čuva stanje baze
SELECT @newPrice;

-- Potvrdjuje promene nastale u transakciji
COMMIT;

/* Ukoliko nije pozvan COMMIT, novo stanje, nakon naredbi, nije sačuvano */

/* Ukoliko uslovnim grananjem unutar upita utvrdimo da postoji razlog zbog kojeg
   treba vratiti stanje baze podataka na ono pre početka transakcije, tada umesto
   izvršavanja COMMIT naredbe, treba da uzvršimo naredbu ROLLBACK koja će stanje
   baze podataka ostaviti nepromenjeno, tj. sve promene u tabelama koje su nastale
   unutar ove transakcije će biti poništene. */
```

Objašnjene pojedinačnih koraka prikazane transakcije je dato u komentarima koda.

8.4.33 Rad sa okidačima - TRIGGER

Okidači su mehanizam koji obezbeđuje da se pre ili nakon obavljene manipulacije podacima u tabelama izvrši grupa naredbi koju definiše korisnik.

Okidači podržavaju ukupno šest scenarija za izvršavanje i to:

- Izvršavanje pre dodavanja novog zapisa BEOFRE INSERT
- Izvršavanje pre izmene postojećeg zapisa BEFORE UPDATE
- Izvršavanje pre brisanja zapisa BEFORE DELETE
- Izvršavanje posle dodavanja novog zapisa AFTER INSERT
- Izvršavanje posle izmene postojećeg zapisa AFTER UPDATE
- Izvršavanje posle brisanja zapisa AFTER DELETE

MySQL podržava definisanje više od jednog okidača za isti scenario za izvršavanje nad istom tabelom. Međutim, u praksi se ova pogodnost ne koristi, već se umesto toga u jednom osnovnom okidaču za jedan scenario izvršavanja nad tabelom poziva grupa ugrađenih procedura, čime se obezbeđuje da se više aktivnosti izvrši za isti scenario izvršavanja okidača nad istom tabelom. Ime okidača treba da bude formirano prema pravilima konvencije imenovanja.

Svedena sintaksa naredbe za pravljenje okidača je prema MySQL uputstvu:

```
CREATE TRIGGER trigger_name { BEFORE | AFTER } { INSERT | UPDATE | DELETE}
ON table_name FOR EACH ROW BEGIN
    trigger_body
END
```

Telo okidača trigger_body su naredbe koje treba da se izvrše u datom scenariju.

Primer korišćenja transakcija je prikazan u nastavku.

Napravite okidač koji će se izvršavati pre unosa novog zapisa kojim se evidentira dodavanje artikla na račun, koji će sprečiti dodavanje na račun ukoliko je broj primeraka dodavanog artikla na stanju manji od količine čije dodavanje se unosom zapisa evidentira

```
CREATE TRIGGER trigger_invoice_article_bi BEFORE INSERT
ON invoice_article FOR EACH ROW
BEGIN
  IF (
    SELECT view__article_stock.quantity_on_stock
    FROM view__article_stock
    WHERE view__article_stock.article_id = NEW.article_id
  ) < NEW.quantity THEN
    SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = "Nema dovoljno komada na stanju!";
  END IF;
END
```

8.4.34 Ugrađene funkcije - FUNCTION

Funkcija predstavlja imenovanu grupu naredbi koja može da se poziva u drugim funkcijama i procedurama, kao i u samostalnim upitima nad bazom podataka.

Svaka funkcija je određena svojim imenom, spiskom parametara i tipom povratne vrednosti. Svaki parametar je određen imenom i tipom podatka. Imena funkcija nisu posebno ograničena konvencijom imenovanja, osim da treba da se pridržavaju opštih pravila za imenovanje elemenata baze podataka.

Svedena sintaksa naredbe za pravljenje funkcije je prema MySQL uputstvu:

```
CREATE FUNCTION
    function_name ([param_name type[,...]])
RETURNS type
BEGIN
    function_body
    RETURN value
END
```

Naredba za uklanjanje funkcije je prikazana u nastavku.

```
DROP FUNCTION [IF EXISTS] function_name;
```

Primeri pravljenja i korišćenja funkcije su prikazani u nastavku.

Napraviti funkciju koja vraća trenutnu cenu artikla čiji ID broj je parametar

```
CREATE FUNCTION
    get_current_article_price(article_id INT)
    RETURNS DECIMAL(10, 2)
BEGIN
    RETURN (
        SELECT article_price.price
        FROM article_price
        WHERE article_price.article_id = article_id
        ORDER BY article_price.created_at DESC
        LIMIT 1
    );
END
```

Prikazani kôd definiše funkciju koja uzima jedan parametar celobrojnog tipa i vraća podatak koji je realna brojčana vrednost. Funkcija kao svoj rezultat vraća podatak dobijen ugnježđenim upitom kojim se doprema vrednost polja cena za jedan najnoviji zapis iz tabele article_price gde je vrednost polja article_price te tabele jednaka vrednosti istoimenog parametra funkcije. Zbog toga što se i polje u tabeli i parametar funkcije zovu isto, u navedenom kodu upita je izričito postavljeno ime tabele iz koje je polje article_id koje se koristi za filtriranje.

Primer koda za pravljenje identične funkcije, bez potrebe za izričitim navođenjem tabele iz koje je polje article_id zahteva da se parametar funkcije imenuje drugačije od imena polja tabele, što je prikazano u nastavku.

```
CREATE FUNCTION
    get_current_article_price(for_article_id INT)
RETURNS DECIMAl(10, 2)
BEGIN
    RETURN (
        SELECT price
        FROM article_price
        WHERE article_id = for_article_id
        ORDER BY created_at DESC
        LIMIT 1
    );
END
```

Korišćenje funkcije za dopremanje trenutne cene artikla je prikazano u nastavku.

```
SELECT get_current_article_price(2);
```

8.4.35 Ugrađene procedure - PROCEDURE

Procedura predstavlja imenovanu grupu naredbi koja može da se poziva u drugim procedurama, ali i kao samostalna naredba. Svaka procedura je određena svojim imenom i spiskom parametara. Za razliku od funkcija, procedure ne vraćaju vrednost. Parametri procedure mogu da budu ulazni, izlazni ili istovremeno i ulazni i izlazni.

Svaki parametar je određen imenom i tipom podatka. Imena procedura nisu posebno ograničena konvencijom imenovanja, osim da treba da se pridržavaju opštih pravila za imenovanje elemenata baze podataka.

Svedena sintaksa naredbe za pravljenje procedure je prema MySQL uputstvu:

```
CREATE PROCEDURE
    procedure_name ([[ IN | OUT | INOUT ] param_name type [,...]])
BEGIN
    procedure_body
END
```

Naredba za uklanjanje procedure je prikazana u nastavku.

```
DROP PROCEDURE [IF EXISTS] procedure_name;
```

Primeri pravljenja i korišćenja procedure su prikazani u nastavku.

Napraviti proceduru za prodaju jednog artikla sa popustom klijentu

Kôd za ovu proceduru moguće je napisati na osnovu primera koda prikazanog u odeljku o transakcijama, uz izmenu načina dopremanja potrebnih ID brojeva i iznosa popusta koji se odobrava za izabrani artikal.

```
CREATE PROCEDURE sell_with_discount(
    IN for_client_id INT,            -- ID broj klijenta
    IN for_article_id INT,           -- ID broj artikla
    IN with_quantity DECIMAL(10, 2), -- Kolicina za prodaju
    IN with_discount DECIMAL(10,2)   -- Procenat popusta
)
BEGIN
    START TRANSACTION; -- Zapicinje novu transakciju za izvrsavanje ove procedure

    -- Smešta iznos trenutne cene artikla u promenljivu @price (korisi funkciju)
    SET @price = get_current_article_price(for_article_id);

    -- Definise novu promenljivu cija vrednost je izrazunata izrazom
    SET @newPrice = @price * with_discount;

    -- Dodaje novu cenu za artikaa, nakon primene popusta na cenu
    INSERT article_price SET article_id = for_article_id, price = @newPrice;

    -- Dodaje novi račun za klijenta
    INSERT invoice SET client_id = for_client_id;

    -- Uzima podatak o ID broju novog računa korišćenjem funkcije LAST_INSERT_ID()
    SET @lastInvoiceId = LAST_INSERT_ID();

    -- Pod ID broj novog računa dodaje trazenu kolicinu, po vazecoj ceni sa popustom
    INSERT invoice_article
      SET invoice_id = @lastInvoiceId,
          article_id = for_article_id,
          quantity = with_quantity;

    -- Dodaje novu cenu za artikal, koja je jednaka početnoj iz promenljive @price
    INSERT article_price SET article_id = for_article_id, price = @price;

    COMMIT; -- Potvrdjuje promene nastale u transakciji
END
```

Način poziva procedure je prikazan u nastavku.

```
CALL sell_with_discount(1, 1, 2, 0.50);
```

Napomena: *Procedure se pozivaju naredbom CALL iza koje sledi ime procedure i spisak parametara navedenih u zagradi iza imena. Parametri koji mogu da budu izričito definisane vrednosti, kao u datom primeru, ili imena promenljivih. Parametri procedure koji su označeni kao izlazni ili istovremeno kao izlazni i ulazni bi trebalo da budu promenljive kojima procedura može da dodeli vrednost tokom svog izvršavanja.*

8.4.36 Zakazivanje odloženih poslova - EVENT

Odloženi poslovi ili događaji su grupe naredbi koje su određene da se izvrše jednom ili u određenim vremenskim intervalima nakon određenog vremsnkog perioda i mogu da traju do tačno određenog zakazanog vremena.

Koriste se u situacijama kada je grupu naredbi potrebno pokretati periodično, u fiksnim vremenskim intervalima, ili nakon određenog vremenskog perioda, tj. odloženo u odnosu na trenutak definisanja.

Imena odloženih poslova nisu posebno ograničena konvencijom imenovanja, osim da treba da se pridržavaju opštih pravila za imenovanje elemenata baze podataka.

Svedena sintaksa naredbe za definisanje odloženih poslova je prema uputstvu:

```
DELIMITER ;;

CREATE EVENT [IF NOT EXISTS]
  event_name
ON SCHEDULE
    AT timestamp [+ INTERVAL interval] ... | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]
[ON COMPLETION [NOT] PRESERVE]
DO BEGIN
    event_body
END

DELIMITER ;

-- Definicija za "interval" je:

    quantity {YEAR | QUARTER | YEAR_MONTH  | MONTH    | WEEK       |  DAY |
              HOUR | MINUTE  | SECOND      | DAY_HOUR | DAY_MINUTE | DAY_SECOND |
              HOUR_MINUTE    | HOUR_SECOND | MINUTE_SECOND}
```

Razlog za redefinisanje simbola za razdvajanje naredbi (DELIMITER) u dupli simbol ; (tačka zarez) izvršen pre početka definisanja zakazanog posla je taj što je moguće da u telu zakazanog posla postoji više od jedne naredbe, koje se u SQL jeziku razdvajaju sa jednim simbolom ; (tačka zarez), pa je potrebno da neki drugi simbol bude korišćen za završetak naredbe definisanja zakazanog posla.

Naredba za uklanjanje procedure je prikazana u nastavku.

```
DROP EVENT [IF EXISTS] event_name;
```

Napomena: *Da bi se zakazani poslovi izvršavali, na MySQL serveru mora da bude uključen mehanizam za izvršavanje zakazanih poslova ili događaja. Ovaj mehanizam se aktivira ili deaktivira dodelom vrednosti globalnoj promenljivoj event_scheduler.*

Naredba za aktiviranje izvršavanja zakazanih poslova je prikazana u nastavnu.

```
SET GLOBAL event_scheduler = ON;
```

Primeri pravljenja i korišćenja zakazanog posla su prikazani u nastavku.

Napraviti zakazani posao koji će da se pokreće svakih 10 minuta i koji će najmanje prodavanom artiklu u tom trenutku da smanji cenu za 5%

```
-- Definisanje novog symbola razdvajanja, jer sledi više naredbi u spolu

DELIMITER ;;

CREATE EVENT
  event_smanji_cenu_najmanje_prodavanog_artikla
ON SCHEDULE
  EVERY 5 MINUTE
DO BEGIN

    SET @najmanje_prodavan_article_id = (
        SELECT
           article_id
        FROM
            view__article_sales
        ORDER BY
            quantity_sold ASC
        LIMIT 1
    );

    SET @trenutna_cena = get_current_article_price(@najmanje_prodavan_article_id);

    SET @nova_cena = @trenutna_cena * 0.95;

    INSERT article_price
        SET
            article_id = @najmanje_prodavan_article_id,
            price = @nova_cena;

END;;

-- Vraćanje starog symbola za razdvajanje naredbi

DELIMITER ;

-- 4F 69 6B 67 57 6D 45 67 62 57 39 71 64 53 42 7A 5A 58 4E 30
-- 63 6E 55 67 55 6E 58 46 76 6D 6C 6A 64 53 45 67 50 44 4D 3D
```

8.4.37 Često korišćene funkcije

Osim funkcija agregacije koje su izdvojene u odeljku GROUP BY klauzule i funkcija datuma koje su ranije navedene, MySQL implementacija SQL jezika podržava veliki broj dodatnih funkcija opšte i specifične namene koje su dostupne programerima i administratorima baze podataka. Za više informacija o načinima korišćenja pojedinačnih funkcija, preporučuje se uvid u zvaničnu dokumentaciju na Internetu.

Ime funkcije	**Opis funkcije**
NOW	Vraća trenutni vremenski žig servera.
CURDATE	Vraća trenutni datum servera.
CURTIME	Vraća trenutno vreme servera.
DATE_ADD	Dodaje određeni vremenski interval na vrednost datuma.
DATE_SUB	Oduzima određeni vremenski interval od vrednosti datuma.
DATEDIFF	Računa razliku između dva datuma u danima.
DATE_FORMAT	Prikazuje datum u definisanom formatu.
FORMAT	Vraća string dobijen na osnovu definisanog formata.
TRIM	Uklanja sve beline ispred i iza vidljivih karaktera u stringu.
LOWER	Pretvara sva velika slova u mala u datom stringu.
UPPER	Pretvara sva mala slova u velika u datom stringu.
SUBSTRING	Vraća deo stringa sa određene pozicije i broja karaktera.
REVERSE	Vraća okrenut string.
CONCAT	Vraća string dobijen spajanjem više tekstualnih vrednosti.
CONCAT_WS	Vraća string spajanjem tekstualnih vrednosti separatorom.
SIN	Vraća rezultat trigonometrijske funkcije sin.
COS	Vraća rezultat trigonometrijske funkcije cos.
ASIN	Vraća rezultat trigonometrijske funkcije asin.
ACOS	Vraća rezultat trigonometrijske funkcije acos.
ATAN	Vraća rezultat trigonometrijske funkcije atan.
TAN	Vraća rezultat trigonometrijske funkcije tan.
ATAN2	Vraća rezultat trigonometrijske funkcije atan za 2 argumenta.
COT	Vraća rezultat trigonometrijske funkcije ctan.
ABS	Vraća apsolutnu vrednost datog broja.
CEIL	Vraća broj zaokružen na višu celobrojnu graničnu vrednost.
FLOOR	Vraća broj zaokružen na nižu celobrojnu graničnu vrednost.
MOD	Vraća ostatak pri deljenju dva broja.
ROUND	Zaokružuje realnu vrednost na određeni broj decimala.
TRUNCATE	Odseca decimale iza određene pozicije, bez zaokurživanja.
CONV	Konvertuje zapis broja iz jedne osnovice (baze) u drugu.
DEGREES	Konvertuje ugao iz radijana u stepene.
RADIANS	Konvertuje ugao iz stepeni u radijane.
LN	Vraća vrednost prirodnog logaritma.
LOG	Vraća vrednost prirodnog logaritma.
LOG10	Vraća vrednost logaritma osnovice 10.

LOG2	Vraća vrednost logaritma osnovice 2.
SIGN	Vraća znak date vrednosti.
DIV	Koristi se za celobrojno deljenje.
EXP	Koristi se za stepenovanje.
POW	Vraća vrednost broja stepenovanog na određeni broj.
SQRT	Vraća kvadratni koren date vrednosti.
RAND	Vraća pseudo-nasumičnu realnu vrednost.
LENGTH	Vraća dužinu stringa u bajtovima.
ASCII	Vraća ASCII numeričku vrednost prvog karaktera.
CHAR	Vraća karakter koji predstavlja data vrednost.
SPACE	Vraća string sa definisanim brojem razmaka.
REPEAT	Umnožava dati string dati broj puta.
REPLACE	Menja pominjanje dela teksta u datom tekstualnom podatku.
STRCMP	Vrši poređenje dva tekstualna podatka.
BIN	Vraća string koji sadrži binarni prikaz određenog broja.
HEX	Vraća heksadecimalni prikaz datog broja.
OCT	Vraća oktalni zapis datog broja.
UNHEX	Vraća vrednost konvertovanu iz heksadecimalnog zapisa.
LPAD	Vraća tekst određene širine dopunom simbolima levo.
RPAD	Vraća tekst određene širine dopunom simbolima desno.
LTRIM	Uklanja beline ispred teksta.
RTRIM	Uklanja beline iza teksta.
LEFT	Vraća dati broj karaktera sa početka datog stringa.
RIGHT	Uklanja beline iza teksta.
CHAR_LENGTH	Vraća broj karaktera u datom stringu.
FROM_BASE64	Koristi se za dekodiranje teksta iz BASE64 kodiranog zapisa.
TO_BASE64	Koristi se za kodiranje teksta u BASE64 kodirani zapis.
MATCH	Obavlja tekstualnu pretragu u datom tekstualnom podatku.
LIKE	Obavlja uparivanje teksta po prostom šablonu.
RLIKE	Obavlja uparivanje teksta po regularnom izrazu.
ENCODE	Kodira string
DECODE	Dekodira string kodiran naredbom ENCODE
PASSWORD	Računa PASSWORD heš string datog teksta
SHA1	Računa SHA1 kontrolni string
SHA2	Računa SHA2 kontrolni string
INET_ATON	Konvertuje tekstualni zapis IP adrese u brojčanu vrednost
INET_NTOA	Konvertuje brojčanu vrednost zapisa IP adrese u tekst
IS_IPV4	Utvrđuje da li je dati argument ispravna IPv4 adresa
INET6_ATON	Konvertuje tekstualni zapis IPv6 adrese u brojčanu vrednost
INET6_NTOA	Konvertuje brojčanu vrednost zapisa IPv6 adrese u tekst
IS_IPV6	Utvrđuje da li je dati argument ispravna IPv6 adresa
UUID	Vraća jedinstveni univerzalni identifikator (UUID)
UUID_SHORT	Vraća smračeni jedinstveni univerzalni identifikator (UUID)
UUID_TO_BIN	Konvertuje tekstualni zapis UUID u binarni

8.4.38 Upravljanje korisnicima - USER MANAGEMENT

MySQL je sistem za upravljanje relacionim bazama podataka koji se sastoji iz serverske i klijentske komponente. Da bi se klijent uspešno povezao sa serverom, potrebno je da izvrši uspostavu veze. Tokom uspostave veze, on dostavlja korisničko ime i lozinku na osnovu kojih server otvara sesiju sa adekvatnim pravima.

MySQL podržava više izvora za uspostavu veze. Izvori zavise od javnih adresa na koje je vezan MySQL server, ukoliko je podešen za mrežni rad. Dva osnovna izvora koji se u praksi najčešće sreću kod MySQL servera podešenog za rad u mrežnom okruženju su: lokalni host i konkretna IP adresa iz opsega u kojem je IP adresa servera.

MySQL podržava mogućnost da se isto korisničko ime koristi sa više izvora uspostave veze ka serveru. Za svaku kombinaciju korisničkog imena i izvora uspostave veze je moguće postaviti zasebnu lozinku. Takođe, svakoj kombinaciji korisničkog imena i izvora uspostave veze ka serveru je moguće dodeliti drugačije setove prava nad bazama podataka i tabelama u tim bazama podataka.

Svedena sintaksa za pravljenje naloga korisnika je prema MySQL uputstvu:

```
CREATE USER
  [IF NOT EXISTS]
  user_and_source
[ IDENTIFIED BY 'password_string' ]
[ DEFAULT ROLE role [, role ] ... ]
[ WITH {
      MAX_QUERIES_PER_HOUR     count
    | MAX_UPDATES_PER_HOUR     count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS     count
  }
]
[ PASSWORD EXPIRE [ DEFAULT | NEVER | INTERVAL N DAY ] | ACCOUNT { LOCK | UNLOCK ] ]
```

Sintaksa za uklanjanje naloga korisnika je prikazana u nastavku.

```
DROP USER
  [IF EXISTS]
  user_and_source [, user_and_source]
  ...
```

Sintaksa ua preimenovanje naloga korisnika je prikazana u nastavku.

```
RENAME USER old_ user_and_source TO new_user_and_source
        [, old_ user_and_source TO new_user_and_source]
...
```

Sintaksa ua izmenu naloga korisnika je prikazana u nastavku.

```
ALTER USER
  [IF EXISTS]
  user_and_source
[ IDENTIFIED BY 'password_string' ]
[ WITH {
      MAX_QUERIES_PER_HOUR     count
    | MAX_UPDATES_PER_HOUR     count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS     count
  }
]
[ PASSWORD EXPIRE [ DEFAULT | NEVER | INTERVAL N DAY ] | ACCOUNT { LOCK | UNLOCK } ]
```

Sintaksa ua promenu lozinke naloga korisnika je prikazana u nastavku.

```
SET PASSWORD [FOR user_and_source] = 'password_string'
[ REPLACE 'current_ password_string' ]
```

Primeri naredbi za rad sa nalozima korisnika su prikazani u nastavku.

Napraviti nalog za korisnika "milan" kojem treba omogućiti pristup samo sa lokala i postaviti mu proizvoljan tekst lozinke

```
CREATE USER "milan"@"localhost"
IDENTIFIED BY "3 25 15 21 23 1";
```

Definisati da lozinka naloga "milan" koji pristupa sa lokala istekne za 14 dana

```
ALTER USER "milan"@"localhost" PASSWORD EXPIRE INTERVAL 14 DAY;
```

Obrisati nalog korisnika "milan" koji ima pristup severu samo sa lokala

```
DROP USER "milan"@"localhost";
```

Napraviti nalog za korisnika "milan" kojem treba omogućiti pristup samo sa IP adrese 192.168.88.24 i postaviti mu proizvoljan tekst lozinke

```
CREATE USER "milan"@"192.168.88.24" IDENTIFIED BY "tekst-lozinke--Xaidia-draconis";
```

8.4.39 Upravljanje pravima korisnika - PRIVILEGE

Smo pravljenjem naloga korisnika nije omogućeno korisnicima da upravljaju strukturom i sadržajem baze podataka. Kako bi se omogućio pristup određenoj bazi podataka, neophodno je odobriti pravo ili grupu prava korisniku. Osim toga, moguće je korisniku ukinuti pravo ili grupu prava nad bazom ili tabelama baze podataka.

Svedena sintaksa za odobravanje prava korisniku je prema MySQL uputstvu:

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON [{ TABLE | FUNCTION | PROCEDURE }]
    { * | *.* | db_name.* | db_name.table_name | table_name
    | db_name.function_name | db_name.procedure_name }
TO { user_and_source | role_name } [, user_and_source | role_name] ...
[ WITH GRANT OPTION ]
```

Svedena sintaksa za ukidanje prava korisnika je prema MySQL uputstvu:

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON [{ TABLE | FUNCTION | PROCEDURE }]
    { * | *.* | db_name.* | db_name.table_name | table_name
    | db_name.function_name | db_name.procedure_name }
FROM { user_and_source | role_name } [, { user_and_source | role_name }] ...
```

Primeri naredbi za rad sa pravima korisnika su prikazani u nastavku.

Dodeliti SELECT pravo nad svim tabelama baze kompanija korisniku "milan" koji pristupa sa lokala (de facto read-only pristup)

```
GRANT SELECT ON kompanija.* TO "milan"@"localhost";
```

Dodeliti korisniku "milan" koji pristupa sa lokala pravo da pregleda i dodaje podatke u tabeli article_price u bazi podataka kompanija

```
GRANT SELECT, INSERT ON kompanija.article_price TO "milan"@"localhost";
```

Oduzeti pravo korisniku "milan" koji pristupa sa IP adrese 192.168.88.24 prava dodavanja, izmene i brisanja podataka iz svih tabela baze kompanija

```
REVOKE INSERT, UPDATE, DELETE ON kompanija.* FROM "milan"@"192.168.88.24";
```

Dodeliti korisniku "milan" koji pristupa sa IP adrese 192.168.88.24 sva prava nad bazom podataka kompanija, uz dodatno pravo da dodeljuje druga prava

```
GRANT ALL ON kompanija.* TO "milan"@"192.168.88.24" WITH GRANT OPTION;
```

9 RELACIJE LOŠE STRUKTURE I NORMALIZACIJA

Najkompleksnija stvar u razvoju baza podataka je pravilno modelovanje. Cilj je da se iz realnog sveta koji se modeluje uoče objekti, njihove veze i odnosi, kao i ograničenja. U relacionom modelu podataka nakon modelovanja nastaju šeme relacija, koje imaju svoje nazive, odgovarjuće atribute, atributi imaju svoje domene u kojima su sadržana ograničenja nad pojedinim atributima, a između relacija postoje ograničenja koja se odnose na veze između primarnjih i spoljašnjih ključeva, što čini referencijalni integritet. Pravilan dizajn podrazumeva da dobijene šeme relacija imaju dobru strukturu. Dobra struktura šema relacija podrazumeva da se u toku održavanja baze podataka (unos, izmena, brisanje) ili kroz postavljene upite ne uočavaju nepogodnosti. Ukoliko se pri projektovanju dobiju šeme relacija loše strukture, moguće je da u toku rada sa bazom nastanu greške i pored ispravne primene SQL-a.

Dobre šeme relacija ispunjavaju sve zahteve relacionog modela podataka (unikatnost atributa u šemi relacije i unikatnost n-torki, uslov identifikacionog integriteta za primarni ključ, uslov referenciojalnog integriteta za spoljašnji ključ, primarni ključ je jedan jedini ili izabrani kandidat ključ, uvek postoji super ključ i sl.). U navedenim razmatranjima nema posebnih zahteva prema ostalim atributima, što može da dovede do pogrešnog zaključka da ostali atributi i njihove vrednosti mogu da budu proizvoljni. Ukoliko u šemama realacija postoje neke dodatne zavisnosti koje odstupaju od osnovnih postavki realacionog modela podataka (dodatne zavisnosti između drugih atributa ili zavisnosti između delova primarnog ključa i takvih atributa) nastaju tzv. relacije loših struktura. U radu sa takvim relacijama mogu da nastanu problemi koji mogu da dovedu do pogrešnih rezultata ili čak do gubitka podataka.

Navedeni problemi mogu da se razmatraju preko teorije funkcijskih zavisnosti. Iz ove teorije su definisane normalne fomre. Normalizacija je postupak kojim se dekomponuju (rastavljaju) šeme relacija loše strukture na dve ili više šema koje su u skladu sa željenom normalnom formom. Nakon normalizacije, u toku rada sa novodobijenim relacijama izbegavaju se tzv. anomalije, a kroz upite se dobijaju ispravne informacije na osnovu postojećih podataka u bazi.

Inicijalno su promovisane tri normalne forme, *prva (1NF), druga (2NF) i treća (3NF) normalna forma*. Kasnije su R.Boyce i E.F.Codd promovisali strožu definiciju treće normalne forme koja se naziva *Boyce-Codd normalna forma (BCNF)*. Sve normalne forme se baziraju na funkcionalnim zavisnostima između atributa tabele. Promovisane su i više normalne forme koje prevazilaze BCNF, i to su *četvrta (4NF)* i *peta (5NF) normalna forma*. Ipak, ove forme se bave situacijama koje su vrlo retke.

Tabela koja ne ispunjava zahteve normalizacije mora se rastaviti na dve ili više tabela od kojih svaka pojedinačno ispunjava zahteve normalizacije. Važno je napomenuti da je za kreiranje tabela u relacionom modelu podataka kritična samo 1NF.

Sve ostale forme su opcione. Ipak, da bi se izbegle anomalije ažuriranja, preporučuje se normalizacija najmanje do 3NF.

U najopštijem smislu, normalizacija je postupak kojim se proizvoljna nenormalizovana relacija transformiše u skup manjih normalizovanih relacija. U okviru normalizacije ne sme doći do gubitaka informacija sadržanih u relaciji. Drugim rečima, mora biti moguće rekonstruisati prethodne relacija na osnovu novodobijenih.

Dekompozicija šeme relacije R(A1,A2,...,An) je zamena šeme relacije R sa skupom šema relacija {R1, R2, ... , Rk} za koje važi $R_i \subset R$ i $R1 \cap R2 \cap ... \cap Rk = R$. Ako je r relacija zadata na R a relacije r1,r2, ... ,rk su dobijene projekcijama relacije r na skupove atributa R1, R2, ... , Rk onda se prirodnim spajanjem mora dobiti relacija r.

Normalizacija predstavlja sistemski metod za osiguravanje da je struktura baze podataka pogodna za upite opšteg tipa, da se smanji mogućnost redundanse podataka, da se smanji mogućnost pojave anomalija unosa, izmena i brisanja koje bi mogle da dovedu do gubitka podataka. Normalizacija je iterativni proces tokom koga se vrši reorganizacija baze podataka u cilju izbjegavanja redundanse i povećanja stabilnosti baze podataka. Normalizacijom baze podataka eliminišu se sledeći atributi :

- atributi koji sadrže više od jedne vrednosti,
- atributi koji su dupli ili se ponavljaju,
- atributi koji ne opisuju tabele,
- atributi koji sadrže redundantne podatke,
- atributi koji su izvedeni iz ostalih atributa.

Potpuna normalizacija nije obavezna ali je preporučljiva. Uzdržavanje od pune normalizacije obično podrazumeva predviđanje problema koji bi se mogli pojaviti. Proces normalizacije je zasnovan na hijerarhiji normalnih formi. Svaka normalna forma se zasniva na prethodnoj i definiše rešenje problema koji prethodnik nije pokrio. Prve tri normalne forme je, kao i pravila za uspostavljanje relacionog modela, ustanovio Ted Kod. Baza se smatra normalizovanom ukoliko poštuje pravila iz prve tri normalne forme. Sa druge strane, što višu NF baza podržava više je relacija u njoj, više JOIN operatora kod upita, zbog čega je teže doći do specifičnog podatka, a baza koristi više resursa da bi odgovorila na zahtev.

9.1 REDUNDANSA I ANOMALIJE AŽURIRANJA

Jedan od najvažnijih ciljeva prilikom projektovanja relacione baze podataka je pravilno grupisanje atributa po tabelama, što rezultuje minimalnim ponavljanjem podataka i smanjenjem prostora potrebnog za skladištenje fajlova u kojima se čuvaju podaci. Ponavljanje podataka je pojava da se isti podaci koji se odnose na neki entitet nepotrebno pojavljuju u dve ili više tabela. U tabelama koje sadrže podatke koji se

ponavljaju mogu da se jave problemi poznati kao višestruko unošenje, višestruko menjanje i višestruko uklanjanje. Pored toga, javljaju se takozvane anomalije unošenja i brisanja.

9.1.1 Višestruko unošenje podataka

Prilikom unosa novih podataka koji se odnose na jedan entitet, u tabelu koja sadrži podatke koji se ponavljaju moraju se uneti i podaci koji se odnose na druge entitete čiji se podaci nalaze u istoj tabeli, što može dovesti do nekonzistentnosti ako se načini greška prilikom unosa. Ako se jedan te isti podataka višestruko unosi mora da se vodi računa da uvek bude ispravno unesen (npr. latinični karakteri). Da bi se ovo ostvarilo neminovno se usložnjava deo softvera za unos podataka, postavljaju se odgovarajuće kontrole i sl.

9.1.2 Višestruko brisanje podataka

Brisanje poslednjeg zapisa koji se odnosi na jedan entitet iz tabele koja sadrži podatke koji se ponavljaju može dovesti do kompletnog gubitka podataka o drugom entitetu čiji se podaci nalaze u istoj tabeli, u istom zapisu. Ako se neki podatak u realciji ponavlja, a potrebno je da se obriše iz baze, neophodno je prvo pronaći na kojim se mestima (zapisima) pojavljuje, a tek tada izvršiti brisanje. Navedene aktivnosti takođe usložnjavaju kod aplikacije: umesto da se kroz SQL naredbe kaže ukloni odgovarajući zapis, prvo mora da se detektuje koji su to sve zapisi.

9.1.3 Višestruke izmene podataka

Prilikom izmene vrednosti atributa koji se odnosi na jedan entitet, u tabeli koja sadrži podatke koji se ponavljaju moraju se promeniti i svi zapisi koji sadrže taj promenjeni atribut, kao i podaci koji se odnose na druge entitete, a koji stoje u direktnoj vezi sa promenjenim atributom. Ako se ne izvrše sve ove promene, baza postaje nekonzistentna.

Primer 1:

Pretpostavimo da postoji šema relacije pod nazivom Ispit, koja obuhvata imena studenata, predmete koje su položili (svaki predmet ima svoju šifru) i dobijene ocene:

Ispit (BrInd, Ime, Prezime, SifP, Ocena).

Jedan student ima ocene iz više predmeta, a jedan predmet je polagalo više studenata. U navedenom primeru šema relacije Ispit ima lošu strukturu i poslužiće da se objasne sve njene slabosti. Neka je trenutni sadržaj relacije Ispit:

BrInd	Ime	Prezime	SifP	Ocena
100	Marko	Marković	BP	8
200	Petar	Petrović	BP	9
100	Marko	Marković	K1	7
300	Dragan	Marković	RM	10
100	Marko	Marković	RM	6

Osnovna mana relacije Ispit je redudansa (višestruko ponavljanje). Ovaj nedostatak izaziva probleme kod sva tri tipa ažuriranja:

- Višestruko unošenje: ime i prezime studenta unose se onoliko puta koliko je ispita polagao. Često se uz ime i prezime evidentiraju i neki drugi podaci o studentu (adresa, kontakt telefon, mejl i sl.), čime bi ovaj problem bio još izraženiji;
- Višestruko menjanje: eventualna promena imena studenta treba da bude sprovedena na svim mestima gde se pojavljuje;
- Višestruko uklanjanje: ako se želi poptuno uklanjanje podataka o studentu, vrši brisanjem redova i to onoliko puta koliko je ispita polagao.

Ako se pokuša izbegavanje unosa imena i prezimena studenata više puta (samo prvi put se upisuje, a u ostalim zapisima da se unosi NULL), gube se neke informacije koje bi trebale da se dobijaju preko upita. Na primer:

BrInd	Ime	Prezime	SifP	Ocena
100	Marko	Marković	BP	8
200	Petar	Petrović	BP	9
100	NULL	NULL	K1	7
300	Dragan	Marković	RM	10
100	NULLo	NULL	RM	6

U razmatranom slučaju nemoguće je postaviti sledeće upite:

- Pronaći sve studente koji koji su položili npr. RM (računarske mreže);
- Izdvojiti sve predmete koje je položio student Marko Marković i sl.

Navedene mane u relaciji Ispit su očigledne. Dovode do kompleksnosti kod ažuriranja, a eventualni upiti bi mogli da daju pogrešne rezultate. Međutim, u navedenoj relaciji loše strukture Ispit postoje dva drastična nedostatka:

- Anomalija unošenja: ne mogu se uneti podaci o studentu, a da se pri tome ne unesu podaci o bar jednom njegovom položenom ispitu;
- Anomalija uklanjanja: ukoliko je neki student položio samo jedan ispit, uklanjanjem podataka o tom jednom ispitu koji je polagao student, uklonili bi se i svi podaci o studentu (broj indeksa, eventualno adresa, kontakt telefon, mejl i sl.).

Anomalije unošenja i uklanjanja su očigledne u lošim šemama relacija kod baza za poslovne informacione sisteme. Ukoliko se u jednoj šemi realcije nađu i podaci o robi i podaci o njihovim dobavljačima (adrese, kontakti, telefoni, mejlovi i sl.) može se desiti da se ne može uneti kontakt za novog dobavljača sve dok on ne isporuči neku količinu robe. Takođe, ako je na lageru robe u nekom trenutku ostao samo jedan proizvod od tog dobavljača, njegovom prodajom (brisanjem sa lagera) mogu se ukloniti i svi kontakt podaci o tom dobavljaču.

U našem primeru, da bi šema relacije bila dobra, potrebno je da se imena studenata (i njegovi ostali podaci) unose samo jednom, nezavisno od podataka o ispitima i ocenama. Razmatrani problemi su posledica loše strukture u šemi relacije Ispit, kao i posebnih funkcijskih zavisnosti koje postoje između ključnih i neključnih atributa.

9.2 FUNKCIONALNA ZAVISNOST

Funkcionalna zavisnost opisuje odnose između atributa u tabeli i predstavlja jedan od glavnih koncepata vezanih za normalizaciju. Osnovni razlog zbog koga se pristupa uočavanju funkcionalnih zavisnosti za tabelu je potreba određivanja ograničenja za očuvanje integriteta. Verovatno najvažnije ograničenje za očuvanje integriteta je uočavanje ključeva kandidata, od kojih se jedan bira za primarni ključ tabele. U procesu njihovog definisanja, naročito je značajno pronaći one funkcijske zavisnosti koje važe za sve moguće vrednosti atributa jedne tabele, jer one predstavljaju tipove ograničenja za očuvanje integriteta. Tipovi ograničenja za očuvanje integriteta definišu vrednosti koje tabela može legitimno da primi.

U toku procesa normalizacije analizira se posebno svaka relacija, identifikuju se zavisnosti u relaciji i ako je potrebno pristupa se normalizaciji. Na osnovu identifikovanih zavisnosti pristupa se postepenom rastavljanju relacija u set novih relacija (tabela). Funkcijske zavisnosti u relacijama je definisao kanadski matematičar William W. Armstrong u radu iz 1974. godine *Dependency Structures of Data Base Relationships*. Ove zakonitosti su poznate pod nazivom Armstrongove aksiome (reflektivnost, pravilo uvećanja, tranzitivnost, unija, dekompozicija, pseudo-tranzijentnost). Na osnovu početnog skupa funkcijskih zavisnosti, poštujući

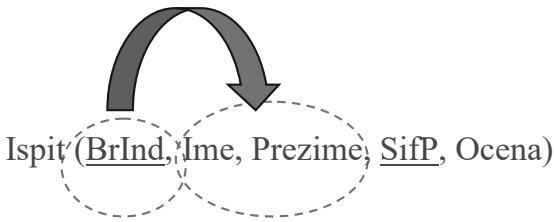
Armstrongove aksiome, mogu se izvesti nove funkcijske zavisnosti nad jednom šemom relacije.

Uzrok problema u primeru 1.

U prethodnom primeru diskutovani su problemi sa šemom relacije:

Ispit (BrInd, Ime, Prezime, SifP, Ocena).

U šemi relacije Ispit definisan je primarni ključ koga čine dva atributa: broj indeksa i šifra predmeta koji se polaže (BrInd, SifP). Prema striktnim pravilima realcionog modela podataka sigurno važi da primarni ključ na jedinstven način određuje svaki zapis u realciji, precizno određuje ime i prezime studenta i dobijenu ocenu na nekom predmetu. Pored toga, po svojim karakteristikama, primarni ključ je minimalni skup atributa koji ima tu osobinu. Međutim, u razmatranom primeru u šemi relacije Ispit, zna se da samo BrInd precizno određuje ime i prezime studenta, tj. nije nam potrebna i vrednost atributa SifP. Dakle:



Slika 9.1: Slabost šeme relacije Ispit

Svi problemi u šemi realcije Ispit nastaju zbog toga što deo primarnog ključa radi isto što i on (BrInd na jedinstven način određuje ime i prezime studenta, umesto što se za to koristi kompletan primarni ključ BrInd, SifP). U šemi relacije Ispit postoji jedna dodatna funkcijska zavisnost između atributa, koja nije predviđena relacionim modelom. Ne može deo primarnog ključa da radi isto što i on, pošto je po definiciji primarni ključ minimalan skup atributa sa svojstvima jedinstvenog određivanja željene vrednosti u zapisima.

Navedena zavisnost je primer tzv. parcijalne funkcijaske zavisnosti, gde neki ne ključni atribut ne zavisi totalno od vrednosti primarnog ključa, već od njegovog dela. Ova zavisnost je suvišna i potrebno je šemu relacije u kojoj se pojavljuje rastaviti (normalizovati), kako bi se tzv. parcijalna zavisnost izgubila.

Primer 2:

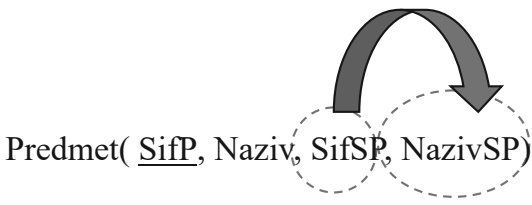
Neka je formirana posebna šema relacije koja objedinjava podatke o predmetima i studijskim programima::

Predmet (SifP, Naziv, SifSP, NazivSP.

Pretpostavimo da je organizacija predmeta i studijskih programa takva da kada se unese šifra predmeta, ona na jedinstven način određuje i naziv tog predmeta i studijski program, a svaki studijski program ima svoju šifru. Na primer, ako se unese šifra predmeta za Programiranje to definiše stidijski program iz Računarstva. Ako se unese šifra predmeta Bankarstvo, to definiše studijski program iz Ekonomije itd. Naravno da je ovakva šema relacije loša, ali će nam poslužiti da se pokažu dodatne slabosti koje se mogu javiti u radu sa njom. Neka je trenutni sadržaj relacije Predmet:

SifP	Naziv	SifSP	NazivSP
BP	Baze podataka	SPR	Računarstvo
BA	Bankarstvo	SPE	Ekonomija
TP	Turizam i prostor	SPTH	Turizam i hotelijerstvo
RM	Računarske mreže	SPR	Računarstvo
K1	Kriptologija 1	SPR	Računarstvo

Kada bi se u ovakvu tabelu zahtevao unos novog predmeta, npr. Programski jezici, umesto da se samo unese novi zapis, čak četiri puta bi pored toga morali da unesemo: SPR, Računarstvo. Pored toga, precizno bi morali da unesemo vrednosti SPR i Računarstvo, kako ne bi bazu doveli u nekonzistentno stanje. Pored višestrukog unošenja podataka u relaciju Predmet, ovde postoji jedna specifična zavisnost: svaki put kada se kao vrednost unese SifSP, njoj odgovara jedna i samo jedna vrednost NazivSP. Grafički prikaz ovog problema je sledeći:



Slika 9.2: Slabost šeme relacije Predmet

U šemi relacije Predmet, pored toga što SifP, kao primarni ključ, na jedinstven način određuje vrednosti svih zapisa, i vrednost neključnog atributa SifSP na jedinstven način određuje NazivSP. Ovo nije u skladu sa definicijom relacionog modela. Opisno bi se u šemi relacije Predmet moglo reći da jedan ne ključni atribut radi isto što i primarni ključ i to jeste slabost koju treba eliminisati. U navedenom primeru, pored uočene slabosti, može se izvesti i jedna nova zavisnost. Naime, kako SifP kao primarni ključ na jedinstven način određuje SifSP, a SifSP određuje NazivSP, zaključuje se da NazivSP tranzitivno zavisi od primarnog ključa. Navedene tranzitivne zavisnosti se tretiraju tzv. trećom normalnom formom, koja ima za cilj da ih u postupku normalizacije eliminiše.

9.3 POSTUPAK NORMALIZACIJE

Neka polazna šema relacije nije u određenoj normalnoj formi ako u skupu funkcijskih zavisnosti F postoji bar jedna koja narušava definiciju normalne forme.

U svakom koraku normalizacije:

- Uočava se jedna takva zavisnost ($X \rightarrow Y$)
- Vrši se dekompozicija u cilju uklanjanja takve zavisnosti
- Ako je u polaznoj važilo $X \rightarrow Y$, dekompozicijom nastaju dve relacije. U prvoj se gube atributi Y, a druga nastaje nad atributima X i Y.
- Ne dozvoljava se gubitak podataka

U postupku normalizacije uočavaju se nepoželjne funkcijske zavisnosti, a zatim se strogo formalnim postupkom rastavljanja dobijaju nove šeme relacija koje ne sadrže početne neželjene zavisnosti. U postupku rastavljanja relacija važno je da se očuvaju svi atributi, tj. da ne dođe do njihovog gubitka. Postupak normalizacije je dobar, ako se iz rastavljenih šema relacija može rekonstrisati polazna. Reverzibilnost ovog procesa se proverava primenom Dekartovog proivoda i spajanjem po zajedničkim atributima. Normalizacija nije samo mehanički postupak. Naime, posebna pažnja mora da se posveti tzv. prirodnim zavisnostima. U primeru 1. koji je razmatran u ovom poglavlju, atribut broj indeksa (BrInd) mora uvek da pokazuje na ime i prezime studenta. U datom primeru ova veza je bila slabost šeme relacije Ispit, ali ova veza je prirodna i nikada se ne sme izgubiti. Tačnije, bilo kakvim rastavljanjem, u jednoj od novodobijenih šema relacija prirodna funkcijska zavisnost mora biti sačuvana.

Krajnji cilj normalizacije je najčešće treća normalna forma. U većini slučajeva ona predstavlja najbolji kompromis između ekstrema koji se odnose na funkcionalnost i lakoću implementacije. Nivoi iznad 3NF u praksi mogu da iskomplikuju dizajn baze podataka do tačke da smetaju funkcionalnosti. Svi nivoi normalizacije su kumulativni što znači da baza koja se nalazi u 2NF takođe mora da ispunjava i sve uslove zadate prvom normalnom formom.

Proces analize šeme relacije je proces primene serije testova na šemu relacije da bi se proverilo da li ispunjava sva pravila definisana za određenu normalnu formu. Normalne forme pomažu projektantu baze podataka da ostvari željeni kvalitet relacione baze podataka.

9.3.1 Prva normalna forma (1NF)

Prva normalna forma se ne odnosi na funkcijske zavisnosti, već na vrednosti atributa koje moraju da budu atomske i moraju da budu skalarnog tipa (a ne vektorskog). Atomske vrednosti su takve da se ne mogu dalje deliti, odnosno u vrednostima se ne smeju naći tzv. viševrednosna značenja. Na primer, ako je projektovanjem šeme relacije predviđeno da se za vrednost atributa može uneti BP02, što može da znači da se to odnosi na Baze podataka koje se realizuju na 2. godini studija (prva dva karaktera su identifikacija predmeta, a sledeće dve cifre su godina studija), onda je to nije u prvoj normalnoj formi. Loša praksa je da su informacije praktično u programima umesto u bazama podataka.

Vektorski atributi nisu dozvoljeni prvom normalnom formom. Na primer, šema relacije ProfesorPredmet (IdProf, Ime{SifP, Sati}) nije u prvoj normalnoj formi. Navedena struktura šeme relacije ProfesorPredmet treba da sadrži podatke o profesorima koji predaju odgovarajuće predmete, a svaki predmet ima nedeljni fond časova. Ima profesora koji predaju samo jedan predmet, neki predaju dva, a neki i više. Zbog toga, za datu strukturu šeme relacije, neminovno je da zapisi budu promenljive dužine, što nije dozvoljeno relacionim modelom. Relacioni model je definisan tako da omogućava direktan pristup podacima i time potpuno odgovara organizaciji na savremenim elektronskim medijumima. Organizacija podataka sa promenljivom dužinom zapisa bi odgovarala tzv. sekvencijalnom pristupu podacima i zastarelim magnetnim memorijskim medijumima.

Šema relacije R je u prvoj normalnoj formi ako i samo ako domeni relacije R sadrže samo proste (atomske) vrednosti. U 1NF nisu dozvoljeni viševrednosni i kompozitni atributi i njihove kombinacije tj. nisu dopuštene relacije u relacijama ili relacije kao atributi n-torki.

Primer: Šema relacije ProfesorPredme t(IdProf, Ime, {SifP, Sati}) sadrži ugnježdenu šemu relacije Predmet (ŠifP,Sati). U originalu, ona sigurno nije u prvoj normalnoj formi. Neka su trenutni podaci takvi da Marko predaje tri predmeta, Petar dva, a Janko jedan. Ovakvo rešenje podrazumeva promenljivu dužinu sloga, te je onemogućen rad sa direktnim pristupom. Ako bi se unapred fiksirala dužina sloga, ograničio bi se maksimalni broj predmeta koji može da predaje jedan profesor. Upiti nad ovakvom relacijom su praktično nemogući.

JMBG	Ime	SifP	Sati
123	Marko	P1	3
		P3	2
		P5	2
222	Petar	P3	2
		P4	4
333	Janko	P1	3

Tabela 9.1: Relacija ProfesorPredmet koja nije u 1NF

Da bi ova šema relacije bila u 1NF nad njom se definiše relacija u kojoj neminovno dolazi do ponavljanja vrednosti atributa. Zapisi su fiksne dužine, a u svakoj ćeliji postoji vrednost. Unesene vrednosti su skalarnog tipa.

JMBG	Ime	SifP	Sati
123	Marko	P1	3
123	Marko	P3	2
123	Marko	P5	2
222	Petar	P3	2
222	Petar	P4	4
333	Janko	P1	3

Tabela 9.2: Relacija ProfesorPredmet u 1NF

Dobijena šema relacije jeste u prvoj normalnoj formi, ali i dalje ima lošu strukturu. Naime, atribut SifP kao deo primarnog kluča šeme relacije ProfesorPredmet na jedinstven način određuje atribut Sati, što je tzv. parcijalna funkcijska zavisnost. Ovu zavisnost tretira sledeća, druga normalna forma.

9.3.2 Druga normalna forma (2NF)

Druga normalna forma se bazira na konceptu *potpune funkcijske zavisnosti*. Šema relacije u drugoj normalnoj formi ako svi atributi entiteta, koji nisu primarni ključevi, zavise totalno od celog primarnog ključa. Drugim rečima, ne dozvoljavaju se parcijalne zavisnosti nekog neključnog atributa od bilo kog kandidat ključa date šeme relacije..

Druga normalna forma se odnosi na tabele sa složenim ključevima, tj. tabele čiji se primarni ključevi sastoje iz dva ili više atributa. Tabela čiji primarni ključ sadrži samo jedan atribut je automatski u najmanje 2NF. U tabeli koja nije u 2NF mogu da se pojave anomalije ažuriranja. Tabela je u 2NF ako je u 1NF i ako je svaki atribut koji nije primarni ključ potpuno funkcionalno zavisan od primarnog ključa.

Na primer, za šemu relacije:

ProfesorPredmet (IdProf,SifP, Ime,Sati)

Uočavaju se sledeće funkcijske zavisnosti:

{IdProf,SifP→Ime, Sati} - (superključna zavisnost)

{SifP→Sati} - (parcijalna zavisnost)

U postupku normalizacije prvo se eliminiše parcijalna zavisnost, tako što od nje nastaje nova šema relacije Predmet(SifP, Sati), iz prethodne šeme relacije se gubi atribut Sati, a na kraju ostaje: ProfesorPredmet (Idprof, SifP, Ime). U novodobijenim šemama relacija nema nepogodnih funkcijskih zavisnosti, a šema relacije Predmet je sačuvala prirodnu funkcijsku zavisnost da Predmet određuje broj Sati.

9.3.3 Treća normalna forma (3NF)

Kod šema relacija koje su normalizovane do 2NF, u relacijama mogu da se pojave anomalije ažuriranja zbog *tranzitivnih zavisnosti*, koje se moraju ukloniti iz tabele postupkom normalizacije do 3NF.

Tranzitivna zavisnost je stanje u kome su A, B i C atributi tabele takvi da, ako A→B *(B je funkcionalno zavisno od A)* i B→C *(C je funkcionalno zavisno od B)*, onda je C *tranzitivno zavisno od A preko B.*

Šema relacije je u 3NF ako je u 1NF i u 2NF i ako u njoj ne postoji ni jedna funkcijska zavisnost po kojoj neki ne ključni atribut tranzitivno zavisni od bilo kog kandidat ključa (što se naravno odnosi i na zavisnost od primarnog ključa).

Normalizacija šeme relacije koja nije u III normalnoj formi se vrši uklanjanjem tranzitivnih zavisnosti, tj. tranzitivno zavisni atributi se izmeštaju u novu tabelu. Postupak normalizacije je isti kao i kod prethodnog postupka sa normalizacijom kod druge normalne forme. Uočava se tranzitivna zavisnost, od atributa koji učestvuju u njoj se formira nova relacija, a atributi koji su bili tranzitivno zavisni se gube iz prethodne relacije. Zatim se otklanjaju i ostale zavisnosti. Na kraju, sve dobijene šeme relacija su u trećoj normalnoj formi, a time su sigurno i u drugoj normalnoj formi. Prva normalna forma se podrazumeva.

9.3.4 Boyce-Codd normalna forma (BCNF)

Druga i treća normalna forma eliminišu parcijalne i tranzitivne zavisnosti za primarne ključeve. Međutim, u 3NF mogu da postoje i tranzitivne zavisnosti za

ključeve kandidate, pa je stoga promovisana stroža normalna forma poznata kao *Boyce-Codd normalna forma (BCNF)*, koja eliminiše dodatne tranzitivne zavisnosti.

Da bi se odredilo da li je tabela u BCNF, potrebno je da sve uočene funcijske zavisnosti $X \rightarrow Y$ budu trivijalne ($Y \subseteq X$) ili superključne ($X \rightarrow R$).

Šema relacije se transformiše u BCNF tako što se vrši njena dekompozicija na dve ili više novih šema. Međutim, ponekad nije poželjno normalizovati šemu do BCNF, zato što se ovakvom normalizacijom mogu izgubiti neke prirodne funkcijske zavisnosti. Kao što smo rekli, u takvim slučajevima se u samom kodu moraju postaviti odgovarajuća ograničenja, kako se baza podataka u toku ažuriranja ne bi dovela u nekonzistentno stanje.

9.4 DENORMALIZACIJA

Denormalizacija je prevođenje tabele u slabiju normalnu formu. Za razliku od šema relacije loše strukture i pravilima za normalizaciju, ovde je reč o namernom smanjenju normalizacije - najčešće zbog performansi. Denormalizacija jer proces koji može dovede do anomalija. Kako bi se anomalije izbegle, neophodno je da se postave ograničenja u softverima za unos podataka.

Ako baza radi sa slabim performansama to ne mora uvek da znači da je treba denormalizovati. Možda je potrebno redizajniranje baze podataka zbog pogrešno osmišljenog modela, koji ne može da ponudi zahtevane performanse. Denormalizacija baze ne mora da znači poboljšanje karakteristika. Problemi često mogu ostati isti kao ranije. Pošto denormalizacija znači promenu strukture baze podataka moguće je da se pokvari postojeća funkcionalnost sistema, ili napraviti probleme u izveštavanju (važni upiti mogu početi raditi sporo usled promene). Čak i ako performanse nisu oštećene izmenama, može doći do anomalija unosa, izmene i brisanja, čime se ugrožava referencijalni integritet baze. U tom slučaju moraju se kreirati ograničenja koja obezbeđuju referencijalni integritet, umesto da većina problema bude rešena normalizacijom.

Prednosti normalizacije:

- Smanjenje fizičkog prostora za smeštanje podataka (nema redudanse);
- Bolja organizacija podataka, nema neželjenih funkcijskih zavisnosti, nema anomalija ažuriranja:
- Promene podataka se rade na nivou samo jednog zapisa

Mane noramalizacije:

- Fizički prostor na disku je danas jeftin i postaje sve manje bitan (izuzev možda kod velikih skladišta podataka – Data Warehouse);
- Normalizacijom se dobija veliki broj malih šema relacija – minimizacija, koje karakteriše tzv. visoka granularnost. Kdo takvih relacija se SQL JOIN upiti veoma sporo izvršavaju.
- Nakon normalizacije dobijaju se relacije koje karakteriše visoka kompleknost sa stanovišta dizajnera i programera. Naknadne ispravke koda se usložnjavaju.

10 TRANSAKCIJE

Danas je veoma bitan i značajan koncept baze podataka po kome je to, u stvari, zajednički resurs koga istovremeno (konkurentno) koristi veći broj programa, jer se pravi efekti baze podataka ispoljavaju kada se radi u mrežnom okruženju. DBMS je upravo tu da upravlja konkurentnim radom više korisnika i da obezbeđuje sinhronizaciju njihovog rada.

Takođe, DBMS ima funkciju da spreči štetne posledice (narušen integritet baze, nekonzistentno stanje baze...) pri promenama (transakcijama) koje se vrše nad bazom podataka u višekorisničkom okruženju, a za to koristi tehnike zaključavanja podataka, vremenskog markiranja, odlaganja i slično. Dalje, u tom smislu posebno je značajno upravljanje istovremenim (konkurentnim) transakcijama.

Baze podataka kontinuirano skladište informacije koje opisuju trenutno stanje organizacije. Na primer, baza podataka banke skladišti trenutni bilans na svakom računu deponenta. Kada se u stvarnom svetu dogodi nešto što menja stanje organizacije, mora da se uradi odgovarajuća promena informacija u bazi podataka. Sa dostupnim DBMS-om, ove promene se dešavaju u realnom vremenu, uz pomoć programa koji se nazivaju transakcije koje deluju kada dođe do promena u stvarnom svetu. Na primer, kada klijent stavlja novac u banku (događaj u stvarnom svetu), izvršava se transakcija depozita. Svaka transakcija mora biti uređena tako da održava preciznost veze između stanja baze podataka organizacije koje je kreira i stvarnog sveta. Pored toga što menja stanje baze podataka, transakcija sama po sebi može da inicira neke događaje u stvarnom svetu. Na primer, izdvojena transakcija kod bankomata, inicira događaj odliva novca.

Odobravanje kreditne kartice je samo jedan primer transakcije koja može biti sprovedena za vreme npr. godišnjeg odmora u inostranstvu. Pripreme za let uključuju transakciju sa bazom podataka za rezervaciju karata, prolazom kroz pasošku kontrolu na aerodrom, uključujemo i transakciju sa bazom podataka imigracione službe, a sa prijavljivanjem u hotelu uključili smo i transakciju sa hotelskom bazom podataka za rezervacije soba. Čak i telefonski poziv koji se obavi iz telefonske sobe uključuje transakciju sa hotelskom bazom podataka zajedno sa međunarodnim prenosnikom koji uspostavlja vezu. Drugi primeri transakcija koje se redovno sprovode u svakodnevnom životu, uključuju i bankomate, skener sistem u supermarketu, prijave na univerzitetima i sl.

U većini aplikacija baze podataka se koriste kako bi se modelovalo stanje nekog stvarnog preduzeća. U takvim aplikacijama transakcija predstavlja program koji posreduje sa bazom podataka, tako da podržava slaganje stanja preduzeća i stanja baze podataka. Praktično rečeno, transakcija ažurira bazu podataka tako da prikazuje događaje koji su se odrazili na stanje stvarnog preduzeća. Jedan primer bi bila transakcija polaganja novca u banku. Klijent daje blagajniku novac kao depozit. Transakcijom se ažurira informacija o računu klijenta u BP kako bi se prikazao depozit.

10.1 DEFINICIJA TRANSAKCIJE

Obično se transakcijom naziva niz operacija nad bazom podataka koji odgovara jednoj logičkoj jedinici posla u realnom sistemu. Važno je istaći da ta logička jedinica posla se izvršava do kraja ili se poništava u celini. Drugim rečima, zahteva se da transakcija bude atomska (nedeljiva) i da sve instrukcije jedne transakcije moraju biti izvršene ili nijedna. U tom smislu, transakcija predstavlja osnovnu programsku jedinicu kojom se obezbeđuje očuvanje konzistentnosti baze.

Naravno, u jednom trenutku nad bazom podataka se može izvršavati više transakcija, i imajući u vidu gore rečeno, transakciona obrada označava grupisanje izmena sadržaja baze podataka u „paket“ koji se potom obrađuje kao neraskidiva celina. Obrada se uvek odvija tako da se uspešno izvrše ili sve transakcije, ili nijedna od njih.

Transakciona obrada je korisna u aplikacijama u kojima jedna akcija mora da se izvrši u vezi sa jednom ili više drugih akcija. Transakciona obrada je uobičajena u bankarskim, računovodstvenim i mnogim drugim aplikacijama.

Na primer, kada u nekoj aplikaciji za bankarsko poslovanje premeštamo iznos sa jednog računa na drugi, ne bismo želeli da stanje na jednom računu povećamo, a da ga pri tom na drugom računu ne smanjimo. Zato ćemo te dve izmene grupisati u jednu transakciju.

Transakciona obrada je izuzetno važna u višekorisničkim aplikacijama. Kada više korisnika istovremeno unosi izmene u bazu podataka, više se ne možemo pouzdati u to da će uvek jedna izmena biti trajno upisana u bazu pre nego što započne naredna, osim ako određenu grupu izmena ne „uokvirimo“ u transakciju. Zbog toga bi u višekorisničkom okruženju trebalo da koristimo transakcionu obradu.

10.2 OSOBINE TRANSAKCIJA

Sve transakcije imaju sledeće osobine:

- atomnost (atomicity),
- konzistentnost (consistency),
- izolacija (izolation),
- trajnost (durability).

Atomnost podrazumeva skup aktivnosti nad bazom podataka po principu „sve ili ništa“. Ili su sve aktivnosti uspešno obavljene ili je baza podataka ostala nepromenjena. DBMS, pored atomnosti transakcija, mora da garantuje atomnost svake aktivnosti (pojedinačne operacije). Primetimo da obični programi ne moraju imati osobinu atomnosti. Npr. ako je sistem pao u trenutku dok je program ažurirao neki fajl, kada

smo podigli sistem, fajl je ostao delimično promenjen. Atomnost izvršenja znači da je svaka transakcija ili potvrđena, ili smo odustali od nje.

Konzistentnost znači da transakcija treba da prevede bazu podataka iz jednog u drugo konzistentno stanje. Za vreme obavljanja transakcije konzistentnost baze podataka može da bude narušena. Ukoliko u toku transakcione obrade dođe do greške, podaci moraju biti vraćeni u stanje pre početka transakcije. Transakcijom se mora pristupati i ažurirati BP tako da sva ograničenja integriteta BP ostanu zaštićena. Svaka realna organizacija je organizovano u odnosu na određena pravila koja ograničavaju moguća stanja organizacije.

Npr. broj studenata prijavljenih za nastavu ne može preći broj mesta namenjenih za tu nastavu. Kada takvo pravilo postoji, moguća stanja BP su ograničena na isti način. Ograničenja integriteta, u odnosu na pomenuta pravila, potvrđuju da broj registrovanih studenata koji se unosi u polje BP ne sme da pređe vrednost polja BP koja odgovara veličini sale. Dakle, kada se transakcija registracije završi, BP mora da zadovolji ovo ograničenje integriteta (pretpostavljajući da je ograničenje zadovoljeno na početku transakcije).

Izolacija znači da kada se dve ili više transakcija izvršavaju istovremeno, njihovi efekti moraju biti međusobno izolovani. Efekti koje izazovu transakcije koje se obavljaju istovremeno moraju biti jednaki efektima nekog njihovog serijskog (jedna posle druge) izvršenja. Zbog povećanja paralelizma u obradi transakcija dozvoljavaju se različiti nivoi izolovanosti.

Prilikom rasprave o konzistentnosti BP, usredsredili smo se na efekat jedne transakcije. Sledeće što ćemo ispitivati je efekat niza transakcija. Rekli smo da se niz transakcija izvodi sekvencijalno, ili serijski, ako se jedna transakcija niza izvrši pre nego što druga počne. Dobra strana kod serijskog izvršavanja je da ako su sve transakcije konzistentne i baza podataka je inicijalno u konzistentnom stanju. Serijsko izvršavanje čuva konzistentnost. Kada prva transakcija niza započne, BP je u konzistentnom stanju, a s obzirom da je transakcija konzistentna i nakon njenog izvršenja BP će biti u konzistentnom stanju. Zbog toga što je BP konzistentna pre početka druge transakcije, i druga će se obaviti korektno.

Serijsko izvršavanje je adekvatno za aplikacije čiji su zahtevi skromni. Međutim, mnoge aplikacije imaju stroge zahteve vremena odziva i pristupa, i često jedini način da se zahtevi zadovolje je konkurentno izvršavanje transakcija. Moderni sistemi mogu da istovremeno izvršavaju više od jedne transakcije, a ovaj metod izvršavanja nazivamo konkurentnim. Konkurentno izvršavanje je adekvatno kada se sistemom za obradu transakcija služi više korisnika. U tom slučaju biće dosta aktivnih, delimično završenih transakcija u svakom trenutku. Kada se transakcije izvršavaju konkurentno, konzistentnost svake od njih nije dovoljna da obezbedi da baza posle izvršenja obe transakcije u potpunosti prikazuje stanje preduzeća.

Trajnost znači da kada se transakcija završi (potvrđene promene), njeni efekti ne mogu biti izgubljeni, čak i ako se neposredno po njenom okončanju desi neki ozbiljan otkaz sistema. Ovaj zahtev sistema za obradu transakcija odnosi se na to da se informacije ne izgube. Sistem mora da osigura da transakcija, koja je jednom potvrđena, svoj efekat prenese na BP, pa čak i ako kompjuter, ili medijum na kojem je BP smeštena, iznenada prestane da radi.

Npr. ako ste se uspešno prijavili za kurs, očekujete da sistem zapamti vašu registraciju pa čak i ako posle toga prestane da radi. Možemo da primetimo da obični programi ne moraju imati osobinu trajnosti. Npr. ako se pojave problemi pošto je program izvršio promenu nekog fajla, fajl može biti vraćen u stanje pre izvršene promene.

10.3 COMMIT I ROLLBACK

Obezbeđenje *ACID* (akronim od reči ***A****tomicity*, ***C****onsistency*, ***I****solation*, ***D****urability*) osobina transakcije se radi upotrebom određenih metoda i instrukcija:

- transakcija počinje sa BEGIN TRANSACTION,
- završava se sa COMMIT, čime se potvrđuju promene u bazi podataka ako su sve instrukcije uspešno izvršene,
- završava se sa ROLLBACK, ako sve instrukcije nisu uspešno završene.

U stvari, postupak transakcije počinje pozivanjem metode *BeginTrans*, čime se označava početak niza operacija koje čine jednu logičku jedinicu. Metoda *CommitTrans* preuzima sve izmene načinjene od poslednjeg mesta na kome je bila pozvana metoda *BeginTrans* i upisuje ih na disk. Metoda *RollbackTrans* deluje na suprotan način od *CommitTrans* – ona poništava sve izmene i vraća stanje kakvo je bilo pre poslednjeg poziva metode *CommitTrans*.

SUBP inicira iz bilo kog razloga ROLLBACK instrukciju, ako dođe do neplaniranog završetka transakcije.

S obzirom da jedan program može predstavljati kolekciju transakcija, transakcije mogu biti i ugnježđene. U tom slučaju, COMMIT instrukcija se izvršava od najnižeg do najvišeg nivoa, a dovoljno je da se ROLLBACK pojavi samo na jednom mestu i poništavaju se promene svih transakcija.

SUBP poseduje i održava dnevnik transakcija (tj. dnevnik aktivnosti, *log file*). Za svaku transakciju i za svaki objekat baze podataka koji je ona ažurirala čuva se:

- vrednost pre ažuriranja (*before-image*)
- vrednost posle ažuriranja (*after-image*)

Na naredbu ROLLBACK, SUBP koristi vrednosti ***pre*** za datu transakciju. Pre Commit naredbe sistem prvo upisuje vrednosti ***pre*** i ***posle*** u log fajl. Ako se prekine COMMIT naredba, mogu se pročitati vrednosti ***posle*** sa log fajla, što omogućava očuvanje konzistentnog stanja.

10.4 KONKURENTNO IZVRŠAVANJE TRANSAKCIJA

Nad modernim bazama podataka transakcije se ne obavljaju u izolovanosti već konkurentno. Više transakcija mogu istovremeno zahtevati iste resurse, isti zapis baze podataka, itd. U takvim situacijama otvara se mogućnost da nekontrolisan međusobni uticaj transakcija dovede do nekonzistentnog stanja.

Već je nagovešteno da je jedan od zahteva SUBP da upravlja konkurentnim radom više korisnika, obezbeđuje sinhronizaciju njihovog rada... a sve u cilju sprečavanja štetnih posledica pri promenama koje se vrše nad bazom podataka u višekorisničkom okruženju.

Komponente SUBP koje učestvuju u ovom procesu su:

- Planer *(Scheduler)*,
- Menadžer transakcija *(Transaction manager)*.

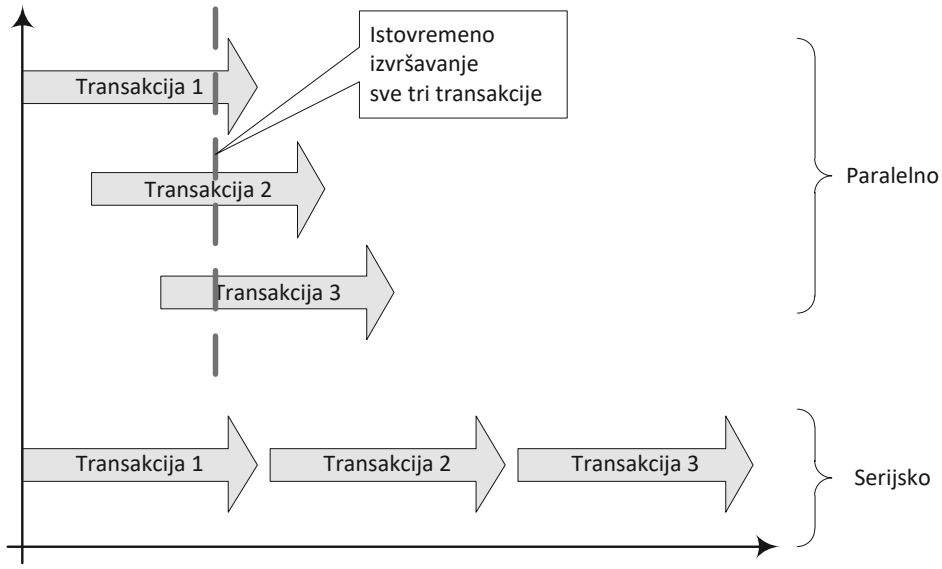
Planer vodi računa o redosledu akcija kod više konkurentnih transakcija. Ako čitanje ili upisivanje može da naruši integritet baze podataka, zahtev se ili vremenski odlaže ili se poništava cela transakcija.

Menadžer transakcija upravlja celokupnim izvršenjem transakcija.

Idealan slučaj izvršavanja transakcija je serijsko izvršavanje. Posledica je korektan rezultat. Serijsko izvršavanje transakcija, u stvari, znači da:

- nema preplitanja transakcija,
- prvo se završi jedna, zatim druga...

Konkurentno izvršavanje transakcija je serijabilno (linearno) ako daje isti rezultat kao i serijsko izvršavanje svih transakcija.



Slika 10.1: Paralelno i serijsko izvršavanje transakcija

Interesantno je da kada se koriste transakcije, povećava se stepen integriteta izmena koje korisnik unosi, ali se smanjuje konkurentnost, odnosno mogućnost da pomoću date aplikacije veći broj korisnika istovremeno menja podatke, jer ima više zapisa koji su zaključani duže vreme.

10.5 PROTOKOL ZAKLJUČAVANJA

Velika je verovatnoća da, ako nema ograničenja, izvršavanje transakcija neće biti serijabilno, a provera serijabilnosti se teško može obaviti u realnom vremenu. Zato se vrši jedan način ograničavanja, tj. zaključavanje i otključavanje podataka.

U stvari, **mehanizam zaključavanja** (*locking*) predstavlja upravo nasilno ostvarivanje serijabilnosti. Transakcija zaključava objekat baze podataka kome je pristupila, čime je onemogućeno drugim transakcijama da nekorektno operišu.

Postoje dva osnovna **nivoa zaključavanja**, na nivou stranice i na nivou zapisa. Zaključavanje na nivou stranice je u stvari zaključavanje čitave stranice sa zapisima, a ne zaključavanje samo pojedinih zapisa, što je slučaj sa zaključavanjem na nivou zapisa. To znači da, na primer, kada se primenjuje zaključavanje na nivou zapisa, više korisnika može istovremeno da ažurira zapise u istoj tabeli, nasuprot slučaju kada bi bila zaključana cela tabela sa svim zapisima u njoj.

Zaključavanje na nivou zapisa obezbeđuje znatno bolje mogućnosti višekorisničkog pristupa podacima. Međutim, treba voditi računa da to ne znači uvek i bolje performanse. Kada se istovremeno ažurira veliki broj zapisa, zaključavanje na nivou stranice može da obezbedi bolje performanse. Ipak, u većini slučajeva, bolje mogućnosti višekorisničkog pristupa podacima, nadoknađuju nešto slabije performanse.

Postoje dva osnovna pristupa u strategiji zaključavanja objekata baze podataka, odakle se izdvajaju dve vrste zaključavanja:

- Ekskluzivno (*exclusive lock* ili *write lock*) ili pesimističko – XL
- Deljivo (*shared lock* ili *read lock*) ili optimističko – SL

Ekskluzivno zaključavanje znači da u svakom datom trenutku samo jedan korisnik može da menja objekat baze podataka. Drugim rečima, ako jedna transakcija postavi XL katanac na objekat baze podataka to znači da ne može ni jedna druga transakcija da postavi katanac, i ne može se postaviti ni jedan drugi katanac. U slučaju ako se primenjuje zaključavanje na nivou stranice, to je veliki problem, jer u svakom trenutku samo jedan korisnik može da ažurira bilo koji zapis koji se nalazi na zaključanoj stranici.

Deljivo zaključavanje omogućava da više korisnika istovremeno ažurira iste zapise uz manji broj sukoba oko zaključavanja zapisa. Drugim rečima, ako jedna transakcija postavi SL katanac na objekat baze podataka to znači da druga transakcija može da postavi SL katanac na isti objekat, i ni jedna druga ne može da postavi XL katanac na taj objekat. Međutim, time se povećava rizik nastajanja sukoba pri upisivanju podataka. Sukob pri upisivanju nastaje kada:

- prvi korisnik počinje da ažurira objekat baze podataka.
- drugi korisnik upisuje u bazu podataka izmene objekta koje je on uneo.
- prvi korisnik pokuša da u tom trenutku upiše svoje izmene.

Sukob pri upisivanju je štetan, jer on znači da prvi korisnik ažurira drugačiji objekat od onoga sa kojim je počeo da radi.

Verovatno je u izboru vrste zaključavanja prihvatljivije ekskluzivno zaključavanje, da bi se izbegli sukobi pri upisivanju. Međutim, uvek treba imati na umu korisnike koji duže vreme zaključavaju objekte baze podataka. U tom slučaju bi bilo dobro razmotriti upotrebu deljivog zaključavanja. Ovaj problem delimično može biti rešen i upotrebom ekskluzivnog zaključavanja sa vremenskim ograničavanjem. Na primer, nekom obrascu se može dodati mogućnost vremenskog ograničavanja tako da izmene koje korisnik nije snimio posle deset minuta automatski bivaju poništene.

Protokol zaključavanja obuhvata sledeća pravila:

1. Transakcija koja čita neki objekat baze podataka mora da postavi SL na taj objekat
2. Transakcija koja želi da ažurira neki objekat mora da postavi XL. Ako je ta transakcija prethodno postavila SL treba da ga promeni u XL
3. Ako transakcija nije postavila „katanac“, zato što je to pre uradila neka druga, prelazi u stanje čekanja
4. Transakcija oslobađa XL i SL na kraju, sa COMMIT ili ROLLBACK naredbom

Oba katanca XL i SL se postavljaju implicitno.

10.6 OPORAVAK BAZE PODATAKA

Oporavak baze podataka (*RECOVERY*) predstavlja proces vraćanja baze podataka u korektno stanje. Sasvim je realno, i dešava se, da usled otkaza sistema mora da se uradi oporavak baze podataka. Uzroci otkaza mogu biti različiti: greške u programiranju, greške u operativnom sistemu, nestanak napajanja...

Proces oporavka se zasniva na redundansi podataka, tj. postojanje rezervnih kopija, koje mogu da se čuvaju na disku, traci... Tako, u slučaju otkaza sistema, oštećena baza podataka se rekonstruiše u ispravno stanje na osnovu poslednje kopije, a nekonzistentno stanje se rešava tako što se poništavaju nekonzistentne promene, a transakcije se ponavljaju.

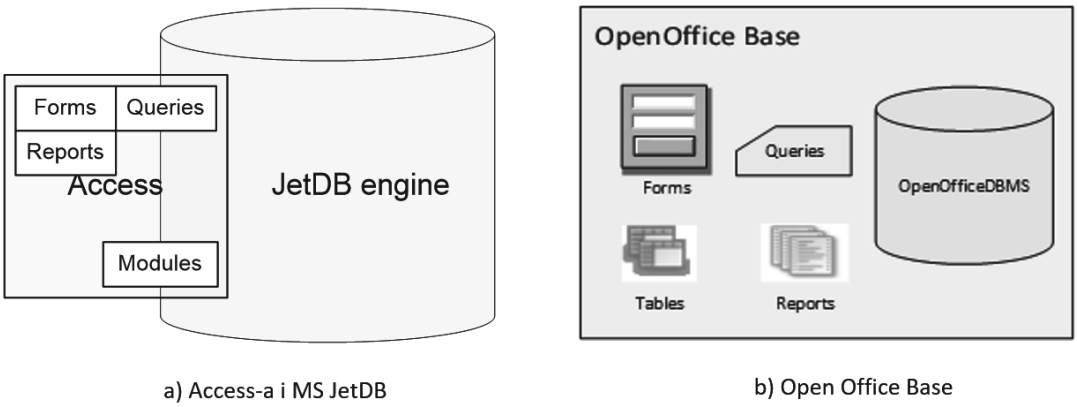
Trajnost podrazumeva da nijedna promena u bazi podataka koju je napravila započeta transakcija, ne sme biti izgubljena. Iz tog razloga, a pošto hard disk može da zakaže, baza podataka se mora čuvati na različitim hard diskovima. Jednostavan primer postizanja trajnosti jeste čuvanje različitih kopija baze podataka na različitim diskovima (koji se, po mogućstvu, napajaju iz različitih izvora energije). Pošto je istovremeni pad oba diska malo verovatan, velika je verovatnoća da će barem jedna kopija hard diska biti uvek dostupna. Duplikat, ili disk imidž, podržava ovakav pristup. Slika hard-diska (duplikat – *mirrored disk*) je sistem čuvanja po kome, kad god aplikacija zahteva da disk izvrši operaciju upisa, sistem simultano beleži istu informaciju na dva različita diska. Tako je prvi disk identična kopija, ili disk imidž, onog drugog.

U transakcijama koje obrađuju aplikacije, sistem disk imidža može da dostigne izuzetnu dostupnost sistema. Ukoliko jedan disk imidž padne, sistem može da nastavi dalje koristeći drugi, bez usporavanja ili zaustavljanja. Kada se zameni disk koji je zakazao, sistem mora da ponovo sinhronizuje oba. Nasuprot tome, kada se trajnost postiže samo pomoću LOG fajla (dnevnika), proces oporavka nakon pada hard diska može trajati znatno duže, a u toku tog perioda sistem je nedostupan korisnicima. Ipak, treba podsetiti da čak i kada transakcija u procesu koristi disk imidž, i dalje se mora koristiti dnevnik kako bi se postigla atomnost – na primer, da bi se poništila transakcija nakon pada.

11 BAZE PODATAKA I APLIKACIJE

Nije redak slučaj da proizvođači savremenih sistema za upravljanje bazama podataka (u daljem tekstu SUBP), pored serverske aplikacije (koja omogućava administriranje korisnika, kontrolu pristupa, održavanje referencijalnog integriteta i konzistentnost podataka BP), najčešće nude i klijentske aplikacije (alate). Primeri ovakvih sistema su Access za Microsoft JetDB i MS SQL, Apache Open Office Base za MySQL, PostgreSQL, a korišćenjem HSQLDB i za mnoge druge SUBP.

Klijentski programi predstavljaju prijateljska grafička okruženja koja omogućavaju brzo kreiranje upita (QBE[4]), ugnježdenih procedura, izveštaja i formi za unos podataka. Prethodno navedene prednosti omogućavaju brzu izradu uslužnih aplikacija.



Slika 11.1: Primeri čvrste sprege klijentskih programa i SUBP

MS Access predstavlja dobro poznati komercijalni primer takvog programa, dok je OO Base besplatan (Slika 11.1 a i b). Nedostatak ovako koncipiranih aplikacija je da su čvrsto povezane za razvojno okruženje (high coupling) i da je komunikacija sa okruženjem (ostalim aplikacijama) obično teško izvodljiva, ili nemoguća (Slika 11.1). Posledica ovakvog (jednoslojnog, ili dvoslojnog) modela je da se aplikacije dizajnirane na ovaj način teško održavaju (modifikacija postojećeg rešenja, unapređenje dodavanjem novih komponenti, ažuriranje novim verzijama serverskih i klijentskih platformi i sl.). Drugu manjkavost predstavlja činjenica da je dizajn korisničkog interfejsa i implementacija logike obrade podataka ograničeno na mogućnosti klijentske tehnologije. Na primer, ne mogu se menjati svi parametri ekranskih formi (izgled kontrola), otežano je ubacivanje kontrola koje eksplicitno nisu ponuđene,

[4] QBE – query by example, alat koji omogućava kreiranje SQL upita bez potrebe poznavanja sintakse SQL jezika. Primer QBE je Access-ov query designer i query wizard

otežana je izrada novih vrsta kontrola. Programiranje je ograničeno mogućnostima programskih jezika ugrađenog u klijentski alat (npr. Visual Basic for Access i Scripting Framework za OpenOffice ne podržavaju sve koncepte objektno orjentisanog programiranja).

Pojavom objektno orjentisanog programiranja (u daljem tekstu OOP) omogućeno je potpuno razdvajanje podataka od logike njihove obrade i od interfejsa prema korisnicima podataka. Aplikacija se gradi od objekata, od kojih svaki preuzima odgovornost za obavljanje specifičnih funkcionalnosti aplikacije. Tako se izdvajaju grupe objekata prema funkcionalnosti. Na primer, formira se grupa objekata od kojih se gradi korisnički interfejs, ili, grupa objekata koji ostvaruju konekciju sa BP, izvršavaju upite nad bazom i prihvataju rezultate upita. Objekti međusobno komuniciraju preko funkcionalnih poziva, pri čemu fizički mogu biti razdvojeni (na različitim računarskim platformama), a aplikacija time postaje distribuirana. Ova pojava grupisanja objekata prema osnovnim funkcionalnostima je nazvana raslojavanje aplikacije.

11.1 SLOJEVITA STRUKTURA APLIKACIJA

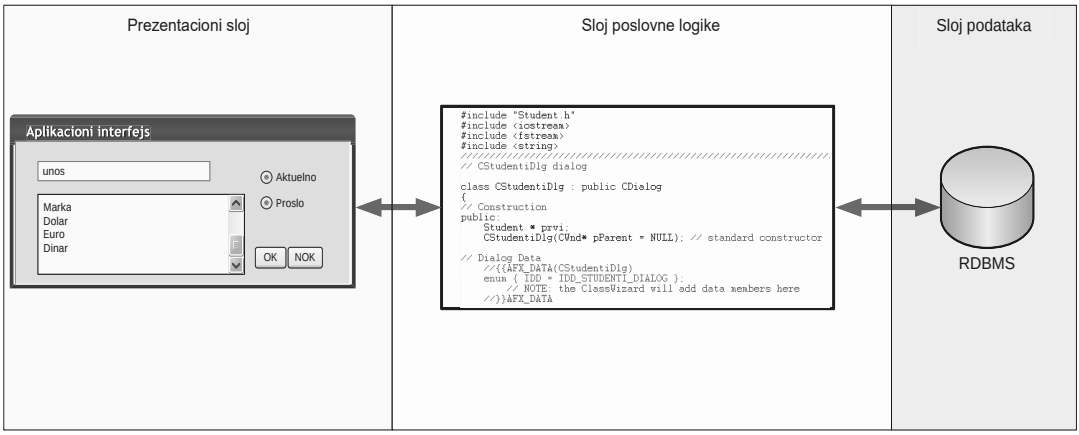
Raslojavanje aplikacije predstavlja odvajanje njenih delova prema funkcionalnosti. U OOP, kao što je već napomenuto, grupišu se objekti srodnih funkcionalnosti (u daljem tekstu slojevi). Grupisanjem je postignuto da se komunikacija između slojeva (funkcionalni pozivi) svedu na najmanju moguću meru i time olakša održavanje aplikacije. Promene u funkcionalnosti (programskom kodu) objekata jednog sloja neće imati posledice na objekte u ostalim slojevima, a imaće sporadične efekte čak i za objekte u istom sloju. Jedno od pravila dobrog dizajna aplikacija je da se u između objekata (klasa) u istom sloju postigne visoka kohezija (high cohesion), a slaba sprega između slojeva (low coupling).

11.1.1 Troslojni model kao osnovni model

Osnovni aplikacioni model je troslojni model, koji nameće dizajnerima i programerima zahtev da aplikacija ima razdvojene tri celine (Slika 11.2):

1. Prezentacioni sloj (*presentation layer*)
2. Sloj poslovne logike (*buisness logic layer*)
3. Sloj podataka (*data layer*)

Nazivi slojeva implicitno otkrivaju njihove funkcionalnosti. Objekti prezentacionog sloja su zapravo objekti korisničkog interfejsa: forme za unos, promenu, brisanje i pregled podataka. Obradu podataka vrši sloj poslovne (aplikacione) logike. Ovaj sloj sadrži ključne objekte sistema, koji sinhronizuju procese prezentacionog i sloja podataka. Sloj podataka čine objekti koji komuniciraju sa BP, preciznije sistemom sa upravljanje BP.

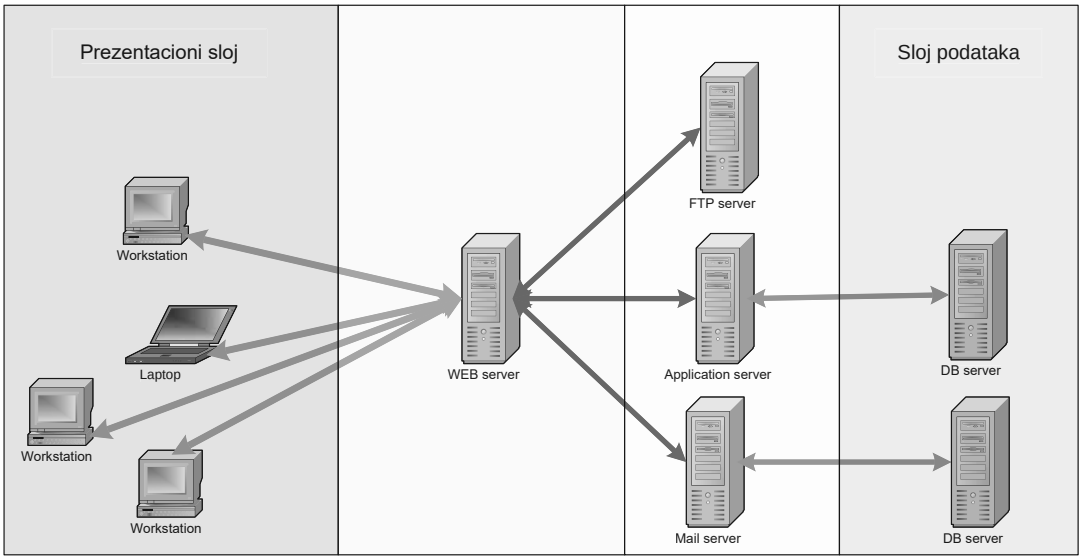


Slika 11.2: Troslojni aplikacioni model

Slojevi komuniciraju funkcionalnim pozivima. Pošto su podaci u troslojnom modelu dostupni (korisnicima posredstvom interfejsa prezentacionog sloja) preko sloja poslovne logike (indirektno), može se reći da su mogućnosti zaštite integriteta podataka mnogo veće nego kod dvoslojnih aplikacionih modela.

11.1.2 Višeslojni modeli

Aplikacije mogu imati više od tri sloja (Slika 11.3). Ova pojava je vezana za distribuirane poslovne sisteme, kod kojih podaci mogu biti razdvojeni na više različitih mesta, i koji sadrže više nivoa obrade. Najčešći primer višeslojnih distribuiranih sistema su Web aplikacije. Kod Web aplikacija, korisniku su vidljive samo Web stranice, isporučene na klijentski pretraživač od strane Web servera i komponovane od različitih sadržaja. Komponente stranice mogu biti kontrole za interakciju sa korisnikom ili veze (linkovi) prema posebnim aplikacijama (modulima). Ove softverske komponente mogu biti ugnježdene na konkretnom serveru, ali isto tako mogu biti distribuirane na različite platforme.



Slika 11.3: Višeslojni aplikacioni model

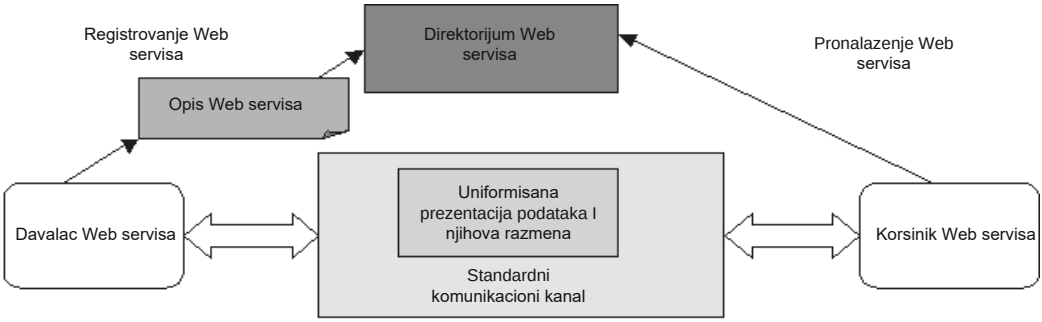
Distribuiranjem aplikacija vrši se rasterećenje hardverskih (serverskih) platformi i veza (linkova) između korisnika i aplikacija. Pošto se različite grupe funkcionalnosti odvijaju na različitim mestima, distribuiranjem je olakšano upravljanje i održavanje aplikacija. U primeru na prethodnoj slici, prezentacioni sloj je Web pretraživač na klijentskoj platformi, a sloj podataka čine 2 servera BP. Web server je zadužen za isporučivanje zahtevanih Web stranica klijentima. Takođe odgovoran je za upravljanje korisničkom sesijom. Drugi međusloj čine različiti aplikacioni serveri (aplikacije): za servisiranje elektronske pošte, za upload i download fajlova, aplikacioni server koji omogućava dinamičko komponovanje Web stranica i transakcionu obradu prema BP, itd.

11.1.3 Aplikacije servisi (nezavisne softverske komponente)

Iako su Web aplikacije višeslojne i distribuirane, njene softverske komponente su i dalje uzajamno zavisne i predstavljaju deo jedne celine. Da bi se omogućilo da softverska komponenta bude dostupna različitim aplikacijama, bila je potrebna formalizacija interfejsa koja bi omogućila komuniciranje sa komponentom. Web servisi su zasnovani na tri osnovna standarda: XML [5](za prikazivanje podataka), SOAP

[5] XML – extensible markup language

[6](za razmenu podataka između davalaca i korisnika servisa) i, za potrebe opisa servisa, definisan je poseban jezik – WSDL[7] i način uspostave veze.



Slika 11.4: Arhitektura Web servisa

Za postojanje servisa potrebne su 3 komponente: davalac servisa, korisnik servisa i provajder. Davalac Web servisa registruje svoj Web servis kod direktorijuma Web servisa (Slika 11.4). Registrovanje bi značilo da se Web servis formalno opisuje WSDL-om (naziv servisa, lokacija servisa, način pristupa servisa), kako bi korisnici mogli da ga eksploatišu. Registrovanjem Web servisa, davalac ga zapravo publikuje – svoju aplikaciju čini dostupnom za sve zainteresovane korisnike. U ulogama korisnika Web servisa pojavljuju se druge aplikacije koje na taj način proširuju svoje funkcionalnosti koje pružaju krajnjim korisnicima. Što je dokumentacija za API[8] kod konvencionalnih aplikacija, to je WSDL opis Web servisa kod njihovih davalaca. Najčešći način razmene podatka između davalaca i korisnika Web servisa je korišćenjem SOAP-a. Format podataka koji se razmenjuju je u XML formatu.

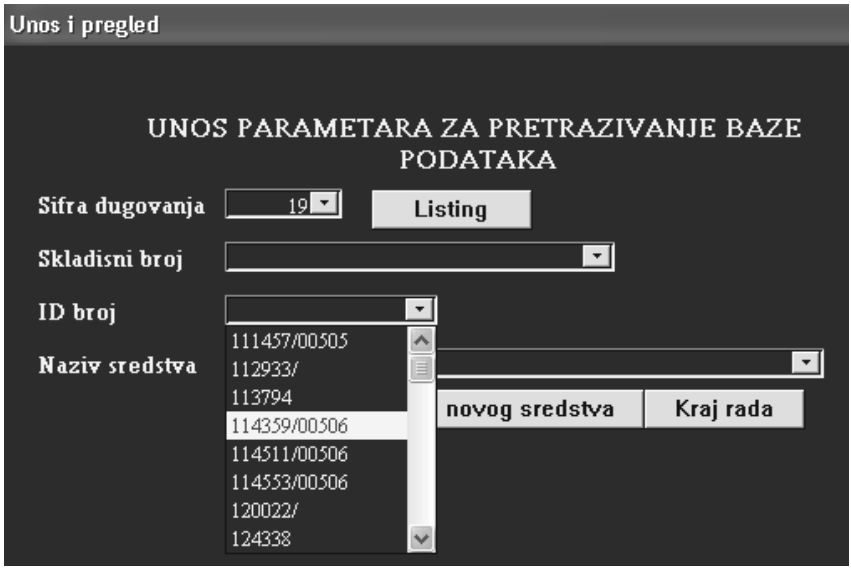
Web servisi predstavljaju tehnologiju budućnosti, jer omogućavaju povezivanje različitih aplikacija, tehnologija, postavljenih na različitim platformama. Formalizacija opisa servisa omogućava njihovu mašinsku čitljivost tako da aplikacije postaju uzajamno kooperativne bez posredovanja čoveka. Web servisi ukidaju i barijere između tzv. desktop i Web aplikacija.

[6] SOAP – simple object access protocol
[7] WSDL – Web Service Definition Language
[8] API – Application Programmable Interface

11.2 SPECIFIČNOSTI PRISTUPA BP IZ RAZLIČITIH APLIKACIONIH SLOJEVA

11.2.1 Pristup podacima iz prezentacionog sloja

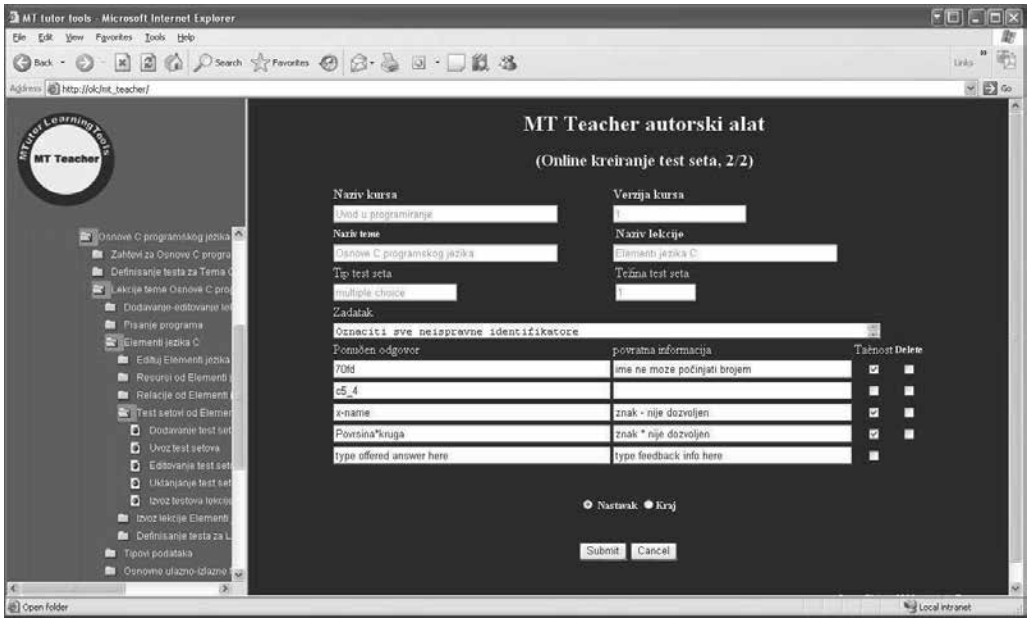
Prezentacioni sloj sadrži objekte korisničkog interfejsa. Bez obzira da li se radi o Desktop ili Web aplikacijama, to su uokvireni prozori sa naslovnom linijom i koji sadrže kontrole za interakciju sa korisnikom (Slika 11.5).



Slika 11.5: Izgled korisničkog interfejsa Desktop aplikacije

Svaka kontrola prezentacionog sloja predstavlja poseban objekat, koji se može dodati prevlačenjem sa palete – tzv. Drag&Drop (ako se koristi vizuelni alat za dizajn), ili unošenjem programskog koda u datoteku (ako se koristi običan tekstualni editor). Fizički gledano, formiraju se fajlovi (datoteke) određenog tipa, koji sadrže kod koji se kompajlira, linkuje i izvršava, ili se interpretira (od strane računara) generišući pri tom korisnički interfejs.

Korisnički interfejsi kod Web aplikacija dizajniraju se u skladu sa ograničenjima Web čitača koji se koriste na klijentskoj strani (Internet Explorer, Netscape navigator, Modzila i sl). Čitač interpretira HTML skript koji je primljen od strane Web servera i otvara Web stranicu koja u sebi može da sadrži formu sa kontrolama. Ovi interaktivni elementi omogućavaju korisniku unos podataka i akcije slično kao kod desktop aplikacija.



Slika 11.6: Izgled korisničkog interfejsa Web aplikacije

Izgled i funkcionalnosti interfejsa određeni su odgovarajućim programskim kodom sadržan u datotekama aplikacija. U ove datoteke mogu da se dodaju i funkcionalnosti za pristup podacima iz BP. Za slučaj kada se direktne funkcije za čitanje, ažuriranje i dodavanje podataka iz/u BP definišu u fajlovima u kojima se definiše korisnički interfejs kažemo da predstavlja *pristup podacima iz prezentacionog sloja*.

Access-ove datoteke predstavljaju najčešći primer pristupa podacima iz prezentacionog sloja. U fajlovima koji imaju *mdb* ekstenziju integriše se kompletna BP, sa formama, izveštajima, makroima i modulima - kodiranim procedurama u VBA[9] jeziku.

```
1:Private Sub Form_Close()
2:DoCmd.RunSQL "UPDATE KolicineSred SET [KOLIC] =
3:      Forms![TSredstva]![RecSum] WHERE KolicineSred.ID_BR =
4:      Forms![TSredstva]![ID_BR] AND
5:      KolicineSred.SifDug=Forms![TSredstva]![SifDug];"
6:End Sub
```

Fragment 1: Pristup tabeli korišćenjem VBA skripta koji sadrži SQL naredbu

[9] VBA – Visual Basic for Access

Ovakve aplikacije su obično vođene događajima korisničkog interfejsa (*Event Driven Applications*). Prethodni primer (Fragment 1) predstavlja proceduru koja je vezana za formu (korisn.interfejs) i koja se aktivira na događaj zatvaranja forme. U linijama 2-5 predstavljena je jedna SQL naredba koja ažurira tabelu *KolicineSred* postavljajući polje *KOLIC* u zapisu prema uslovima u WHERE klauzuli (*id broj* i *šifra dugovanja*) na vrednost sadržanu u kontroli *RecSum* u formi *Tsredstva*.

Na sličan način funkcioniše pristup podacima iz prezentacionog sloja u Web aplikacijama. Kao što je rečeno, ove aplikacije takođe sadrže forme popunjene kontrolama koje omogućavaju unos i pregled podataka.




```
1  <html><body><form method="post" action="demo_add.asp">
2  <table>
3  <tr>
4  <td>Naziv kompanije:</td>
5  <td><input name="firma"></td>
6  </tr><tr>
7  <td>Adresa:</td>
8  <td><input name="adresa"></td>
9  </tr><tr>
10 <td>Postanski broj:</td>
11 <td><input name="postkod"></td>
12 </tr>
13 </table>
14 <p align="left">
15 <input type="submit" value="Dodaj">
16 <input type="reset" value="Odustani">
17 </p></form></body></html>
```

Slika 11.7: Izgled Web forme i HTML kod koji je formira

Forme u Web aplikacijama predstavljene su stranicama koje su kodirane u HTML jeziku (Slika 11.7). Web čitač na klijentskoj mašini interpretira ovaj kod i prikazuje stranicu u svom prozoru (Slika 11.7 levo). U gornjem primeru prikazane su kontrole za unos teksta (*firma, adresa, postkod*). Kada korisnik unese potrebne podatke, pritiskom na dugme dodaj, pokreće se akcija u zaglavlju stranice (lin.1 - Slika 11.7 desno).

```
1:  <html>
2:  <body>
3:  <%
4:  set conn=Server.CreateObject("ADODB.Connection")
5:  conn.Provider="Microsoft.Jet.OLEDB.4.0"
6:  conn.Open "d:/webdata/partneri.mdb"
7:  sql="INSERT INTO kupci (naz_firme, adresa, postbroj)"
8:  sql=sql & " VALUES "
9:  sql=sql & "('" & Request.Form("firma") & "',"
10: sql=sql & "'" & Request.Form("adresa") & "',"
11: sql=sql & "'" & Request.Form("postkod") & "')"
12: on error resume next
13: conn.Execute sql,recaffected
14: if err<>0 then
15:   Response.Write("Nemate prava na dodavanje podataka!")
16: else
17:  Response.Write("<h3>Klijent " & Request.Form("firma") 18:
& " je dodat</h3>")
19: end if
20: conn.close
21: %>
22: </body>
23: </html>
```

Fragment 2: Sadržaj demo_add.asp

Razlika kod Web aplikacija je što se kod za pristup BP izvršava na Web serveru. Podaci koje je korisnik uneo prenose se u vidu HTTP[10] zahteva na server, gde se koriste pri izvršavanju fajla *demo_add.asp*. Navedena datoteka kreirana je u ASP[11] tehnologiji i predstavlja samo jednu od velikog broja Web tehnologija koje omogućavaju dinamičko generisanje sadržaja. Ova datoteka (Fragment 2), pored standardnog HTML koda, sadrži i programski kod pisan u nekom od jezika proizvedenih u Microsoft-u (C++, C#, VisualBasic), koji je korišćen za unos podataka u BP. Nakon uspostave konekcije sa Access-ovom BP *partneri.mdb* (lin.4-6), komponuje se SQL naredba koja treba da izvrši dodavanje podataka u tabeli *kupci* koje je korisnik uneo preko prethodne Web stranice (Slika 6, desno, lin.7-11). Izvršenje stringa SQL naredbe vrši se preko objekta konekcije *conn* (lin.13). Ako je dodavanje zapisa uspešno, server isporučuje klijentskom čitaču zahtevanu stranicu s porukom *Klijent* <naziv> *je dodat*. Ako dodavanje nije uspelo, prikazaće se poruka *Nemate prava na dodavanje podataka*.

Osnovna karakteristika skripta koji se koristi u kreiranju Web stranica je da se isti zasniva na tzv. *tag*-ovima (npr. <html>, ili </body>). Proizvođači *framowork*-ova za

[10] HTTP – Hypertext Transfer Protokol – protokol koji omogućava transfer podataka između Web stranica

[11] ASP – *Active Server Pages*, Microsoft tehnologija za generisaje dinamičkih Web stranica

razvoj Web aplikacija nude i *tag*-ove za komunikaciju sa BP. Na primer za JSP[12] postoji biblioteka *tag*-ova JSTL[13], koja sadrži *sql tag*-ove. Korišćenjem *sql tag*-ove, postiže se neposrednost i standardni pristup u razmeni podataka aplikacije i BP. Da bi se *sql tag*-ovi koristili u Web stranicama, potrebno je uključiti biblioteku tagova i izvršiti konekciju prema BP.

```
1:  <sql:query var="upit1">
2:   SELECT * FROM moja_tabela
3:  </sql:query>
4:  <c:forEach var="naziv_polja" items="${upit1.columnNames}">
5:    <th><c:out value="${naziv_polja}"/></th>
6:  </c:forEach>
7:  <c:forEach var="red" items="${upit1.rows}">
8:    <tr>
9:      <c:forEach var="kolona" items="${red}">
10:      <td><c:out value="${kolona.value}"/></td>
11:     </c:forEach>
12:   </tr>
13: </c:forEach>
```

Fragment 3: Korišćenje SQL tag-ova za generisanje upita

U prethodnom primeru (Fragment 3) predstavljen je upit korišćenjem *sql tag*-a ***query***. Sql tag predstavlja poseban entitet (lin.1-3), koji je imenovan kao *upit1*. U ovom tagu je ugnježden standardni SQL upit (lin.2). Pošto je upit izvršen, dalje se predstavljaju rezultati upita. U tu svrhu koriste se *c tag*-ovi iz JSTL. Najpre se formira zaglavlje tabele u koje se smeštaju nazivi kolona (lin.4-6) koji se dobijaju iz *sql tag*-a pozivom njegove metode *columnNames*. Nakon toga se pomoću 2 for petlje, ugnježdene jedna u drugoj (lin.7-13 – spoljnja, lin.9-11- unutrašnja), formiraju red po red, korišćenjem *sql tag* metode *rows*, a u svakom redu se, korišćenjem *sql tag* metode *value*, popunjavaju konkretne vrednosti podataka za svako polje (kolonu).

PHP je Web orijentisan programski jezik, ali i jedna od najmasovnije korišćenih tehnologija za kreiranje Web aplikacija. U početku čist proceduralan jezik, koji jako podseća na C, prerasta sa verzijama 4 i 5 u objektno orjentisan jezik koji se može integrisati sa drugim tehnologijama.

[12] JSP – *Java Server Pages*, Java tehnologija za generisaje dinamičkih Web stranica
[13] JSTL – *JSP Standard Tag Library*

```
1: mysql_connect("biblioteka.snemanja.net:3617",$username,$password);
2: @mysql_select_db("biblioteka") or die( "Nema konekcije sa BP");
3: $result = mysql_query("SELECT * FROM knjige");
4: $num = mysql_numrows($result);
5: mysql_close();
6: $i=0;
7: while ($i < $num) {
8:   $naslov = mysql_result($result,$i,"naslov");
9:   $autor = mysql_result($result,$i,"autor");
10:  $i++;
11:}
```

Fragment 4: Korišćenje SQL tag-ova za generisanje upita

U prethodnom kodu (Fragment 4) predstavljen je upit pisan u PHP-u. Varijable su označene prefiksom *$* (*$result, $num* itd.). PHP jezik sadrži izuzetno bogatu podršku za MySQL SUBP. Postoji veliki broj ugrađenih funkcija koje jako ubrzavaju pisanje koda a time i produktivnost izrade aplikacija: *mysql_connect* – za konekciju na SUBP (lin.1); *@mysql_select_db* – za izbor BP (lin.2); *mysql_query* – za izvršenje SQL *INSERT/UPDATE/SELECT* naredbe (lin.3); *mysql_numrows* – za dobijanje broja zapisa kao rezultata (lin.4); *mysql_close* – za zatvaranje konekcije sa BP i SUBP (lin.5). Za razliku od ostalih jezika, PHP omogućava pristup podacima u rezultujućem setu nakon raskida konekcije (lin.8,9), tako da je konekcija vremenski minimalna.

Prednosti pristupa podacima u BP iz prezenatacionog sloja je jednostavnost i brzina implementacije. On je pogodan za *jednostavne* aplikacije (sve je *na jednom mestu*), kao što su aplikacije za testiranje i isprobavanje. Ovakav pristup je pogodan i u slučajevima kada se izvršavaju jednostavne SQL naredbe (naredbe koje ne sadrže ugnježdavanje, ne obuhvataju kombinovane podatke iz više tabela) i kada je ciljni SUBP unapred poznat i nepromenljiv.

Za različite vrste SUBP, pa čak i za različite verzije SUBP od istih proizvođača, postoje razlike u sintaksi SQL naredbi. U slučaju promene, ili instalacijom nove verzije SUBP, neretko je potrebno menjati kod SQL naredbi koji se nalazi u objektima korisničkog interfejsa. Ovo je jedna od ključnih slabosti pristupa podacima iz prezentacionog sloja.

Poslovne aplikacije su znatno složenije. Za složenije[14] aplikacije, ovakav pristup bi doveo do konfuzije već u procesu implementacije, jer bi se preklapali poslovi dizajnera i programera. Održavanje i upravljanje ne raslojenim softverom je mukotrpno i često ispod očekivanih rezultata.

[14] Složenost aplikacije, između ostalog, predstavljena je brojem različitih fukcionalnosti koje aplikacija može da obavi

11.2.2 Pristup podacima iz sloja poslovne logike

Ovo je najčešće korišćen pristup kod višeslojnih aplikacija. U sloju poslovne logike postoje entiteti (klase ili moduli) koji su zaduženi za komunikaciju sa BP. Preostali delovi aplikacije razmenjuju podatke sa BP isključivo preko ovih entiteta. U OOP, klase koje omogućavaju interakciju sa BP, obično su već dizajnirane i implementirane od strane proizvođača SUBP, ili proizvođača softverskih alata za razvoj aplikacija. Takve klase su *CDatabase, CRecordset* klase iz Microsoft (MFC) fondacije klasa, *ResultSet, Connection* klase u Java-inom paketu *java.sql.**. Posao programera je, ili da koristi navedene klase u originalnom obliku, ili da kreira nove klase izvođenjem iz ponuđenih, modifikujući im funkcionalnosti prema specifičnim potrebama aplikacije.

U sledećem primeru (Slika 11.8), predstavljen je kod pisan u C++ koji koristi *CDatabase* klasu u osnovnom obliku da bi ostvario konekciju s BP (lin.4 – BP zvana *baza*) i izvedenu klasu *ProizvodiSet* iz *CRecordset* klase za preuzimanje podataka iz tabele proizvoda. Ako je konekcija s BP uspostavljena, dinamički se kreira *pSet* – objekat klase *ProizvodiSet* (lin.6). Funkcija *Open* naasleđena je iz *CRecordset* klase. Ova funkcija ima opcioni broj argumenata. Jedan od argumenata je SQL naredba koja treba da se izvrši u BP. Ako nema argumente, podrazumeva se da se uzima celokupan skup zapisa iz tabele proizvoda.

```
CDatabase db;
ProizvodiSet * pSet=NULL;
try{
    if(db.Open("baza"))
    {

        pSet=new ProizvodiSet();
        pSet->Open();
        while(!pSet->IsEOF())
        {
            m_cLNaziv.AddString(pSet->m_Naziv);
            pSet->MoveNext();

        }
        pSet->Close();
        delete pSet;
        db.Close();

    }
}catch(CDBException* pe){
    AfxMessageBox(pe->m_strError);
    if(pSet->IsOpen()){
        pSet->Close();
    }
delete pSet;
}
```

Korišćenje MFC C++ klasa u komunikaciji s BP

```
try{
  java.sql.Connection conn;
  java.sql.Statement stmt;
  java.sql.ResultSet rset;
  java.sql.ResultSetMetaData rsm;
  conn=this.cpool.checkOut();
  stmt=conn.createStatement();
  stmt.executeUpdate("INSERT INTO " +
          "t_mtutor_groups" +
          " VALUES ( 0, '"+group_name+"')");
  stmt.close();
  this.cpool.checkIn(conn);
}catch(Exception ex){
  ex.printStackTrace();
}
```

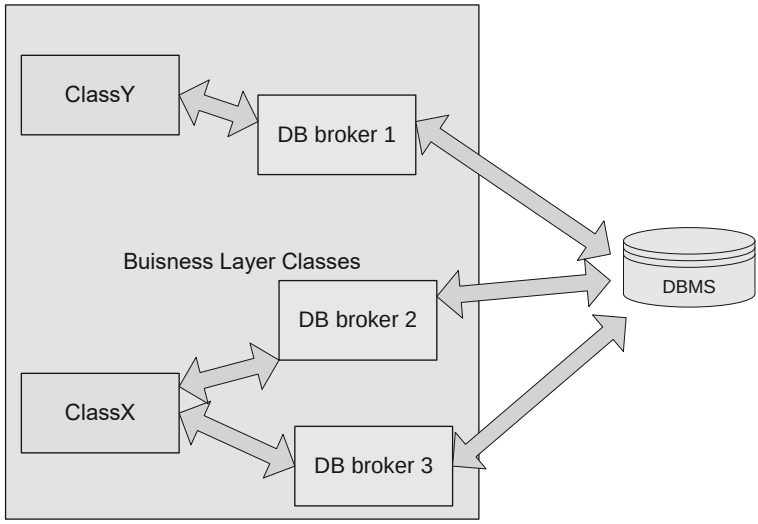
Korišćenje Java klasa iz paketa java.sql u komunikaciji sa BP

Slika 11.8: Komunikacija sa BP iz C++(levo) i Java (desno) aplikacije – primeri

Objekat *pSet* prihvata sve podatke dobijene iz BP. Konkretnim podacima pojedinačnih zapisa se pristupa u cikličnoj strukturi (*while* petlja, lin.8-12). Podaci se dobijaju posredno, preko *pSet* objekta, koji sadrži podatke koji odgovaraju poljima odgovarajuće tabele. Na primer, podatak *m_Naziv* u klasi *ProizvodiSet* (lin.10)

odgovara polju *naziv* u tabeli *proizvodi*. Nakon preuzimanja podataka jednog zapisa, poziva se funkcija *MoveNext* nasleđena od klase *CRecordset,* tako da se u sledećoj iteraciji preuzimaju podaci sledećeg zapisa iz tabele. Nakon preuzimanja podataka, zatvara se i uništava *pSet* objekat (lin.13,14) i konekcija s bazom (lin.15). Ostale naredbe (lin.17 do kraja) namenjene su za rešavanje grešaka tokom konekcije s BP.

U sledećem fragmentu (Slika 11.8) predstavljeno je korišćenje Java klasa iz paketa *java.sql*. Nakon instanciranja potrebnih objekata (*Connection, ResultSet, ResultSetMetaData, Statement*), vrši se povezivanje s bazom (lin.7), a zatim izvršenje SQL naredbe za dodavanje novog zapisa u tabelu *t_mtutor_groups* (lin.8). Nakon toga zatvara se konekcija sa BP, a sve naredbe se obmotavaju u *try catch* blok radi kontrole greške.



Slika 11.9: Pristup BP iz klasa poslovne logike

Oba primera ukazuju na veoma sličnu metodologiju. Najpre se instanciraju potrebni objekti, zatim se uspostavi konekcija s ciljnom BP, obavi se željeni transfer podataka. Pri tome, podaci se uzimaju/menjaju/dodaju iz/u tabele BP, korišćenjem objekata u sloju poslovne logike. Pre završetka metoda konekcija s BP se zatvara na način predviđen standardnim protokolom, a cela operacija se nadzire korišćenjem *try catch* blokova za kontrolu neželjenih izuzetaka. Prednost ovakvog pristupa je što se objekti koji razmenjuju podatke sa BP dizajniraju potpuno nezavisno od prezentacionog sloja (Slika 11.9). Ovi objekti (tzv. *data brokers*) preuzimaju ulogu posrednika između entiteta u BP i ostalih objekata aplikacije.

11.2.3 Pristup iz sloja podataka (poziv ugnježdenih procedura)

Pristup podacima u BP iz sloja poslovne logike ima nekoliko nedostataka. Ovi nedostaci proizilaze iz činjenice da se SQL naredbe (za upite, brisanja, dodavanja i izmene podataka) nalaze direktno u izvornom kodu aplikacije (zapravo u klasama sloja poslovne logike). Takozvano *hardcode*-iranje narušava optimizovanost koda - time i cele aplikacije. Pristup BP iz sloja podataka povećava obim koda, čime se otežava njegovo ažuriranje. Na primer ako se vrši promena strukture, ili naziva jedne tabele u BP, ovo ažuriranje mora da se izvede u svim SQL naredbama u izvornom kodu, u kojima se referenciraju podaci te tabele. Tranzijentno, aplikacija mora ponovo da se generiše, instalira i podešava. Kod aplikacija u velikim poslovnim sistemima, ovakav pristup može da stvori mnoge probleme.

Rešenje za ovakav problem je *izmeštanje* SQL naredbi iz izvornog koda aplikacije u SUBP. SQL naredbe se *ugnježđavaju* kao procedure (*stored procedure*) u ciljnu BP (u jednom SUBP može biti više BP). Većina današnjih SUBP omogućava kreiranje ugnježđenih procedura.

```
1: CREATE PROCEDURE `spUsedTestSets`(IN u_id INTEGER(11))
2: BEGIN
3:     SELECT * FROM `t_mtutor_used_test_sets` WHERE ( user_id = u_id );
4: END;
```

Fragment 5: Definisanje ugnježđene procedure – primer

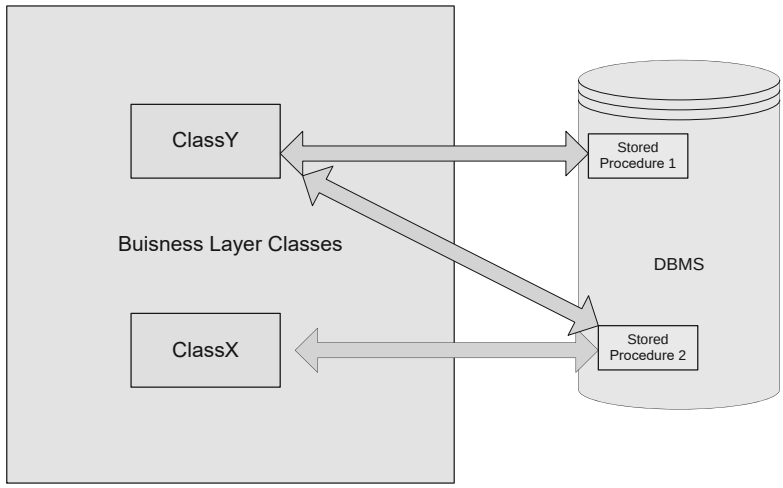
U gornjem fragmentu (Fragment 5) prikazana je definicija 1 ugnježđene procedure u SUBP MySQL 5.x. Ugnježđena procedura slična je bilo kojoj funkciji, izuzev što ne vraća vrednosti, već sadrži samo parametre. Parametri mogu biti ulazni – označeni službenom rečju *IN*, i izlazni – označeni službenom rečju *OUT*. Kod procedura koje vraćaju više zapisa, često se definišu samo ulazni parametri. Umesto izlaznih parametara, rezultati (zapisi) se prihvataju korišćenjem klase koja enkapsulira skup zapisa (npr. *ResultSet,* ili *CRecordset*). Pri definisanju, procedura se najpre imenuje i deklarišu se njeni parametri (lin.1). Implementacija započinje službenom rečju *BEGIN*, a završava službenom rečju *END*. Između početka i kraja je telo procedure u vidu SQL izraza koji treba da se izvrši (lin.3). Ulazni parametri procedure se u telu koriste najčešće kao kriterijumi SQL izraza (*u_id*). Definisane procedure se pozivaju iz aplikacije, prosleđivanjem argumenata i prihvatanjem rezultata njihovog izvršenja.

U sledećem primeru (Fragment 6) prikazan je način korišćenja ugnježđene procedure u Java kodu. Najpre se (lin.1) preko objekta konekcije sa BP (*conn*) vrši njeno pozivanje (naziv procedure je *spUsedTestSets*) i preuzimanje od strane objekta aplikacije (*cs*). Zatim se preko istog objekta prosleđuju parametri (lin.2).

```
1: cs = conn.prepareCall("{call spUsedTestSets(?)}");
2: cs.setInt("user_id", u_id);
3: rs = cs.executeQuery();
4: while( rs.next() ){
5:    int test_id = rs.getInt("test_set_id");
6:    Date test_dat = rs.getDate("date");
7: }
```

Fragment 6: Definisanje ugnježdene procedure - primer

Upit se izvršava eksplicitnim pozivanjem (lin.3). Rezultati upita se prihvataju u objekat klase *Recordset* (*rs*). Kroz cikličnu strukturu (lin.5-7), preuzimaju se pojedinačni podaci iz skupa zapisa dobijenih upitom.



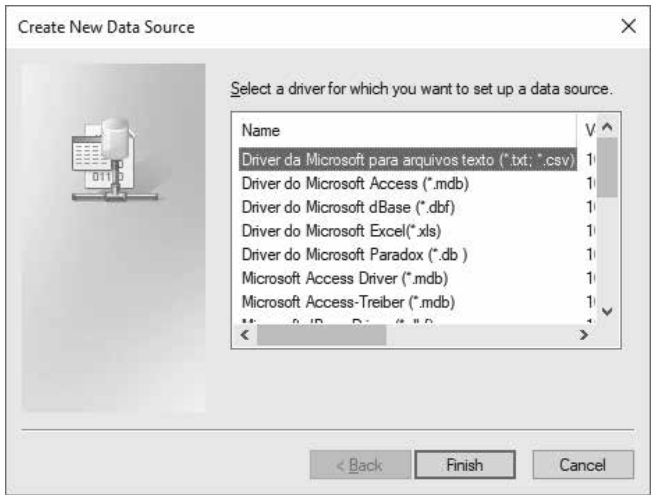
Slika 11.10: Pristup BP iz ugnježdenih procedura

Korišćenje ugnježdenih procedura smanjuje kompleksnost sloja poslovne logike (Slika 11.10). S druge strane, povećava se kompleksnost BP i opterećuje se SUBP. Posledica toga je da se deo programerskih poslova prebacuje na administratore BP. Ugnježdavanje procedura ima još jednu značajnu prednost. Procedure se prave za ciljnu vrstu i verziju SUBP. One se testiraju bez potrebe povezivanja s bilo kakvom aplikacijom. Na taj način je olakšano održavanje i proširivanje složenih sistema na nivou podataka.

11.3 TEHNOLOGIJE KOJE OMOGUĆAVAJU RAZMENU PODATAKA IZMEĐU BP I APLIKACIJA

11.3.1 ODBC (Object Database Connectivity)

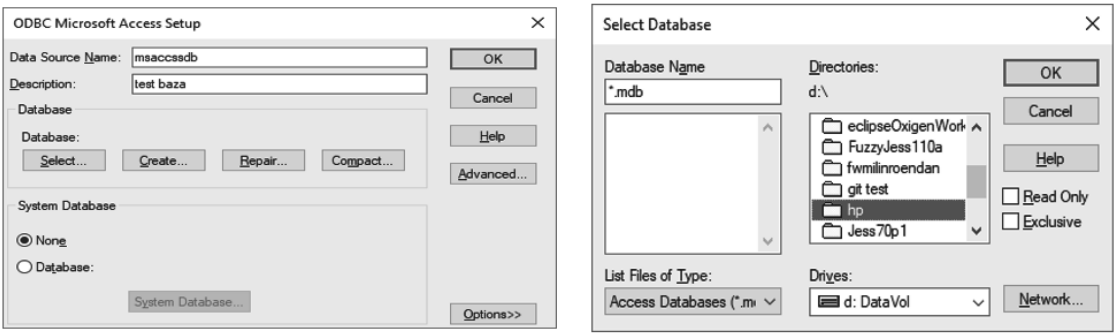
ODBC veznik se ostvaruje korišćenjem ODBC *driver*-a. ODBC *driver* je sistemski softverski proizvod specijalne namene. ODBC *driver*-i su podprogrami koji se sami za sebe ne mogu koristiti. Aplikacije mogu pristupati podacima u SUBP preko ODBC veznika koji se definiše nad specifičnim ODBC *driver*-om. Za svaku verziju sistema za upravljanje BP i operativnog sistema dizaniraju se zasebni ODBC *driver*-i. To znači da se, na primer ne može koristiti isti ODBC *driver* za verziju 3 i za verziju 5 SUBP MySQL. Isto tako, ODBC *driver* za istu verziju SUBP (npr.MySQL v5) ne može se koristiti na različitim platformama (Linux, Windows OS).



Slika 11.11: Postojeći ODBC veznici u sistemu

Proizvođači OS obično u instalaciji OS nude već niz veznika za njihove SUBP. Na primer, Microsoft uz Windows OS instalira na sistem ODBC veznike za njegove SUBP (Access, SQL server, FoxPro). Spisak veznika je dostupan iz administratorske konzole *Control Panel*-a (Slika 11.11).

Tehnologija ODBC-a funkcioniše na sledeći način. Najpre se bira odgovarajući ODBC *driver*. Nakon toga se bira baza podataka s kojom aplikacija treba da razmenjuje podatke. Pre kreiranja aplikacije potrebno je izvršiti registrovanje BP kojoj će se pristupati posredstvom ODBC veznika (Slika 11.12 a i b).



Slika 11.12: Podešavanje konekcije ka BP

Registracija je obavezna i obuhvata dodelu naziva ODBC vezniku i označavanje BP koja će predstavljati izvor podataka u ODBC vezniku. Naziv ODBC veznika ne mora imati nikakve veze sa stvarnim nazivom BP u SUBP. Nakon ovog kratkog postupka, ODBC veznik je spreman za korišćenje.

```
CZabeleskeSet::CZabeleskeSet(CDatabase* pdb)
    : CRecordset(pdb)
{
    m_ID = 0;
    m_poruka = _T("");
    m_nFields = 3;
    m_nDefaultType = snapshot;
}

CString CZabeleskeSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=baza");
}

CString CZabeleskeSet::GetDefaultSQL()
{
    return _T("[Zabeleske]");
}

void CZabeleskeSet::DoFieldExchange(CFieldExchange* pFX)
{
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Long(pFX, _T("[ID]"), m_ID);
    RFX_Date(pFX, _T("[datum]"), m_datum);
    RFX_Text(pFX, _T("[poruka]"), m_poruka);
}
```

Slika 11.13: Korišćenje ODBC veznika iz C++ aplikaciji

U aplikaciji (Slika 11.13) se poziva izvor podataka (ODBC veznik) po svom imenu. Dalje se pristupa tabelama u BP preko naziva tabela, poljima u tabelama preko naziva polja. U primeru sa slike naziv ODBC je *baza*. Tabela kojoj se pristupa naziva se *Zabeleske*, a *ID, datum, poruka* su polja u tabeli. Komunikacija (pregledanje, dodavanje, uklanjanje i editovanje zapisa) vrši se korišćenjem SQL jezika specifičnog za SUBP koji sadrži BP čije podatke aplikacija koristi.

11.3.2 ADO (ActiveX Data Objects)

ADO – ActiveX Data Objects predstavlja tehnologiju koja omogućava pristup svemu što može da poseduje podatke (e-mailovi, Excel tabele, datoteke). ADO dakle poseduje mnogo veću fleksibilnost (u odnosu na vrstu izvora podataka) nego ODBC veznici. ADO tehnologija je dizajnirana da bolje podrži savremene zahteve za distribuiranjem različitih vidova multimedijalnih podataka.

ADO sloj predstavlja nadgradnju nad OLE[15] radi uprošćavanja pristupa podacima. Podacima kao što su video, audio klipovi, ilustracije, mnogobrojni različiti formati dokumenata, moguće je prići iz korisničke aplikacije korišćenjem tzv. ActiveX komponenti. Postoji nekoliko vrsta komponenti:

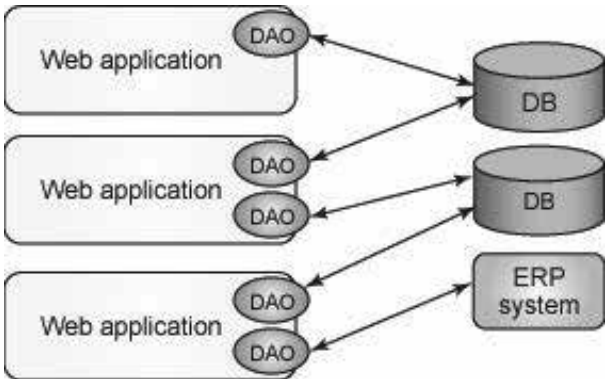
1. **Automatizacioni serveri** (*Automation Servers*) i kontroleri – komponente davaoci (serveri) i potražioci servisa (kontroleri); Primer ovakvog pristupa su aplikacije – mail agenti, koje koriste funkcionalnosti Microsoft Outlook-a; pristup automatizacionim serverima vrši se korišćenjem *IDispatch* interfejsa.
2. **ActiveX Kontrole** –kontrole su ekvivalent OLE Controls (datoteke sa ekstenzijom OCX); one su najčešće namenjene proširenju funkcionalnosti korisničkih interfejsa, npr. omogućavaju prevlačenje, premeštanje grafičkih objekata, tabelarnu prezentaciju podacima iz BP itd.
3. **COM Objekti** – u Windows operativnim sistemima postoji na stotine ovih objekata; svaki od njih sadrži više specifičnih interfejsa kojima se pristupa iz korisničke aplikacije. COM objekti se proizvode za vrlo različite namene, pre svega za korisničke interfejse; najčešće spominjani su renderovanje 3D grafike i promena izgleda korisničkog interfejsa.
4. **ActiveX Dokumenta** –kreiraju se i edituju u ActiveX serverskim aplikacijama (kao što su MSWord, MSExcel), a prikazuju se u ActiveX kontejnerskim aplikacijama (npr. Internet Explorer);
5. **ActiveX Kontejneri –** to su najsloženije ActiveX komponente koje u sebe mogu da prihvate automatizacione servere, kontrole i dokumenta.

[15] OLE (*Object Linking and Embeding*) – ova tehnologija je prethodnica ActiveX tehnologiji, i omogućavala je integraciju podataka različitih formata i iz raznorodnih dokumenata u jednom dokument kontejneru. Razlika u odnosu na ActiveX je što je omogućavala pregled ali ne i editovanje podataka iz drugog izvorišta.

Korišćenjem ADO objekata u aplikacijama smanjuje se kompleksnost aplikacije (broj linija koda napisanih radi ostvarivanja neke funkcionalnosti). Time se smanjuje vreme potrebno za izgradnju aplikacije i povećava produktivnost programiranja.

11.3.3 DAO (Data Access Objects)

DAO predstavlja komponentu koja obezbeđuje zajednički interfejs između aplikacija i određenog skladišta podataka (ciljnog SUBP). Mesto koje zauzima DAO u višeslojnoj arhitekturi aplikacija je na granici sloja poslovne logike i sloja podataka (Slika 11.14).



Slika 11.14: Aplikacije i DAO

DAO omogućava automatizaciju, odnosno potpunu nezavisnost objekata aplikacije od prezentacije podataka u ciljnoj BP. DAO tehnologija izbegava potrebu registrovanja kao što je to slučaj sa ODBC veznicima. Fokusirana je isključivo na BP kao izvore podataka. Bazi podataka preko DAO objekata pristupa se direktno. DAO objekti u aplikaciji ponašaju se kao klijenti u odnosu na SUBP. Omogućava potpuniju kontrolu i jednostavniji pristup svim entitetima u SUBP.

```
1: try{
2:  CDaoDatabase db;
3:  db.Open("s_erw05.mdb");
4:  db.Execute("INSERT INTO t_masters VALUES ('test','test_pwd',0)");
5:  db.Close();
6: }catch(CDaoException* pe){..;}
```

Korišćenje DAO objekata za unos podataka u tabelu

```
1: CDaoDatabase db; CDaoMasterSet ms;
2: try{
3:  ms.Open(AFX_DAO_USE_DEFAULT_TYPE,
4:          "SELECT * FROM t_masters WHERE status = 0 ");
5:  while (!ms.IsEOF()){
6:    fillCombo(ms.m_sUserName);
7:    ms.MoveNext();
8:  }
9: }catch(CDaoException* pe){..;}
```

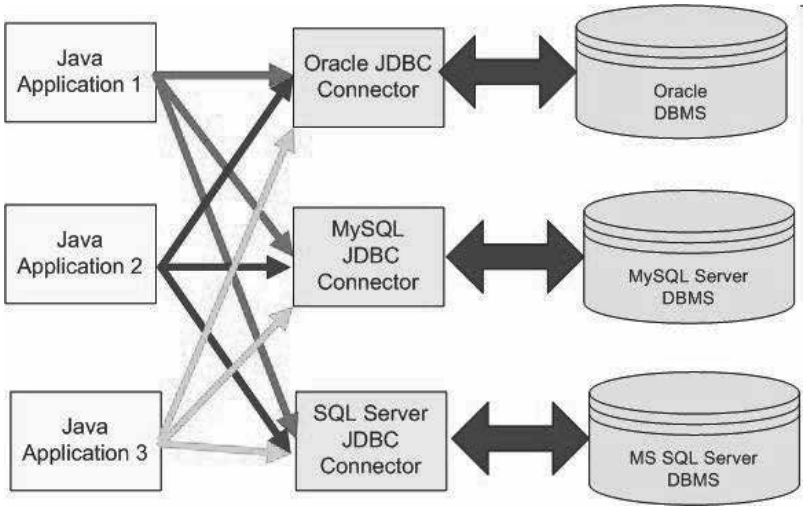
Korišćenje DAO objekata za upit u BP

Slika 11.15: Korišćenje DAO

DAO tehnologija vrši obavijanje aplikacionim objektima svih objekata u SUBP: čitav sistem (SUBP), BP, tabele, polja, indeksi, upiti, korisničke grupe, pojedinačni korisnički nalozi itd. Na taj način izbegava se potreba pristupa nekom posredničkom entitetu (kao što je to ODBC). Unos podataka u tabelu iziskuje svega 6 linija koda (Slika 11.14, gore), dok preuzimanje podataka po zadatom kriterijumu zahteva nekoliko linija koda više (Slika 11.15 dole).

11.3.4 JDBC (Java Database Connectivity)

U paleti tehnologija koje se koriste za povezivanje aplikacija sa BP izdvajaju se kao posebna grupa Java tehnologije. Ove tehnologije zasnovane su na specifičnim svojstvima Java programskog jezika. Platformska nezavisnost, orijentisanost prema mrežnom programiranju i Internetu, kao i izgradnji distribuiranih informacionih sistema učinilo je Java-u možda najdominantnije korišćenim jezikom u izgradnji aplikacija u zadnjim godinama prošlog milenijuma i u ovom milenijumu. Java tehnologije su svugde: od aplikacija za velike poslovne sisteme do mobilnih aplikacija koje same migriraju sa jednog na drugu računarsku platformu (mobilni softverski agenti) ili aplikacija za mobilnu telefoniju.



Slika 11.16: JDBC tehnologija povezivanja aplikacija i SUBP

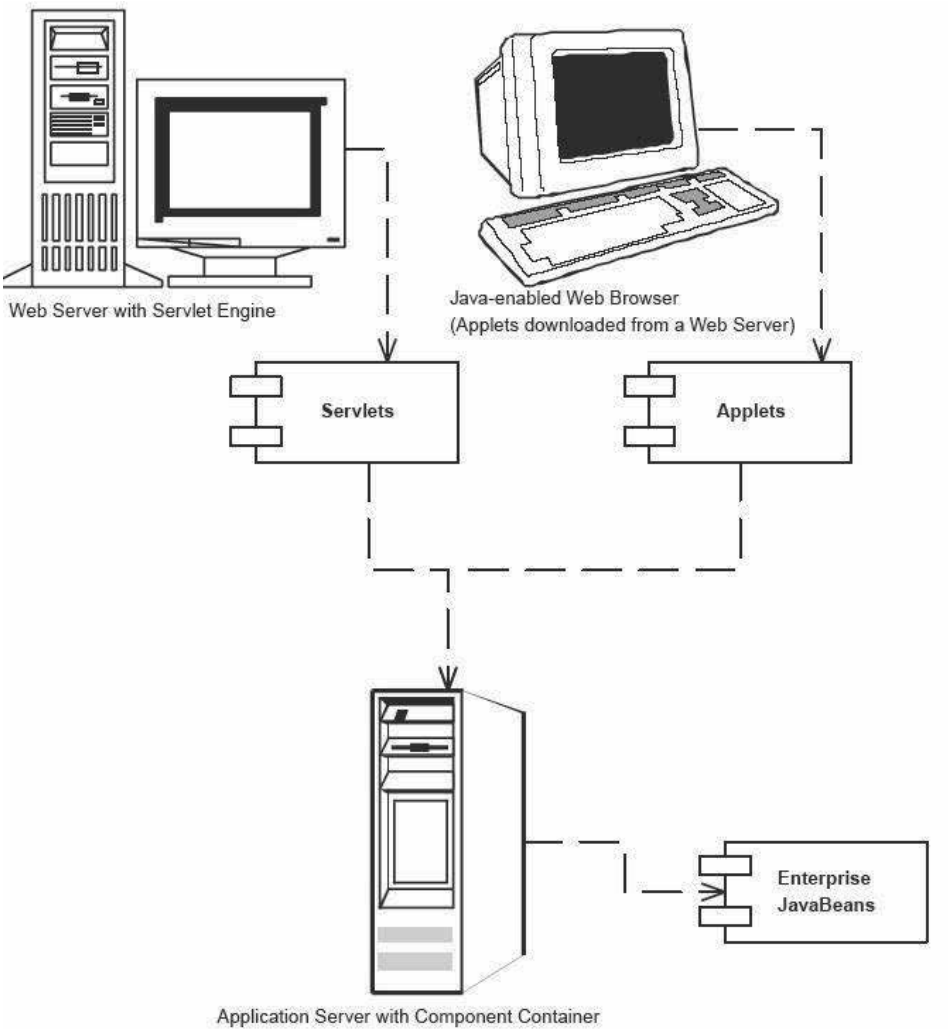
Dominantna tehnologija za povezivanje Java aplikacija na SUBP je JDBC (Slika 11.16). Postoji velika sličnost između ODBC i JDBC konektora. Suštinska razlika je da JDBC konektor ne treba da se registruje na sistem da bi bio operativan i da se konektor pravi prema ciljnom SUBP. JDBC konektor ne koristi resurse OS već resurse Java virtualne mašine (JVM[16]). Isti JDBC konektor koriste konkurentski različite java aplikacije. Na tržištu se mogu pronaći različiti JDBC konektori za iste SUBP. Najpouzdanije rešenje je korišćenje konektora koji za Java-u nudi proizvođač SUBP.

11.3.5 Enterprise Java Beans

Enterprise Java Beans (EJB) predstavljaju nadgradnju koja koristi JDBC konektore kombinovane sa drugim Java tehnologijama (npr. RMI, CORBA) u cilju postizanja visoke produktivnosti u izgradnji distribuiranih IS zasnovanih na Java tehnologijama. Postoji 2 vrste EJB objekata: EJB sesije i EJB entiteta. EJB sesije su klijentski orjentisani i traju koliko i sesija između klijentske i serverske aplikacije (vidi poglavlje 11.2). EJB entiteta predstavlja posrednika između aplikacije i SUBP.

EJB entiteta su perzistentni (postojani) objekti koji predstavljaju *poglede* na željene podatke iz BP. Oni su smešteni u EJB kontejneru (delu serverske aplikacije) i postojani su čak i ako dođe do pada JVM. Pri ponovnom podizanju sistema dolazi i do njihovog ponovnog instanciranja sa podacima do momenta pada. EJB entiteta interaguju posredstvom JDBC sa SUBP na isti način kao i Java klase koje ne koriste *Eneterprise* okruženje.

[16] JVM – Java Virtual Machine predstavlja posrednika između platformskog OS i Java aplikacija



Slika 11.17: Mesto EJB tehnologije u distribuiranim IS

Motivacija za korišćenje EJB je u mogućnostima kombinovanja raličitih tehnologija (RMI, messaging, itd) i jednostavnijeg načina obezbeđivanja boljih performasi sistema (transakciona podrška, perzistentnost podataka).

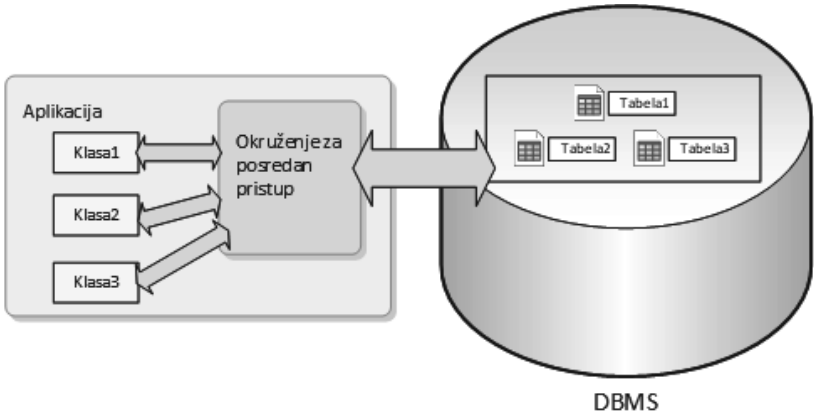
Pristup bazama podataka iz aplikacija može se rešiti na veliki broj različitih načina. Mnogobrojne tehnologije imaju za cilj da vreme izgradnje složenih poslovnih aplikacija što više skrate, a s druge strane da poboljšaju kvalitet aplikacija u smislu performansi, pouzdanosti, fleksibilnosti i sigurnosti podataka. U jednostavnijim aplikacijama i u radu sa transparentnijim podacima može se koristiti pristup podacima iz korisničkog interfejsa. Složeni sistemi koji su najčešće i distribuirani zahtevaju da se podacima pristupa iz sloja poslovne logike ili pozivanjem ugnježdenih procedura u

SUBP. Platformski OS, programski jezik u kome se sistem implementira, korisnički zahtevi u pogledu funkcionalnosti aplikacije, ograničiće dizajnere i programere na izbor konkretne tehnologije za povezivanje aplikacije sa BP. Svaka od predstavljenih tehnologija ima različite prednosti i nedostatke koje implementatori moraju da razumeju kako bi napravili pravi izbor. Dobar izbor tehnologije predstavlja osnovu dobrog početka izgradnje IS.

11.3.6 Okruženja za indirektan pristup SUBP (ORM Frameworks)

Prethodno opisani načini pristupanja DBMS su po svojoj prirodi neposredni, bez obzira da li se radi o pristupu iz sloja koriničkog interfejsa, ili sloja poslovne logike. Drugim rečima, izvorni kod aplikacije sadrži SQL naredbe koje su specifične za određeni tip DBMS. Ova karakteristika predstavlja potencijalnu slabost u slučaju da se podaci migriraju iz jednog tipa DBMS u neki drugi, od potpuno drugog proizvođača. Na primer ako je poslovni zahtev da se podaci više ne čuvaju u DBMS otvorenog koda već u nekom komercijalnom sistemu. Migracija je moguća, ali pored prebacivanja podataka, neophodno je menjati SQL skriptove u izvornom kodu aplikacije radi prilagođenja novom DBMS, što najčešće predstavlja najveći deo posla.

Ovakave scenarijo može da se izbegne zahvaljujući postojanju okruženja za indirektan pristup (Slika 11.18). Ova okruženja (frameworks) su implementirana kao softverski moduli – biblioteke klasa, ili funkcija koje se uključuju a time postaju sastavni delovi aplikacija. Okruženja za indirektan pristup se zasnivaju na objektno – relacionom mapiranju pa se još nazivaju i ORM okruženjima (ORM). Druim rečima svaka tabela u BP koju koristi aplikacija se mapira na odgovarajuću klasu (tip podatka u OOP) aplikacije. Na primer, ako u BP postoji tabela *proizvod*, njoj odgovara klasa *Proizvod* definisana u kodu aplikacije. Tako da prilikom startovanja aplikacije, podacima jednog zapisa iz tabele *proizvod* ORM okruženje puni jednu instancu (objekat) klase *Proizvod* u radnoj memoriji aplikacije. Dalje, u toku rada aplikacije se manipulacija podacima u izvornom kodu vrši indirektno – umesto da se menjaju podaci u zapisima tabela BP, to se vrši u objektima – instancama klasa u aplikacij, a ORM okruženje vrši po zahtevu njihovu sinhronizaciju sa podacima odgovarajućeg zapisa u tabeli BP.



Slika 11.18: Posredan pristup aplikacije DBMS-u

Pošto ORM okruženja pri korišćenju postaju deo aplikacije, prilikom izbora mora da se vodi računa da su pisana u programskom jeziku u kom je pisana i aplikacija. Tako da najpoznatiji primeri okruženja za indirektan pristup iz Java programskog jezika su *Hibernate* i *Eclipse Link*, za Pyton programski jezik to su *Django*, *SQLAlchemy* i *Storm*, za C# to je Microsoft-ovo rešenje *ADO Entity Framework* i Nhibernate (*open source* rešenje).

12 XML (I) BAZE PODATAKA

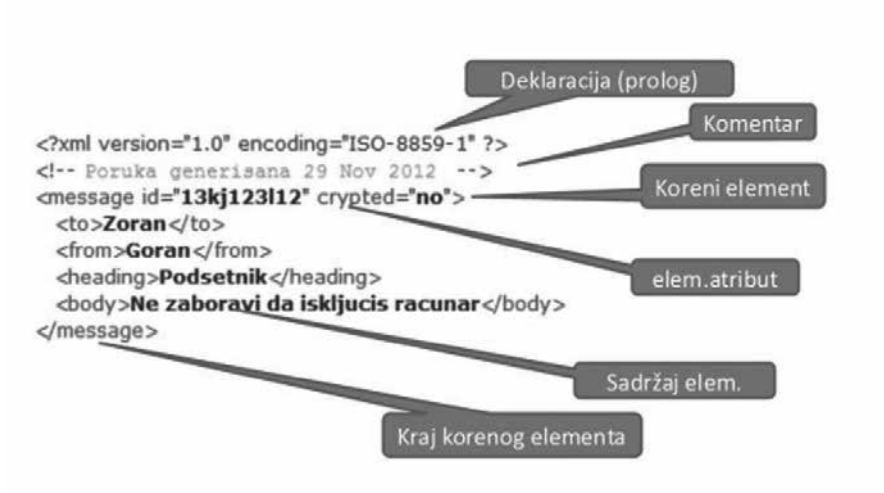
Savremeni informacioni sistemi (IS) predstavljaju mrežne aplikacije zasnovane na internet tehnologijama, bilo da su korisnički interfejsi realizovani u vidu desktop formi, Web stranica, ili posebnih formi za prenosne uređaje kao što su pametni telefoni i tablet računari. Usled čestih promena u preduzećima (npr. unutrašnje organizacione promene, ili kupoprodaja čitavih kompanija), postoji potreba i za udruživanjem (integracijom) IS. Jezgro svakog IS su njegove baze podataka (BP). Poslovni IS skladište slične, neretko iste podatke, koji su struktuirani i formatirani na različite načine. To je jedan od najčešćih problema integracije IS.

Kao rešenje prepoznate su tehnologije zasnovane na XML-u (*Extensible Markup Language*). Format i struktura podataka u BP ostaju nepromenjeni, menja se samo način njihovog predstavljanja - njihova prezentacija van BP. Drugim rečima - različite aplikacije u okviru istog, ili različitih IS razmenjuju podatke kao XML dokumenta, čiji je sadržaj hijerarhijski struktuiran i opisan rečnikom podataka tako da su moguće brze transformacije sadržaja između XML tekstualnog formata i formata ciljne BP. Proizvođači sistema za upravljanje BP (SUBP) su otišli i korak napred - omogućeno je skladištenje podataka u XML formatu, omogućeni su direktni upiti, manipulacija i transakcije nad XML formatiranim podacima (*Native XML* BP). Ovakvo rešenje je prepoznato naročito kao potreba unutar kompanija (unutar istog IS) jer se izbegava dvostruka transformacija podataka (između aplikacija - učesnika u razmeni).

Baze podataka koje, bilo da samo podržavaju transformaciju podataka u / iz XML formata, ili obezbeđuju i skladištenje, upite i manipulaciju podacima u XML formatu nazivaju se XML bazama podataka (XML BP).

12.1 OPŠTE O XML-U

XML predstavlja jezik koji je namenjen za predstavljanje podataka i meta - podataka i koji je proširljiv (*Extensible Markup Language*). XML formatirani podaci se čuvaju u vidu XML dokumenata - u tekstualnim datotekama, koje su prepoznatljive po ekstenziji u nazivu: *xml*. Iako se radi o tekstualnom sadržaju, podaci u XML dokumentu imaju hijerarhijsku struktuiranost. To se postiže XML elementima. Na sledećem primeru objašnjena je struktura XML dokumenta (Slika 12.1).



Slika 12.1: Primer jednostavnog XML dokumenta

Na prethodnoj slici predstavljen je jednostavan XML dokument koji se koristi za razmenu kratkih poruka između aplikacija. Kao što se može videti, na početku XML dokumenta neophodna je deklaracija (*prolog*), koji ne sadrži podatke, već opisuje način na koji su podaci struktuirani (*XML version*) i predstavljeni (*encoding*). Ove meta - podatke koristi aplikacija kako bi verodostojno interpretirala stvarne podatke u XML dokumentu. Svaki XML dokument mora da ima jedan i samo jedan koreni XML element - u ovom slučaju to je element nazvan *message* (poruka). Broj svih ostalih elemenata je proizvoljan. U konkretnom slučaju to su elementi nazvani *to*, *from, heading* i *body*. Sadržaj XML elemenata nalazi se između njegovog startnog i stopnog obeležja (*tag*-ovi). U gornjem primeru sadržaj elementa *heading* je "Podsetnik", a sadržaj elementa *body* je "Ne zaboravi da iskljucis racunar". Elementi ne moraju da imaju sadržaj. Isto tako sadržaj elementa može da bude kompleksan - na primer koreni element *message* sadrži sve ostale elemente i njihove sadržaje. Elementi mogu da imaju atribute. Atributi se navode u startnom tag-u elementa iza naziva i predstavljaju naziv - vrednost par. Na koreni element *message* sadrži 2 atributa: *id* čija je vrednost "13kj123|12" i *crypted* čija je vrednost "no". Pored toga XML dokumenta mogu da sadrže komentare koji se ne uzimaju u obzir prilikom interpretacije sadržaja XML dokumenta, ali mogu da posluže za neke specijalizovane namene (npr. dodatni podaci o sadržaju dokumenta, instrukcije za njegovu obradu i sl.).

Pošto XML dokumenata mogu biti generisana iz različitih izvora unutar IS ili između njih, neophodna je provera ispravnosti (validacija) pre nego što započne interpretacija sadržaja dokumenta. Da bi se izvršila validacija, potrebno je da validator (softverski modul) ima specifikaciju sadržaja (opis) XML dokumenta koji proverava. U tu svrhu najčešće se koriste dve vrste specifikacija:

- DTD (*Document Type Definition*) specifikacija
- XSD (*XML Schema Definition*) specifikacija

Specifikacije se nalaze u posebnim datotekama i prepoznatljive su na osnovu istoimenih ekstenzija u nazivima datoteka - *dtd* i *xsd*. DTD predstavlja stariju, ali jednostavniju specifikaciju. Iz tog razloga i danas se koristi za opise jednostavnih XML dokumenata. U sledećem primeru predstavljen je XML dokument kataloga knjiga i njegova DTD specifikacija (Slika 12.2).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<knjige>
    <knjiga id="1edf12">
        <naslov>Na Drini ćuprija</naslov>
        <autor>Ivo Andrić</autor>
    </knjiga>
    <knjiga id="09ad8f2">
        <naslov>Aska i vuk</naslov>
        <autor>Ivo Andrić</autor>
    </knjiga>
</knjige>
```

test.xml

```dtd
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT knjige (knjiga)*>
<!ELEMENT knjiga (autor|naslov)*>
<!ATTLIST knjiga
    id CDATA #IMPLIED
  >
<!ELEMENT naslov (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
```

test.dtd

Slika 12.2: ML dokument i njegova DTD specifikacija

Odmah je uočljivo da je DTD specifikacija (*test.dtd*) sličan regularnom XML dokumentu, ali nema koreni element, svaki XML element ima isti prefiks (počinje sa *<!*) i elementi nemaju stopna obeležja. Svaki element DTD specifikacije predstavlja opis elementa XML dokumenta (*test.xml*) ili atributa odgovarajućeg elementa.

DTD specifikacija je optimalno rešenje za jednostavne sadržaje, međutim nema dovoljnu izražajnost, zahtevanu u naprednim primenama, gde se podaci najpre agregiraju iz različitih izvora, a zatim isto tako distribuiraju različitim korisnicima (aplikacijama) na različite načine. Primer ovakvih aplikacija su socijalne mreže (Facebook, LinkedIn, Twitter i sl.) koje sarađuju međusobno, ili sa drugim sličnim aplikacijama (npr. Webmail provajderi) u nastojanju da prate aktivnosti korisnika i o tome obaveštavaju druge korisnike. Iz tog razloga, W3C (WWW konzorcijum) razvio je XSD specifikaciju.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="knjige">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="knjiga"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="knjiga">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="naslov"/>
        <xs:element ref="autor"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="naslov" type="xs:string"/>
  <xs:element name="autor" type="xs:string"/>
</xs:schema>
```

Slika 12.3: Izgled XSD specifikacije za isti XML dokument (test.xml)

Na prethodnoj ilustraciji predstavljena je XSD specifikacija za isti fajl kao u prethodnom primeru (*test.xml*). Očigledna je veća kompleksnost, ali zato mnogo detaljnija specifikacija elemenata, čime se postiže veća izražajnost specifikacije. Za razliku od DTD specifikacije, *xsd* datoteka je u potpunosti formatirana po pravilima XML jezika: sadrži jedan koreni element (*xsd:schema*), a elementi pored naziva mogu da imaju atribute i sadržaj (koji predstavljaju ugnježdeni drugi XML elementi).

Nakon definisanja DTD, ili XSD specifikacije, da bi se izvršila validacija, neophodno je da se u XML dokumentu ta specifikacija referencira. Zavisno od tipa specifikacije, različiti su načini referenciranja. DTD specifikacija se referencira kao XML komentar (Slika 12.4) koji započinje rečju *DOCTYPE*, na koji se nastavlja naziv korenog XML elementa (*knjige*), način pristupa (*system*) i putanja do specifikacije (fajl *katalog.dtd*).

```
<?xml version='1.0' ?>
<!DOCTYPE knjige SYSTEM "file://C:\...\katalog.dtd">
<knjige>
    <knjiga id="ledf11">
        <naslov>Na Drini ćuprija</naslov>
        <autor>Ivo Andrić</autor>
    </knjiga>
    <knjiga id="09sd8f2">
        <naslov>Aska i vuk</naslov>
        <autor>Ivo Andrić</autor>
    </knjiga>
</knjige>
```

Slika 12.4: Fajl test.xml koji sadrži podatke za validaciju pomoću DT

U slučaju XSD specifikacije (Slika 12.5), ona se dodaje u koreni element XML dokumenta, u vidu 2 atributa: *xmlns:xsi* - definicija XML imenovanja (*xmlns* - XML Name Space), koja predstavlja standard po kome je napravljena XML shema (*xsi* - XML Schema Instance), i *xsi:noNamespaceSchemaLocation* - putanja do datoteke koja sadrži specifikaciju (*katalog.xsd*). Atribut *noNamespaceSchemaLocation* ima značenje da ne postoji neka posebno definisana specifikacija, već je postojeća urađena u potpunosti po W3C XSD standardu.

```
<?xml version="1.0"?>
<knjige xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="file:///c:/.../katalog.xsd">
    <knjiga id="ledf11">
        <naslov>Na Drini ćuprija</naslov>
        <autor>Ivo Andrić</autor>
    </knjiga>
    <knjiga id="09sd8f2">
        <naslov>Aska i vuk</naslov>
        <autor>Ivo Andrić</autor>
    </knjiga>
</knjige>
```

Slika 12.5: Fajl test.xml koji sadrži podatke za validaciju pomoću XSD

Bilo da se radi o DTD ili XSD specifikaciji, postoji paleta softverskih alata koji su besplatni i omogućavaju automatsko generisanje *dtd* i *xsd* fajlova na osnovu *xml* fajlova i obrnuto. To znači da se lako može rekonstruisati specifikacija (meta - podaci) za bilo kakav *xml* fajl. Ovo svojstvo je značajno u pogledu automatskog kreiranja interfejsa između raznorodnih sistema koji uzajamno razmenjuju podatke u vidu XML dokumenata (na primer, automatsko kreiranje klijentskih aplikacija za priključivanje na Web servise koji koriste XML format poruka).

12.2 XML PROŠIRENJE ZA SQL (SQL/XML)

XML proširenje za SQL (SQL/XML) predstavlja deo SQL 2003 standarda (specifikacije). Ono predstavlja skup dodatnih SQL funkcija koje omogućavaju kreiranje XML dokumenata na osnovu podataka dobijenih SQL upitima nad BP. Vodeći proizvođači SUBP su učestvovali u izgradnji ovog proširenja (Oracle, IBM, Microsoft), tako da je standard ugrađen u većinu aktuelnih distribucija SUBP.

SQL/XML proširenje obuhvata sledeće koncepte:

- Funkcije za kreiranje XML dokumenta (*XML Publishing Functions*)
- XML tipove podataka
- Pravila za mapiranje

12.2.1 Funkcije za kreiranje XML dokumenta

Funkcije za kreiranje XML dokumenta omogućavaju da se rezultati SQL upita predstave u XML dokumentu struktuiranom na željeni način. U sledećoj tabeli su date SQL/XML funkcije

xmlelement()	Kreira XML elemenat, omogućavajući specifikaciju imena
xmlattributes()	Kreira XML atribute od kolona, koristeći naziv kolone kao naziv XML atributa
xmlroot()	Kreira koreni čvor XML dokumenta
xmlcomment()	Kreira XML komentar
xmlpi()	Kreira XML instrukciju za procesiranje
xmlparse()	Parsira string kao XML i vraća rezultujuću XML strukturu
xmlforest()	Kreira XML elemente iz kolona, koristeći naziv kolone kao naziv XML elementa
xmlconcat()	Kombinuje listu pojedinačnih XML vrednosti radi kreiranja jedne vrednosti koja sadrži podatke grupisane konkatenacijom (XML forest)
xmlagg()	Kombinuje kolekciju redova, koji sadrže po jednu XML vrednost, radi kreiranja jedne vrednosti dobijene agregacijom (XML forest)

Tabela 12.1: SQL/XML funkcije

U sledećem primeru predstavljen je način korišćenja funkcija za kreiranje XML dokumenta. Pretpostavimo da BP sadrži dve tabele: *t_korisnici* i *t_narudzbenice* (Slika 12.6) i da, koristeći funkcije SQL/XML proširenja, želimo da automatski generišemo XML dokument koji bi sadržao imena i mesta prebivanja korisnika.

t_korisnici

ID_kor (PK)	korisnik	mesto
1	Goran	Beograd
2	Aleksandar	Jagodina
3	Zoran	Kraljevo

t_narudzbenice

ID_nar (PK)	ID_kor (SK)	datum
1	1	22.2.2013
3	1	17.5.2013

Slika 12.6. Primer baze za objašnjenje SQL/XML

Na osnovu prikazanog sadržaja *t_korisnici* običan SQL upit kao rezultat vratiće sledeći tekst (Slika 12.7).

```
select k.korisnik as ime, k.mesto from t_korisnici k
```

ime	mesto
Goran	Beograd
Aleksandar	Jagodina
Zoran	Kraljevo

Slika 12.7: Običan SQL upit i njegov rezultat (Oracle sintaksa)

Ako bi želeli da generišemo XML dokument istog sadržaja, koristimo funkciju *xmlelement*. kao što je predstavljeno sledećim SQL skriptom (Slika 12.8).

```
select xmlelement ( name "korisnik",
        xmlelement ( name "ime", k.korisnik ),
        xmlelement ( name "mesto", k.mesto ))
from t_korisnici k
```

Slika 12.8: SQL upit sa SQL/XML funkcijom *xmlelement*

Rezultat upita će u ovom slučaju biti tekstualni redovi od kojih svaki sadrži podatke jednog korisnika predstavljene u XML formatu (Slika 12.9).

```
<korisnik>
    <ime>Goran</ime>
    <mesto>Beograd</mesto>
</korisnik>
```

Slika 12.9: Rezultat prethodnog SQL upita

Komponovanjem ovakvih rezultata mogu da se izgrađuju kompletna XML dokumenta. Potpuno isti rezultat dobio bi se i korišćenjem *xmlforest* funkcije (Slika 12.10).

```
select xmlelement ( name "korisnik",
        xmlforest (k.korisnik as ime, k.mesto))
from t_korisnici k
```

Slika 12.10: Upit sa SQL/XML funkcijom *xmlforest*

Sve ostale SQL/XML funkcije (Tabela 12.1) koriste se na sličan način, kombinovanjem, kako bi se dobila dobro formatirana XML dokumenta od podataka u rezultujućim skupovima zapisa dobijenih SQL upitima. Sledeći primer ilustruje dobijanje hijerarhijski složenijeg XML dokumenta korišćenjem složenog upita i SQL/XML funkcija (Slika 12.11).

```
select xmlelement ( name "korisnik",
        xmlattributes ( k.ID_kor as id ),
        xmlforest (k.korisnik as ime, k.mesto)) ,
        xmlelement ( name "narudzbenice",
                select xmlagg ( xmlelement ( name "narudzbenica",
                        xmlattributes ( n.ID_nar as nid ),
                        xmlelement ( name "datum", n.datum )))
                from t_narudzbenice n
                where k.ID_kor = n.ID_kor ))) as "narudzbenice_korisnika"
from t_korisnici k
```

Slika 12.11: Primer složenog upita korišćenjem SQL/XML funkcija

Kombinovani upit ima 2 SQL *select* naredbe. Spoljna naredba namenjena je dobijanje svih korisnika iz *t_korisnici*, pri čemu je identifikator korisnika *ID_kor* predstavljen kao atribut XML elementa *korisnik* nazvan *id*. Unutrašnji upit uzima podatke *ID_nar* i *datum* narudžbenice korisnika iz tabele *t_narudzbenice* po *ID_kor* koji je prosleđen *where* klauzuli upita. Pošto SQL/XML funkcije u unutrašnjim (podsekventnim) pozivanja inicijalno vraćaju samo jedan zapis (samo bi vratile prvu narudžbenicu), neophodno je korišćenje funkcije *xmlagg. P*odatak *ID_nar* predstavljen je kao atribut *nid*, a podatak *datum* narudžbenice kao poseban element koji je ugnježđeni u element *naarudzbenica.* Kao rezultat upita dobija se XML dokument predstavljen na sledećoj ilustraciji (Slika 12.12).

```
<narudzbenice_korisnika>
    <korisnik id="1">
        <ime>Goran</ime>
        <mesto>Beograd</mesto>
        <narudzbenice>
            <narudzbenica nid="1">
                <datum>22.2.2013</datum>
            </narudzbenica>
            <narudzbenica nid="3">
                <datum>17.5.2013</datum>
            </narudzbenica>
        </narudzbenice>
    </korisnik>
    <korisnik id="2">
        <ime>Aleksandar</ime>
        <mesto>Jagodina</mesto>
    </korisnik>
    <korisnik id="3">
        <ime>Zoran</ime>
        <mesto>Kraljevo</mesto>
    </korisnik>
</narudzbenice_korisnika>
```

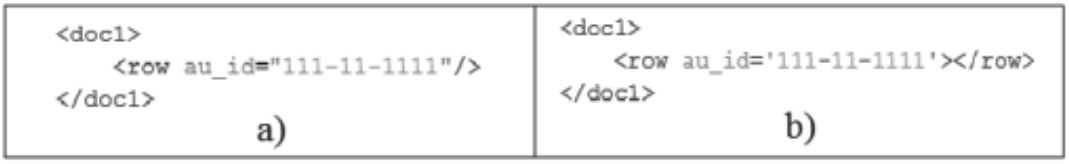
Slika 12.12: XML dokument - rezultat prethodnog upita

12.2.2 XML tip podataka u SUBP

Obzirom da XML/SQL omogućava kreiranje XML dokumenata, neophodno je da postoji definisani XML tip podataka u SUBP (*XML Datatype*). Pošto je XML dokument zapravo tekstualni fajl, XML tip podataka je najčešće implementiran kao karakterski niz *varchar* ili *CLOB* (*Character - based Large Object*), nad kojim je definisan XML/SQL skup funkcija (vidi prethodno poglavlje).

U poljima koja su definisana nad XML tipom podataka može se naći čitav XML dokument, ili njegov fragment. To znači da sadržaj polja može da bude pojedinačan XML element (tag). Ovako fleksibilan pristup ima i ograničenja. Atributi XML tag-ova ne mogu se pojavljivati samostalno, već samo u okviru tagova. Takođe nije specificirano čuvanje samostalnih XML komentara, niti instrukcija za procesiranje. Nad poljem koje ima XML tip podataka ne može se definisati primarni ključ.

Upoređivanje 2 XML dokumenta može da predstavlja složen zadatak. Na nivou kompletnog sadržaja (teksta) oni mogu da budu različiti, ali na nivou podataka da budu isti. Sledeći trivijalan primer (Slika 12.13) ilustruje ovaj slučaj.



Slika 12.13 Primer XML dokumenata koji imaju iste podatke

Ako bi se XML dokument pod *a)* i pod *b)* upoređivali na osnovu poklapanja sadržaja (*pattern matching*), očigledno je da su različiti obzirom da je zapis elementa *row* različit u ova dva slučaja. Ali oni imaju identičnu strukturu (elementi *doc1* i *row*), vrednost atributa *au_id* elementa *row* je takođe identična u oba slučaja (string "*111-11-1111*"). To znači da na nivou podataka, ova dva dokumenta su ista. Ovo je jedan od glavnih razloga zašto ne može da se definiše primarni ključ nad poljem koje sadrži XML tip podataka.

12.2.3 Pravila za mapiranje

Funkcije za kreiranje XML dokumenta predstavljene u poglavlju 12.2.1 koriste SQL vrednosti za kreiranje XML vrednosti podržanih od strane WWW konzorcijuma (*W3C XML Schema Types*). Da bi se omogućila navedena konverzija definisana su pravila za mapiranje, od kojih su najznačajnija:

- SQL skup karaktera se konvertuje u Unicode skup karaktera i obrnuto,
- SQL identifikatori se mapiraju u XML nazive elemenata i obrnuto,
- SQL tipovi podataka se mapiraju u *XML Schema* tipove podataka,
- SQL tabele mogu da se mapiraju u XML dokumenta, ili *XML Schema* dokumenta,
- SQL šeme mogu da se mapiraju u XML dokumenta, ili *XML Schema* dokumenta i
- SQL katalozi mogu da se mapiraju u XML dokumenta, ili *XML Schema* dokumenta

Pored navedenih pravila, moguće je da se izvrši parametrizacija konverzije u smislu definisanja ciljnog prostora imenovanja (*XML namespace*), definisanje strategije u rukovanju NULL vrednostima (npr. ignorisanje, ili predstavljanje sa *xsi:nil* vrednošću), ili / i definisanje strategije za mapiranje SQL tabela u pojedinačna dokumenta ili dekomponovanje u grupe XML dokumenata.

12.3 MANIPULACIJA NAD BP KOJE SADRŽE XML FORMATIRANE PODATKE (NATIVE XML BP)

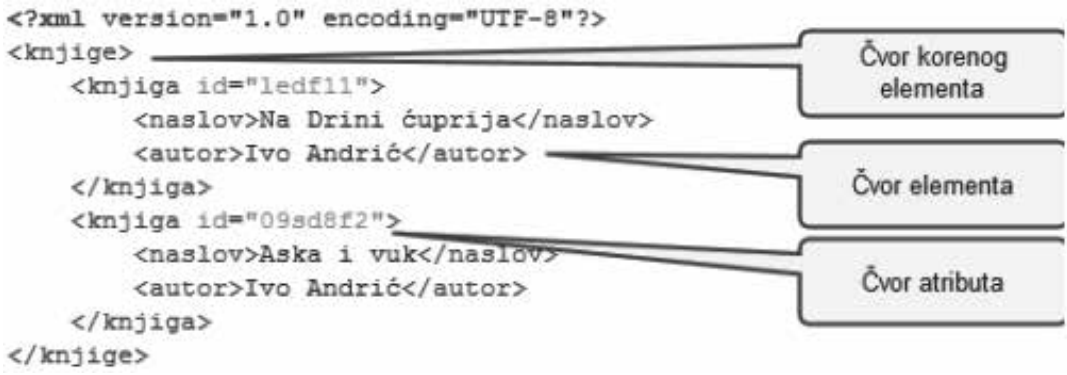
Za razliku od XML proširenja za SQL (SQL/XML) koji predstavlja skup dodatnih SQL funkcija koje omogućavaju kreiranje XML dokumenata na osnovu podataka dobijenih SQL upitima nad BP, za rad nad pravim BP koji sadrže XML formatirane podatke (*Native XML DB*), koriste se posebno dizajnirani jezici koji su prilagođeni modelu i tipu podataka u XML dokumentima. To su XPath i XQuery jezici. U sledeća dva poglavlja predstavljeni su osnovni principi i elementi ova dva jezika, kao i primeri njihovog korišćenja.

12.3.1 XPath

XPath (*XML Path Language*) je upitni jezik namenjen za laku navigaciju (obilaženje) kroz elemente XML dokumenata prema zadatim kriterijumima. Zasnovan je na hijerarhijskoj (stablastoj) strukturi XML dokumenata. Osnovni koncept XPath jezika je čvor *(node).* Postoje čvorovi:

- čvor XML elementa,
- čvor atributa XML elementa,
- tekstualni čvor (sadržaj XML elementa),
- čvor prostora imenovanja (*XML namespace*),
- čvor instrukcije za obradu i
- čvor komentara.

Na sledećem primeru označeni su karakteristični čvorovi, koji se najčešće koriste u izrazima (Slika 12.14).



Slika 12.14: Koncept čvorova u XPath jeziku

Da bi se omogućila navigacija, čvorovi se nalaze u međusobnim relacijama: roditelj, deca, braća, preci i naslednici. Na primer čvor *knjiga* ima roditeljski čvor *knjige* i čvorove decu *naslov* i *autor*. Čvorovi *naslov* i *autor* su *braća* čvorovi, čiji su preci čvorovi *knjiga* i *knjige.* To znači da čvor *knjige* ima naslednike *knjiga, naslov* i *autor*. Osnovni binarni operator je kosa crta - '/' (*slash*), kojom se specificira putanja u XML dokumentu na kojoj se traži neki podatak. Na gornjem primeru da bi se selektovali svi čvorovi *knjiga* - deca čvora *knjige* dovoljno je napisati sledeći izraz:

/knjige/knjiga, ili ***//knjiga***

Drugi slučaj je mnogo opštiji jer omogućava da se selektuju svi čvorovi *knjiga* bez obzira gde se nalaze u dokumentu. Da bi se dobio tekući čvor dovoljno je da izraz sadrži samo tačku '**.**', a da bi se dobio roditeljski čvor - izraz treba da sadrži dve tačke '**..**'. Može se zaključiti da su za osnovnu navigaciju po XML dokumentu, u XPath jeziku iskorišćeni isti principi i operatori kao za navigaciju u fajl sistemu pod Linux OS.

Pored osnovnih postoji niz drugih operatora. Da bi se dobili atributi tekućeg čvora, dovoljno je da izraz sadrži znak '**@**'. Ako nam treba vrednost određenog atributa u čvorovima, onda se specificira i njegov naziv. Na primer, da bismo dobili vrednosti svih atributa imenovanih *id* svih čvorova u XML dokumentu, dovoljan je izraz '***//@id***'. Ako želimo da selektujemo samo prvi čvor *knjiga* onda se koristi izraz sa predikatom '***knjige/knjiga[1]***'. Za selekciju poslednjeg čvora koristi izraz sa predikatom '***knjige/knjiga[last()]***'. Da bi se dobili svi čvorovi *knjiga* kod kojih je autor *Ivo Andrić*, dovoljan je izraz sa predikatom '***knjige/knjiga[autor='Ivo Andrić']***'.

XPath jezik omogućava i formiranje kombinovanih izraza, koji obuhvataju čvorove više putanja. U tu svrhu koristi se operator **I** (*AND*) čiji je zapis uspravna linija '|'. Na primer, sledeći izraz omogućava dobijanje vrednosti čvorova *naslov* i *autor* svih čvorova *knjiga* u dokumentu:

//knjiga/naslov | //knjiga/autor

XPath jezik ima izražajnost ispoljenu i kroz takozvane XPath ose (*axes*). Ose omogućavaju da se dobijaju podaci čvorova u hijerarhijskoj strukturi oko tekućeg čvora. To znači da nije potrebno eksplicitno poznavanje i navođenje naziva čvorova. Sledeća tabela prikazuje sve XPath ose i objašnjenje njihove namene.

XPath osa	Namena
ancestor	selektuje sve čvorove pretke tekućeg čvora
ancestor-or-self	selektuje sve čvorove pretke i tekući čvor
attribute	selektuje sve atribute tekućeg čvora
child	selektuje sve čvorove decu tekućeg čvora
descendant	selektuje sve čvorove naslednike tekućeg čvora
descendant-or-self	selektuje sve čvorove naslednike i tekući čvor
following	selektuje sve čvorove do kraja XML dokumenta nakon tekućeg čvora
following-sibling	selektuje sve čvorove 'braću' tekućeg čvora
namespace	selektuje sve čvorove prostora imenovanja tekućeg čvora
parent	selektuje čvor roditelja tekućeg čvora
preceding	selektuje sve čvorove koji prethode tekućem čvoru od početka XML dokumenta
preceding-sibling	selektuje sve čvorove 'braću' koji prethode tekućem čvoru
self	selektuje tekući čvor

Tabela 12.2: XPath ose

Ose takođe mogu da se koriste u izrazima koji specificiraju nazive čvorova. U tu svrhu koristi se operator dosega '**::**' (inače preuzet iz C++ jezika). Da bi dobili sve čvorove koji prethode čvoru *naziv* (primer na slici 12.14), dovoljan je izraz ***preceding*::*naziv*** (kao rezultat dobijaju se čvorovi *knjiga* i *knjige*. Da bi dobili zaista naslove knjiga iz navedenog primera potrebno je upotpuniti upit

Pored navedenih, u XPath jezik ugrađeni su gotovo svi standardni aritmetički, logički i relacioni operatori. Podrška za navigaciju, različite načine filtriranja sadržaja XMLdokumenata i različiti pristupi podacima koje elementi XML dokumenta sadrže doprineli su neizostavnoj upotrebi XPath jezika, samostalno, ili unutar naprednih XQuery upita nad XML podacima.

12.3.2 XQuery

XQuery je funkcionalni jezik za pravljenje izraza kojima se vrši obrada i pravljenje upita nad XML podacima. Glavni motiv dizajnerima iz W3C konzorcijuma (XQWG - XML Query Working Group) u kreiranju XQuery jezika je pojednostavljivanje načina manipulacije podacima u XML dokumentima i XML bazama podataka. Za razliku od SQL/XML specifikacije (poglavlje 12.3) koja praktično predstavlja proširenje SQL jezika (*ANSI/ISO SQL 2003* specifikacija), XQuery je potpuno novi jezik koji koristi isključivo XML za modelovanje i specifikaciju tipova podataka. I pored toga, XQuery pored XML dokumenata ima mogućnosti upita i nad podacima u relacionoj BP (tzv. XML pogledi).

XQuery omogućava pronalaženje i ekstrakciju elemenata i atributa XML dokumenata. U praktičnim primenama to bi značilo da je namenjen za:

- izvlačenje informacija dobijenih putem Web servisa,
- generisanje sumarnih izveštaja,
- transformisanje podataka iz XML u XHTML i
- pretragu Web dokumenata u potrazi za relevantnim informacijama

Za ekstrakciju podataka iz XML dokumenata XQuery koristi ugrađene funkcije u kombinaciji sa XPath izrazima. Svaka obrada podataka XML dokumenata započinje obavezno njegovim otvaranjem. Ovo vrši XQuery funkcija *doc*. Na primer izraz:

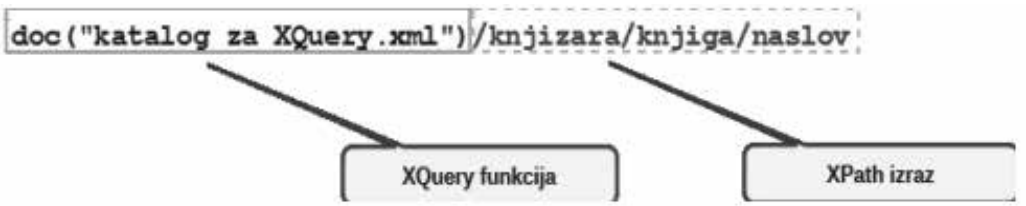
doc ("katalog za XQuery.xml")

otvoriće za obradu XML dokument na tekućem direktorijumu nazvan "*katalog za XQuery.xml*". Sadržaj ovog fajla predstavljen je na sledećoj ilustraciji (Slika 12.15).

```
<?xml version="1.0" encoding="UTF-8"?>
<knjizara>
    <knjiga kategorija="KUVANJE">
        <naslov lang="en">Everyday Italian</naslov>
        <autor>Giada De Laurentiis</autor>
        <godina>2005</godina>
        <cena>305.00</cena>
    </knjiga>
    <knjiga kategorija="DECA">
        <naslov lang="en">Harry Potter</naslov>
        <autor>J K. Rowling</autor>
        <godina>2005</godina>
        <cena>1345.54</cena>
    </knjiga>
    <knjiga kategorija="WEB">
        <naslov lang="en">Learning XML</naslov>
        <autor>Erik T. Ray</autor>
        <godina>2003</godina>
        <cena>839.95</cena>
    </knjiga>
</knjizara>
```

Slika 12.15: Sadržaj dokumenta "katalog za XQuery.xml"

Da bi smo dobili sve naslove iz kataloga knjižare potrebno je izvršiti sledeći XQuery upit (Slika 12.16).



Slika 12.16: Primer XQuery izraza

Upit ima dva dela: na levoj strani je XQuery funkcija *doc* kojoj je prosleđen kao argument naziv fajla, a na desnoj strani se nalazi XPath izraz kojim se specificira podatak koji se traži. Rezultat prethodnog XQuery izraza predstavlja XML formatiran podatak (Slika 12.17).

```
<naslov lang="en">Everyday Italian</naslov>
<naslov lang="en">Harry Potter</naslov>
<naslov lang="en">Learning XML</naslov>
```

Slika 12.17: Rezultat prethodnog XQuery izraza

Gradnja XQuery upita sa kriterijumima (filterima), je takođe jednostavna. Dodajući odgovarajuće predikate na kraj XPath izraza omogućava se filtriranje rezultata. Na primer, modifikujući upit iz prethodng primera dobićemo sve knjige čija je cena manja od 1000 dinara (Slika 12.18).

```
xquery version "1.0";
doc("katalog_za_XQuery.xml")/knjizara/knjiga[cena<1000]
```

Slika 12.18: XQuery upit za dobijanje svih knjiga čija je cena manja od 1000 dinara

Rezultat ovako modifikovanog upita je XML fragment koji sadrži sve elemente *knjiga* zajedno sa pod- elementima *naslov, cena* i *autor*, a koje zadovoljavaju zadati kriterijum pretrage (Slika 12.19).

```
<knjiga kategorija="KUVANJE">
    <naslov lang="en">Everyday Italian</naslov>
    <autor>Giada De Laurentiis</autor>
    <godina>2005</godina>
    <cena>305.00</cena>
</knjiga>
<knjiga kategorija="WEB">
    <naslov lang="en">Learning XML</naslov>
    <autor>Erik T. Ray</autor>
    <godina>2003</godina>
    <cena>839.95</cena>
</knjiga>
```

Slika 12.19: Rezultat XQuery upita sa slike 12.18

Da bi dobili sve knjige namenjene deci, u konkretnom primeru koristimo atribut *kategorija* XML elementa knjiga. Izgled XQuery upita za dobijanje takvog rezultata je kao na sledećoj ilustraciji (Slika 12.20)

```
xquery version "1.0";
doc("katalog_za_XQuery.xml") /knjizara/knjiga[@kategorija='DECA']
```

Slika 12.20: XQuery upit za dobijanje svih dečjih knjiga

Da bi dobili sve knjige namenjene deci, u konkretnom primeru koristimo atribut *kategorija* XML elementa knjiga. Izgled XQuery upita za dobijanje takvog rezultata je kao na sledećoj ilustraciji (Slika 12.21)

```
<knjiga kategorija="DECA">
    <naslov lang="en">Harry Potter</naslov>
    <autor>J K. Rowling</autor>
    <godina>2005</godina>
    <cena>1345.54</cena>
</knjiga>
```

Slika 12.21: Rezultat XQuery upita sa slike 11.20

Upit može da se dodatno sužava tako da dobijamo samo deo podataka u rezultatu. Na primer, da bi smo dobili samo naslove knjiga čija je cena manja od 1000 dinara modifikujemo jedan od prethodnih XQuery upita (Slika 12.22).

```
xquery version "1.0";
doc("katalog_za_XQuery.xml") /knjizara/knjiga[cena<1000]/naslov
```

Slika 12.22: XQuery upit sa filtriranjem kriterijuma i ispisa

daće rezultat (Slika 12.23).

```
<naslov lang="en">Everyday Italian</naslov>
<naslov lang="en">Learning XML</naslov>
```

Slika 12.23: Rezultat XQuery upita sa slike 11.22

Korišćenjem klauzula *for, where, order* i *return*, XQuery omogućava uređivanje rezultata upita po različitim kriterijumima. Na sledećem primeru XQuery upit nad dokumentom "*katalog za XQuery.xml*" omogućava dobijanje naslova svih knjiga sortiranih po godini izdanja.

```
xquery version "1.0";
for $x in doc("katalog_za_XQuery.xml") /knjizara/knjiga
order by $x/godina
return $x/naslov
```

Slika 12.24: XQuery upit kojim se uređuju rezultati pretrage

Da bi izraz bio funkcionalan koriste se promenljive koje su prepoznatljive po prefiksu '*$*'. U gornjem kodu se koristi pomoćna promenjiva *$x*, koja kroz *for* petlju prihvata jedan po jedan element *knjiga* i uređuje ga prema godini izdanja i vraća zatim uređene naslove knjiga. Rezultat su elementi *naslov* čiji redosled je izmenjen u odnosu na originalni (Slika 12.25).

```
<naslov lang="en">Harry Potter</naslov>
<naslov lang="en">Everyday Italian</naslov>
<naslov lang="en">Learning XML</naslov>
```

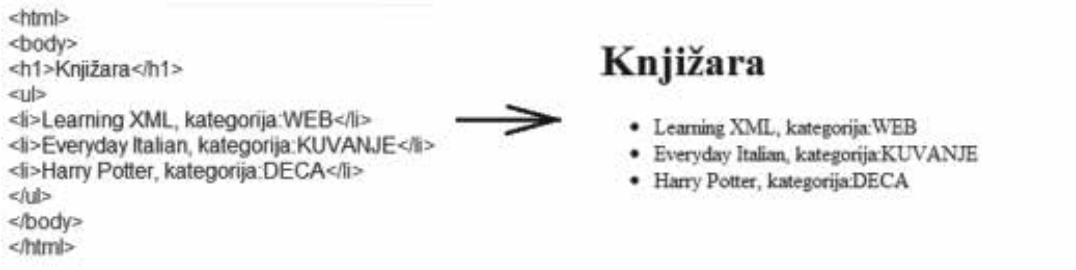
Slika 12.25: Rezultat prethodnog XQuery upita

U slučaju da rezultati upita (podaci) treba da se prikažu na nekoj Web (HTML) stranici, potrebno je njihovo dodatno uređivanje. U tu svrhu dovoljna je modifikacija prethodnog upita kako bi se dobila dobro formatirana HTML stranica (Slika 12.26).

```
xquery version "1.0";
<html>
  <body>
    <h1>Knjižara</h1>
    <ul>
    {
    for $x in doc("katalog_za_XQuery.xml") /knjizara/knjiga
    order by $x/godina
    return <li>{data($x/naslov)}, kategorija:{data($x/@kategorija)}</li>
    }
    </ul>
  </body>
</html>
```

Slika 12.26: Modifikovan XQuery upit radi uređivanja prikaza u HTML format

XQuery upitu se dodaju HTML elementi (uokvireni crvenim), a sam upit se uokviruje vitičastim zagradama *'{'* i *'}'*kao poseban blok. Korišćenjem funkcije *data* omogućena je ekstrakcija (izdvajanje) podataka XML elemenata *knjiga* - tekst naslova i vrednost atributa *kategorija*. Kao rezultat dobija se HTML dokument koji je čitljiv u Web pretraživaču (Slika 12.27).



Slika 12.27: Rezultat prethodnog XQuery upita i njegov prikaz u Web pretraživaču

U dosadašnjim primerima, radi demonstracije i objašnjavanja načina korišćenja XQuery upita korišćena je funkcija *doc* kojom se učitavao sadržaj XML dokumenta u radnu memoriju. U slučaju da se sadržaji XML dokumenata čuvaju u XML bazi podataka, onda su oni uskladišteni u poljima definisanim nad XML tipovima, koji su specifični, razlikuju se u zavisnosti od proizvođača SUBP. Na primer IBM DB2 sistemi imaju jedan XML tip - *XML kolonu* (polje). Prilikom definisanja polja koje je tipa XML kolona nema specifikacije veličine podataka. Sve XML kolone su limitirane na 2 GB (u razmeni podataka). Pored navedenog, postoje i drugi SUBP: besplatna rešenja kao što je *eXist-db,* univerzitetska rešenja kao što je *Berkeley DB XML,* ili skupa komercijalna rešenja poput *Mark Logic*-a.

13 SKLADIŠTENJE PODATAKA (DATA WAREHOUSING)

U savremenim poslovnim sistemima postoji potreba korišćenja podataka iz različitih izvora. Do takve situacije najčešće se dolazi usled promena kao što su udruživanja različitih kompanija, reorganizacija i restrukturiranja delova sistema zbog optimizacije poslovnih procesa, usled kupovine/prodaje delova ili čitavih poslovnih sistema itd. Kao posledica, podaci bitni za procese upravljanja u takvim složenim sistemima nisu više na jednom mestu, na jednoj BP, niti su deo istog IS. Postoje različiti slučajevi distribuiranosti podataka od kojih su karakteristični:

- u različitim BP,
  - na jednoj računarskoj platformi, u okviru istog SUBP,
  - na različitim SUBP u okviru istog IS,
  - na različitim SUBP različitih IS i
- u vidu dokumenata, van BP, struktuirani ili ne, dostupni preko mreže, u različitim formatima.

U svim slučajevima migracija (prebacivanje) podataka sa "pridruženih" sistema na "matični", radi njihovog integirisanja može da prouzrokuje nestabilnost, nepouzdanost i pad performansi IS, a otklanjanje posledica može da bude proces koji troši puno resursa (vremenski zahtevan, uz angažovanje velikog broja zaposlenih). Uprkos problemima, rukovodstvo kompanije nastoji da u procesima odlučivanja obuhvati što više ulaznih informacija, iz različitih izvora kako bi odluke bile objektivnije (odgovarale stvarno situaciji) i ispravnije.

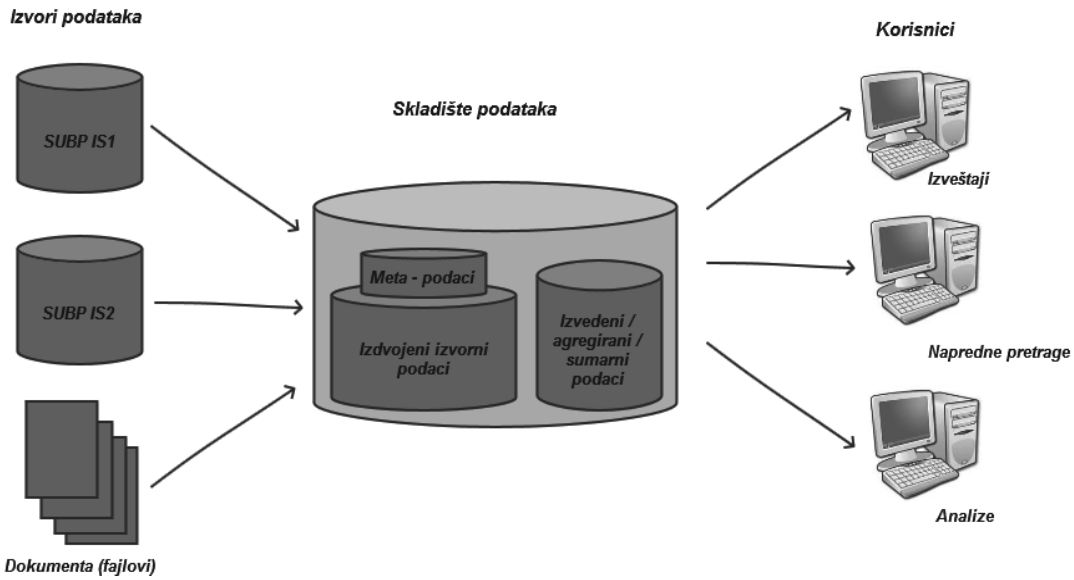
Proizvođači SUBP i poslovnih IS su podržali opisane potrebe fleksibilnim softverskim rešenjima namenjenim za podršku u odlučivanju (SPO - sistemi za podršku u odlučivanju). Ovi sistemi nisu namenjeni za transakcionu obradu podataka kao što je to slučaj konvencionalnih IS i SUBP (*OLTP - Online Transactional Processing*), već su namenjeni za ekstrakciju, transport, transformaciju (*ETL*) podataka iz različitih izvora, zatim njihovu analitičku obradu u realnom vremenu (*OLAP - Online Analytical Processing*). Ovi procesi su obuhvaćeni jednim terminom - "Skladištenje podataka" (*Data Warehousing*). Kao rezultat navedenih procesa dobijaju se skladišta podataka koja se razlikuju od konvencionalnih BP po sledećim karakteristikama:

- mnogo više se koristi indeksiranje nego kod konvencionalnih BP - čime se povećava brzina pristupa kompleksnim strukturama podataka,
- povezivanje (spajanje) podataka iz različitih izvora su retka,
- podaci nisu u potpunosti normalizovani - najčešće su redundantni, nisu u 3. normalnoj formi kako bi se povećala brzina izvršenja upita,

- često je korišćenje izvedenih i agregiranih podataka,
- upiti nad podacima nisu predefinisani kao kod konvencionalnih BP, već se formiraju po zahtevu (*ad hoc*),
- podaci se koriste isključivo za izveštavanje (upiti), dok se kod konvencionalnih BP pored upita podaci mogu menjati, dodavati i uklanjati,
- u izvršenju upita nad skladištima podataka, pretraga se vrši na mnogo većem broju zapisa nego kod konvencionalnih BP i
- u izvršenju upita nad skladištima podataka obuhvataju se podaci iz mnogo dužih vremenskih razdoblja nego kod konvencionalnih BP.

13.1 ARHITEKTURA SKLADIŠTA PODATAKA

Skladišta podataka mogu da budu realizovana na različite načine. Za podršku osnovnim procesima, navedeni u prethodnom poglavlju neophodna sistemska arhitektura ima 3 sloja (Slika 13.1) pri čemu skladište podataka ima ulogu posrednika između različitih izvora podataka s jedne strane i poslovnih korisnika s druge.



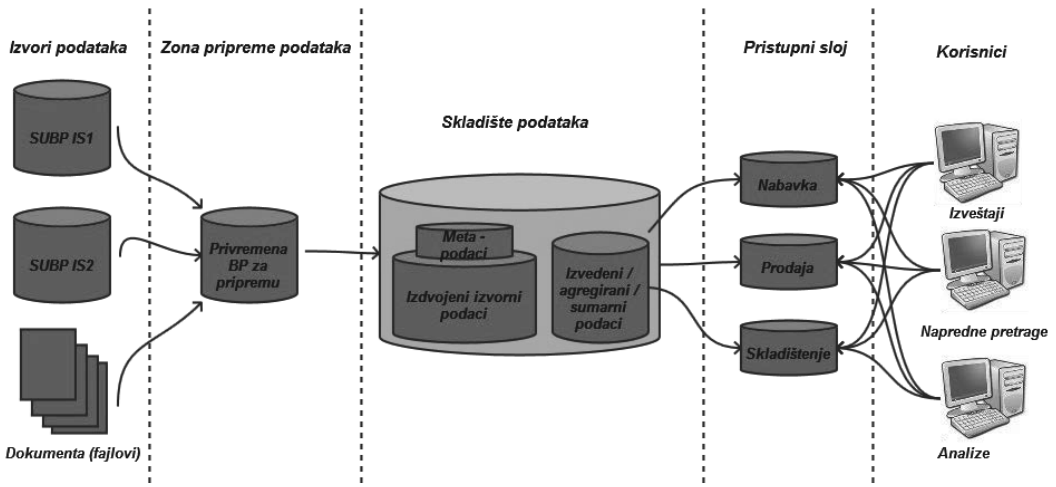
Slika 13.1: Osnovna arhitektura skladišta podataka

Očigledan problem kod osnovne arhitekture su različite performanse sistema koji su izvori podataka. Izvorni podaci su u različitim SUBP, u okviru istih, ili različitih IS, neki se nalaze u datotekama, manje ili više struktuirani, tako da je brzina izdvajanja

(ekstrakcije) podataka nestabilna i nepredvidiva. To za posledicu ima smanjivanje kvaliteta servisa koji skladište može da ponudi krajnjim korisnicima. Poseban problem predstavlja i to što se korisnički zahtevi ne znaju unapred, samim tim i upiti ne mogu da budu predefinisani, već se vrše tzv. *ad hoc* upiti - upiti po zahtevu. Ovi upiti se, posmatrano na duži vremenski rok, mogu tipizirati obzirom da u različitim delovima kompanije (odeljenjima, jedinicama) važne su različite grupe podataka. Na primer, deo koji se bavi nabavkom, da bi se odlučio za dobavljače, koristi podatke iz kataloga ponuđača, bilo da se radi o tekstualnim, tabelarnim sadržajima, ili BP koje su dostupne za distribuciju korisnicima. S druge strane, deo koji se bavi prodajom mora da prati situaciju na tržištu, konkurenciju, kretanje cena, pojavu novih sličnih proizvoda i mnoge druge podatke. To znači da izvedeni podaci dobijeni obradom u skladištu mogu takođe da se unapred grupišu kako bi se skratilo vreme izvršenja *ad hoc* upita.

13.1.1 Proširena arhitektura

Da bi se navedeni problemi prevazišli, u cilju optimizacije, osnovna arhitektura se najčešće proširuje (Slika 13.2) tako da se operativni podaci iz različitih izvora najpre pripreme, kako bi se omogućila njihova integracija. Ovaj sloj se naziva zona pripreme podataka (*staging area*). U njoj se vrši prikupljanje podataka iz različitih izvora i skladištenje u privremenu BP. Uvođenjem ovog međusloja, izbegava se pojava vremena 'čekanja' na podatke i 'sinhronizuju' se različite brzine dobijanja podataka iz različitih izvora. Isto tako, predobrada u zoni pripreme podataka omogućava plansko korišćenje izvora podataka (*scheduled data retrivial*) i njihovo "brže oslobađanje" usled optimizacije.



Slika 13.2: Proširena arhitektura

Kao što je napomenuto *ad hoc* upiti se mogu tipizirati prema delovima kompanije, što omogućava i grupisanje već obrađenih podataka u skladištu. Ove grupe podataka predstavljaju skladišta 'za sebe' obzirom da su podaci u različitim grupama izolovani (*data marts*). Ona se nalaze u tzv. pristupnom sloju (*access layer*). U njima su obrađeni podaci, grupisani po kriterijumima od interesa za poslovne korisnike - skupovi podataka prema poslovnim linijama, ili timovima. U primeru sa prethodne ilustracije to su odeljenja za nabavku, prodaju i skladištenje. U ovim pod - skladištima mogu da se nađu isti podaci, što otežava njihovo održavanje (*data refreshing*) (ažuriranje redundantnih podataka) i što predstavlja cenu povećanja performansi prema korisnicima (sistemima za izveštavanje, analitičkim alatima i aplikacijama za napredno pretraživanje - *data mining*).

13.2 LOGIČKI DIZAJN SKLADIŠTA PODATAKA

Logičkim dizajnom skladišta podataka dobija se njegov konceptualni model - shema skladišta podataka. Za razliku od logičkog dizajna konvencionalnih BP, u kojima se za opis pretežno koriste dijagrami objekti - veze (*ER - entity relationships* dijagrami), u dizajniranju skladišta koriste se tzv. dimenzioni modeli.

13.2.1 Formiranje dimenzija

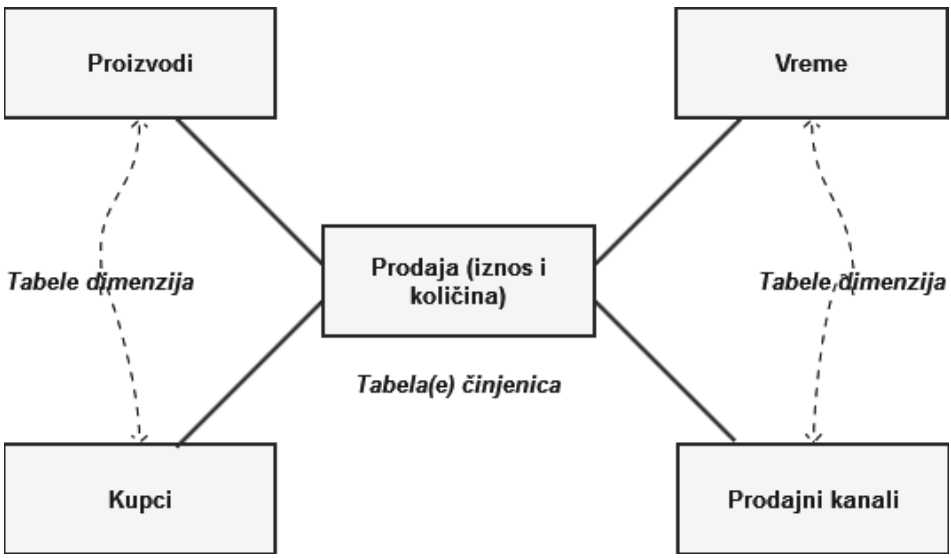
Dimenzioni modeli su zasnovani na izvedenim strukturama podataka koje se grade kako bi skladište moglo da odgovori na *ad hoc* upite korisnika i koje se zovu jednim imenom *dimenzije*. Koristeći prethodna razmatranja (Slika 13.2), a da bi objasnili kreiranje dimenzija, uzećemo primer formiranja grupe podataka 'Prodaja'. Da bi menadžment prodaje mogao da analizira prodaju na svim prodajnim mestima tokom različitih vremenskih perioda i da bi mogao da sagleda uticaj promotivnih kampanja, pored činjenica (*facts*), potrebno je da se definišu i dimenzije skladište podataka, kojima bi se omogućilo njihovo grupisanje prema poslovima prodaje. U konkretnom slučaju kandidati za dimenzije koje treba da budu uključeni u analizu prodaje su:

- proizvodi,
- klijenti,
- vreme,
- prodajni kanali i
- promotivne kampanje

13.2.2 Shema zvezde

Shema skladišta se formira povezivanjem odabranih dimenzija na tabelu činjenica. Pri formiranju sheme skladišta obavezno se u razmatranje moraju uzeti sheme izvora podataka na osnovu kojih se formira skladište. Najčešće korišćena shema je tipa

zvezde (*star*) (Slika 13.3). U centru zvezde se smešta tabela činjenica (*facts table*). Ova tabela sadrži najveću količinu podataka (u odnosu na tabele dimenzija). Ti podaci su u najčešćem slučaju mere poslovanja, na primer vrednost i količina prodaje, cene, zarada i sl. Ako tabela činjenica sadrži agregirane (zbirne) podatke, onda oni treba da budu na istom nivou agregacije. U konkretnom primeru tabela činjenica sadrži podatke o novčanim iznosima i količini prodaje. Još jedan termin je prihvaćen kao naziv za tabelu činjenica - kocka (*cube*). Ovaj naziv ima veze sa najčešćom grafičkom reprezentacijom tabele činjenica i njenih tabela dimenzija. U praksi broj dimenzija nije limitiran (kocka ih ima samo tri), tako da će u daljem tekstu i dalje biti korišćen termin tabela činjenica.



Slika 13.3: Shema zvezde sa 4 dimenzije za analizu prodaje

Kao kraci zvezde na centar se povezuju odabrane dimenzije. U primeru to su objekti koji su direktno povezani sa podacima prodaje: prodati proizvodi, kupci, kanali prodaje i vremenski periodi koji se razmatraju. Tabele dimenzija se još nazivaju i tabelama za pronalaženje (*lookup tables*). Ove tabele najčešće sadrže reference na konkretne podatke, koje omogućavaju kategorizaciju podataka kroz agregiranje. Na taj način dobijaju se različiti nivoi agregacije, a dimenzione tabele dobijaju hijerarhijsku uređenost. Na primer, kupci mogu da se kategorizuju prema mestima stanovanja, a zatim dalje prema regionima prodaje u kojima su uključena mesta stanovanja. Za tu namenu u tabeli dimenzije se koristi referenciranje "Na sebe" (Slika 13.4).

ID	NAZ	NIVO_AGR	REF
1	str. Vlada	prodajno mesto	4
2	Fortuna d.o.o.	prodajno mesto	4
3	Vaxi a.d.	prodajno mesto	6
4	Kruševac	mesto	7
5	Kraljevo	mesto	7
6	Trstenik	mesto	7
7	Zapadno-moravski	region prodaje	0



Slika 13.4: Princip formiranja agregirane (hijerarhijske) dimenzione tabele ' prodajni lanci'

Da bi se kreirala hijerarhijska struktura, dimenziona tabela '*prodajni lanci*' mora da ima najmanje četiri polja. Nad poljem *ID* definisan je primarni ključ, polje *NAZ skladišti* naziv objekta u tabeli, a polje *NIVO_AGR* skladišti agregacioni (hijerarhijski) nivo objekta u tabeli. Ključno polje je polje *REF*, koje čuva reference objekata na druge objekte iz iste tabele, ali koji se nalaze na višem nivou agregacije. U gornjem primeru postoje 3 nivoa agregacije: *prodajno mesto, mesto, region prodaje*. Prva 3 zapisa na primeru su zapisi prodajnih mesta (*str.Vlada, Fortuna d.o.o. i Vaxi a.d.*). Sledeća 3 su mesta (gradovi). Na osnovu podataka u polju *REF* zaključuje se da se prodavnice *str.Vlada* i *Fortuna d.o.o.* nalaze u Kruševcu, a prodavnica *Vaxi a.d.* se nalazi u Trsteniku. Sva tri grada *Trstenik, Kruševac* i *Kraljevo* pripadaju *Zapadno - moravskom* regionu prodaje.

Drugi način kreiranja hijerarhije podataka u tabelama dimenzija je da, umesto primarnih ključeva, kao reference se koriste konkretni nazivi objekata. Minimalan broj polja zavisi u ovom slučaju od broja nivoa *hijerarhije*. Redizajnirana tabela iz prethodnog primera bi, umesto da ima izražene nivoe agregiranosti kao vrednosti polja (*NIVO_AGR*, Slika 13.4), u stvari morala da ima definisana polja za svaki nivo agregacije (hijerarhije).

ID	prodajno mesto	mesto	region prodaje
1	str. Vlada	Kruševac	Zapadno-moravski
2	Fortuna d.o.o.	Kruševac	Zapadno-moravski
3	Vaxi a.d.	Trstenik	Zapadno-moravski

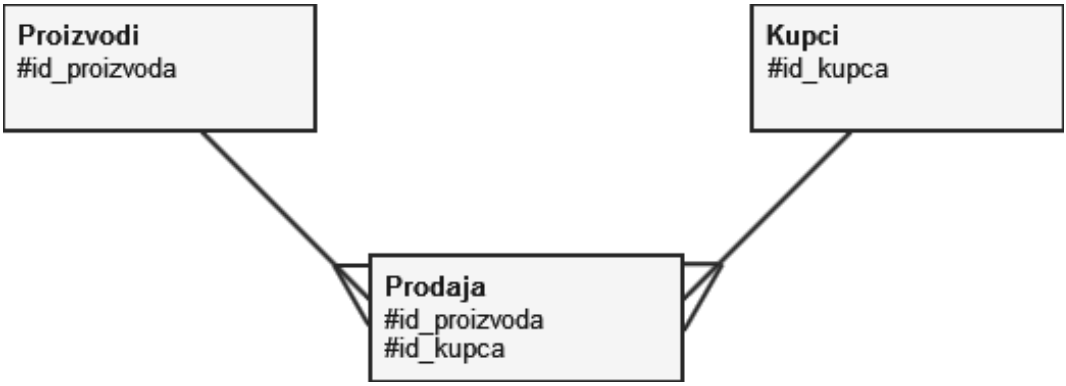
Slika 13.5: Hijerarhijska struktura definisana preko polja dimenzione tabele

Nedostatak drugog pristupa je nefleksibilnost sistema, jer na primer u slučaju rasta kompanije van granica države, shema skladišta bi morala da se permanentno menja. U konkretnom slučaju, trebalo bi da se dodaju na primer polja: za pripadnost državi, kontinentalnom regionu, kontinentu. U slučaju definisanja hijerarhijske strukture preko polja dimenzione tabele, skladišta moraju da sadrže u pozadini softversku podršku za održavanje referencijalnog integriteta podataka. To su posebne tabele referenci na izvorne podatke kako bi se izbegla dvosmislenost u tumačenju izvedenih podataka. Na primer, za hipotetičku internacionalnu kompaniju koja posluje u Češkoj i Srbiji, može se desiti da ima dva istoimena regiona prodaje: Zapadno - moravski regioni, ili ako kompanija ima prodajna mesta i u Mađarskoj mogu da se pojave dva Podunavska regiona.

Na prikazane načine kreiraju se tabele i za druge dimenzije. Na primer, vremenska dimenzija može da se razmatra na časovnom, dnevnom, mesečnom, kvartalnom i godišnjem nivou u zavisnosti od potreba menadžmenta. Mogu da se analiziraju na primer, u koje doba dana, kojim danima u nedelji, mesecima, ili kvartalima u godini prodaja beleži maksimalne, ili minimalne vrednosti, razmatrano prema mestima i regionima prodaje. Kao što je napomenuto, u slučaju agregacije podataka u dimenzionim tabelama, neophodno je da sadrže podatke istog (sličnog) nivoa agregiranosti. Na primer, besmisleno je da menadžment analizira kupovinu jednog kupca na mesečnom, kvartalnom ili godišnjem nivou, već je u tom slučaju agregiranost po mestu i regionu prodaje neophodna.

Tabela činjenica koja se nalazi u središtu šeme skladišta povezuje se sa tabelama dimenzija konceptom spoljnih i primarnih ključeva. Tako da pored izvedenih podataka, tabela činjenica sadrži spoljnje ključeve za održavanje veza prema tabelama dimenzija. Kao primarni ključ tabele činjenica uobičajeno je da se koristi kompozit spoljnih ključeva prema tabelama dimenzija. U predstavljenom primeru tabela činjenica sadrži podatke o prodaji a povezana je na dimenzije *proizvodi* i *kupci* (Slika 13.6)).

Tabela činjenica *prodaja* predstavlja tabelu sastavnicu u odnosu na tabele dimenzija *proizvodi* i *kupci*. To znači da su veze *proizvodi* - *prodaja* i *kupci* - *prodaja* tipa jedan prema više, tako da se jedna vrsta proizvoda može naći u više prodaja, kao i da jedan kupac može da obavi više kupovina. Primarni ključevi *id_proizvoda* i *id_kupca* predstavljaju spoljnje ključeve u tabeli *prodaja* i istovremeno su delovi kombinovanog primarnog ključa ove tabele činjenica.

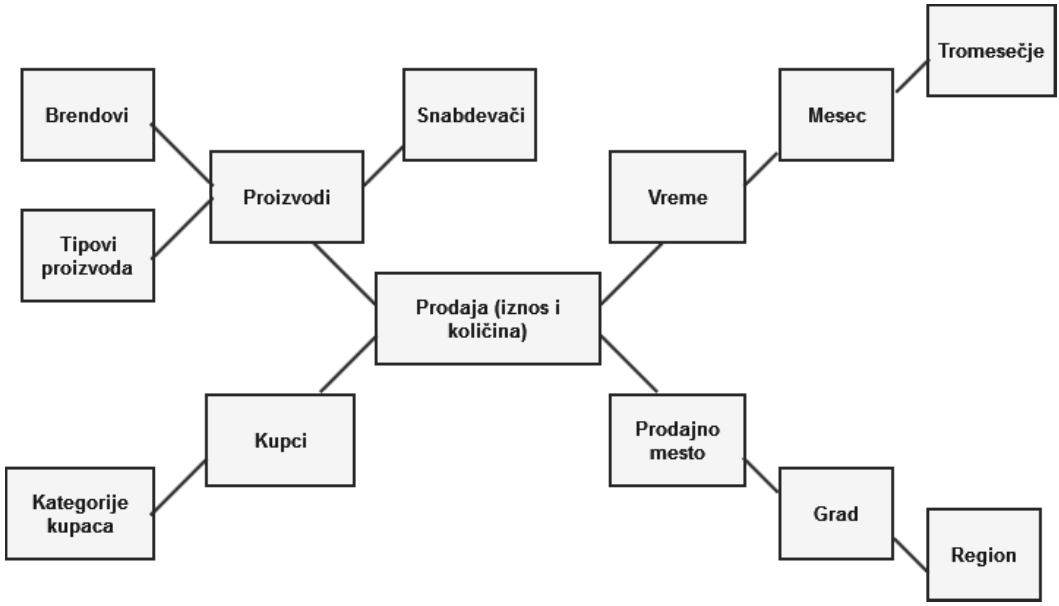


Slika 13.6: Veza tabele činjenica sa tabelama dimenzija

Jednostavan izgled sheme zvezde dobijen je višestrukom agregacijom podataka u nekoliko dimenzija koje su bitne za analizu i donošenje poslovnih odluka. Da bi se ostvario visok stepen grupisanja podataka, neophodno je da zahtevi poslovnih korisnika budu dobro definisani. Samo u tom slučaju mogu da se očekuju kvalitetni izvedeni podaci i validni zaključci, neophodni za donošenje ispravnih poslovnih odluka. Shema zvezde omogućava vrlo brze performanse sistema, jer se upiti izvode na već pripremljenim izvedenim (agregiranim) podacima. S druge strane, proizvođači skladišta nude i alate koji omogućavaju korisnicima da mogu da izvrše i proveru tih izvedenih podataka 'po dubini', do svakog pojedinačnog podatka dobijenog iz konkretnog izvora.

13.2.3 Shema pahuljice

Pored sheme zvezde koriste se i shema pahuljice (*snowflake*). Motiv za ovu shemu je očuvanje normalizacije podataka dimenzija u skladištu. Shema pahuljice ima razgranatu stablastu strukturu. Od centra prema krajevima, entiteti se po kracimaređaju od pojedinačnog ka opštijem. Na sledećoj ilustraciji predstavljena je shema pahuljice razvijena za dati primer analize prodaje (Slika 13.7). Dimenzije više ne predstavljaju agregate, ili izvode podataka, već su dekomponovane (normalizovane). Na primer dimenzija 'Prodajni lanci' je dekomponovana na niz povezanih objekata (tabela) - prodajno mesto, grad i region. Vremenska dimenzija se razmatra kao vreme i datum pojedinačne kupovine, zatim kao mesečni i kvartalni vremenski period. Proizvodi koji su predmet prodaje dekomponovani su prema objektima od interesa - brendovima (robnim markama), snabdevačima i tipovima proizvoda. Takođe kupci se razmatraju po kategorijama, tako da je dimenzija 'Kupci' takođe dekomponovana.



Slika 13.7: Formiranje sheme pahuljice na osnovu datog primera

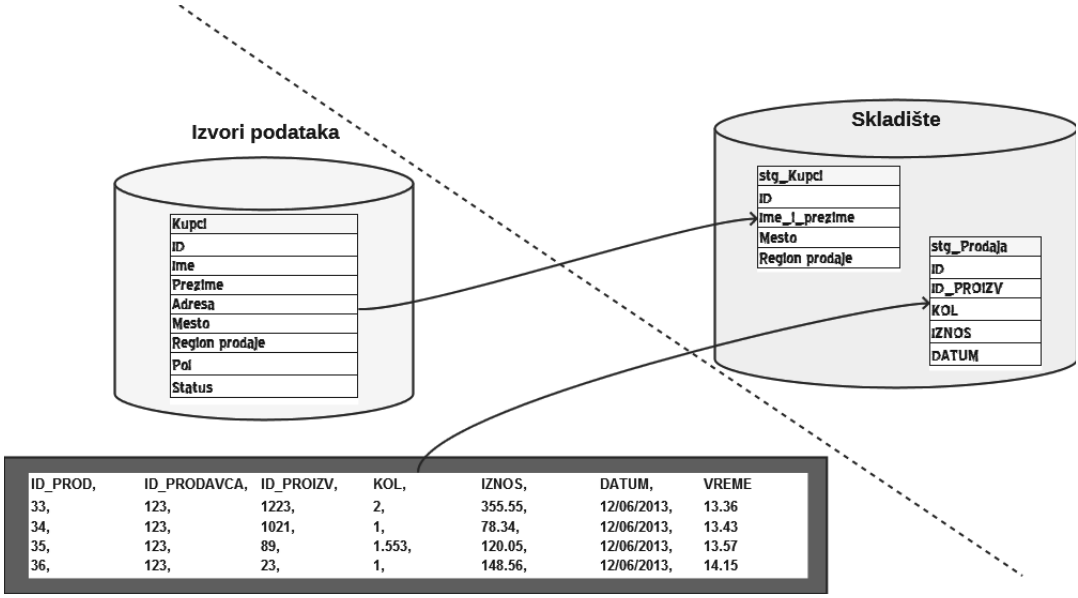
Pošto je shema pahuljice kompleksnija od sheme zvezde, postoji potencijalni problem pada performansi sistema (procesiranja podataka i brzine izvršenja *ad hoc* upita), tako da proizvođači sugerišu korišćenje sheme zvezde kad god je to moguće. Jedan od načina prevazilaženja ovog problema je dekompozicija skladišta na više manjih koji imaju shemu zvezde umesto jednog koje ima shemu pahuljice.

13.3 IZDVAJANJE, TRANSPORT, TRANSFORMACIJA I UČITAVANJE PODATAKA (ETL PROCES)

Prva faza u pripremi podataka za skladištenje je njihovo izdvajanje, transport, transformacija i učitavanje (tzv. *ETL - Extraction, Transport, Transformation and Loading* proces). Ovaj proces je jako bitan jer se moraju prikupiti podaci iz različitih izvora. To su najčešće BP na različitim SUBP, Excel fajlovi, XML fajlovi i obični tekstualni fajlovi. Iako ETL proces može da se izvede programski, u većini aktuelnih programskih jezika, pisanje programskog koda (*hard coding*) 'od nule' (*from scratch*) pokazalo se vrlo neproduktivno. Proizvođači SUBP i skladišta podataka su zato razvili napredne alate koji u velikoj meri automatizuju ETL proces. Primeri takvih alata su *Oracle Data Warehouse Bulder, Microsoft SQL Server Integration Services* i *IBM Information Server*. Pored toga postoje kompanije koje su se specijalizovale za proizvodnju alata za naprednu integraciju podataka iz različitih izvora, što uključuje ETL proces. Primer takvih alata su *SAP Business Objects, SAS Data Integration Studio, Informatica PowerCenter, Altova MapForce* i *DatabaseSpy,* itd.

13.3.1 Izdvajanje (ekstrakcija) podataka

Osnovni princip ETL procesa je da se za svaki izvor podataka pravi posebna tabela u skladištu. Tabele u skladištu ne moraju da obuhvataju sve izvorne podatke, već one koji su bitni za poslovne korisnike. Proces izdvajanja podataka iz izvora i njihova priprema za transport u skladište naziva se ekstrakcija podataka. Na sledećoj ilustraciji predstavljen je primer ekstrakcije na 2 tipa izvornih podataka: tabela *Kupci* koja se nalazi u izvornoj BP i fajl koji sadrži podatke prodaje na nekom od prodajnih mesta zapisane u tekstualnom formatu (Slika 13.8).



Slika 13.8: ETL proces obuhvata podatke iz raznih izvora

Prvi slučaj predstavlja ekstrakciju podataka iz tabele BP u tabelu skladišta podataka. To je najjednostavniji slučaj. Ako je izvorišni SUBP i skladište od istog proizvođača, tipovi podataka su isti, tako da je ETL proces najbrži i najlakše ostvarljiv. Alati za ETL procesiranje omogućavaju čak da se najpre importuju meta - podaci (sheme) za kompletne BP, a zatim da se vrši njihova modifikacija i mapiranje sa shemom tabela u skladištu. Prilikom ekstrakcije podataka iz različitih BP, koje mogu da budu i na različitim lokacijama, udruživanje može da se vrši kroz isti upit. Sledeći primer (Slika 13.9) koji je napisan u *Oracle SQL*Plus* jeziku ilustruje taj slučaj. U skladištu se kreira nova tabela *zap_mor_mesta_prodaje* na osnovu rezultata upita iz 2 različite BP: *bp_lokacije* i *bp_klijenti.* Ove BP ne moraju biti čak niti na istim platformama, već distribuirane bilo gde na mreži. U novoj tabeli biće predstavljena sva mesta prodaje u kojima je izvršena kupovina u okviru *Zapadno-moravskog* regiona.

```
CREATE TABLE zap_mor_mesta_prodaje AS SELECT distinct t1.region_prodaje, t2.mesto
FROM regioni@bp_lokacije t1, prodajno_mesto@bp_klijenti t2
WHERE t1.region_id = t2.region_id
AND t1.region_prodaje='Zapadno-moravski';
```

Slika 13.9: Primer upita za ekstrakciju podataka iz više BP

U slučaju da su SUBP i skladište podataka od različitih proizvođača, zahvaljujući standardima, nema puno razlika u tipovima podataka. I u tom slučaju postoji dobra podrška za konverzije (transformacije) podataka. Transport podataka iz izvorne u ciljnu tabelu se obavlja rutinski: najvažnije je da se izvrši tačno mapiranje izvorišnih u ciljna polja. Prilikom mapiranja nije obavezno da sva polja izvorne tabele budu izdvojena (ekstrahovana) i učitana u odgovarajuću tabelu u skladištu. Nakon specifikacije konekcije prema izvorišnoj BP i ciljnom skladištu, alat za ekstrakciju vrši transport podataka iz BP u skladište, primenjujući transformaciju podataka polja nad kojima je specificirana konverzija. Transferi iz tabela BP u tabelu skladišta podataka su po pravilu najbrži.

Drugi slučaj predstavlja ekstrakciju podataka iz tekstualnog fajla u tabelu skladišta podataka. Ovaj slučaj je nešto složeniji jer uključuje proceduru parsiranja podataka iz izvornog fajla. Parsiranjem se tekstualni podaci transformišu u tabelu (na isti način kao dobro poznata MS Word transformacija - *Convert Text to Table*). Tekst mora da sadrži separatore polja i redova. U gornjem primeru (Slika 13.9) zarez (',') je separator polja, a oznaka za novi red (specijalni ASCII karakter CR, ili CRLF) je separator redova. Treba zapaziti da se u prvom redu nalaze nazivi polja (*ID_PROD, ID_PRODAVCA,* itd.). Nazivi polja ne predstavljaju neophodne podatke, pošto mogu eksplicitno da se definišu neposredno pre kreiranja tabele skladišta u koju će se primiti željeni podaci. Alati za ETL procesiranje omogućavaju podešavanje navedenih opcija.

Pored navedenih formata, alati za ETL procesiranje omogućavaju ekstrakciju podataka iz mnogobrojnih drugih formata kao što su MS Excel radne knjige, XML fajlovi itd. Takođe najčešće su podržani uvozi podataka iz svih aktuelnih SUBP.

13.3.2 Transport podataka

Nakon ekstrakcije, podaci se transportuju iz izvorišnih BP / datoteka u privremenu BP (*staging area*), ili direktno u skladište što zavisi od arhitekture sistema (Slika 13.1 i 2). Transport je zapravo migracija podataka s jedne na drugu platformu. Najčešće se podaci transportuju kao fajlovi posredstvom FTP konekcije između izvora podataka i skladišta. To mogu biti:

- ravni (*flat*) fajlovi u nekom opštem predefinisanom formatu, pri čemu su dodatne informacije o izvornom objektu neophodne.
- fajlovi u nekom specifičnom formatu - što je čest slučaj kada su izvorni SUBP i skladišta od istog proizvođača.
- transport kroz distribuirane operacije. Na primer upiti koji obuhvataju tabele u BP na udaljenim mašinama (Slika 13.9). Za razliku od prethodnih načina koji predstavljaju Offline procesiranje, distribuirane operacije omogućavaju Online dobijanje podataka - rezultata upita, direktno, u jednom koraku.
- specifična, visoko optimizovana rešenja, kao što su *Oracle* transportabilne tabele, koje se koriste i za transport podataka između pripremne BP (*staging area*) i skladišta.

Za razliku od ekstrakcije podataka, koja uključuje heterogene izvore podataka, strategija transporta podataka u ciljno skladište treba da bude jedinstvena u smislu izbora načina transporta. To znači da transportovani podaci treba da imaju jedinstven format. ETL proces je dinamičan i može da se izvršava:

- periodično, najčešće dnevno / nedeljno / mesečno u zavisnosti od dinamike, obima poslova i performansi platforme na kojoj je skladište podataka
- po eksplicitnom zahtevu poslovnih korisnika, ili
- kad nastane potreba (usred velikih promena podataka u sistemu).

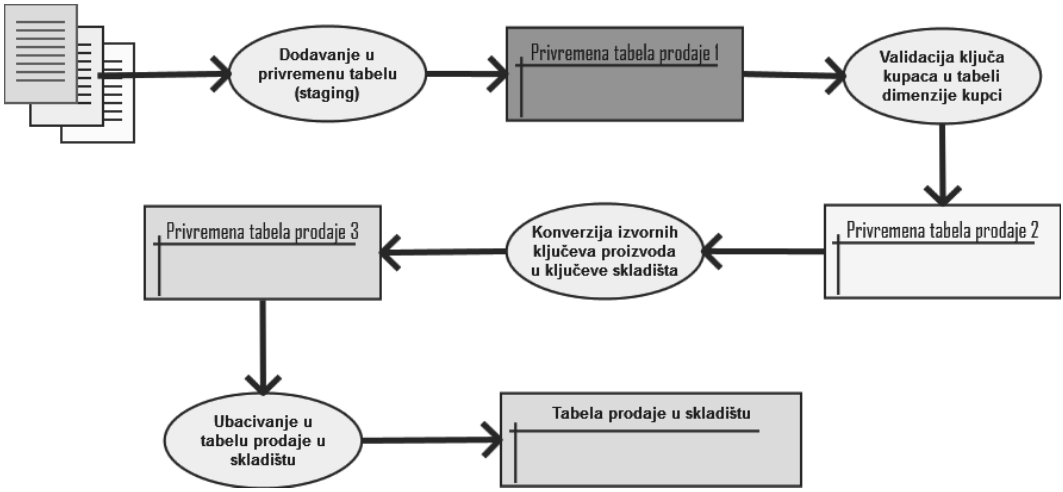
Zajednički format za transportovanje pozitivno utiče na stabilne performanse sistema.

13.3.3 Transformacija i učitavanje podataka

Transformacija podataka predstavlja najzahtevniji deo ETL procesa. Postoje dve tehnike transformacije podataka:

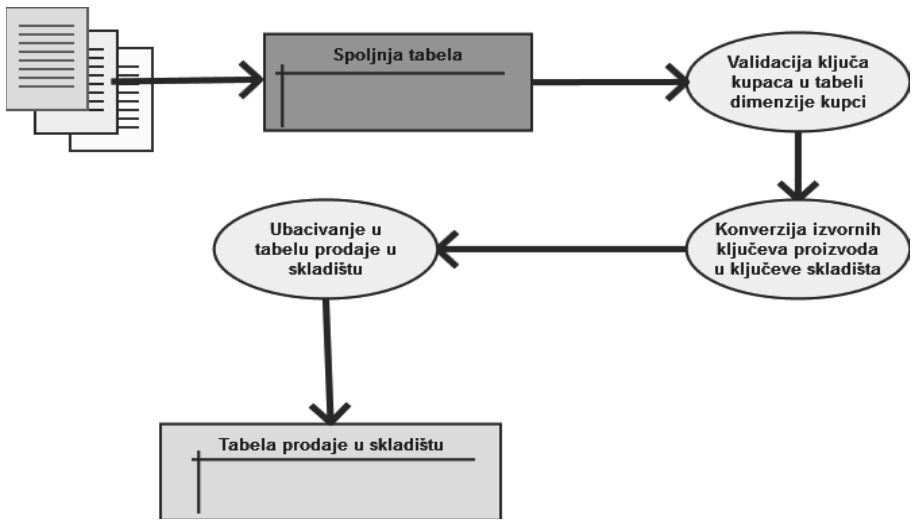
- višestepena tehnika (*multistage data transformation*)
- kanalska tehnika (*pipeline data transformation*)

Višestepena tehnika je najčešće korišćena tehnika, pogodna za arhitekture sa pripremnim zonama, u kojima se vrše najveći deo transformacije. Na sledećem primeru predstavljena je višestepena transformacija podataka u slučaju da se transport podataka iz izvora obavio posredstvom ravnih fajlova (Slika 13.10). Pre nego što se podaci prodaja nađu u tabeli činjenica, spremni za analizu neophodno ih je transformisati u potreban format. Podaci iz ravnih datoteka se dodaju u privremenu tabelu (u konkretnom slučaju to je tabela prodaje) u zoni pripreme (*staging area*). Dalje se vrši inkrementalna obrada tih podataka SQL operacijama, kojima se vrši provera validnosti podataka, zamena ključeva i preslaganje podataka prema potrebi. Obično je broj faza u okviru kojih se odvija transformacija jednak broju SQL operacija, jer se čitav proces postupno odvija i tabela inkrementalno dobija svoje konačno uređenje. Na kraju se tabela iz zone pripreme ubacuje u skladište. Isti je način transformacije u slučajevima tabele činjenica i tabela dimenzija.



Slika 13.10: Višestepena transformacija podataka (primer)

Kanalska tehnika transformacije podataka se primenjuje u osnovnoj arhitekturi skladišta (Slika 13.1), u kojoj ne postoji zona pripreme, već postoje direktne veze skladišta i izvora podatka. Ova tehnika je brža u odnosu na višestepenu tehniku, ali je potrebna velika koherentnost izvora podataka i skladišta. Ne samo da treba da budu od istog proizvođača, već treba voditi računa i o interoperabilnosti verzija ovih sistema. Na sledećoj ilustraciji (Slika 13.11) predstavljen je isti primer, ali se u ovom slučaju vrši transformacija kanalskom tehnikom. Izvorna tabela se transportuje ravnim fajlom (fajlovima ako je veća količina podataka), i transformacija se vrši bez formiranja privremenih tabela, već se direktno dobija transformisana tabela koja se zatim dodaje u skladište.



Slika 13.11: Kanalska tehnika transformacije podataka (primer)

Na nivou instrukcija, transformacija se najčešće realizuje kroz tzv. CTAS (*Create Table As Select*) SQL naredbe. Ove naredbe su kompozitne, pošto sadrže tipičan SQL upit na nivou podataka (*Select DML - Data Manipulation Language statement*) i SQL naredbu za kreiranje tabele na nivou meta - podataka (*Create DDL - Data Definition Language statement*). Sledeći primer prikazuje jednu CTAS naredbu (Slika 13.12), korišćenu u višestepenoj transformaciji za kreiranje tabele *privremena_prodaja_2* od tabela *privremena_prodaja_1* i *proizvod*, tako da se UPC (*Uniform Price Codes*) kod proizvoda zamenjuje identifikatorom proizvoda *proizvod_id* koji se koristi unutar sistema (pogledati slike Slika 13.10 i 13.11 - konverzij*a izvornih ključeva proizvoda u ključeve skladišta*).

```
CREATE TABLE privremena_prodaja_2 NOLOGGING PARALLEL AS SELECT prodaja_transaction_id,
proizvod.proizvod_id prodaja_proizvod_id, prodaja_kupac_id, prodaja_vreme_id,
prodaja_kanal_id, prodaja_kolicina_prodata, prodaja_novcani_iznos
FROM privremena_prodaja_1, proizvod
WHERE privremena_prodaja1.upc_kod = proizvod.upc_kod;
```

Slika 13.12: Transformacija CTAS SQL naredbom

Postoje i složenije transformacije koje mogu da se izvrše nad podacima u cilju njihovog boljeg organizovanja. Na primer, izvori podataka mogu da budu u tabelarnom formatu, ali to ne moraju da budu relacione tabele. U sledećem primeru obrađen je jedan takav slučaj (Slika 13.13). Predstavljeni su podaci vezani za kupovinu određenih proizvoda od strane kupaca (predstavljen je samo jedan od kupaca čiji je ID =123) razmatranu prema danima u nedelji. U ulaznoj tabeli svaki dan ima sopstveno polje za čuvanje iznosa kupovine kupca 123 za određeni proizvod. Cilj je da se izvrši transformacija u relacionu tabelu koja će imati jedno polje za iznose i jedno polje za datume.

ulazna_tabela_prodaja

ID_PROIZV	ID_KUPCA	NEDELJA	IZNOS_PON	IZNOS_UTO	IZNOS_SRE	IZNOS_CET	IZNOS_PET	IZNOS_SUB	IZNOS_NED
1223	123	3/6/13	355.55	80.34	79.34	356.55	81.34	80.34	357.55
1021	123	10/6/13	78.34	122.05	121.05	79.34	123.05	122.05	80.34
89	123	17/6/13	120.05	358.55	357.55				

izlazna_tabela_prodaja

ID_PROIZV	ID_KUPCA	DATUM	IZNOS
1223	123	3/6/13	355.55
1223	123	4/6/13	80.34
1223	123	5/6/13	79.34
1223	123	6/6/13	356.55
1223	123	7/6/13	81.34
1223	123	8/6/13	80.34
1223	123	9/6/13	357.55
1021	123	10/6/13	78.34
1021	123	11/6/13	122.05
1021	123	12/6/13	121.05
1021	123	13/6/13	79.34
1021	123	14/6/13	123.05
1021	123	15/6/13	122.05
1021	123	16/6/13	80.34

Slika 13.13: Transformacija podataka

Predstavljena transformacija se naziva obrtanje (*pivoting*). Na nivou instrukcija, *pivoting* transformacija se realizuje SQL naredbom za dodavanje zapisa (Slika 13.14).

```
INSERT ALL
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA, ulazna_tabela_prodaja.IZNOS_PON)
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA+1, ulazna_tabela_prodaja.IZNOS_UTO)
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA+2, ulazna_tabela_prodaja.IZNOS_SRE)
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA+3, ulazna_tabela_prodaja.IZNOS_CET)
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA+4, ulazna_tabela_prodaja.IZNOS_PET)
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA+5, ulazna_tabela_prodaja.IZNOS_SUB)
INTO izlazna_tabela_prodaja (ID_PROIZV, ID_KUPCA, DATUM, IZNOS)
VALUES (ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA+6, ulazna_tabela_prodaja.IZNOS_NED)
SELECT ulazna_tabela_prodaja.ID_PROIZV, ulazna_tabela_prodaja.ID_KUPCA, ulazna_tabela_prodaja.NEDELJA,
   ulazna_tabela_prodaja.IZNOS_PON,ulazna_tabela_prodaja.IZNOS_UTO, ulazna_tabela_prodaja.IZNOS_SRE,
   ulazna_tabela_prodaja.IZNOS_CET, ulazna_tabela_prodaja.IZNOS_PET, ulazna_tabela_prodaja.IZNOS_SUB,
   ulazna_tabela_prodaja.IZNOS_NED
FROM ulazna_tabela_prodaja;
```

Slika 13.14: SQL naredba kojom se omogućava pivotiranje ulazne u izlaznu tabelu

Cilj transformacije, bez obzira na način na koji se izvodi, je punjenje skladišta potrebnim podacima. Za učitavanje se koriste posebni moduli: SQL punioci (*SQL loader)*. Ovi moduli vrše pri učitavanju potrebne transformacije podataka: od konverzija tipova, rukovanja NULL vrednostima do sortiranja, filtriranja, dobijanja izvedenih podataka i redizajna podataka u tabeli. Brzina ETL procesa u velikoj meri zavisi od kvaliteta SQL punilaca.

13.4 PROMENA (AŽURIRANJE) PODATAKA U SKLADIŠTU

ETL procesiranje priprema podatke radi dalje obrade u smislu agregacije i dobijanja sumarnih podataka u skladu sa zahtevima poslovnih korisnika. Korisnici mogu dalje da kreiraju različite oblike izveštaja, da vrše napredna pretraživanja i analizu tih podatka u realnom vremenu. Bez obzira na izvor, podaci se u skladištima nalaze u tabelama, koje sadrže primarne i spoljnje ključeve i koje su indeksirane. Identifikatori zapisa (primerni ključevi) u tabelama skladišta često nisu isti kao oni korišćeni u izvornim tabelama. Obzirom da se podaci u izvornim tabelama neprekidno menjaju, a da ta promena ne može da se odrazi automatski u tabelama skladišta (ne mogu automatski da se primene mehanizmi održavanja referencijalnog integriteta), u skladištima je potrebna drukčija strategija i tehnika ažuriranja podataka.

Najčešće se podaci u skladištu ažuriraju u zadatim vremenskim intervalima: dnevno (npr. svake noći), nedeljno, ili mesečno. Obzirom da tabele u skladištima sadrže istorijske podatke, koji se najčešće više ne menjaju, i obzirom da imaju veliki broj zapisa (naročito tabele činjenica), strategija ažuriranja se svodi na dodavanje samo novih a ne svih zapisa. Ovo je slučaj, koji važi za situaciju da u međuvremenu nije došlo do promene sheme bilo izvornih BP, bilo sheme skladišta. Radi optimizacije manipulisanja podacima tabela skladišta, proizvođači su omogućili da se vrši

kontrolisano deljenje tabela (*partitioning*) po zadatom kriterijumu. Pošto je ažuriranje periodično, vremenski kriterijum je najbolji za deljenje tabela. Na taj način se u memoriju učitavaju samo podaci dela (particije) tabele za vremenski period od interesa. Sladeći primer ilustruje ovaj tipičan postupak ažuriranja.

13.4.1 Primer ažuriranja tabele skladišta

U skladištu postoji tabela *prodaja* koja predstavlja tabelu činjenica. Ona se ažurira periodično - na mesečnoj bazi. Isto tako, radi optimizacije, tabela je podeljena na mesečne particije. To znači da se svakog prvog u mesecu dodaju zapisi iz izvornih tabela za prethodni mesec. Zadatak je da se dodaju u tabelu *prodaja* zapisi za mesec jun 2013-te. Ažuriranje se odvija u 3 faze:

1. najpre se kreira nova, privremena tabela od zapisa za mesec jun *prodaja_06_2013*;
2. zatim se na privremenoj tabeli *prodaja_06_2013* primeni indeksiranje i ograničenja koja važe za tabelu skladišta *prodaja*;
3. na kraju se podaci privremene tabele *prodaja_06_2013* dodaju tabeli *prodaja* u 2 koraka:
   - u tabeli *prodaja* kreira se nova particija za mesec jun,

     ```
     ALTER TABLE prodaja ADD PARTITION prodaja_06_2013
     ```

   - u ovu particiju se zatim dodaje čitava tabela *prodaja_06_2013*, čime ona prestaje da postoji kao posebna tabela, već postaje deo - particija tabele *prodaja*

```
ALTER TABLE prodaja EXCHANGE PARTITION prodaja_06_2013 WITH TABLE prodaja_06_2013
```

Ovaj postupak omogućava uštedu u korišćenju resursa, jer se indeksiranje i provera referencijalnog integriteta (ograničenja) nije izvršilo na čitavoj tabeli *prodaja*, već samo na privremenoj tabeli *prodaja_06_2013*. Pored dodavanja, ažuriranje omogućava uklanjanje i arhiviranje podataka. Na primer, ako su za analizu i odlučivanje važni podaci ne stariji od tri godine, to znači da bi za prethodni primer, tabela *prodaja* sadržala 36 particija; dodavanje particije za novi mesec će automatski izvršiti arhiviranje, ili kompletno uklanjanje particije najstarijeg meseca iz tabele.

13.5 AGREGIRANJE PODATAKA U SKLADIŠTU

Agregacija podataka predstavlja osnovni koncept zbog kojeg se skladišta koriste. Za tu namenu upotrebljeni su SQL upiti u kojima je *GROUP BY* klauzula proširena na različite načine:

- *CUBE* i *ROLLUP* ekstenzijama,
- funkcijama za grupisanje i
- skupovima za grupisanje.

Ove tehnike omogućavaju jednostavno i brzo izvršenje upita i formiranje izveštaja u cilju višedimenzionalne (višekriterijumske) analize podataka, koja predstavlja osnov sistemima za podršku u odlučivanju.

13.5.1 ROLLUP ekstenzija

ROLLUP ekstenzija je namenjena za dobijanje agregacija na različitim nivoima u vidu suma (SUM), količine/brojeva (COUNT), maksimalnih (MAX), minimalnih (MIN) i srednjih (AVG) vrednosti za zadate grupe dimenzija. Predstavlja jednostavnu ekstenziju *GROUP BY* klauzule. Struktura SQL upita sa *ROLLUP* ekstenzijom izgleda:

```
SELECT ... FROM ... WHERE .... GROUP BY ROLLUP (LISTA DIMENZIJA PO KOJIMA SE VRŠI GRUPISANJE)
```

Grupisanje se vrši listom dimenzija, u kojoj je bitan redosled. Na primer ako su u listi pobrojani redom lanac prodaje, vreme i iznos, to znači da će rezultujući zapisi isto tako biti grupisani. Sledeći primer to demonstrira (Slika 13.15).

```
SELECT prodajni_lanci.NAZ, vreme.kalendar_mesec_naziv,
drzave.iso_kod_drzave,
TO_CHAR(SUM(iznos_prodaje), '9,999,999,999') prodaje$
FROM prodaje, kupci, vreme, prodajni_lanci, drzave
WHERE prodaje.vreme_id = vreme.id
AND prodaje.kupac_id = kupci.id
AND kupci.id_drzave = drzave.ID
AND prodaje.id_lanca = prodajni_lanci.ID
AND prodajni_lanci.NAZ IN ('Direktna prodaja', 'Internet')
AND vreme.kalendar_mesec_naziv IN ('maj 2013', 'jun 2013')
AND drzave.iso_kod_drzave IN ('RS', 'MK')
GROUP BY
ROLLUP(prodajni_lanci.NAZ, vreme.kalendar_mesec_naziv, drzave.iso_kod_drzave);
```

Slika 13.15: Primer SQL agregirajućeg upita sa ROLLUP ekstenzijom

U primeru svi podaci koji se selektuju su zapravo grupisani pomoću *ROLLUP* ekstenzije. U *WHERE* klauzuli su dati kriterijumi za povezivanje podataka iz tabela u skladištu (prema identifikatorima zapisa) i filteri za *ROLLUP* grupisanje - obuhvaćeni su prodajni lanci zvani *Internet* i *Direktna prodaja*, meseci *maj* i *jun* u državama Srbiji (*RS*) i Makedoniji (*MK*). Rezultat ovakvog upita izgledaće kao na sledećoj ilustraciji (Slika 13.6).

prodajni_lanci.NAZ	vreme.kalendar_mesec_naziv	drzave.iso_kod_drzave	prodaje
Direktna prodaja	maj 2013	MK	235,342.00 €
Direktna prodaja	maj 2013	RS	1,023,553.00 €
Direktna prodaja	maj 2013		1,258,895.00 €
Direktna prodaja	jun 2013	MK	183,423.00 €
Direktna prodaja	jun 2013	RS	782,234.00 €
Direktna prodaja	jun 2013		965,657.00 €
Direktna prodaja			2,224,552.00 €
Internet	maj 2013	MK	156,783.00 €
Internet	maj 2013	RS	524,234.00 €
Internet	maj 2013		681,017.00 €
Internet	jun 2013	MK	126,445.00 €
Internet	jun 2013	RS	835,454.00 €
Internet	jun 2013		961,899.00 €
Internet			1,642,916.00 €
			3,867,468.00 €

Slika 13.16: Rezultat prethodnog SQL upita sa ROLLUP ekstenzijom

Podaci u rezultujućoj tabeli grupisani su s leva u desno. U poslednjoj koloni (*prodaje*) predstavljeni su agregirani podaci - novčani iznosi mesečne prodaje za različite kombinacije grupisanja. Vrednosti koje nisu uokvirene predstavljaju iznose po zemljama prodaje na mesečnom nivou u okviru različitih prodajnih lanaca. Crveno uokvireni su iznosi agregirani na nivou države, zeleno uokvireni su iznosi agregirani na mesečnom nivou i plavo uokviren je iznos prodaje agregiran na nivou prodajnih lanaca. Ovako dobijeni podaci dalje mogu da se interpretiraju na različite načine analitičkim alatima (filtrirani, vizualizovani u vidu dijagrama, itd.).

13.5.2 CUBE ekstenzija

CUBE ekstenzija je veoma slična *ROLLUP* ekstenziji, s tim da omogućava u jednoj naredbi kalkulaciju svih mogućih kombinacija agregacija. Ova prednost ima cenu u intenzivnom korišćenju resursa platforme radi procesiranja podataka. *CUBE* ekstenziju zato treba izbegavati kad god je to moguće. Međutim u nekim situacijama predstavlja najoptimalniji izbor - dobijanje izveštaja iz unakrsnih upita nad više tabela. To znači da je treba koristiti kada su u upitima intenzivno kombinovane kolone (polja) više dimenzija, a izbegavati kada kolone predstavljaju različite nivoe iste dimenzije. Struktura SQL upita sa *CUBE* ekstenzijom izgleda:

```
SELECT ... FROM ... WHERE .... GROUP BY CUBE (LISTA DIMENZIJA PO KOJIMA SE VRŠI GRUPISANJE)
```

Kao kod *ROLLUP* upita, *CUBE* ekstenzijom grupisanje se vrši listom dimenzija, u kojoj je bitan redosled. Na primer ako su u listi pobrojani redom lanac prodaje, vreme i iznos, to znači da će rezultujući zapisi isto tako biti grupisani. Sledeći primer to demonstrira (Slika 13.17).

```
SELECT prodajni_lanci.NAZ, vreme.kalendar_mesec_naziv, drzave.iso_kod_drzave,
TO_CHAR(SUM(iznos_prodaje), '9,999,999,999') prodaje$
FROM prodaje, kupci, vreme, prodajni_lanci, drzave
WHERE prodaje.vreme_id = vreme.id AND prodaje.kupac_id = kupci.id
AND kupci.id_drzave = drzave.ID AND prodaje.id_lanca = prodajni_lanci.ID
AND prodajni_lanci.NAZ IN ('Direktna prodaja', 'Internet')
AND vreme.kalendar_mesec_naziv IN ('maj 2013', 'jun 2013')
AND drzave.iso_kod_drzave IN ('RS', 'MK')
GROUP BY CUBE (prodajni_lanci.NAZ, vreme.kalendar_mesec_naziv, drzave.iso_kod_drzave);
```

Slika 13.17. Primer SQL agregirajućeg upita sa CUBE ekstenzijom

Iako se gornji *CUBE* upit razlikuje od *ROLLUP* upita (Slika 13.15) samo po nazivu ekstenzije, razlika između rezultata je mnogo očiglednija (Slika 13.18). Rezultat *ROLLUP* upita imao je 15 redova, dok rezultat *CUBE* upita sadrži 27 redova. Podaci su agregirani u svim mogućim kombinacijama. Prvi red predstavlja agregat svega - ukupan iznos prodaje. Sledeća dva su agregati - iznosi prodaja po državama. Četvrti red predstavlja mesečni agregat - ukupan iznos prodaje u mesecu maju. Sledeća dva reda su agregati - iznosi prodaja po državama u mesecu maju. Od sedmog do devetog reda su iznosi prodaja za jun - najpre ukupnim iznosom prodaje za jun, a zatim iznosi prodaje za taj mesec po državama. Zatim slede agregati - po prodajnim lancima. Na primer red u kome se pojavljuje u prvoj koloni prvi put vrednost *Direktna prodaja* predstavlja ukupan iznos ostvaren direktnom prodajom za sve mesece i države. Zatim slede agregati - iznosi direktne prodaje po državama, zatim po mesecima, koje se dalje raščlanjuju po mesecima i državama. Na isti način formirani su i rezultati prodaje preko interneta.

prodajni_lanci.NAZ	vreme.kalendar_mesec_naziv	drzave.iso_kod_drzave	prodaje
			3,867,468.00 €
		MK	701,993.00 €
		RS	3,165,475.00 €
	maj 2013		1,939,912.00 €
	maj 2013	MK	392,125.00 €
	maj 2013	RS	1,547,787.00 €
	jun 2013		1,927,556.00 €
	jun 2013	MK	309,868.00 €
	jun 2013	RS	1,617,688.00 €
Direktna prodaja			2,224,552.00 €
Direktna prodaja		MK	418,765.00 €
Direktna prodaja		RS	1,805,787.00 €
Direktna prodaja	maj 2013		1,258,895.00 €
Direktna prodaja	maj 2013	MK	235,342.00 €
Direktna prodaja	maj 2013	RS	1,023,553.00 €
Direktna prodaja	jun 2013		965,657.00 €
Direktna prodaja	jun 2013	MK	183,423.00 €
Direktna prodaja	jun 2013	RS	782,234.00 €
Internet			1,642,916.00 €
Internet		MK	283,228.00 €
Internet		RS	1,359,688.00 €
Internet	maj 2013		681,017.00 €
Internet	maj 2013	MK	156,783.00 €
Internet	maj 2013	RS	524,234.00 €
Internet	jun 2013		961,899.00 €
Internet	jun 2013	MK	126,445.00 €
Internet	jun 2013	RS	835,454.00 €

Slika 13.18: Rezultat prethodnog SQL upita sa CUBE ekstenzijom

Za veliki broj dimenzija, korišćenje *CUBE* ekstenzije može da bude problematično i po pitanju veličine i preglednosti rezultata. Jedno od rešenja problema je delimično korišćenje *CUBE* ekstenzije tako što se iza *GROUP BY* klauzule doda naziv polja / kolone pre *CUBE* ekstenzije. Sledeća modifikacija predstavlja parcijalno korišćenje *CUBE* ekstenzije:

```
SELECT  ... FROM ... WHERE .... GROUP BY prodajni_lanci.NAZ, CUBE (vreme.kalendar_mesec_naziv, drzave.iso_kod_drzave);
```

Uočljivo je da u odnosu na prethodni SQL upit sa *CUBE* ekstenzijom (Slika 13.17), parcijalna *CUBE* ekstenzija ne obuhvata sve dimenzije za grupisanje. U konkretnom slučaju izbačena je iz liste dimenzija *prodajni lanci* koja je zapravo samo premeštena kao kriterijum za grupisanje odmah iza *GROUP BY* klauzule. Kao rezultat, dobiće se

tabela koja, umesto 27 sadrži 18 polja. U odnosu na prethodni rezultat (Slika 13.18), neće postojati redovi koji nemaju vrednost za polje *prodajni_lanci.NAZ*.

13.5.3 Funkcije i skupovi za grupisanje

Funkcije za grupisanje omogućavaju programsku obradu rezultata. Omogućavaju eliminisanje NULL vrednosti, sortiranje agregiranih vrednosti i filtriranje rezultata. Za ovu svrhu koristi se klauzula *GROUPING* koja omogućava da se grupišu rezultati već agregirani nekom od tehnika (*GROUP BY*, *ROLLUP* i *CUBE* ekstenzijama) i prikažu samo one vrednosti koje su bitne za analizu u datom trenutku. Na primer, uzevši rezultate sa prethodnih ilustracija (Slike 13.16. do 13.18) funkcijama za grupisanje mogli bi se izdvojiti podaci samo na godišnjem nivou po dva preostala kriterijuma (dimenzije) država i prodajni lanac (Slika 13.19).

prodajni_lanci.NAZ	drzave.iso_kod_drzave	prodaje
	MK	701,993.00 €
	RS	3,165,475.00 €
Direktna prodaja		2,224,552.00 €
Internet		1,642,916.00 €
		3,867,468.00 €

Slika 13.19: Rezultat primene funkcija za grupisanje
da bi se dobili rezultati na godišnjem nivou

Primenom funkcija za grupisanje umesto velikog broja zapisa i nepreglednih rezultata, dobijena je rezultujuća tabela koja sadrži samo manji deo agregiranih podataka. Rezultati su mnogo pregledniji i jasniji za tumačenje od strane menadžmenta.

Skupovi za grupisanje omogućavaju preciznu specifikaciju grupisanja podataka u slučaju korišćenja više dimenzija i bez korišćenja zahtevnog procesiranja *CUBE* ekstenzijom u SQL upitu. To znači da je jednim upitom moguće izvršiti istovremeno više grupisanja podataka na različite načine. Sintaksa naredbe za definisanje skupova za grupisanje data je sledećim izrazom:

```
SELECT ... FROM ... WHERE .... GROUP BY GROUPING SETS ((GRUPA DIMENZIJA 1), (GRUPA DIMENZIJA 2), ... , (GRUPA DIMENZIJA N))
```

Klauzula GROUPING *SETS* je karakteristična za formiranje skupova za grupisanja i predstavlja kraći zapis unije rezultata višestrukog grupisanja *GROUP BY* klauzulama. Skupovi za grupisanje omogućavaju da podaci u skladištu budu predstavljeni na različite načine.

13.6 ANALITIČKE OPERACIJE

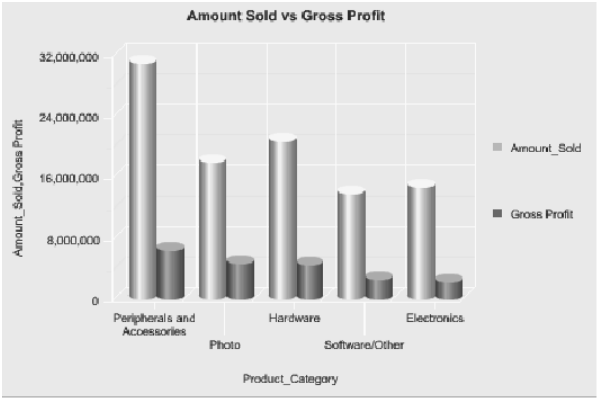
Izvedeni i agregirani podaci u skladištu podataka dalje se koriste za analitičko procesiranje (*OLAP - Online Analitical Processing*) i za napredno pretraživanje podataka (*DM - Data Mining*). Ova dva procesa se suštinski razlikuju u pristupu podacima. *OLAP* se izvodi na ukrštenim podacima na višim nivoima agregacije, dok je *DM* namenjen traganju za skrivenim pravilima / obrascima u konkretnim podacima. Različitost *OLAP* i *DM* ih čini (uzajamno dopunjivim) komplementarnim.

13.6.1 OLAP funkcije

U prethodnoj sekciji predstavljene su osnovne tehnike kojima se omogućuje dobijanje izvedenih podataka u cilju njihove analize. *OLAP* koristi višedimenzionalni model podataka nad kojim obavlja statističke, matematičke i finansijske analize nad istorijskim podacima. Proizvođači skladišta nude dodatne funkcije kojima se obezbeđuje proširenje SQL upita radi obrade agregiranih i izvedenih podataka u skladištu. Ove funkcije mogu da se grupišu u kategorije:

- *agregacione* funkcije - namenjene za pronalaženje srednjih vrednosti, suma, najmanjih i najvećih vrednosti, rangiranje zapisa i brojanje zapisa po zadatim kriterijumima
- hijerarhijske funkcije - za analizu hijerarhijskih struktura - odnosi između rezultata na bazi naslednici / prethodnici, roditelji / deca, određivanje nivoa hijerarhije, dubine, hijerarhijskog poretka i sl.,
- *lag* funkcije - namenjene za otkrivanje zakonitosti promene i razlika vrednosti podataka najčešće između različitih vremenskih perioda
- funkcije za *rangiranje* - namenjene za rangiranje podataka u odnosu na zadate kriterijume; rang može da bude prosečan po periodima, može da ima frekvenciju pojavljivanja podataka u određenom rangu i sl.,
- funkcije *raspodele* - namenjene za dobijanje raspodele / učešća podataka u odnosu na njihovu kumulativnu masu, ili karakteristične vrednosti ako se radi o hijerarhijskoj strukturi podataka (roditelj, prvi / koreni predak)
- *prozorske* funkcije - predstavljaju podskup *agregacionih* funkcija koje se izvršavaju na specifičan način: formira se 'prozor' koji obuhvata podskup vrednosti koje se analiziraju; na taj način moguće je izvršiti analizu kretanja računanjem suma / aritmetičkih sredina / najmanjih i najvećih vrednosti u prozoru.

Rezultati OLAP analiza se najčešće predstavljaju u vidu različitih dijagrama i tabela agregiranih podataka, kako bi trendovi promena, karakteristične vrednosti bile lako uočljive. Na sledećoj ilustraciji su prikazani neki od tipova dijagrama koji se koriste.





	Americas	Asia	Europe	Oceania
Product_Category	Gross Profit	Gross Profit	Gross Profit	Gross Profit
Electronics	$1,267,454	$212,503	$647,881	$92,845
Hardware	$2,373,452	$513,427	$1,370,214	$190,690
Peripherals and Accessories	$3,688,919	$679,203	$1,860,473	$254,658
Photo	$2,497,317	$519,079	$1,380,126	$201,451
Software/Other	$1,587,793	$228,127	$716,466	$92,597

Slika 13.20: Različiti tipovi dijagrama za prikaz rezultata OLAP analize

Na levoj strani je primer stubičastog dijagrama koji prikazuje raspodelu prodaje i ostvarene profite po proizvodima. Ovaj tip dijagrama je pogodan za analize kojima se bavi menadžment. Gore desno su takozvani 'merači' dijagrami (*gauges*), koji predstavljaju odlično sredstvo za svakodnevni uvid u trenutnu situaciju u kompaniji po različitim kriterijumima. Oni imaju kazaljke koje mogu da se nađu u crvenoj, žutoj, ili zelenoj zoni. Na taj način korisnik može brzo da odredi prioritete u radu: najpre (urgentno) rešava stvari (probleme) na koje ukazuju kazaljke u crvenoj zoni. Redovni zadaci su oni na koje ukazuju kazaljke u žutoj zoni, a aktivnosti na koje ukazuju kazaljke u zelenoj zoni mogu i da se produže / odlože u slučaju potrebe. Dole desno je tabela agregiranih podataka, najčešće na najvišem nivou agregacije i sa hiperlinkovima prema pod - tabelama koje učestvuju u agregatu. Podaci su grupisani tako da omogućavaju korisnicima (menadžmentu) jasno i precizno upoređivanje (komparaciju) podataka iz različitih grupa. Pored predstavljenih postoje i drugi tipovi dijagrama: linijski dijagrami najviše odgovaraju u situacijama kada treba da se istaknu trendovi promene podataka; pita dijagrami su pogodni za prikaz raspodela i učešća pojedinih grupa podataka u ukupnoj masi; tačkasti dijagrami omogućavaju sagledavanje grupisanja podataka oko nekih mera centralnih tendencija; površinski dijagrami su namenjeni za prikaz vrednosti u 3 dimenzije prikaza.

13.6.2 Napredno pretraživanje podataka

Pored *OLAP* analize podaci u skladištu se analiziraju i naprednim pretraživanjem podataka (u daljem tekstu *DM - Data Mining*). Kao što je već napomenuto u ovom poglavlju, objekti *DM-a* su konkretni podaci u skladištu, dobijeni od izvora, a ne kumulativni agregati. *DM* omogućuje da se iz mnoštva konkretnih podataka izvuku zaključci koji na prvi pogled nisu uočljivi. Na primer, podaci kao što su faktori koji su uticali na loše poslovanje, otkrivanje potencijalnih klijenata koji žele da promene

davaoca usluga (kompaniju), otkrivanje varljivog ponašanja i sl. Funkcije koje omogućavaju napredno pretraživanje su:

- klasifikacija - podaci se sortiraju prema najčešće unapred utvrđenim kategorijama (klasama),
- regresija - aproksimacija i predviđanje kontinualnih numeričkih vrednosti,
- klasterovanje - prirodno grupisanje podataka oko mera centralne tendencije (centroidi, euklidske distance), pri čemu novi podaci unose promene u klasterima,
- udruživanje (asocijacija) - otkrivanje 'vezanih' proizvoda / usluga / pojava, onih koji se prodaju / pojavljuju koincidentno,
- otkrivanje važnosti atributa - identifikacija podataka (polja) koja su najmerodavnija za predikciju rezultata i
- izdvajanje svojstava - svojstva su izvedeni podaci, najčešće kombinacija atributa.

Praktična implementacija *DM-a* se svodi na pretragu jedne, ili više kolona tekstualnih podataka u tabelama skladišta. Predstavljanje rezultata *DM*-a je isto kao i u slučaju *OLAP* analize - tabelarno i grafički. Pored toga koriste se stablasti dijagrami u slučaju klasifikacija i asocijativni grafovi u slučaju funkcija udruživanja.

14 NoSQL BP

14.1 UVOD

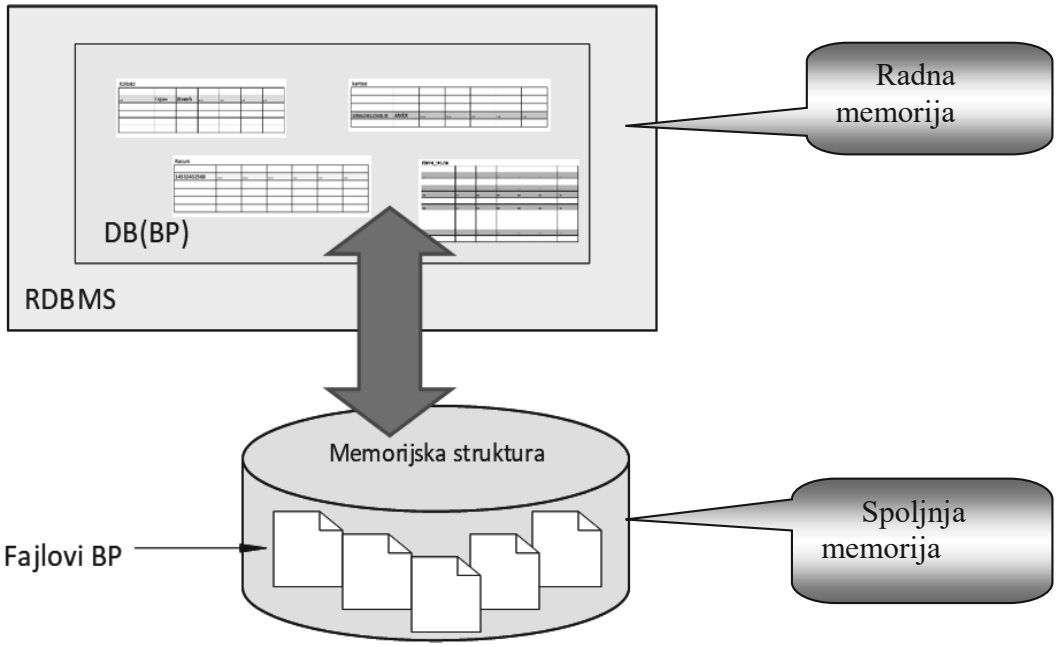
NoSQL DBMS (*Not-only-SQL Database Management Systems*) su skalabilni sistemi otvorenog koda koji skladište i omogućavaju manipulaciju heterogenim sadržajima koji ne mogu da se normalizuju i koji su distribuirani na Web-u. NoSQL DBMS omogućavaju da se u istoj BP čuvaju podaci od nekoliko bajtova do nekliko desetina megabajta. Dalje, NoSQL sistemi omogućavaju da se podaci distriburaju na Webu, da imaju kopije hostovane na različitim lokacijama, a da i dalje budu dostupni za manipulaciju: pretrage i izmene podataka. NoSQL DBMS su fleksibilni sistemi koji omogućavaju obradu veoma heterogenih podataka. U NoSQL DBMS da bi se podaci skladištili, ne mora da postoji šema BP (*schema free systems*). Na kraju NoSQL DBMS su sistemi koji imaju vrlo jednostavan API (obzirom da nema složenih referentnih modela), koji omogućava brze replikacije BP i rad sa velikom količinom podataka.

14.2 MOTIVI NASTANKA NOSQL DBMS

Kod sistema za upravljanje relacionim bazama podataka (u daljem tekstu RDBMS – *Relational Database Management System*), kod kojih dolazi tokom eksploatacije do nagomilavanja podataka koji imaju slabu struktuiranost (nizak stepen normalizacije), beleži se pad performansi manifestovan kroz sporije vreme odziva sistema. Najčešći scenario u kome se pojavljuje ovaj simptom je da RDBMS čuva veliku količinu neindeksiranih podataka u potpuno nedistribuiranom sistemu (1 BP na 1 hostu). Primer takvih RDBMS su arhivski sistemi koji sadrže dokumenta u raznim formatima (MS Office, OpenOffice, PDF, ili neki drugi XML format) pa čak i skenirana dokumenta koja se čuvaju u nekom multimedijalnom formatu i koja su opisana metapodacima. U takvim sistemima tabele sadrže i polja u kojima se skladište već pomenuta dokumenta, ili drugi multimedijalni sadržaji kao bajtovski nizovi, koji mogu veoma da variraju u veličini, a mogu i da predstavljaju strukturu za sebe. Na primer BP biblioteke koja čuva elektronska izdanja knjiga od nekoliko desetina stranica, ali i knjiga od preko 1000 stranica; još slikovitiji primer je da bi ta ista BP sadrži knjige koje sadrže isključivo tekstualni sadržaj, ali i knjige koje sadrže veliki broj ilustracija i drugih multimedijalnih sadržaja.

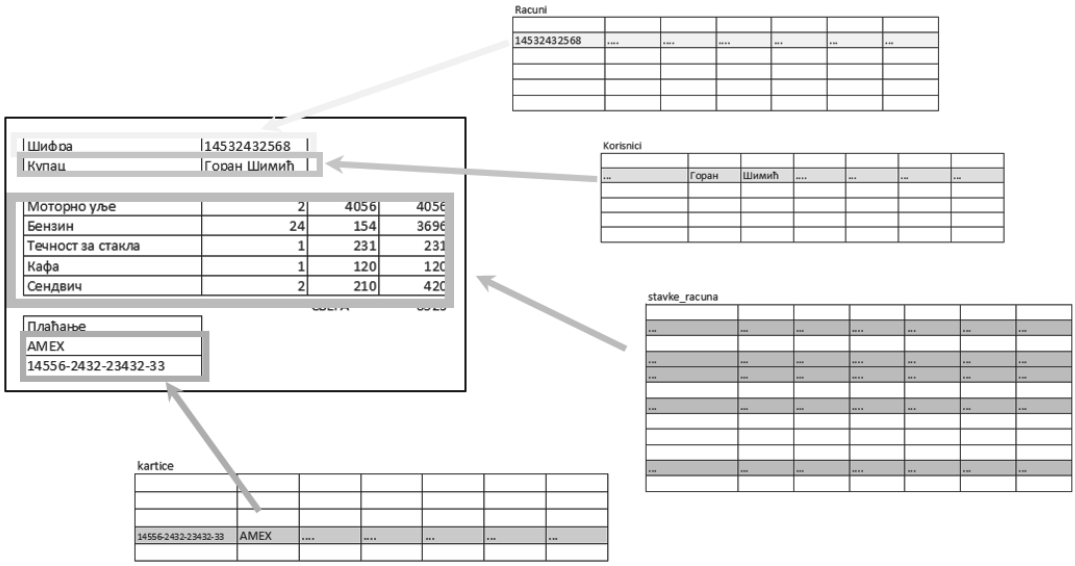
Većina savremenih RDBMS nema adekvatnu podršku za ovakve slučajeve. Uzrok problema je zavisnost od dostupnih resursana hostu: u ograničenjima spoljnje i radne memorije kao i u procesnoj snazi hosta. Fizički, podaci se čuvaju u fajlovima na spoljnjoj memoriji, koja predstavlja jednodimenzioni adresni prostor sa sekvencijalnim pristupom (npr. pristup magnetne glave na mesto u sektoru na kome započinje prvi bajt podatka od interesa). Rezultat toga je da nedistribuirana BP fizički predstavlja jednu, ili nekoliko datoteka na takvom medijumu, vreme pretrage sadržaja fajla (*seeking* time) se

uvećava proporcionalno uvećanjem podataka koji fajl BP sadrži. Pored toga, da bi se podaci obradili, potrebno je da se konvertuju najpre u model podataka koji podržava RDBMS: na primer, u tabelama podaci imaju 2D referenciranje (polje i zapis), a mogu biti proizvoljno i višestruko referencirani iz drugih tabela u BP. To znači da jednodimenzioni model na fizičom medijumu mora da se konvertuje (mapira) u višedimenzionalnu složenu strukturu šeme BP u radnoj memoriji dodeljenoj RDBMS-u (Slika 14.1).



Slika 14.1: Konverzija iz fizičkog modela u model podataka relacione BP

Dakle, jedan deo problema je u vezi različitog načina na koji OS organizuje podatke na spoljnjim nosiocima u odnosu na način na koji to čini RDBMS. Dizajneri aplikacija izloženi su još jednom problemu – ako model podataka u RDBMS nije identičan modelu podataka koji se koristi u aplikacijama, u tom slučaju postoji potreba za još jednom konverzijom formata podataka u samoj aplikaciji. Ovaj problem je poznat pod nazivom *impendance msimatch*. Sledeći primer (Slika 14.2) prikazuje jedan takav slučaj: da bi aplikacija generisala račun (model podatka koji koristi aplikacija), potrebno je da se prikupe i agregiraju podaci iz različitih tabela (na primeru to su tabele *kartice, korisnici, stavke racuna, racuni*). Drugim rečima šema BP ne podržava poslovni model podataka u aplikaciji.



Slika 14.2: Generisanje računa - konverzija iz modela podataka RDBMS BP u strukturu podataka u aplikaciji

Operativna (radna) memorija, uprkos neuporedivo većoj brzini pristupa, unosi drugi vid ograničenja: memorijski prostor postaje kritičan resurs pri obradi podatka koji su memorijski zahtevni. Na primer, pad performasi dolazi do izražaja u SQL upitima sa kriterijumom (SELECT naredbe sa WHERE klauzulom). Bez obzira koliko je tip polja uključenog u kriterijum upita jednostavan, da bi se izvršila potpuna pretraga neke tabele u BP, potrebno je ispitati vrednosti po kriterijumu u svim njenim zapisima. Situacija se usložnjava kada se radi o 'velikim' zapisima. Primer takvih zapisa su oni koji sadrže dokumenta, slike, audio, ili video sadržaje.

Proizvođači RDBMS preporučuju indeksiranje polja u tabelama kako bi se ubrzao pristup, ali to nije podrazumevano rešenje na nivou sistema već rešenje koje treba da primeni administrator RDBMS po ukazanoj potrebi i na svoju odgovornost.

Indeksiranje je dobro za sisteme u kojima nema čestih promena podatka jer ubrzava pretrage. Kod sistema kod kojih postoje česte promene (ažuriranja) podataka ovaj pristup se nije pokayao tako efikasnim. Izmena podataka, ili brisanje zapisa u tabeli najčešće je sa kriterijumom, što znači da umesto indeksnog, sistem koristi sekvencialni pristup zapisima. Indeksi takođe gube svrsishodnost ako se definišu za polja koja sadrže samo nekoliko konkretnih vrednosti, što prouzrokuje, pored velikog broja zapisa u tabeli koja se indeksira i veliki broj zapisa u indeksnoj tabeli. Na primer, u tabeli građana polje *pol*, koje ima 2 vrednosti, ili polje *bračni status*, koje takođe ima samo nekoliko vrednosti, u slučaju da se indeksiraju, izazvaće pravu eksploziju zapisa u indeksnoj tabeli, a problem sekvencijalnog pristupa se, pored tabele od interesa, proširuje i na indeksnu tabelu.

Na kraju, efikasno procesiranje velike količine podataka predstavlja isto tako veliki izazov u RDBMS obzirom da je obrada sadržaja tabela takođe sekvencijalna. Simultano (istovremeno) procesiranje podataka deljenjem između više procesora je, ili nemoguće, ili previše komplikovano.

Navedena ograničenja RDBMS su rezultovala da RDBMS dizajneri potraže rešenje u distribuiranosti podataka. Nažalost, kod komercijalnih RDBMS sistema licenca se najčešće naplaćuje po računaru – hostu na koji je RDBMS instaliran. To se pokazalo kao velika investicija čak i za velike kompanije kao što su Google i Amazon koji su od početka ovog milenijuma svoj biznis zasnivali na velikoj količini distiribuiranih podataka, a time umesto *main-frame* pristupa bili primorani da koriste klastere računara. Sve veća potreba za obradom i analizom velike količine podataka (Big Data Analytics), kao i velika heterogenost podataka distribuiranih na Web-u, bili su motiv da se u prvoj deceniji ovog veka započne sa razvojem tzv. NoSQL (*Not only SQL*) sistema za upravljanje BP. Google (BigTable) i Amazon (Dynamo) su prve počele da primenjuju potpuno novi pristup skladištenja podataka razvijajući sopstvena rešenja koja se danas nazivaju NoSQL-om.

14.3 OSNOVNI PRINCIPI NOSQL BP

NoSQL BP zasnivaju se na *simultanoj* (istovremenoj) obradi velike količine podataka koji su distribuirani na mreži. Podaci, iako su fizički distribuirani na više računara - hostova, oni se kombinuju u strukture koje predstavljaju logičke celine 'za sebe' koje se u NoSQL sistemima nazivaju *agregatima* i koje se formiraju prema potrebama korisničkih aplikacija. Dakle hostovi, koji čuvaju elementarne (sirove) podatke i koji učestvuju u davanju podataka, obrazuju logičku grupu nazvanu *klaster*.

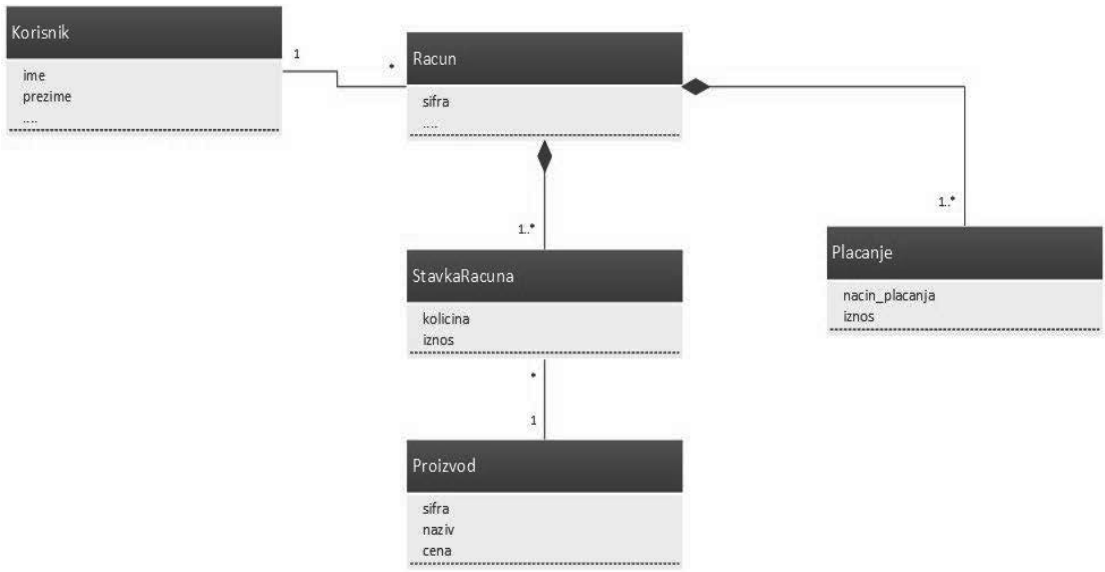
Međutim, ovakva koncepcija nije isključila problem velikog protoka podataka između hostova u klasteru. Obzirom da hostovi mogu biti bilo gde na globalnoj mreži, očigledna je bila potreba da NoSQL sistemi moraju da se optimizuju uvođenjem predobrade, koja bi se izvršavala simultano na svakom pojedinačnom hostu koji sadrži podatke od interesa i koja bi rezultovala smanjivanjem količine podataka u razmeni između hostova. Ovaj problem je rešen Google-ovim Map Reduce konceptom. U ovom poglavlju bavićemo se dakle sa četri osnovna koncepta NoSQL DBMS – agregacionim modelom podataka, organizacijom podataka na hostovima, modelima distribuiranja podataka i Map Reduce konceptom.

14.3.1 Agregacioni model podataka

Kao što je ilustrovano u prethodnoj sekciji (Slika 14.2), relacioni model podatka, koji se zasniva na tabelama koje imaju podatke organizovane po zapisima (*data records*), nije prikladan za modelovanje podataka u NoSQL DBMS. Osnovni problem

je što zapis predstavlja striktno određen model u kome su podaci u potpunosti specificirani (npr. tip vrednosti, veličina i ograničenja), normalizovani barem u 1. normalnoj formi (tzv. 'ravni' podaci) i nalaze se u striktnom redosledu. To znači da dizajneri tabela u RDBMS moraju unapred da znaju preciznu strukturu podataka koje trebaju aplikacijama. Takođe, u RDBMS struktuiranost složenijih podataka je podržana normalizacijom u više normalne forme i uspostavljanjem relacija između zapisa u različitim tabelama.

RDBMS pristup ima značajne prednosti jer eliminiše, ili barem umanjuje redundansu (ponavljanje) podatka u jednoj, ili više tabela, a normalizcija i relacioni model omogućava jednostavno održavanje konzistentnosti podataka i elimininaciju nepotrebnih (nepoželjnih) zavisnosti između podataka. Na žalost, u NoSQL sistemima to nije dovoljno, pošto se radi o podacima koji su distribuirani na Web-u, dakle na više hostova grupisanih u logičke celine - klastere. Umesto relacionog modela, u NoSQL sistemima koristi se mnogo fleksibilniji – agregacioni model podataka. Agregacija je koncept koji je preuzet iz objektno orjentisanog modelovanja – to je tip veze između podataka (objekata) koji oslikava njihovu slabu uzajamnu spregu koja omogućava veliku fleksibilnost i kombinovanje. Sledeća ilustracija (Slika 14.3) predstavlja agregacioni model za primer korišćen u sekciji 10.2 (Slika 2). Smisao modela je sledeći: koncepti *korisnik* i *račun* su u asocijativnoj vezi u odnosu jedan na strani korisnika prema više na strani računa; to znači da korisnik može da ima više kupovina (računa), ali račun se izdaje samo jednom korisniku. Koncept *račun* ima agregacione veze prema konceptima *stavka računa* i *plaćanje*. U slučaju stavki računa to znači da je korisnik kupovao različite vrste usluga i/ili roba. Kako su ovi predmeti kupovine predstavljeni podacima na različitim hostovima, mogu da se razlikuju po strukturi, ali priroda agregacione veze dozvoljava njihovo objedinjavanje. Istom logikom može da se objasni agregaciona veza između računa i plaćanja. Jedan račun može da se plati različitim načinima plaćanja. Na primer jednim delom avansno, a drugim po prijemu robe/usluge. Dalje, može da se plati delom u gotovini, a delom platnom kartic(om/ama) i/ili čeko(m/vima), u jednoj ili više rata, itd. Da bi ovako nešto bilo moguće, model podataka je relaksiran agregacionom vezom koja obezbeđuje slabu spregu između koncepata račun i plaćanje. To znači da u konkretnom slučaju može da bude uključen proizvoljan broj načina plaćanja, za ceo, ili delove iznosa, sa rokovima odmah, ili na odloženi vremenski period.



Slika 14.3: Primer agregacionog modela podataka

Prethodni primer oslikava veliku fleksibilnost NoSQL DBMS koji može da odgovori uspešno vrlo različitim poslovnim zahtevima. Osnovna prednost agregacionog u odnosu na relacioni model podataka je da omogućava proizvoljnu struktuiranost podataka od interesa. Dakle, hostovi čuvaju sirove podatke u nekom od formata, dok NoSQL DBMS kao zajednička infrastruktura za manipulaciju podacima obezbeđuje njihovo udruživanje prema konkretnim poslovnim zahtevima koristeći agregacione modele podataka.

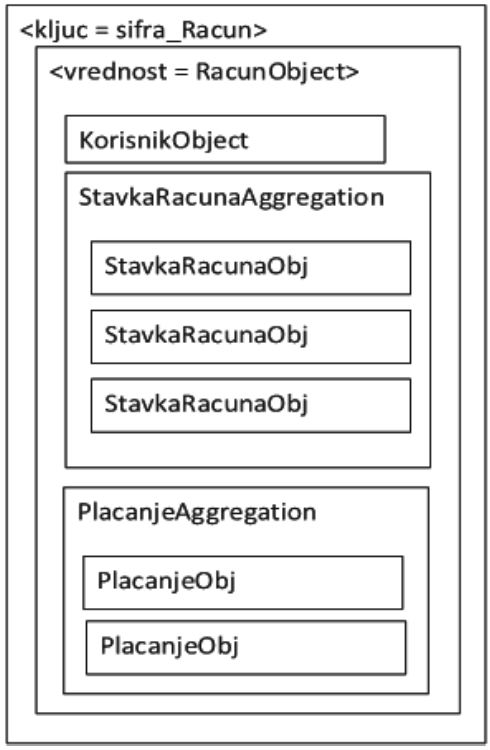
14.4 NOSQL DBMS ORGANIZACIJA PODATAKA

NoSQL DBMS organizacija podataka se odnosi na način na koji NoSQL hostovi skladište podatke kako bi bili lako pretraživi i brzo dostupni korisničkim aplikacijama. Za razliku od RDBMS-ova, kod kojih se organizacija rešava šemom BP, kod NoSQL sistema to nije slučaj. Najčešći načini skladištenja u NoSQL DBMS su skladišta ključ – vrednost parova, skladišta dokumenta i skladišta po familijama kolonama.

14.4.1 Skladišta ključ – vrednost parova

Ovaj tip skladišta predstavalja heš tabelu (*hash table*) u kojoj ključ – vrednost par predstavlja jedan zapis, pri čemu je ključ jedinstven (kao i koncept primarnog ključa definisanog nad jednim poljem u RDBMS), a vrednost može da bude podatak bilo kog tipa – prost kao brojčana vrednost, ili niz karaktera, ali i kompleksan kao neki objekat – dakle složena struktura podataka. Na sledećoj ilustraciji je primer računa koji je

smešten na skladištu kao ključ – vrednost par (Slika 14.4). Šifra računa predstavlja ključ za direktan pristup objektu klase *Racun*. Ovaj objekat je vrednost koja se čuva. On je kompleksan jer sadrži ugnježden objekat klase Korisnik i dve agregacije: jednu za stavke računa, a drugu za način plaćanja.



Slika 14.4: Primer skladištenja računa kao ključ – vrednost par

Sledeći kod predstavlja reprezentaciju ovog modela u ključ – vrednost skladištu (Slika 14.5). Svi ključevi su uokvireni crvenim, a njima odgovarajuće vrednosti (objekti) plavim pravougaonikom. Očigledno je da se svi podaci koji su agregirani u objekat račun čuvaju na ključ – vrednost skladištima. Kao što se određenom računu pristupa posredstvom ključa (šifre računa), tako se pristupa i podacima korisnika kome je izdat račun, stavkama računa i načinima plaćanja. U konkretnom primeru se vidi da je korisnik račun platio delom gotovinski, a delom kreditnom karticom. Važna je napomena da podaci koji su predstavljeni u objektu *račun* predstavljaju rezultat upita koje je NoSQL izvršio nad više skladišta podataka i predstavljaju samo deo podataka koji se čuvaju u distribuiranom sistemu. Na primer, može se uočiti da ne postoji datum izdavanja računa, ne postoji broj kreditne kartice, nema nikakvih informacija o proizvodima, ili uslugama koje je korisnik kupio i slično.



Slika 14.5: Izgled računa na ključ – vrednost skladištu zapisan u JSON formatu

Na prethodnoj ilustraciji se uočava da su podaci koji se skladište zapisani kao tekst u JSON formatu. JSON (JavaScript Object Notation) je format za tekstualni zapis podataka i predstavlja otvoren standard (RFC 8259, ECMA-404) dizajniran kako bi se omogućila efikasna razmena pre svega hijerarhijski struktuiranih podataka. Obzirom da je ovaj standard zasnovan na istom konceptu kao i ključ – vrednost skladišta, veoma je zastupljen kao deo infrastrukture u NoSQL sistemima.

14.4.2 Skladišta dokumenta

Skladišta dokumenta su možda najfleksibilniji NoSQL DBMS obzirom da omogućavaju da se u istu bazu podataka skladište podaci koji se uzajamno razlikuju kako u veličini, tako i u struktuiranosti. Drugim rečima, skladišta dokumenta nemaju

nikakvo znanje o sadržaju koji se skladišti, niti koriste bilo kavu šemu za skladištenje. Ovi NoSQL DBMS su dobili naziv zato što su pogodni za skladištenje dokumenata koji upravo variraju po prethodno navedenim karakteristikama. Sledeći primer to ilustruje (Slika 14.6). Prikazana su 2 jednostavna dokumenta u JSON zapisu (uokvireni zelenom i žutom). Naizgled slični, dokumenti se razlikuju u vrsti i broju atributa. Prvi ima 3 atributa, a drugi 4, pri čemu su svega 2 atributa zajednička – *ime* i *broj_pasosa*. Drugi dokument je pri tome i kompleksniji obzirom da ima veću struktuiranost.

```
{
    "ime":"Jovan Perić",
    "hobi":["plivanje","pecanje"],
    "broj_pasosa":"00034532"
}
```

```
{
    "ime":"Ana Mitrović",
    "putovanja_van_zemlje":["Italija","Austrija","Grčka"],
    "pebivalista":[
        {"grad":"Zrenjanin","ulica":"Đure Jakšića", "ul_br":"22/Ib"},
        {"grad":"Novi Sad","ulica":"Kisačka", "ul_br":"134/B12"}
    ],
    "broj_pasosa":"00034532"
}
```

Slika 14.6: Zapisi u skladištu dokumenata

Uprkos svim navedenim razlikama između dva dokumenta na gornjoj ilustraciji, za njih je zajedničko da sadrže lične podatke osoba koji se mogu prikupiti. Očigledno da se podaci razlikuju u struktuiranosti. Međutim, fleksibilnost skladišta dokumenata omogućava da se ovakvi dokumenti čuvaju u istoj bazi podataka omogućavajući napredne pretrage. Ono što je vrlo uočljivo na datom primeru je da ne postoje jedinstveni identifikatori u zapisu. Drugim rečima dokument mora ostati originalnan. NoSQL skladišta dokumenata ovaj problem prevazilaze internio dodeljenim identifikatorima, koji omogućavaju konzistentnost u ažuriranju zapisa.

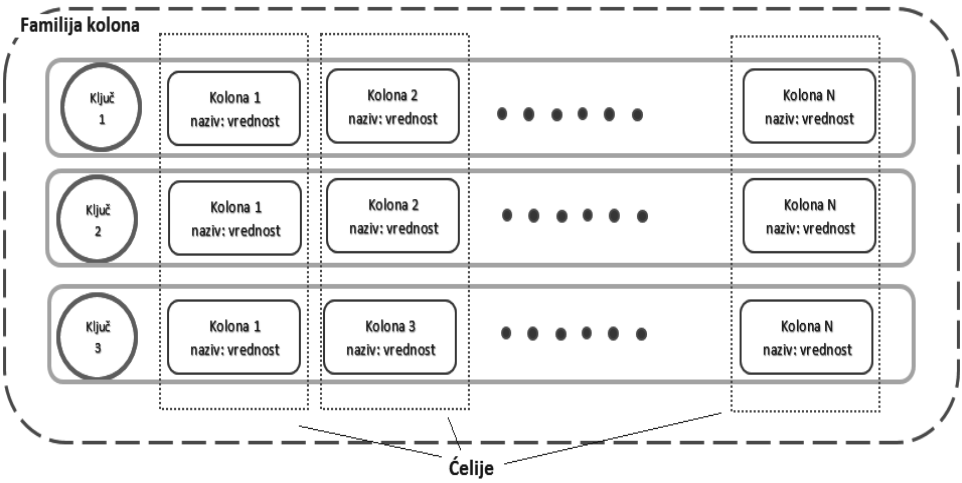
14.4.3 Skladišta familija kolona

Skladišta familija kolona su dobila ime po konceptu zvanom *familija kolona*. Ovaj koncept je sličan konceptu skupa zapisa, ili tabele kod RDBMS sistema s tom razlikom da svi zapisi u istoj tabeli imaju potpuno isti broj, iste tipove, iste nazive i druge karakteristike polja, dok kod familije kolona to nije slučaj: redovi u familiji kolna ne moraju da imaju iste vrste kolona, a time niti podatke koje one sadrže (Slika 7). Kolona je predstavljena parom naziv – vrednost. Redovi u familijama kolona su jednoznačno određeni ključem – konceptom sličnim konceptu primarnog ključa definisanog nad

jednim poljem u tabelama relacionih baza podataka. Ključ reda u familijama kolona uvek ima smisao – transparentan (kao ime i prezime, matični broj i slično), ili skriven (heš vrednost transparentnog smislenog podataka). Pored toga u nekim sistemima uvodi se koncept ćelija kojima se grupišu kolone koje imaju isti naziv. Ćelije omogućavaju da se redovi grupišu što bolje prema sličnositi njihovih kolona, a time i da se formiraju što kvalitetnije familije kolona. Ćelije omogućavaj da se kolone mapiraju, a njihovim imenovanjem dobija se nova dimenzija skladištenja koja se naziva superkolonom, odnosno dobija se novi nivo grupisanja – familija superkolona.

Ova NoSQL skladišta grupišu ključeve redova i superkolona u takozvane prostore ključeva (keyspaces) koji predstavljaju pristupne tačke za pristup srodnim familijama kolona, čineći ih jedinstvenim logičkim celinama podataka – konceptom sličnim bazama podataka kod RDBMS.

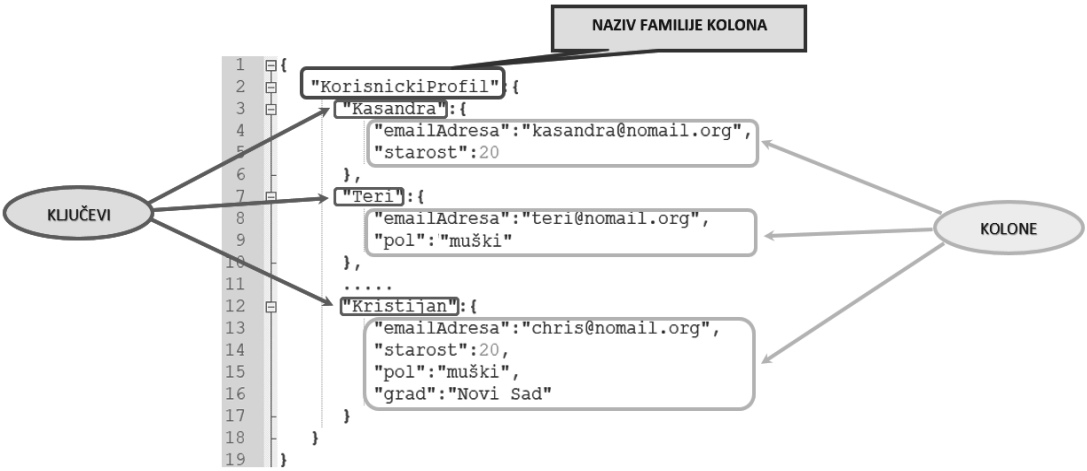
Svrha formiranja familije kolona je da se izvrši grupisanje podataka kojima se često pristupa zajedno. Kod tabela u relacionim bazama podataka nema nikakvih veza između susednih zapisa. Na primer, u tabeli građana zapisi koji predstavljaju članove iste porodice, ili zaposlene u istoj firmi su u potpunosti izolovani, dok susedni zapisi u gotovo svim slučajevima nemaju ništa zajedničko. Da bi se omogućilo bilo kakvo grupisanje, RDBMS pretražuje podatke kroz iterativni proces koji je vrlo neefikasan za ogroman broj zapisa. Familija kolona predstavlja NoSQL skladište kojim se rešava upravo ovaj problem. To znači da u konceptualnom modelu predstavljenom na slici ključevi 1,2, i 3 jesu jednoznačni, ali su redovi koje identifikuju uzajamno povezani i predstavljaju logičku grupu podataka. Na primer, familiju kolona NoSQL DBMS može da formira od naučnika koji su objavili više zajedničkih naučnih radova, muzičara koji su imali zajedničke nastupe na velikom broju koncerata, osoba koje imaju slična interesovanja, ili zajedničke prijatelje i sl.



Slika 14.7: Familija kolona – konceptualni model

U pretragama, zahvaljujući ključevma redova i nazivima kolona vrlo brzo se dobijaju rezultati i kod velikog broja redova. S druge strane ovaj koncept omogućava da se lako nalazi sličnosti između podataka (redova) koji se nalaze u različitim familijama.

Sledeći JSON formatiran kod (Slika 14.8) predstavlja praktičan primer skladišta familije kolona koja se zove *KorisnickiProfil*.



Slika 14.8: Primer familije kolona

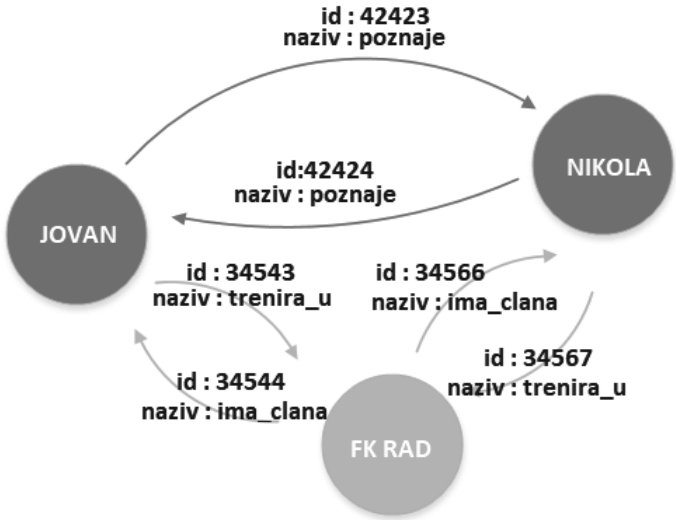
Korisničko ime predstavlja ključ reda u familiji. U samom redu se nalaze kolone predstavljene nazivom kolone i njenom vrednošću. Na primer ključ prvog reda na gornjem primeru je *Kasandra*, njen red ima 2 kolone. Prva ima naziv *emailAdresa* i vrednost *kasandra@nomail.org*, a druga kolona ima naziv *starost* i vrednost *20*. Na primeru se vidi da kolona koja je zajednička za sve redove je *emailAdresa*. Poslednji red u familiji ima četri kolone, dakle sadrži najpotpuniji korisnički profil.

14.4.4 Grafovska skladišta podataka

Pored navedenih načina organizovanja podataka na NoSQL skladištima postoje i drugi, od kojih je najčešće pominjan model grafa. Graf predstavlja strukturu podataka koja se sastoji od čvorova i lukova. U NoSQL grafovskim skladištima u čvorovima se nalaze podaci, najčešće složenog tipa (objekti), a lukovima su predstavljene njihove uzajamne relacije.

Sledeća slika prikazuje jednostavno grafovsko skladište podataka (Slika 14.9). Skladište sadrži samo 3 čvora i šest lukova – između svakog čvora po dva koji su

obrnuto usmereni. Od podataka u čvorovima predstavljeno je samo ime osobe kao identifikator obzirom da su ostali podaci u čvorovima irelevantni za objašnjenje. Ono što grafovsko skladište poseduje, a sva ostala prethodno opisana skladišta nemaju su opisi relacija između podataka (čvorova). Na prikazu je zadržan minimalistički pristup, tako da je svaka relacija opisama samo sa 2 parametra: ima svoj jedinstveni identifikator na skladišti i naziv relacije. Naziv relacije je *predikat* u trojci gde čvor iz koga relacija počinje predstavlja *subjekat,* a čvor na koji relacija pokazuje je objekat. Na primer u relaciji čiji id je 34543 *Jovan* je subjekat a *FK Rad* predikat (*Jovan trenira u FK Rad*). Relacija u suprotnom smeru (id : 34544) ima drukčiju strukturu: subjekat je *FK Rad*, predikat je *ima_clana*, a *Jovan* je u ovoj relaciji objekat. Kao posledica, ova relacija nosi i drukčiju semantiku (smisao): *FK Rad ima člana* koji se zove *Jovan*).



Slika 14.9: Grafovska BP

Međutim dve relacije između 2 objekta mogu da imaju istu semantiku iako su različito usmerene. Na gornjem primeru to su relacije id : 42423 i id : 42424. One odražavaju poznanstvo između Nikole i Jovana iako u prvoj Jovan predstavlja subjekat, a Nikola objekt u relaciji, dok je u drugoj obrnuto.

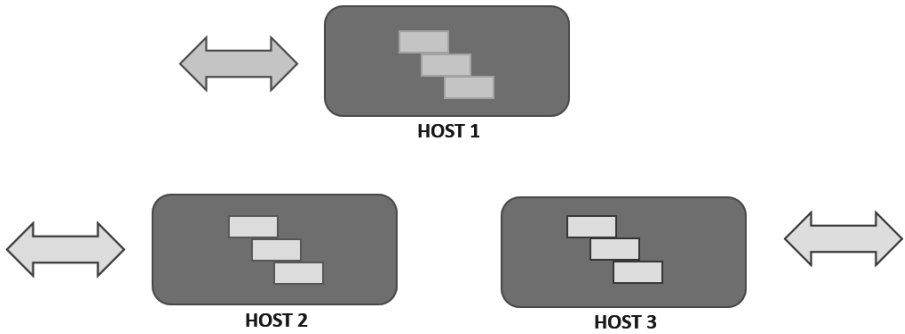
Zahvaljujući relacijama, grafovska skladišta imaju tu sposobnost da oslikavaju širi kontekst postojanja podatka koji je predstavljen čvorom grafa. Broj uvirućih i izvirućih relacija odražavaju njegovu poziciju i značaj u odnosu na ostale podatke (čvorove) na skladištu. Sva grafovska skladišta omogućavaju prikupljanje podataka iz susednih čvorova (analiza direktnih relacija), ali primenom algoritama za pretragu grafa za dubine veće od 1 (pretraga putanja i čvorva koji nisu u neposrednoj vezi) aplikacije mogu da pronađu podatke koji se ne mogu prikupiti na drugi način.

14.5 NOSQL DBMS – MODELI DISTRIBUIRANJA PODATAKA

Jedan od najvažnih zadataka koje treba da ispunjavaju NoSQL sistemi jeste održavanje podataka distribuiranih u klasteru. Postoje različiti načini distribuiranja:

- Potpuno deljenje
- Master – slave replikacije
- Replikacije na ravnopravnim hostovima
- Kombinacija deljenja i repliciranja

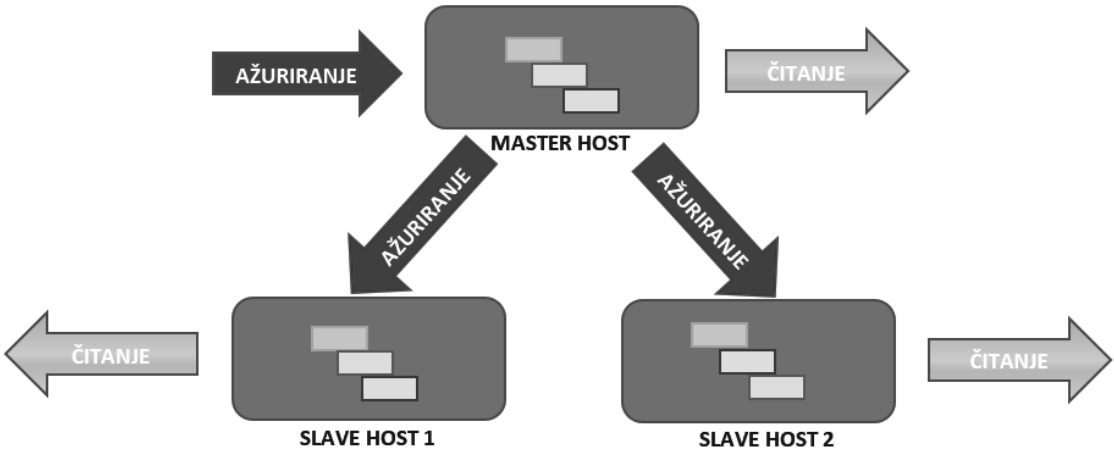
Potpuno deljenje je model kod koga se na svakom hostu u klasteru nalaze različiti podaci (Slika 14.10). Ne postoje kopije podataka na drugim hostovima tako da je konzistentnost podataka lako održiva. Ovaj model je pogodan u slučajevima da aplikacije koje pristupaju lokalnim NoSQL DBMS hostovima takođe imaju lokalni karakter. Drugim rečima one uopšte ne potražuju, ili veoma retko potražuju podatke sa drugih hostova u klasteru.



Slika 14.10: Potpuno deljenje podaka

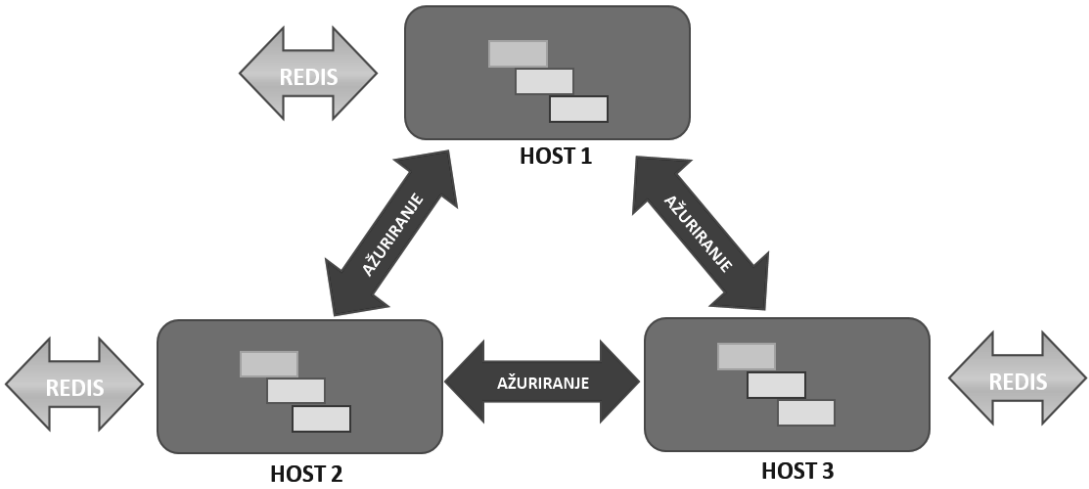
Kod potpunog deljenja svaki host je odgovoran za razmenu (čitanje, upis i ažuriranje) samo podataka koje skladišti.

Master – slave replikacije je model koji predstavlja potpunu suprotnost prethodnom i kod koga se na svim hostovima u klasteru nalaze isti podaci (Slika 14.11). Jedan od hostova u klasteru igra ulogu mastera. To znači da je izmena podataka moguća samo na master hostu, dok je na ostalim hostovima (koji imaju *slave* ulogu) dozvoljeno samo čitanje, ali ne i promena podataka. Iz ovakve podele uloga proizilazi još jedan zadatak mastera – održavanje konzistentnosti podataka, odnosno ažuriranje replika podataka na svim *slave* hostovima. Uslov za uspešnost primene ovog modela je pouzdana mrežna infrastruktura i velike brzine protoka podataka.



Slika 14.11: Master – slave deljenje

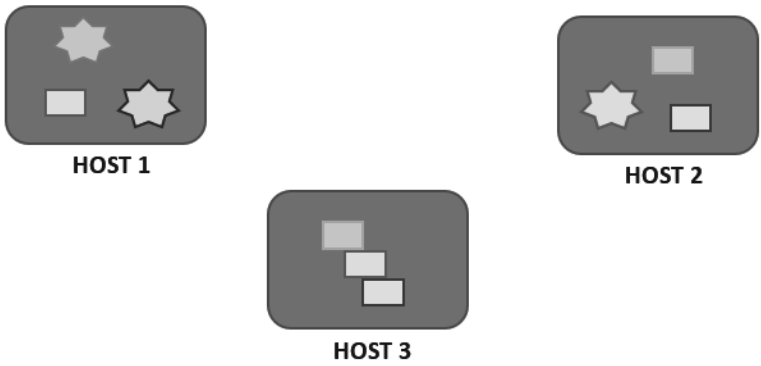
Replikacije na ravnopravnim (peer to peer) hostovima je model koji se razlikuje u odnosu na prethodni po tome što svi hostovi u klasteru mogu da čitaju i da menjaju podatke (označeno kao REDIS – eng. *Read Edit Delete Insert Select*, slika 14.12). Ovaj model znatno usložnjava održavanje konzistentnosti podataka u klasteru jer promena podatka na bilo kom od hostova propagira se i na sve ostale hostove. U tom slučaju NoSQL sistem mora da podrži transakcionu obradu, odnosno da se promene na svim hostovima odigraju u jednoj transakciji.



Slika 14.12: Peer to peer deljenje podataka

Može se zaključiti da su ovakvi sistemi ugroženi u slučaju gubitka komunikacionog kanala za ažuriranje pošto postoji mogućnost distribucije netačnih podataka korisničkim aplikacijama. Permanentno testiranje linkova pre i u toku transakcije radi prihvatanja (commit) ili odbijanja (rollback) transakcije nije dovoljna mera da bi se uspostavila pouzdanost ovakvih distribucionih sistema.

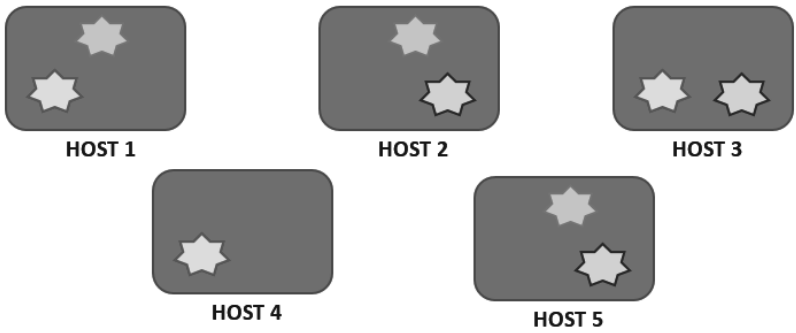
Mogućnosit pojave nekonzistentnih podataka u gore opisanim situacijama doveo je do pojave hibridnog modela koji je nastao kao *kombinacija deljenja i repliciranja*. Postoje 2 varijante ovog modela. U prvoj varijanti (Slika 14.13) hostovi mogu da budu u isto vreme masteri za neke podatke (skladište njihove originale i ažuriraju njihove replike), a za druge podatke da imaju ulogu *slave* hosta (samo skladište kopije). Takođe u verijanti 1 ostovi mogu da budu samo *master*, ili samo *slave* hostovi. Pravilo distribuiranosti je da u klasteru postoji sam jedan original podatka i zna se na kom hostu je skladišten. Samo tako može da se implementira uspešno ažuriranje podataka.



Slika 14.13: Kombinacija deljenja i repliciranja – varijanta 1

Na gornjoj slici host 1 ima ulogu mastera za 2 podatka (prikazani zvezdama) a *slave* hosta za treći. Drugi host je master za jedan podatak a *slave* za 2, dok je host 3 *slave* host.

Druga varijanta kombinacije deljenja i repliciranja podrazumeva ravnoravnost hostova (*peer to peer* hostovi). To znači da svi hostovi imaju ulogu mastera, a replika podataka nema već sve instance podataka predstavljaju originale.

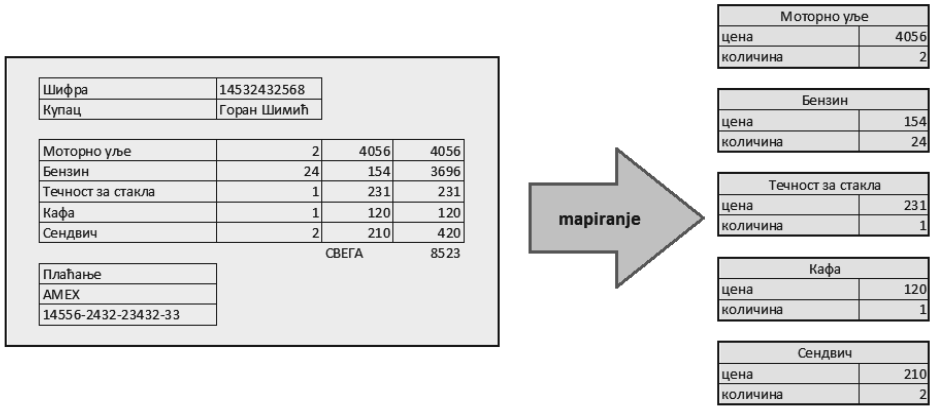


Slika 14.14: Kombinacija deljenja i repliciranja – varijanta 2

Prilikom promene podataka na bilo kom hostu, on preuzima odgovornost za ažuriranje tog podatka na svim ostalim hostovima. Drugim rečima svaki host mora da zna za hostove u klasteru na kojima je skladišten taj isti podatak. U praksi korišćenja NoSQL DBMS podaci su distribuirani faktorom tri, što znači da je svaki distribuiran na 3 čvora u mreži (hosta u klasteru). Na taj način povećana je otpornost sistema na padove mreže, odnoso klaster se deli na delove koji se dalje konsoliduju u prvobitni nakon uspostavljanja normalnog transfera podataka. Da bi se ovo omogućilo, NoSQL DBMS-ovi pamte vremena promene svih podataka koje skladište na klasteru, što rezultira konzistentnom rekonstrukcijom podataka po ponovnom uspostavljanju komunikacionih kanala. Karakteristično za skladišta sa familijama kolona je da koriste ovaj način distribuiranja podataka.

14.6 MAP REDUCE KONCEPT

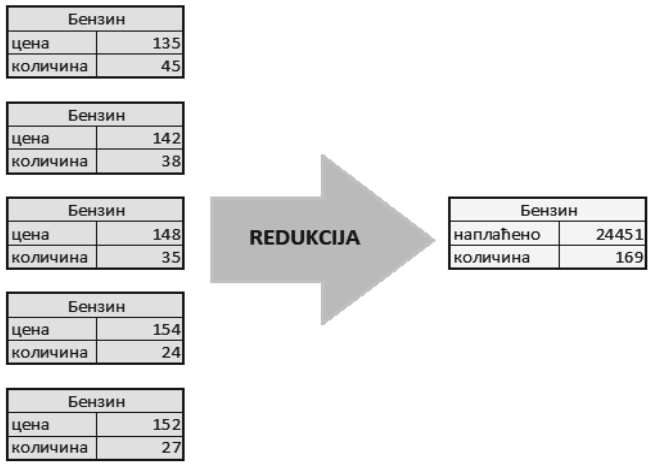
Map Reduce koncept je nastao kao proizvod potrebe da se podaci predstavljeni agregacionim modelom efikasno razmenju između hostova u klasteru. Implementacija ovog koncepta sarži dve faze. Prva je mapiranje. Mapiranje se vrši pomoću funkcije koja kao ulazni parametar ima agregiran podatak, a kao izlaz daje niz podataka predstavljenih kao parovi ključ – vrednost. Sledeći primer to ilustruje (Slika 14.15). Račun koji predstavlja agregiran podatak je ulazni parametar funkcije za mapiranje, a rezultat njenog izvršenja su pojedinačne stavke računa predstavljene po *ključ – vrednost* principu. Ključ svake stavke je naziv proizvoda, a vrednost je struktura koja takođe ima 2 *ključ – vrednost* para: za cenu i za količinu.



Slika 14.15: Mapiranje podataka iz agregata u ključ vrednost parove

Obzirom da se radi o podacima koji predstavljaju agregate, drugim rečima, koji su sastavljeni od distribuiranih podataka na različitim hostovima, funkcija mapiranja se u realnosti izvršava simultano (istovremeno) na svim hostovima na kojima se nalaze podaci od interesa. Ovakav pristup omogućava veliku brzinu procesiranja podataka kako zbog paralelizma obrade, tako i zbog lokalnog karaktera podataka koji se obrađuju. Drugim rečima, host obrađuje u ovoj fazi samo podatke koje skladišti.

Funkcija za mapiranje se izvršava na jednom zapisu – u gornjem primeru to je jedan račun. Ako pretpostavimo da je cilj obrade da se prikupe podaci kupovine određenog proizvoda, funkcija za mapiranje će se izvršiti za sve račune svih korisnika (Slika 14.16). U tom slučaju dobijaju se kao rezultat *ključ – vrednost* parovi čiji broj može da se redukuje u skladu sa zadatkom obrade. Redukcija parova ključ-vrednost predstavlja drugu fazu *Map Reduce* koncepta.



Slika 14.16: Redukcija ključ-vrednost parova

Kao rezultat redukcije umesto velikog broja ključ – vrednost parova, host emituje samo jedan par, koji predstavlja konačan rezultat obrade korišćenjem *Map Reduce* koncepta.

14.7 PRIMERI NOSQL DBMS

Od početka 21. veka nastala su mnogobrojna NoSQL skladišta podataka koja predstavljaju reprezentativne primere koncepata koji su objašnjeni u poglavlju 10. *Google Bigtable* kao prvo skladište bazirano na familiji kolona, namenjeno kao osnovna infrastruktura za pretrae na Web-u, poslužilo je kao inspracija za *HyperTable* i *HBase* skadišta.

Amazon-ovo skadište *Dynamo,* kao prvo skladište na bazi ključ-vrednost parova pravljeno kao infrastruktura za analizu aktuelnih tržišta, motivisalo je razvoj *Facebook*-ovog skladišta *Cassandra* i projekat *Voldemort*.

Jedno od prvih skladišta dokumenata - *Apache*-ov *CouchDB*, (koristi ga *BBC bradcasting)* poslužilo je kao motiv za nova rešenja od kojih se najviše ističe široko prihvaćen *MongoDB*. *ElasticSearch* predstavlja još jedan primer skladišta dokumenata koji je zastupljen u poslovnim primenama. Pored toga, interesantno je da su skladišta dokumenata vrlo primenjena u novijim generacijama geoprostornih sistema, koji se baziraju na fuziji ogromne količine podataka iz različitih izvora (npr. *Google Maps*).

Na kraju pominjemo i reprezentativne primere grafovskih skladišta podataka: *IBM DB2* (od verzije 10), *Neo4J*, *InfiniteGraph*, *InfoGrid* i drugi. Ova skladišta su takođe našla široku primenu u svim domenima u kojima su bitne relacije između podataka: društvene veze, sistemi javnog transporta, putne mape i mrežne topologije.

15 PRIMERI ZA MODELOVANJE

15.1 ADRESAR NA MOBILNOM TELEFONU

U razvojnom centru popularnog proizvođača mobilnih telefona pokrenut je projekat razvoja softverskog modula za upravljanje telefonskim imenikom. U okviru ovog projekta potrebno je razviti model baze podataka u kojoj bi se čuvali zapisi telefonskog imenika. Kao osnovni zahtev pri razvoju modela navedeno je postizanje visokih performansi sa ograničenim procesorskim resursima. Svaki od zapisa u imeniku treba da sadrži naziv, adresu i broj telefona. Dužinu naziva zapisa, adrese i broja telefona treba ograničiti na po 256 bajtova, a skladištenje broja telefona omogućiti i sa prefiksom „+“. U pogledu funkcionalnosti zahtevano je listanje zapisa u imeniku, filtriranje zapisa unosom dela naziva i pronalaženje naziva za uneti broj telefona. Razvijeni model baze podataka bio bi realizovan u okviru *SQLite* ili *MySQL* sistema za upravljanje bazama podataka na Linuks/Android operativnom sistemu.

- **Model baze podataka**

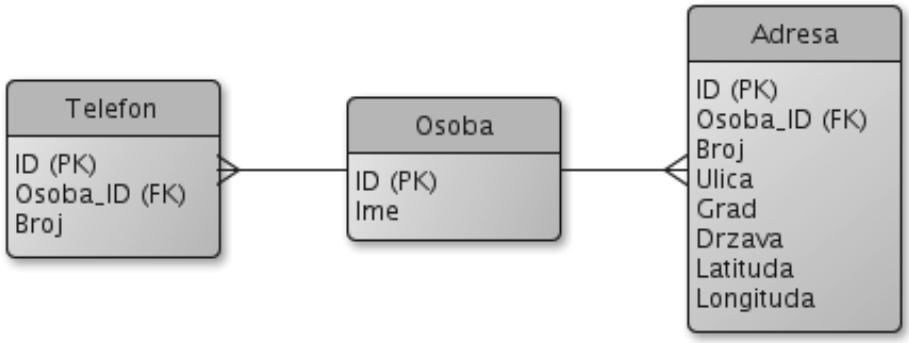


- **Pitanja**
  - Da li je predloženi model normalizovan i koje normalne forme nisu ispoštovane?
  - Da li ima potrebe za denormalizacijom baze u cilju podizanja performansi?
  - Koje su prednosti realizacije rešenja korišćenjem *SQLite* SUBP-a u odnosu na *MySQL* ili fajl-sistem (skladištenje podataka unutar fajla)?
  - Na koji način bi se na nivou *SQL* jezika moglo omogućiti prepoznavanje zapisa čiji je broj skladišten u formatu „+38164...“ a koji je korisnik uneo u formatu „064...“?

- **Naknadni zahtevi**

U nove modele mobilnih telefona ugrađuju se značajno brži procesori nego što je to bio slučaj kod modela za koji je razvijen prethodni sistem za upravljanje telefonskim imenikom. U međuvremenu, kao osnovna mana prethodnog rešenja prepoznato je postojanje više zapisa u imeniku kod osoba za koje se čuva više kontakt telefona ili adresa. Takođe, ugradnjom *GPS* modula javila se ideja za mogućnošću sortiranja zapisa u imeniku po udaljenosti kontakata od trenutne lokacije korisnika mobilnog telefona. Koje su izmene u prethodnom modelu baze podataka potrebne da bi se rešio navedeni problem i omogućilo pomenuto sortiranje?

- **Model baze podataka**



- **Pitanja**

  - Kako će se navedene izmene odraziti na složenost i performanse sistema?
  - Da li se izmenom pojavljuje novi kandidat za primarni ključ u tabeli Osoba?
  - Ukoliko bi se novi sistem za upravljanje telefonskim imenikom primenio na telefone koji već koriste stari sistem, koji su koraci potrebni da ne bi došlo do gubitaka postojećih kontakata?
  - Na koje je sve načine moguće realizovati sortiranje kontakata po udaljenosti od trenutne lokacije uređaja, a koji se mogu smatrati optimalnim?
  - Da li savremeni sistemi za upravljanje bazama podataka imaju podršku za geografske tipove podataka?

15.2 BIBLIOTEKA

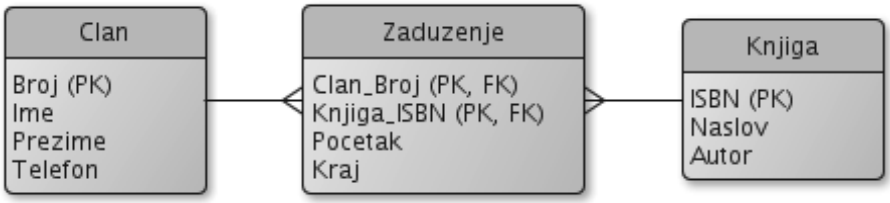
Za potrebe biblioteke jedne srednjoškolske ustanove potrebno je razviti softver za vođenje knjiga, članova i zaduženja. Za softver su planirane samo osnovne funkcionalnosti:

1. vođenje evidencije o postojećim članovima biblioteke,
2. vođenje evidencije o postojećim knjigama u biblioteci i
3. vođenje evidencije o aktivnim zaduženjima.

Podaci o članovima koje je potrebno voditi u bazi podataka su broj članske karte, ime, prezime, adresa i kontakt telefon. Svaka knjiga je određena međunarodnim standardnim knjižnim brojem (engl. *International Standard Book Number, ISBN*). Za naslove koji su se pojavili pre 2007. godine ovaj broj sadrži 10 cifara, a za naslove koji su se pojavili počev od te godine ovaj broj sadrži 13 cifara. Pored ovog broja za svaku od knjiga beleže se još i naslov i autor.

Proces zaduživanja knjige podrazumeva korišćenje kartonske kartice za beleženje broja članske karte člana, *ISBN* broj knjige koju je zadužio, kao i datum kada je zaduživanje izvršeno. Proces razduživanja podrazumeva beleženje datuma razduživanja u za to predviđeno polje na kartici. Pregled aktivnih zaduženja podrazumeva pretragu svih kartica kod kojih je polje za datum razduživanja prazno. Članovima je dozvoljeno paralelno zaduživanje do tri knjige ali se u pojedinim slučajevima odobrava i prekoračenje ovog broja.

- **Model baze podataka**

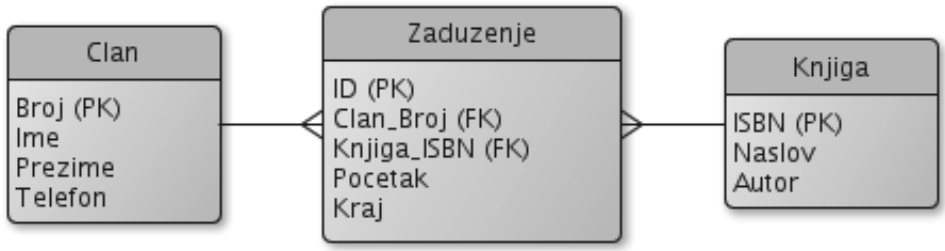


- **Pitanja**
  - Da li predstavljeni model može da odgovori na sve zahteve slučajeve korišćenja?
  - Šta će se dogoditi ukoliko jedan član pokuša dva puta da zaduži istu knjigu?
  - Da li je predloženi model normalizovan i koje normalne forme nisu ispoštovane?
  - Da li ima potrebe za denormalizacijom baze u cilju podizanja performansi?
  - Koje su izmene na modelu potrebne da bi se on mogao primeniti na video-klub ili rent-a-kar agenciju?

- **Naknadni zahtev 1**

Tokom korišćenja rešenja baziranog na prethodno razvijenom modelu uočeno je da on ne podržava slučaj u kome jedan član može da više puta zaduži istu knjigu. Ovo ograničenje je u praksi zaobilaženo brisanjem zapisa o prethodnom zaduživanju pre dodavanja novog ili izmenom početka zaduživanja. Međutim, na ovaj način dolazi do gubitaka podataka, odnosno oštećuje se istorijat zaduživanja. 1. Koje izmene modela baze podataka su potrebne da bi se rešio navedeni problem?

- **Model baze podataka**



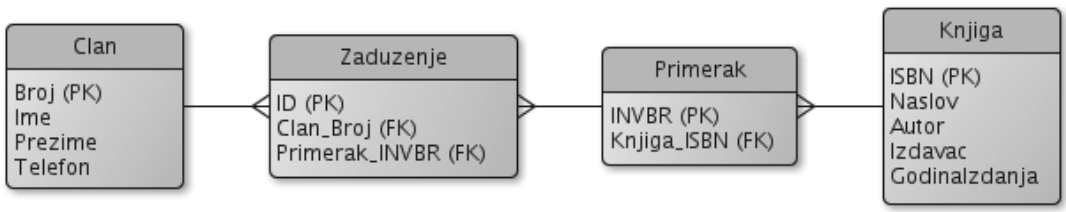
- **Pitanja**

  - Na koji način se izmene mogu ugraditi u postojeći sistem a da se ne izgube trenutni podaci?

- **Naknadni zahtev 2**

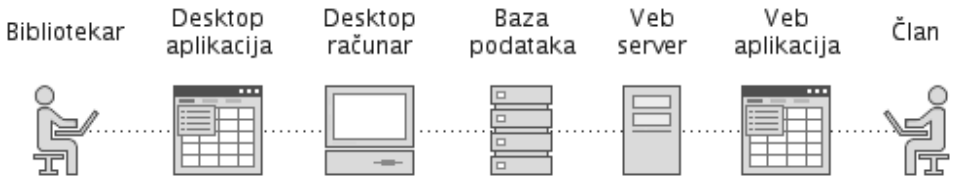
Korišćenjem računarske baze podataka nije moguće utvrditi da li u biblioteci postoji slobodan primerak određenog naslova, ili se pre izdavanja mora sačekati na vraćanje nekog od već izdatih primeraka. Svi primerci knjiga u biblioteci su obeleženi jedinstvenim inventarskim brojem. Dodatno, isti naslovi u biblioteci često postoje izdati od strane različitih izdavača i u raznim godinama izdanja te bi i ove podatke o naslovu trebalo podržati u bazi podataka.

- **Model baze podataka**



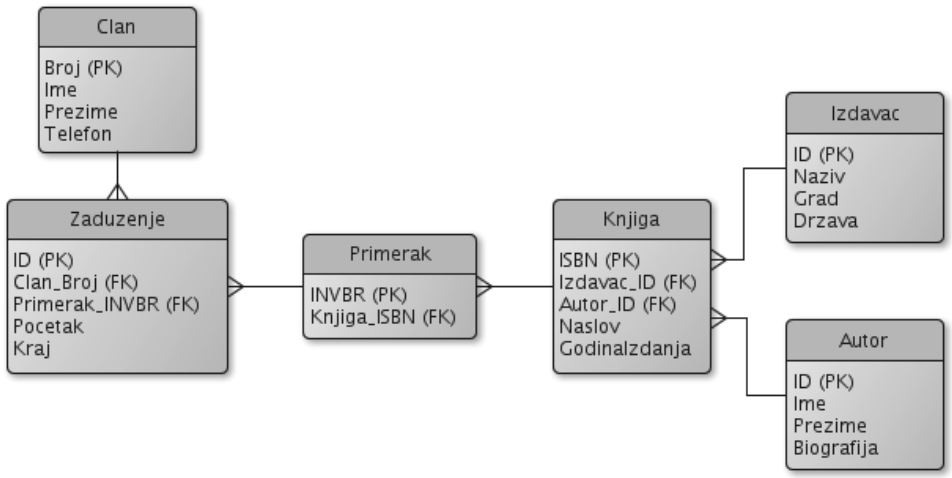
- **Pitanja**
  - Na koji način se izmene mogu ugraditi u postojeći sistem a da se ne izgube trenutni podaci?
- **Naknadni zahtev 3**

Uočena je i nepravilnost u korišćenju programa koja se ogleda u različitim unosima istih autora (ćirilica, latinica, unos na izvornom pismu i jeziku, različiti redosled imena i prezimena, različito korišćenje inicijala...) od strane korisnika programa. Ovo onemogućava pretragu naslova po imenu autora, korisnici uglavnom pogađaju različite moguće kombinacije, a sve češće se dešava da se propusti neki od naslova. Isti problem, iako manje značajan, javio se i kada je u pitanju izdavač knjige.



Dodatno, planirano je proširenje postojećeg rešenja dodavanjem Veb aplikacije koja bi članovima biblioteke omogućila da se preko Interneta informišu o postojećim i slobodnim naslovima u biblioteci. Ova aplikacija bi koristila centralnu bazu podataka (istu kao i aplikacija koju koriste radnici u biblioteci). Kao jedna od mogućnosti Veb aplikacije planirano je i prikazivanje kratke biografije autora što takođe treba podržati u modelu baze podataka.

- **Model baze podataka**



- **Pitanja**

  - Da li je bilo moguće navedeni problem potpuno ili delimično rešiti bez izmene baze podataka, odnosno na sloju logike rešenja?

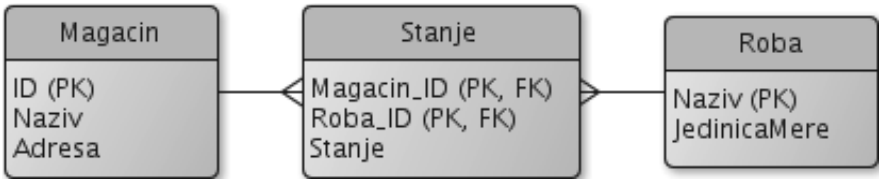
  - Tehnologija u kojoj je izrađen korisnički interfejs aplikacije omogućava reagovanje na događaj pritiska tastera (događaj *onkeypress*) kada se fokus postavi na polje za unos autora za pretragu naslova. Koji upit treba izvršiti u bazi podataka da bi se korisniku ponudila lista svih autora čije ime/prezime sadrži uneti tekst?

  - Da li postoje bezbednosni rizici kod omogućavanja pristupa sadržaju baze podataka putem Interneta? Ukoliko postoje, na koji način bi se mogli otkloniti ili umanjiti?

15.3 MAGACIN

Potrebno je razviti model baze podataka koji bi podržao upravljanje skupom magacina. Svaki od magacina ima svoj naziv i adresu a u njemu se u svakom trenutku može nalaziti proizvoljna količina različitih tipova robe. Količina različitih tipova robe meri se različitim mernim jedinicama (komad, metar, kilogram...). Model treba da omogući da se u svakom trenutku može videti sadržaj svakog od magacina, kao i da se za svaki tip robe mogu videti dostupne količine u različitim magacinima.

- **Model baze podataka**



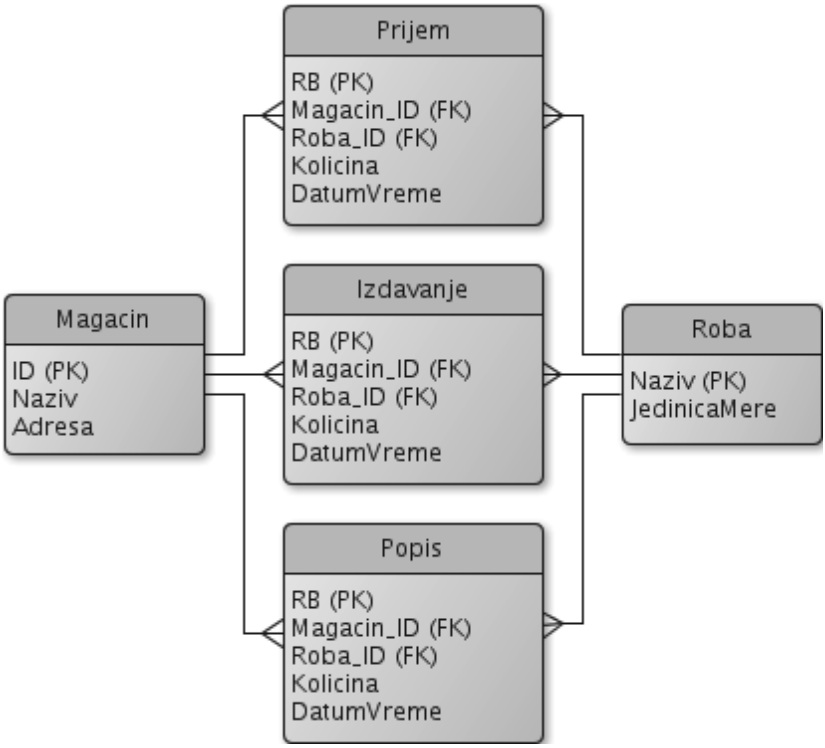
- **Pitanja**

  - Koji tip podataka treba izabrati za kolonu *Stanje* u istoimenoj tabeli?

- **Naknadni zahtev**

Nakon inicijalnog perioda korišćenja razvijenog modela utvrđeno je više njegovih nedostataka. Prvenstveno, prilikom popisa su utvrđene značajne razlike između stanja u bazi podataka i stvarnih stanja u magacinima. Iz tog razloga je postavljen zahtev da se u bazi podataka podrže svi prijemi i izdavanja robe, kao i da se omogući unos razlika evidentiranih prilikom popisa.

- **Model baze podataka**



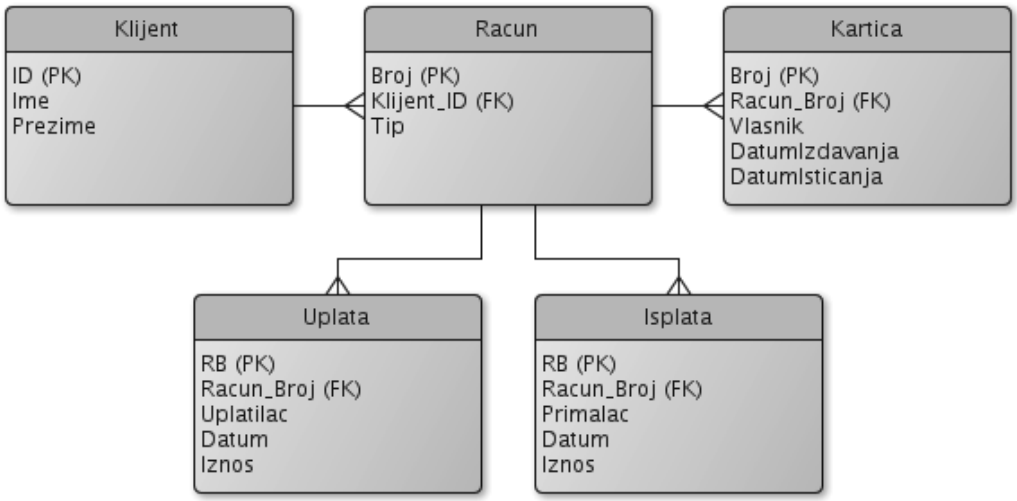
- **Pitanja**

1. U čemu se sve ogleda značaj polja *DatumVreme* u tabelama *Prijem*, *Izdavanje* i *Popis*?
2. Koji su koraci nakon uvođenja predstavljenih izmena potrebni za dobijanje aktuelnog stanja određene robe u određenom magacinu?
3. Koji su koraci potrebni za dobijanje aktuelnog stanja u određenom magacinu za sve tipove robe?
4. Koji su koraci potrebni za dobijanje aktuelnog stanja određenog tipa robe u svim magacinima?
5. Na koji način uvedene izmene utiču na performanse sistema?

15.4 BANKA

Za potrebe određene banke potrebno je razviti osnovni model baze podataka. Model treba da podrži samo vođenje evidencije o klijentima, računima, uplatama, isplatama i izdatim platnim karticama. Na osnovu podataka u bazi potrebno je da se u svakom trenutku može dobiti trenutno stanje na određenom računu.

- **Model baze podataka**



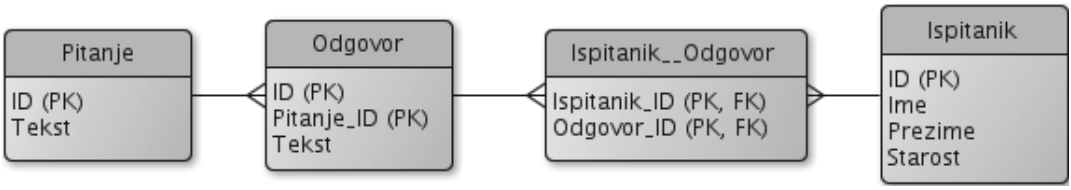
- **Pitanja**

  - Na koji način se preko razvijenog modela utvrđuje trenutno stanje na određenom računu? Šta su prednosti a šta mane takvog pristupa i koje alternative postoje?
  - Koja je prednost centralizovane baze podataka sa aspekta postojanja velikog broja filijala banke? Koji su funkcionalni i bezbednosni izazovi posledica centralizacije?
  - Koje su izmene modela potrebne da bi se podržala i štednja klijenata, odnosno obračunavanje kamate?
  - Koje su izmene modela i celokupnog sistema potrebne da bi se klijentima omogućilo elektronsko bankarstvo (upravljanje računom putem Interneta)? Koji su bezbednosni i funkcionalni izazovi?

15.5 MARKETINŠKA AGENCIJA

Za potrebe marketinške agencije je potrebno razviti model baze podataka koja bi omogućavala sprovođenje anketa putem Interneta. Ispitanici na početku anketiranja unose ime, prezime i starost, a zatim odgovaraju na postavljena pitanja. Pitanja sadrže predefinisane vrednosti kao odgovore koje ispitanici biraju. Nakon sprovođenja ankete vrši se statistička analiza datih odgovora.

- **Model baze podataka**

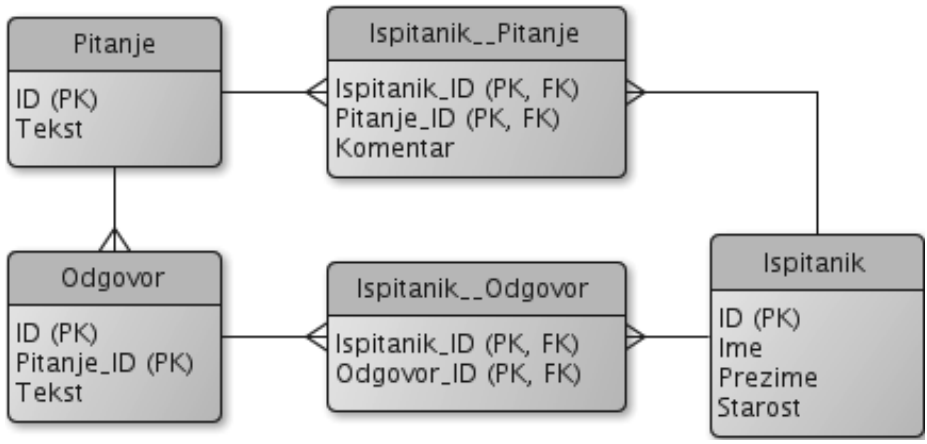


- **Pitanja**
  - Na koje načine se sa postojećim modelom može odrediti redosled prikazivanja pitanja i odgovora?
  - Kojim skupom upita se mogu dobiti procenti davanja oodređenih odgovora?
  - Da li je moguće predstavljeni model iskoristiti i za proveru znanja ispitanika? Koje izmene su potrebne u tom slučaju?

- **Novi zahtev**

Nakon inicijalnog perioda korišćenja razvijenog modela uočen je jedan značajan nedostatak koji se ogleda u tome da ispitanici ne mogu da daju komentare na ponuđena pitanja. U skladu sa tim, potrebno je izmeniti model tako da se za određena pitanja omogući unos slobodnih vrednosti.

- **Model baze podataka**



- **Pitanja**
  - Da li predloženo rešenje omogućava uklanjanje kolona *Ime*, *Prezime* i *Starost* iz tabele *Ispitanik*? Da li je time moguće u potpunosti isključiti tabelu *Ispitanik* iz modela?

15.6 ELEKTRONSKO GLASANJE

Za potrebe održavanja predstojećih izbora potrebno je razviti sistem elektronskog glasanja koji bi eliminisao potencijalne zloupotrebe i izmene rezultata, kao i omogućio praćenje rezultata glasanja u realnom vremenu. U okviru projektovanog rešenja planirana je centralizovana baza podataka i terminali na glasačkim mestima.

Svim punoletnim građanima treba izdati *SMART* kartice sa podrškom za digitalno potpisivanje. Proces izdavanja kartice podrazumeva:

- potvrđivanje identiteta glasača,
- generisanje para ključeva potrebnog za digitalno potpisivanje,
- registraciju identiteta glasača i javnog ključa u centralnoj bazi podataka i
- skladištenje identiteta glasača i tajnog ključa na kartici.

Sam proces glasanja podrazumeva:

1. umetanje kartice u odgovarajući interfejs na terminalu za glasanje,
2. potvrđivanje identiteta glasača i
3. izbor i potvrdu jedne ponuđenih opcija za glasanje.

S obzirom na osnovnu namenu i razlog izgradnje sistema, neophodno je onemogućiti višestruke glasove sa iste kartice, kao i glasanje korišćenjem kartica koje nisu prošle kroz regularnu proceduru izdavanja. Takođe, za svaki glas je potrebno zabeležiti vreme (preciznost na nivou sekunde) i *IP* adresu terminala na kome je glasanje izvršeno, a sama baza ne treba da prihvati glasove kod kojih nisu dostavljeni digitalni potpis i navedeni parametri.

- **Model baze podataka**



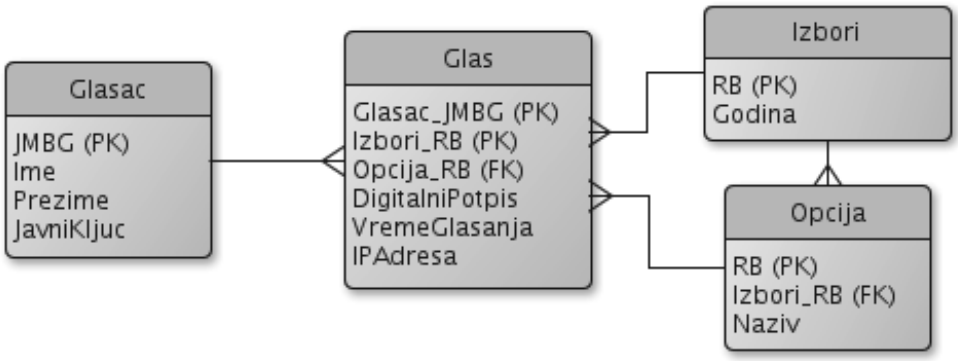
- **Pitanja**
  - Da li je predloženi model normalizovan i koje normalne forme nisu ispoštovane?
  - Da li ima potrebe za denormalizacijom baze u cilju podizanja performansi?
  - Imajući u vidu da pravo da glasa ima oko 7 miliona građana, razmotriti sledeće:
    - Koliki je kapacitet baze potreban za skladištenje identiteta glasača?
    - Koliki je kapacitet baze potreban za skladištenje glasova?
    - Kakvi su resursi potrebni za registrovanje pomenutog broja glasača i izdavanje kartica na 120 mesta u periodu od 3 meseca pre održavanja izbora?
    - Koji je potreban kapacitet celokupnog sistema (centralne baze, komunikacionog kanala i sl.) u slučaju izlaska na glasanje svih birača u periodu od 12 časova?
  - Koje su prednosti i mane skladištenje fotografije glasača u samoj bazi podataka i na fajl sistemu?
  - Imajući u vidu da neka mesta za glasanje nemaju direktnu vezu sa centralnom bazom podataka, na njima se predviđa korišćenje priveremenih priručnih baza sa naknadnom sinhronizacijom. Kakav to ima uticaj na bezbednost sistema i na koji način se mogu eliminisati zloupotrebe (na primer, višestruko glasanje istom karticom)?
  - Za podizanje bezbednosti sistema i smanjenje mogućnosti zloupotrebe, razmatra se upotreba biometrijskih podataka glasača (iris, otisak prsta...). Kakav to može imati uticaj na sistem i koje su izmene potrebne? Na koje načine je moguće zaštititi biometrijske podatke skladištene u bazi podataka? Koje mogućnosti otvara upotreba biometrije?

- Kao nedostatak sistema uočena je mogućnost da se iz baze vidi koji je glasač izabrao koju opciju. Koje su izmene potrebne da bi se zaštitio identitet glasača, odnosno da se na osnovu podataka iz baze ne može utvrditi koji je glasač glasao za koju opciju, sa tim da se i dalje zadrži onemogućavanje višestrukog glasanja?

- **Naknadni zahtev 1**

Na izborima na kojima je sistem prvi put korišćen javila se potreba za drugim krugom, odnosno ponovnim korišćenjem sistema. Uočeno je da prethodno razvijeni model baze podataka to ne omogućava u okviru iste baze podataka, te je kao privremeno rešenje kreirana nova. Međutim, kao trajnije rešenje zahtevana je izmena modela u cilju omogućavanja korišćenja iste baze za višestruke izbore.

- **Model baze podataka**



- **Pitanja**

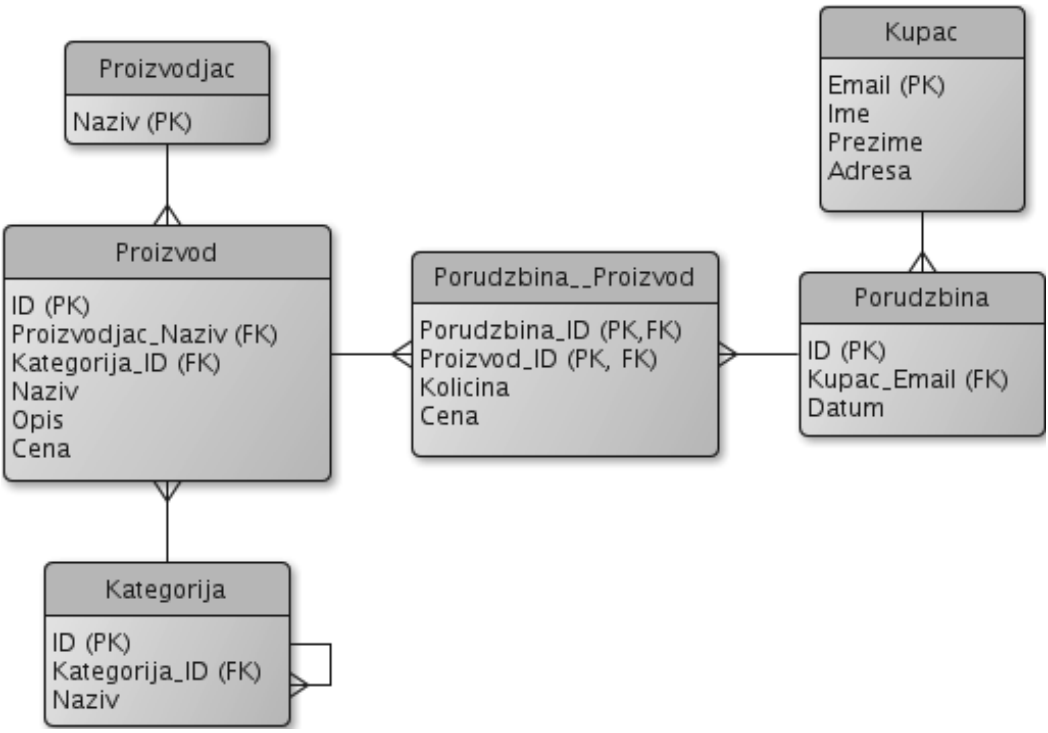
  - Kako će se navedene izmene odraziti na složenost, pouzdanost i performanse sistema?
  - Kako će se navedene izmene odraziti na redne brojeve opcija na narednim izborima? Koja su potencijalna rešenja za to?

Da li se razvijeni model može koristiti i za potrebe anketiranja građana u cilju predviđanja izbora rezultata i koje su izmene potrebne?

15.7 ELEKTRONSKA PRODAVNICA

Potrebno je razviti model baze podataka koja bi se koristila za elektronsku (Veb) prodavnicu. Baza podataka treba da omogući čuvanje podataka o aktuelnim proizvodima (proizvođač, naziv, opis, cena, kategorija) i njihovo poručivanje putem Interneta. Proizvodi se mogu pretraživati preko proizvođača i kategorije u kojoj se nalaze. Kategorije su organizovane hijerarhijski (npr. Alati > Bušilice > Stone bušilice). U okviru jedne porudžbine može se naći više proizvoda. Kupci su obavezni da se registruju i pri tom ostave adresu elektronske pošte, ime, prezime i adresu.

- **Model baze podataka**



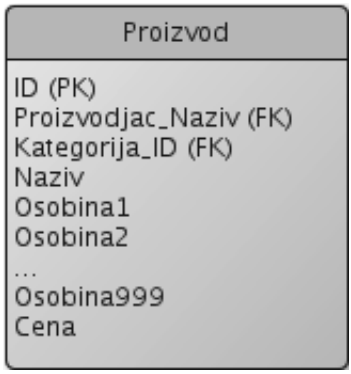
- **Pitanja**
  - Da li je adresa elektronske pošte dobar kandidat za primarni ključ tabele *Kupac*?
  - Zašto je potrebno čuvati cenu proizvoda u okviru tabele *Porudzbina__Proizvod* kada taj podatak već postoji u tabeli *Proizvod*?
  - Koju vrednost za nad-kategoriju (polje *Kategorija_ID*) će imati korene kategorije?
  - Kakvi algoritmi su potrebni za pretraživanje hijerarhijski organizovanih podataka kakve su u ovom slučaju kategorije?

- **Naknadni zahtev 1**

Nakon korišćenja razvijenog modela u početnom periodu rada elektronske prodavnice uočeno je da proizvode nije dovoljno pretraživati po proizvođaču, kategoriji, nazivu i grubom opisu, već da je potrebna preciznija pretraga po njihovim osobinama. U skladu sa tim, potrebno je osobine izdvojiti iz opisa. Pri tom, treba imati u vidu da različiti proizvodi imaju potpuno različite osobine - na primer, kod košulja je značajna veličina, boja i sirovinski sastav, dok su ključne osobine frižidera kapacitet, energetska efikasnost i mogućnost samo-otapanja.

- **Model baze podataka 1**

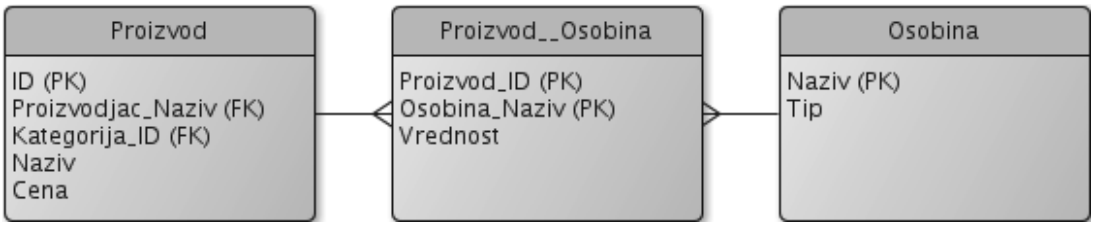
S obzirom na veliku raznolikost osobina različitih proizvoda predložen je model koji primenjuje tzv. „retku matricu“, odnosno koji sadrži tabelu u kojoj će većina polja kod većine redove biti prazna:



- **Pitanja**
  - Da li je predloženo rešenje optimalno? Koji su njegovi nedostaci?
  - Na koji način se vrši proširivanje skupa mogućih osobina? Da li su kod proširivanja potrebne izmene Veb aplikacije? Da li je moguće praviti dinamičke formulare za pretragu proizvoda po osobinama?

- **Model baze podataka 2**

Osnovni model baze podataka je proširen EAV (Entitet-Atribut-Vrednost) modelom za proizvod koji omogućava da se svakom pojedinačnom proizvodu pridruži proizvoljan podskup definisanih osobina sa sopstvenim vrednostima:



- **Pitanja**
  - Koji su algoritamski koraci za pravljenje dinamičkog formulara za pretragu proizvoda?
  - Na koje sve načine je moguće sortiranje osobina, osim po azbučnom/abecednom sortiranju po nazivu? Koje su izmene potrebne?
  - Koji su nedostaci pomenutog modela? Kakav je njegov uticaj na performanse?

LITERATURA

Blagojević Vladimir Relacione baze podataka [Knjiga]. - Beograd : Klub Nikola Tesla, 2001.

Branisalv Lazarević, Z.Marjanović, N. Aničić, S. Babarogić Baze podataka [Knjiga]. - Beograd : FON, 2008.

Carlos Coronel, Steven Morris, and Peter Rob Database Systems: Design, Implementation and Management [Knjiga]. - Boston : Joe Sabatino, 2011. - T. 9th Edition.

David M. Kroenke, David J. Auer Database Concepts [Knjiga]. - New Jersey : Pearson Prentice Hall, 2008. - T. Third Edition.

DuBois Paul MySQL [Knjiga]. - New York : Addison Wesley, 2013. - T. 5th Edition.

Hector Garsia-Molina, Jeffrey D. Ullman, Jennifer Widom Database Systems - the Complete Book [Knjiga]. - New Jersey : Prentice Hall, 2008.

Jeffrey A. Hoffer, Mary B. Prescott, Heikki Topi Modern Database Management [Knjiga]. - New Jersey : Pearson Prentice Hall, 2009. - T. Ninth Edition.

Jeffrez D. Ullamn, Jennifer Widom A First Course in Database Systems [Knjiga]. - New Jersey : Pearson Prentice Hall, 2008. - T. Third Edition.

L.Johnson James Database: Models, Languages, Design [Knjiga]. - London : Oxford University Press, 1997.

Michael Kifer, Arthur Bernstein, Philip M. Lewis Database Systems: An Application-Oriented Approach, Introductory Version [Knjiga]. - New York : Pearson, Addison Wesley, 2004. - T. 2nd Edition.

Murach Joel MySQL [Knjiga]. - US : Mike Murach & Associates Inc., 2012.

Powell Gavin Beginning Database Design [Knjiga]. - Indianapolis : Wiley Publishing, 2005.

Silberschatz A., Korth H., Sudarshan S., Database Systems Concepts [Knjiga], Mc Graw Hill, 2011.

S.Mullinas Craig Admninistracija baza podataka [Knjiga]. - Čačak : Kompjuter biblioteka , 2003.

Toby Teorey, Sam Lightstone, Tom Nadeau, H. V. Jagadish Database Modeling [Knjiga]. - New York : Elsevier, 2011.

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд

004.65(075.8)

BAZE podataka / Mladen Veinović ... [et al.]. - 4. izd. - Beograd : Univerzitet Singidunum, 2025 (Beograd : Caligraph). - XIII, 307 str. : ilustr. ; 24 cm

Tiraž 2.500. - Bibliografija: str. 307.

ISBN 978-86-7912-684-9

1. Веиновић, Младен, 1962- [autor]

a) Базе података

COBISS.SR-ID 173148425

© 2025.

Sva prava zadržana. Nijedan deo ove publikacije ne može biti reprodukovan u bilo kom vidu i putem bilo kog medija, u delovima ili celini bez prethodne pismene saglasnosti izdavača.

www.singidunum.ac.rs



Baze podataka

Mladen Veinović
Goran Šimić
Aleksandar Jevremović
Milan Tair

U ovom udžbeniku, predstavljen je uvod u relacionu algebru, modelovanje, korišćenje upitnog jezika SQL, kao i rad sa NoSQL bazama podataka. Pretežno je razmatran relacioni model, koji je osnova za najveći broj poslovnih aplikacija. Na početku su izneti osnovni koncepti, definicije i relaciona algebra kao osnova upitnih jezika. Zatim su prikazane mogućnosti SQL upitnog jezika na primeru MySQL sistema za upravlja- nje relacionim bazama podataka. Posebna pažnja je posvećena prikazu planiranja i projektovanja modela na konkretnom primeru, koji počinje analizom projektnog zahteva, preko određivanja entiteta, atributa i njihovih relacija, do kreiranja gotovog modela. Posebna poglavlja obrađuju transakcije u bazama podataka, postupak normalizacije modela, kao i rad sa transportnim formatima podataka, poput XML-a. Na kraju, izložen je teorijski deo oko skladištenja podataka i dati su karakteristični primeri za modelovanje i programiranje, poređani od lakših ka težim problemima.

Univerzitet Singidunum