Sveučilište u Zagrebu
Prirodoslovno Matematički fakultet
Matematički odsjek

Robert Manger

BAZE PODATAKA

Skripta

Drugo izdanje
Zagreb, veljača 2011.

Sadržaj

Predgovor

Ova skripta sadrže predavanja iz predmeta „Baze podataka“ koji se predaje studentima matematike na PMF-Matematičkom odsjeku Sveučilišta u Zagrebu. Tekst pokriva sve važne teme koje se obično susreću u udžbenicima o relacijskim bazama podataka, na primjer govori o relacijskom modelu i o jeziku SQL. Ipak, naglasak je stavljen na postupak oblikovanja (projektiranja) baze.

Baza podataka može se pojaviti kao sastavni dio neke određene aplikacije, ili ona može predstavljati samostalni resurs i davati podršku raznim aplikacijama. Čak i u prvom, a pogotovo u drugom slučaju, baza nastaje kao rezultat zasebnog razvojnog postupka koji je u većoj ili manjoj mjeri odvojen od razvoja samih aplikacija.

Oblikovanje baze podataka predstavlja ključni dio njezinog razvoja. Cilj oblikovanja je da se na osnovu utvrđenih potreba za podacima odabere pogodna građa baze. Ta građa u pravilu ne bi smjela biti optimizirana za jednu određenu aplikaciju, već bi trebala odražavati smisao i unutarnju povezanost samih podataka. Na taj način, baza bi dugoročno trebala biti pogodna za promjene i evoluciju u skladu sa zahtjevima budućih aplikacija.

Uobičajeni postupak oblikovanja baze podataka sastoji se od tri faze: to su konceptualno, logičko, odnosno fizičko oblikovanje. U prvoj fazi nastaje konceptualna shema sastavljena od entiteta, atributa i veza; ona zorno opisuje podatke ali još nije pogodna za implementaciju. Druga faza stvara takozvanu logičku ili relacijsku shemu sastavljenu od relacija (tablica), koja je usklađena s mogućnostima uobičajenih softverskih paketa za upravljanje relacijskim bazama podataka. Sastavni dio druge faze je takozvana normalizacija, gdje se logička struktura relacija popravlja tako da bolje izrazi unutrašnje osobine samih podataka. Treća faza daje kao rezultat naredbe u jeziku SQL kojima se realizira potrebna fizička građa baze zajedno s pomoćnim strukturama za pretraživanje, te čuvanje integriteta odnosno sigurnosti podataka.

Ova skripta sastoje se od 7 poglavlja. Poglavlje 1 sadrži uvod u baze podataka: tu se definiraju osnovni pojmovi, te se daje pregled razvojnog ciklusa baze odnosno načina njezinog dokumentiranja. Poglavlje 2 obrađuje konceptualno oblikovanje baze podataka, dakle izradu odgovarajuće sheme entiteta, atributa i veza. Poglavlje 3 objašnjava relacijski model za bazu podataka, te pokriva prvi dio logičkog oblikovanja koji rezultira polaznom relacijskom shemom. U Poglavlju 4 bavimo se nastavkom logičkog oblikovanja, dakle normalizacijom relacijske sheme dobivene metodama iz Poglavlja 3. Poglavlje 5 predstavlja malu digresiju u zamišljenom postupku oblikovanja: tu naime govorimo o postavljanju upita u relacijskim bazama podataka pomoću jezika kao što su relacijska algebra, relacijski račun, odnosno upitni dio SQL-a. Poglavlje 6 odnosi se na fizičko oblikovanje baze podataka u jeziku SQL, no ujedno objašnjava i kako je baza zaista fizički građena, te kako se nad njom obavljaju uobičajene operacije poput izvrednjavanja upita. Zadnje Poglavlje 7 govori o čuvanju integriteta i sigurnosti baze podataka, bilo u fazi njezina oblikovanja bilo tijekom njezine kasnije uporabe.

Ova skripta bogato su opskrbljena primjerima i zadacima koji prate i ilustriraju obrađeno gradivo. Kroz većinu poglavlja provlači se jedan odabrani studijski primjer: baza podataka o fakultetu. Također, u prilogu se nalazi cjelovita dokumentacija za dodatna dva studijska primjera. Unutar svakog poglavlja, zadnje potpoglavlje sadrži zadatke za vježbu. Neki od tih zadataka su samostalni, neki se odnose na studijske primjere, a neki na bazu podataka iz studentovog područja interesa.

Makar ova skripta u potpunosti pokrivaju predavanja iz predmeta „Baze podataka“ na PMF-Matematičkom odsjeku, ona za sada samo djelomično pokrivaju vježbe iz istog predmeta. Naime, od vježbi se očekuje da one, osim praćenja tema s predavanja, puno detaljnije obrade jezik SQL, te pruže studentima mogućnost stvarnog rada s tim jezikom na računalu.

1. Uvod u baze podataka

U ovom predmetu bavimo se problematikom trajnog pohranjivanja većih količina podataka u vanjskoj memoriji računala. To je izuzetno važna problematika: naime unatoč svom nazivu, računala zapravo rijetko služe za *računanje*, a znatno češće za *spremanje i pretraživanje* podataka.

Mogućnost trajnog pohranjivanja podataka u računalima postoji gotovo jednako dugo koliko i sama računala, te je podržana u svim programskim jezicima. Na primjer, program razvijen u jeziku COBOL ili C može stvoriti datoteku na disku i u nju upisati podatke, ili otvoriti postojeću datoteku i iz nje pročitati podatke. Služeći se klasičnim programskim jezicima u principu je moguće stvoriti trajne kolekcije podataka na disku i izgraditi aplikacije koje rabe takve podatke.

Ipak, kad govorimo o bazama podataka, tada mislimo na višu razinu rada s podacima od one koju podržavaju klasični programski jezici. Ustvari mislimo na tehnologiju koja je nastala s namjerom da ukloni slabosti tradicionalne „automatske obrade podataka“ iz 60-tih i 70-tih godina 20. stoljeća. Ta tehnologija osigurala je veću produktivnost, kvalitetu i pouzdanost u razvoju aplikacija koje se temelje na pohranjivanju i pretraživanju podataka u računalu.

1.1. Osnovni pojmovi

Osnovna ideja tehnologije baza podataka je u tome da pojedina aplikacija ne stvara svoje vlastite datoteke na disku. Umjesto toga, sve aplikacije rabe zajedničku i objedinjenu kolekciju podataka. Također, aplikacija ne pristupa izravno podacima na disku. Umjesto toga, ona barata s podacima na posredan način, služeći se uslugama specijaliziranog softvera koji je zadužen da se brine za zajedničku kolekciju. Spomenuta zajednička kolekcija podataka naziva se baza podataka, a specijalizirani softver koji posreduje između aplikacija i podataka naziva se sustav za upravljanje bazom podataka. U nastavku ćemo najprije pokušati preciznije definirati ova dva ključna pojma, a zatim ćemo objasniti i druge pojmove koji su u vezi s njima.

1.1.1. Baza podataka i sustav za upravljanje bazom podataka (DBMS)

Baza podataka je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim programima. Ubacivanje, promjena, brisanje i čitanje podataka obavlja se posredstvom posebnog softvera, takozvanog *sustava za upravljanje bazom podataka* (DBMS-a). Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka, već se referenciraju na neku idealiziranu logičku strukturu baze.

Sustav za upravljanje bazom podataka (*Data Base Management System* - DBMS) je poslužitelj (server) baze podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom. Također, on obavlja u ime klijenata sve operacije s podacima. Dalje, on je u stanju podržati razne baze, od kojih svaka može imati svoju logičku strukturu, no u skladu s istim *modelom*. Isto tako, brine se za sigurnost podataka, te automatizira administrativne poslove s bazom.

Slično kao i operacijski sustav, DBMS spada u temeljni softver kojeg većina korisnika i organizacija ne razvija samostalno već ga kupuju zajedno s računalom. Danas postoji svega nekoliko važnih i široko zastupljenih DBMS-a.

- **DB2.** Proizvod tvrtke IBM, namijenjen prvenstveno velikim *mainframe* računalima.
- **Oracle.** Proizvod istoimene tvrtke, pokriva gotovo sve računalne platforme, na primjer UNIX, Linux i MS Windows.
- **MS SQL Server.** Microsoftov proizvod, namijenjen poslužiteljskim računalima s operacijskim sustavima MS Windows.
- **MySQL.** Besplatni proizvod tvrtke MySQL AB, popularan na raznim platformama, prvenstveno kao podrška web aplikacijama.

Svi ovi proizvodi uz sam DBMS uključuju u sebi i dodatne alate za razvoj aplikacija, administriranje baze i slično.

1.1.2. Modeli za logičku strukturu baze podataka

Model podataka je skup pravila koja određuju kako sve može izgledati logička struktura baze podataka. Model čini osnovu za oblikovanje i implementiranje baze. Točnije rečeno, podaci u bazi moraju biti logički organizirani u skladu s onim modelom kojeg podržava odabrani DBMS.

Dosadašnji DBMS-i obično su podržavali neki od sljedećih modela:

- **Relacijski model.** Zasnovan je na matematičkom pojmu *relacije*. I podaci i veze među podacima prikazuju se tablicama koje se sastoje od redaka i stupaca.
- **Mrežni model.** Baza je predočena mrežom koja se sastoji od *čvorova* i usmjerenih *lukova*. Čvorovi predstavljaju tipove zapisa (slogova podataka), a lukovi definiraju veze među tipovima zapisa.
- **Hijerarhijski model.** Specijalni slučaj mrežnog. Baza je predočena jednim *stablom* (hijerarhijom) ili skupom stabala. Svako stablo sastoji se od čvorova i veza „nadređeni-podređeni“ između čvorova. Čvorovi su tipovi zapisa, a odnos „nadređeni-podređeni" izražava hijerarhijske veze među tipovima zapisa.
- **Objektni model.** Inspiriran je objektno-orijentiranim programskim jezicima. Baza je predočena kao skup trajno pohranjenih *objekata* koji se sastoje od svojih internih „atributa“ (podataka) i „metoda“ (operacija) za rukovanje tim podacima. Svaki objekt pripada nekoj klasi. Između klasa se uspostavljaju veze nasljeđivanja, agregacije, te druge vrste veza.

Hijerarhijski i mrežni model bili su u uporabi u 60-tim i 70-tim godinama 20. stoljeća. Od 80-tih godina pa sve do današnjih dana prevladava relacijski model. Očekivani prijelaz na objektni model za sada se nije desio, tako da današnje baze podataka uglavnom još uvijek možemo poistovjetiti s relacijskim bazama. Svi prije spomenuti poznati DBMS-i koji su danas u širokoj uporabi podržavaju isključivo relacijski model.

1.1.3. Ciljevi koji se nastoje postići uporabom baza podataka

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

- **Fizička nezavisnost podataka.** Razdvaja se logička definicija baze od njezine stvarne fizičke građe. Znači, ako se fizička građa promijeni (na primjer, podaci se prepišu u druge datoteke na drugim diskovima), to neće zahtijevati promjene u postojećim aplikacijama.
- **Logička nezavisnost podataka.** Razdvaja se globalna logička definicija cijele baze podataka od lokalne logičke definicije za jednu aplikaciju. Znači, ako se globalna logička definicija promijeni (na primjer uvede se novi zapis ili veza), to neće zahtijevati promjene u postojećim aplikacijama. Lokalna logička definicija obično se svodi na izdvajanje samo nekih elemenata iz globalne definicije, uz neke jednostavne transformacije tih elemenata.
- **Fleksibilnost pristupa podacima.** U starijim mrežnim i hijerarhijskim bazama, načini pristupanja podacima bili su unaprijed definirani, dakle korisnik je mogao pretraživati podatke jedino onim redoslijedom koji je bio predviđen u vrijeme oblikovanja i implementiranja baze. Danas se podrazumijeva da korisnik može slobodno prebirati po podacima, te po svom nahođenju uspostavljati veze među podacima. Ovom zahtjevu zaista zadovoljavaju jedino relacijske baze.
- **Istovremeni pristup do podataka.** Baza mora omogućiti da veći broj korisnika istovremeno rabe iste podatke. Pritom ti korisnici ne smiju ometati jedan drugoga, te svaki od njih treba imati dojam da sam radi s bazom.
- **Čuvanje integriteta.** Nastoji se automatski sačuvati korektnost i konzistencija podataka, i to u situaciji kad postoje greške u aplikacijama, te konfliktne istovremene aktivnosti korisnika.
- **Mogućnost oporavka nakon kvara.** Mora postojati pouzdana zaštita baze u slučaju kvara hardvera ili grešaka u radu sistemskog softvera.
- **Zaštita od neovlaštene uporabe.** Mora postojati mogućnost da se korisnicima ograniče prava uporabe baze, dakle da se svakom korisniku reguliraju ovlaštenja što on smije a što ne smije raditi s podacima.
- **Zadovoljavajuća brzina pristupa.** Operacije s podacima moraju se odvijati dovoljno brzo, u skladu s potrebama određene aplikacije. Na brzinu pristupa može se utjecati odabirom pogodnih fizičkih struktura podataka, te izborom pogodnih algoritama za pretraživanje.
- **Mogućnost podešavanja i kontrole.** Velika baza zahtijeva stalnu brigu: praćenje performansi, mijenjanje parametara u fizičkoj građi, rutinsko pohranjivanje rezervnih kopija podataka, reguliranje ovlaštenja korisnika. Također, svrha baze se vremenom mijenja, pa povremeno treba podesiti i logičku strukturu. Ovakvi poslovi moraju se obavljati centralizirano. Odgovorna osoba zove se *administrator* baze podataka. Administratoru trebaju stajati na raspolaganju razni alati i pomagala.

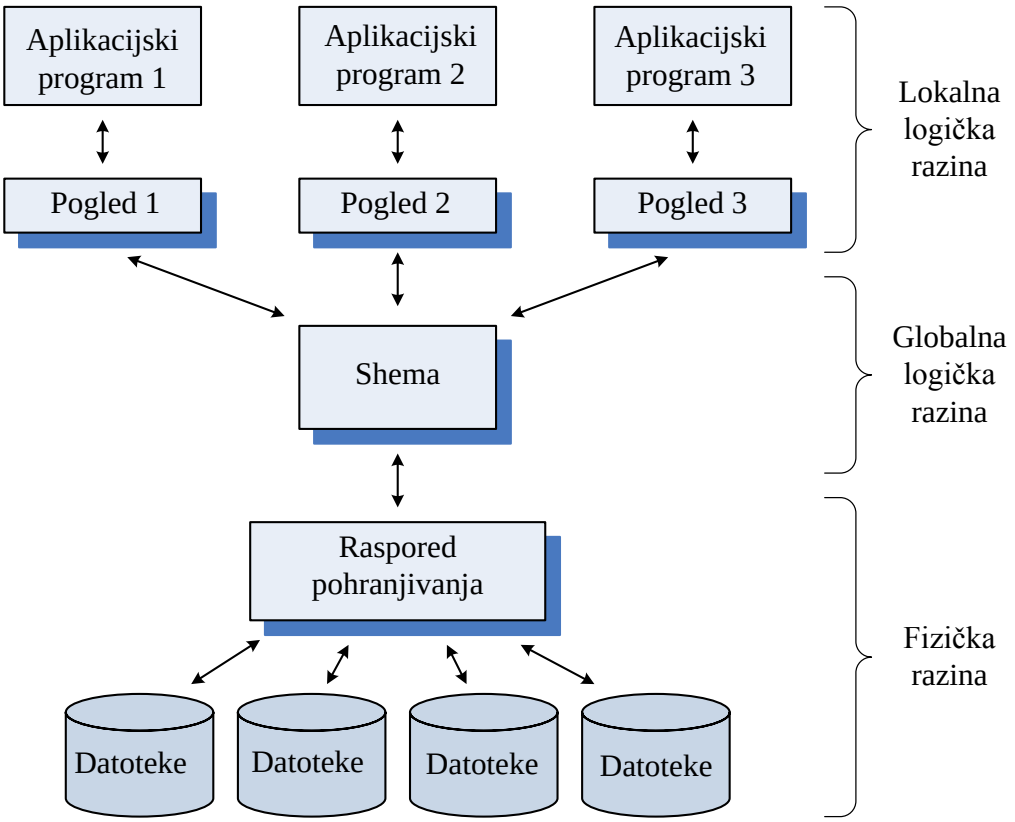
1.1.4. Arhitektura baze podataka

Arhitektura baze podataka sastoji se od tri „sloja“ i sučelja među slojevima, kao što je prikazano na Slici 1.1. Riječ je o tri razine apstrakcije.

- **Fizička razina** odnosi se na fizički prikaz i raspored podataka na jedinicama vanjske memorije. To je aspekt kojeg vide samo sistemski programeri (oni koji su razvili DBMS). Sama fizička razina može se dalje podijeliti na više pod-razina apstrakcije, od sasvim konkretnih staza i cilindara na disku, do već donekle apstraktnih pojmova datoteke i zapisa (sloga) kakve susrećemo u klasičnim programskim jezicima. *Raspored pohranjivanja* opisuje kako se elementi logičke definicije baze preslikavaju na fizičke uređaje.
- **Globalna logička razina** odnosi se na logičku strukturu cijele baze. To je aspekt kojeg vidi projektant baze odnosno njezin administrator. Opis globalne logičke definicije

naziva se *shema* (engleski također *schema*). Shema je tekst ili dijagram koji definira logičku strukturu baze, i u skladu je sa zadanim modelom. Dakle imenuju se i definiraju svi tipovi podataka i veze među tim tipovima, u skladu s pravilima rabljenog modela. Također, shema može uvesti i ograničenja kojim se čuva integritet podataka.

- **Lokalna logička razina** odnosi se na logičku predodžbu o dijelu baze kojeg rabi pojedina aplikacija. To je aspekt kojeg vidi korisnik ili aplikacijski programer. Opis jedne lokalne logičke definicije zove se *pogled* (engleski *view*) ili *pod-shema*. To je tekst ili dijagram kojim se imenuju i definiraju svi lokalni tipovi podataka i veze među tim tipovima, opet u skladu s pravilima rabljenog modela. Također, pogled u svojoj konačnoj realizaciji zadaje i način na koji se iz globalnih podataka i veza izvode lokalni.



Slika 1.1: Arhitektura baze podataka.

Primijetimo da se fizička neovisnost podataka spomenuta u Odjeljku 1.1.3 ustvari postiže time što se pravi razlika između fizičke i globalne logičke razine, dok se logička neovisnost postiže razlikovanjem lokalne logičke razine i globalne logičke razine. Na taj način, opisana troslojna arhitektura omogućuje ispunjavanje dvaju najvažnijih ciljeva koji se nastoje postići uporabom baza podataka.

Za stvaranje baze podataka potrebno je zadati samo shemu i poglede. DBMS tada automatski generira potrebni raspored pohranjivanja i fizičku bazu. Projektant odnosno administrator može samo donekle utjecati na fizičku građu baze, podešavanjem njemu dostupnih parametara.

Programi i korisnici ne pristupaju izravno fizičkoj bazi, već dobivaju ili pohranjuju podatke posredstvom DBMS-a. Komunikacija programa odnosno korisnika s DBMS-om obavlja se na lokalnoj logičkoj razini. To znači da DBMS na transparentan način prevodi korisničke zahtjeve za podacima s lokalne logičke razine na globalnu logičku razinu, a zatim ih dalje realizira kao ekvivalentne operacije na fizičkoj razini.

1.1.5. Jezici za rad s bazama podataka

Komunikacija korisnika odnosno aplikacijskog programa i DBMS-a odvija se pomoću posebnih jezika. Ti jezici tradicionalno se dijele na sljedeće kategorije.

- **Jezik za opis podataka** (*Data Description Language* - DDL). Služi projektantu baze ili administratoru u svrhu zapisivanja sheme ili pogleda. Dakle tim jezikom definiramo podatke i veze među podacima, i to na logičkoj razini. Naredbe DDL obično podsjećaju na naredbe za definiranje složenih tipova podataka u jezicima poput COBOL-a ili C-a.
- **Jezik za manipuliranje podacima** (*Data Manipulation Language* - DML). Služi programeru za uspostavljanje veze između aplikacijskog programa i baze. Naredbe DML omogućuju „manevriranje“ po bazi, te jednostavne operacije kao što su upis, promjena, brisanje ili čitanje zapisa. U nekim softverskim paketima, DML je zapravo biblioteka potprograma: „naredba“ u DML svodi se na poziv potprograma. U drugim paketima zaista se radi o posebnom jeziku: programer tada piše program u kojem su izmiješane naredbe dvaju jezika, pa takav program treba prevoditi s dva prevodioca (DML-precompiler, obični compiler).
- **Jezik za postavljanje upita** (*Query Language* - QL). Služi neposrednom korisniku za interaktivno pretraživanje baze. To je jezik koji podsjeća na govorni (engleski) jezik. Naredbe su ne-proceduralne, dakle takve da samo specificiraju rezultat kojeg želimo dobiti, a ne i postupak za dobivanje rezultata.

Ovakva podjela na tri jezika danas je već prilično zastarjela. Naime, kod relacijskih baza postoji tendencija da se sva tri jezika objedine u jedan sveobuhvatni. Primjer takvog *integriranog* jezika za relacijske baze je SQL: on služi za definiranje podataka, manipuliranje i pretraživanje. Integrirani jezik se može rabiti interaktivno (preko on-line interpretera) ili se on može pojavljivati uklopljen u aplikacijske programe. Svi DBMS-i spomenuti u 1.1.1 koji su danas u širokoj uporabi koriste se isključivo SQL-om za sve tri svrhe.

Naglasimo da gore spomenuti jezici DDL, DML i QL nisu programski jezici. Dakle ti jezici su nam nužni da bismo stvorili bazu i povezali se s njom, no oni nam nisu dovoljni za razvoj aplikacija koje će nešto raditi s podacima iz baze.

Tradicionalni način razvoja aplikacija koje rade s bazom je uporaba klasičnih programskih jezika (COBOL, C, …) s ugniježđenim DML-naredbama. U 80-tim godinama 20. stoljeća bili su dosta popularni i takozvani *jezici 4. generacije* (*4-th Generation Languages* - 4GL): riječ je o jezicima koji su bili namijenjeni isključivo za rad s bazama, te su zato u tom kontekstu bili produktivniji od programskih jezika opće namjene. Problem s jezicima 4. generacije je bio u njihovoj nestandardnosti: svaki od njih je u pravilu bio dio nekog određenog softverskog paketa za baze podataka, te se nije mogao rabiti izvan tog paketa ili baze.

U današnje vrijeme, aplikacije se najčešće razvijaju u standardnim *objektno orijentiranim* programskim jezicima (Java, C++, C#, …). Za interakcije s bazom rabe se unaprijed pripremljene klase objekata. Ovakva tehnika je dovoljno produktivna zbog uporabe gotovih klasa, a razvijeni program se lako dotjeruje, uklapa u veće sustave ili prenosi s jedne baze na drugu.

1.2. Razvojni ciklus baze

Uvođenje baze podataka u neku ustanovu predstavlja složeni zadatak koji zahtijeva primjenu pogodnih metoda i alata, te timski rad stručnjaka raznih profila. To je projekt koji se može podijeliti u pet aktivnosti: utvrđivanje i analiza zahtjeva, oblikovanje (projektiranje), implementacija, testiranje i održavanje. Riječ je o razvojnom ciklusu koji je dobro poznat u softverskom inženjerstvu i koji se u sličnom obliku pojavljuje kod razvoja bilo koje vrste softverskih produkata. No u slučaju baza podataka taj ciklus ima neke svoje specifičnosti. Od navedenih pet aktivnosti, nas u ovim skriptama najviše zanima oblikovanje. Ipak, zbog cjelovitosti izlaganja, u ovom potpoglavlju ukratko ćemo opisati sve aktivnosti.

1.2.1. Utvrđivanje i analiza zahtjeva

Da bi se utvrdili zahtjevi, proučavaju se tokovi informacija u dotičnoj ustanovi. Dakle gledaju se dokumenti koji su u opticaju, prate se radni procesi, razgovara se s korisnicima, proučava se postojeći softver. Uočavaju se *podaci* koje treba pohranjivati i veze među njima.

U velikim organizacijama, gdje postoje razne grupe korisnika, pojavit će se razna tumačenja značenja i svrhe pojedinih podataka, te razni načini njihove uporabe. Analiza zahtjeva treba pomiriti te razlike, tako da se eliminira redundancija i nekonzistentnost. Na primjer, u raznim nazivima podataka treba prepoznati sinonime i homonime, te uskladiti terminologiju.

Analiza zahtjeva također mora obuhvatiti analizu *transakcija* (postupaka, operacija) koje će se obavljati s podacima, budući da to redovito ima utjecaja na sadržaj i konačni oblik baze. Važno je procijeniti frekvenciju i opseg pojedinih transakcija, te zahtjeve na performanse.

Rezultat utvrđivanja i analize zahtjeva je dokument (obično pisan neformalno u prirodnom jeziku) koji se zove *specifikacija*. Taj dokument rijetko se odnosi samo na podatke, on ujedno definira i najvažnije transakcije s podacima, a često i cijele aplikacije.

1.2.2. Oblikovanje (konceptualno, logičko i fizičko)

Cilj oblikovanja je da se u skladu sa specifikacijom oblikuje građa baze. Dok je analiza zahtjeva otprilike odredila koje vrste podataka baza treba sadržavati i što se s njima treba moći raditi, oblikovanje predlaže način kako da se podaci na pogodan način grupiraju, strukturiraju i međusobno povežu. Glavni rezultat oblikovanja trebala bi biti shema cijele baze, građena u skladu s pravilima rabljenog modela podataka, te zapisana na način da ju rabljeni DBMS može razumjeti i realizirati. Kod većih baza, rezultat oblikovanja mogu također biti i pogledi (pod-sheme) za potrebe pojedinih važnijih aplikacija.

Budući da je oblikovanje prilično složena aktivnost, ona se obično dijeli u tri faze koje slijede jedna iza druge i koje ćemo sad ukratko opisati.

- **Konceptualno oblikovanje.** Glavni rezultat prve faze oblikovanja je takozvana *konceptualna shema* cijele baze, sastavljena od entiteta, atributa i veza. Ona zorno opisuje sadržaj baze i načine povezivanja podataka u njoj. Prikaz je jezgrovit, neformalan i pogodan ljudima za razumijevanje, no još je nedovoljno razrađen da bi omogućio izravnu implementaciju.
- **Logičko oblikovanje.** Kao glavni rezultat druge faze oblikovanja nastaje *logička shema*, koja je u slučaju relacijskog modela sastavljena od relacija (tablica). Sastavni dio logičkog oblikovanja je i takozvana *normalizacija*, gdje se primjenom posebnih pravila nastoji popraviti logička struktura samih relacija, tako da se ona bolje prilagodi inherentnim osobinama samih podataka.
- **Fizičko oblikovanje.** Glavni rezultat treće faze oblikovanja je *fizička shema* cijele baze, dakle opis njezine fizičke građe. U slučaju uporabe DBMS-a zasnovanog na jeziku SQL, pojam fizičke razine treba shvatiti uvjetno. Fizička shema zapravo je niz SQL naredbi kojima se relacije iz logičke sheme realiziraju kao SQL tablice. Pritom se dodaju pomoćne strukture i mehanizmi za postizavanje traženih performansi, te čuvanje integriteta i sigurnosti podataka. Također se mogu uključiti i SQL naredbe kojima se pogledi (pod-sheme) za pojedine aplikacije realiziraju kao virtualne tablice izvedene iz stvarnih tablica.

Navedene tri faze oblikovanja detaljno ćemo obraditi u Poglavlju 2, odnosno Poglavljima 3 i 4, odnosno Poglavlju 6 ovog priručnika . Svi rezultati oblikovanja opisuju se u odgovarajućim dokumentima koji zajedno čine *projektnu dokumentaciju* baze. O toj dokumentaciji opširnije ćemo govoriti u Potpoglavlju 1.3.

1.2.3. Implementacija

Implementacija se svodi na fizičku realizaciju oblikovane baze na odgovarajućem poslužiteljskom računalu. U slučaju DBMS-a zasnovanog na SQL-u, pokreću se SQL naredbe koje čine fizičku shemu baze, te se na disku stvaraju prazne SQL tablice sa svim pratećim strukturama i mehanizmima.

Daljnji postupak sastoji se od punjenja praznih tablica s početnim podacima. Takvi podaci obično postoje u nekom obliku, na primjer kao obične datoteke, ili kao tekstualni dokumenti. Većina DBMS-a opskrbljena je alatima koji nam olakšavaju transfer podataka iz takvih izvora u bazu. No postupak je obično ipak mukotrpan i ne da se sasvim automatizirati, zbog potrebe čišćenja, ispravljanja i usklađivanja podataka.

Nakon što su početni podaci uneseni u bazu, razvijaju se aplikacije koje obavljaju najvažnije transakcije s podacima. Time je omogućeno testiranje.

1.2.4. Testiranje

Testiranje baze provodi se tako da korisnici pokusno rade s bazom i provjeravaju zadovoljava li ona svim zahtjevima. Dakle pokreću se najvažnije transakcije s podacima, prati se njihov učinak, te se mjere performanse. Također se nastoji simulirati očekivana frekvencija pojedinih transakcija, da bi se utvrdila stabilnost i pouzdanost rada pod opterećenjem.

Glavni cilj testiranja je otkrivanje i popravak grešaka koje su se mogle potkrasti u svakoj od prethodnih aktivnosti: dakle u analizi zahtjeva, oblikovanju, odnosno implementaciji. Greške u ranijim aktivnostima imaju teže posljedice jer se provlače i kroz kasnije aktivnosti pa

zahtijevaju više truda da se poprave. Na primjer, greška u analizi može uzrokovati da u specifikaciji nedostaje neki važni podatak, a to onda znači da tog podatka neće biti ni projektnoj dokumentaciji ni u implementiranoj bazi, pa popravak treba izvršiti u svim dokumentima i shemama, te u samoj bazi.

Mjerenjem performansi tijekom testiranja nastoji se na utvrditi jesu li zadovoljeni zahtjevi vezani uz performanse, na primjer je li brzina odziva zadovoljavajuća. U slučaju da performanse nisu dovoljno dobre, administrator baze može to pokušati ispraviti podešavanjem određenih parametara fizičke organizacije, na primjer dodavanjem novih pomoćnih struktura podataka (indeksa) ili raspoređivanjem podataka na više diskova. Ipak, loše performanse mogu biti i posljedica grešaka u oblikovanju, na primjer posljedica neuočavanja važnih veza između određenih vrsta podataka. U takvom slučaju slijedi opet popravak grešaka, dakle nova revizija shema, dokumenata i same baze.

1.2.5. Održavanje

Održavanje se odvija u vrijeme kad je baza već ušla u redovitu uporabu. Riječ je o kontinuiranom procesu, gdje su baza i njezini prateći dokumenti podvrgnuti stalnim promjenama. Neki autori proces održavanja opisuju možda i primjerenijim pojmom *evolucije*.

Kao i općenito u softverskom inženjerstvu, tako i kod baza podataka možemo govoriti o nekoliko vrsta održavanja, koje se razlikuju po sadržaju i svrsi traženih promjena. *Korekcijsko održavanje* svodi se na naknadni popravak grešaka koje nisu bile otkrivene tijekom testiranja. *Perfekcijsko održavanje* je mijenjanje sheme baze u svrhu prilagođavanja novim aplikacijama koje nisu postojale tijekom polaznog utvrđivanja i analize zahtjeva. *Adaptacijsko održavanje* je potrebno onda kad želimo bazu prilagoditi novom DBMS-u koji se nije rabio u vrijeme oblikovanja i početne implementacije.

Naglasimo da je održavanje baze nužnost na koju se treba pripremiti već tijekom njezinog razvoja i uzimati je u obzir tijekom cijelog njezinog života. Moramo biti svjesni činjenice da baza koja se ne mijenja vrlo brzo postaje neuporabljiva. Da bi promjene tekle što lakše i bezbolnije, izuzetno je važno da baza od početka ima zdravu građu koja odražava inherentnu logiku i povezanost samih podataka. Kod relacijskih baza, ta zdrava građa znači normaliziranost. Ispravno normalizirana relacijska baza moći će se mijenjati bez većih problema, a promjene će se svoditi na povremeno ubacivanje novih podataka u već postojeće relacije ili dodavanje sasvim novih relacija. Pritom te promjene neće utjecati na ispravni rad već postojećih aplikacija, budući da su one zaštićene svojstvima fizičke i logičke nezavisnosti opisanim u Odjeljku 1.1.3.

1.3. Dokumentacija baze

Svaki softverski produkt, pa tako i baza podataka popraćen je odgovarajućom dokumentacijom. Općenito, razlikujemo *korisničku* dokumentaciju namijenjenu korisnicima i *razvojnu* dokumentaciju namijenjenu softverskim inženjerima. Razvojnu dokumentaciju dalje možemo podijeliti na dokumente koji prate pojedine aktivnosti iz razvojnog ciklusa. U ovim skriptama, mi ćemo se ograničiti isključivo na *projektnu* dokumentaciju za baze podataka, dakle na dokumente koji nastaju tijekom aktivnosti oblikovanja baze. Istaknut ćemo važnost takve dokumentacije, prikazati nekoliko predložaka za njezinu izradu, te spomenuti alate koji se pritom obično rabe.

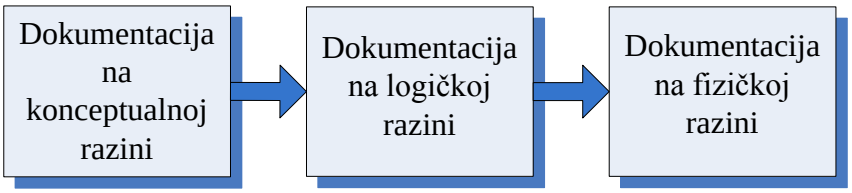
1.3.1. Važnost izrade dokumentacije

Projektna dokumentacija za bazu podataka važna je zato jer ona omogućuje ispravan tijek samog oblikovanja i implementacije u skladu s pravilima struke, te konzistentan prelazak iz pojedine razvojne faze ili aktivnosti u drugu. Također, dokumentacija je neophodna tijekom testiranja i kasnijeg održavanja baze, budući da ona predstavlja jedini relevantni izvor informacija o građi baze. Ne treba zaboraviti da u razvoju i održavanju baze obično sudjeluje veći broj ljudi, te da osobe koje će kasnije mijenjati bazu nisu one iste osobe koje su je stvorile. Projektna dokumentacija predstavlja sponu između svih tih ljudi koji se možda nikada nisu sreli, te njihovu kolektivnu memoriju.

Projektna dokumentacija za bazu podataka prati sve tri faze oblikovanja, i zato se dijeli na tri dijela.

- **Projektna dokumentacija na konceptualnoj razini.** Opisuje konceptualnu shemu baze.
- **Projektna dokumentacija na logičkoj razini.** Dokumentira logičku shemu baze.
- **Projektna dokumentacija na fizičkoj razini.** Sadrži fizičku shemu baze.

Odnos između ta tri dijela prikazan je na Slici 1.2. Strelice na toj slici označavaju da svaki prethodni dokument predstavlja polazište za izradu idućeg dokumenta.



Slika 1.2: Dijelovi projektne dokumentacije.

Završni dio projektne dokumentacije je dokumentacija na fizičkoj razini. Ona se jedina izravno rabi za implementaciju baze. No to ne znači da se dovršetkom fizičke sheme mogu pobrisati prethodni dokumenti, dakle konceptualna i logička shema. Sva tri dijela moraju se čuvati zato jer svaki od njih daje korisnu i komplementarnu informaciju o građi baze. Na primjer, netko tko se tek želi upoznati s bazom lakše će se snaći u konceptualnoj shemi nego u fizičkoj shemi. Također, u slučaju prebacivanja na novi DBMS logička shema može predstavljati bolje polazište nego fizička.

Nakon što se baza implementira i uđe u redovitu uporabu, pripadna projektna dokumentacija mora se čuvati zbog kasnijeg održavanja. Zaista, osoba koja namjerava izvršiti promjenu u bazi prisiljena je rabiti dokumentaciju da bi ustanovila gdje i što treba promijeniti. Kad god dođe do promjene u građi baze, ta promjena mora se unijeti i u dokumentaciju. Štoviše, sva tri dijela dokumentacije moraju se istovremeno mijenjati tako da ostanu međusobno konzistentni i konzistentni s realiziranom bazom. Jedino pod tim uvjetom dokumentacija će ostati relevantna za daljnje održavanje.

Kod manjih baza dovoljno je da projektna dokumentacija u svakom trenutku odražava ažurno stanje. No kod većih i složenijih baza korisno je da se osim ažurnog stanja dokumentira i povijest promjena. To se može realizirati tako da se sami dokumenti čuvaju u više verzija, ili tako da se u jednom dokumentu bilježi tko je i kada izvršio kakvu promjenu.

1.3.2. Predlošci za izradu dokumentacije

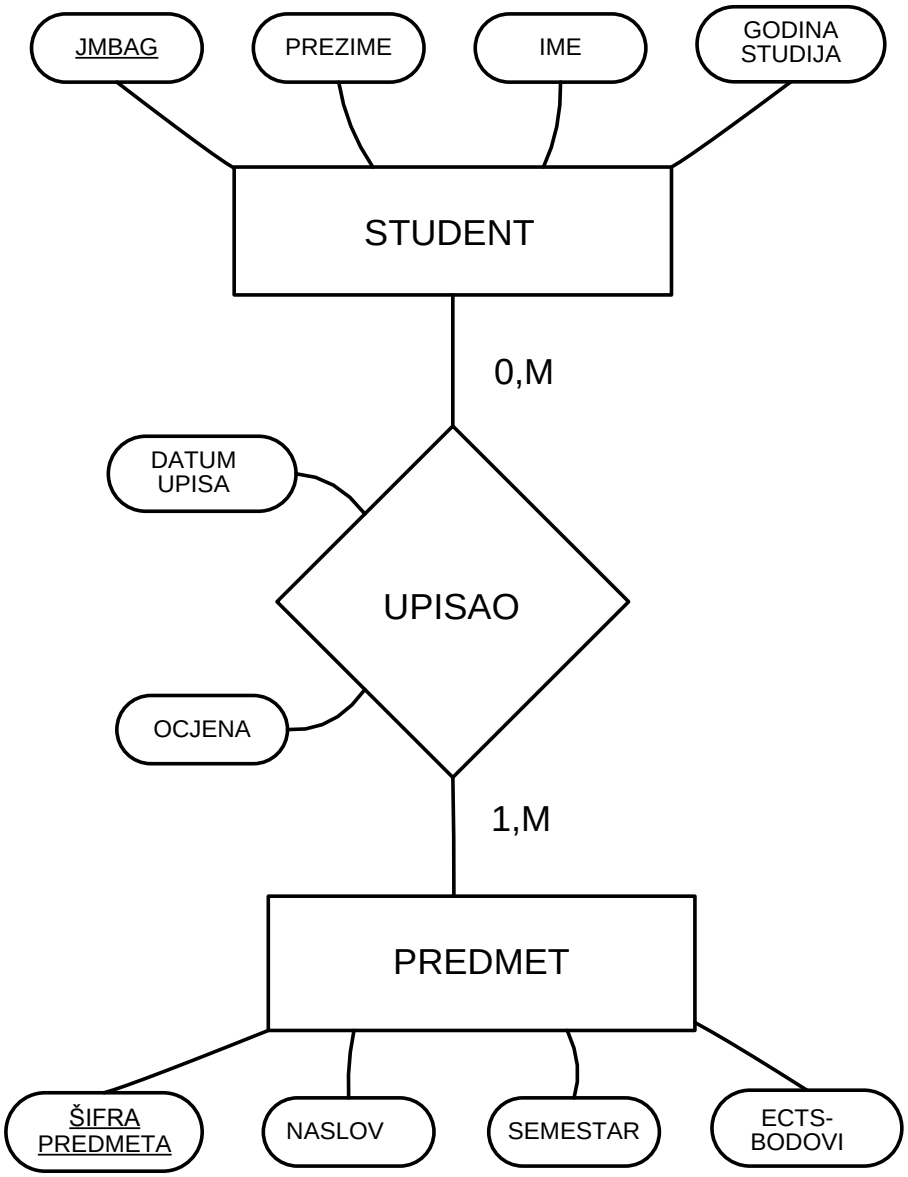
Projektna dokumentacija sastoji se od tekstualnih i grafičkih dijelova. Dokumentacija na konceptualnoj razini uglavnom je grafička, dakle prvenstveno se oslanja na dijagrame. Dokumentacija na logičkoj razini može dijelom biti grafička, no ipak se više oslanja na tekstualne dijelove koji se oblikuju uporabom bogate tipografije (raznoliki fontovi, podcrtavanje i slično). Dokumentacija na fizičkoj razini uvijek je goli ASCII tekst.

Rekli smo da dokumentacija na konceptualnoj razini ustvari opisuje konceptualnu shemu baze koja se sastoji od elemenata koji se zovu entiteti, atributi, odnosno veze. Postoji nekoliko predložaka za prikaz konceptualne sheme, svi se oni sastoje od neke vrste dijagrama, uz kojeg stoji više ili manje tekstualnih dopuna. Opisat ćemo tri najvažnija predloška, preostali se mogu smatrati neznatnim varijacijama tih triju.

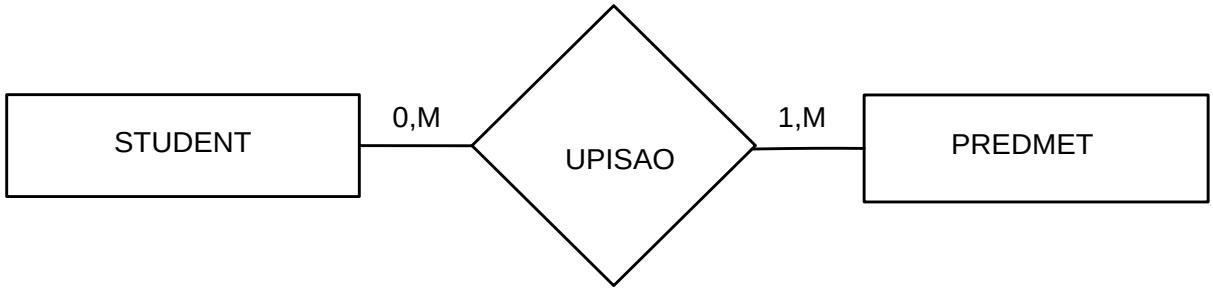
- **Izvorni Chen-ov dijagram.** Kao grafički elementi pojavljuju se pravokutnici, rombovi, „mjehurići“ i spojnice među njima. Pritom pravokutnici označavaju entitete, rombovi veze, a mjehurići atribute. U sam dijagram su kao jedini tekstualni elementi ubačena imena entiteta, veza i atributa, te oznake takozvanih kardinalnosti veza. Prednost ovakvog načina prikazivanja je da je sva informacija prikazana na dijagramu. Nedostatak je da dijagram može postati nepregledan i prenatrpan mjehurićima ukoliko imamo mnogo atributa.
- **Reducirani Chen-ov dijagram.** Riječ je o pojednostavnjenoj vrsti izvornog Chen-ovog dijagrama, gdje su zbog bolje preglednosti nacrtani samo pravokutnici (entiteti), rombovi (veze) i spojnice među njima, a izbačeni su mjehurići (atributi). I dalje su na dijagramu prisutna imena entiteta i veza, te oznake kardinalnosti veza. Nedostatak informacije o atributima na dijagramu nadomještava se tekstom koji prati dijagram.
- **UML-ov *class* dijagram.** UML je standardizirani i danas vrlo popularan grafički jezik koji se rabi u objektno-orijentiranim metodama za razvoj softvera. *Class* dijagram je jedan od standardnih UML dijagrama i on originalno služi za prikaz klasa objekata i veza između tih klasa. Taj dijagram možemo upotrijebiti za prikaz konceptualne sheme baze na taj način da entitet interpretiramo kao posebnu vrstu klase koja ima atribute ali nema operacije. Entitet se tada crta kao pravokutnik s upisanim imenom entiteta na vrhu i upisanim imenima svih atributa u sredini. Veza (ili asocijacija po UML-ovoj terminologiji) crta se kao spojnica između pravokutnika s upisanim imenom na sredini i upisanim oznakama kardinalnosti (ili multipliciteta po UML-ovoj terminologiji) na krajevima. Dijagram sadrži svu potrebnu informaciju, nema potrebe za tekstualnim nadopunama.

Sljedeće četiri slike prikazuju konceptualnu shemu jedne te iste baze prikazanu na tri načina, uporabom opisanih triju obrazaca. Riječ je vrlo jednostavnoj bazi koja sadrži podatke o studentima i predmetima na nekom fakultetu, te pamti koji student je upisao koji predmet. Slika 1.3 prikazuje shemu u obliku izvornog Chen-ovog dijagrama. Na Slikama 1.4 i 1.5 vidimo ekvivalentni reducirani Chen-ov dijagram s popratnim tekstom. Slika 1.6 daje istu informaciju u obliku UML-ovog class dijagrama.

Od tri spomenuta predloška za prikaz konceptualne sheme, mi ćemo u nastavku ovih skripti rabiti isključivo reducirani Chen-ov dijagram s popratnim tekstom. Naime, taj predložak se u praksi pokazuje najspretnijim budući da on predstavlja dobar kompromis između izražajnosti i jednostavnosti.



Slika 1.3: Izvorni Chen-ov dijagram.



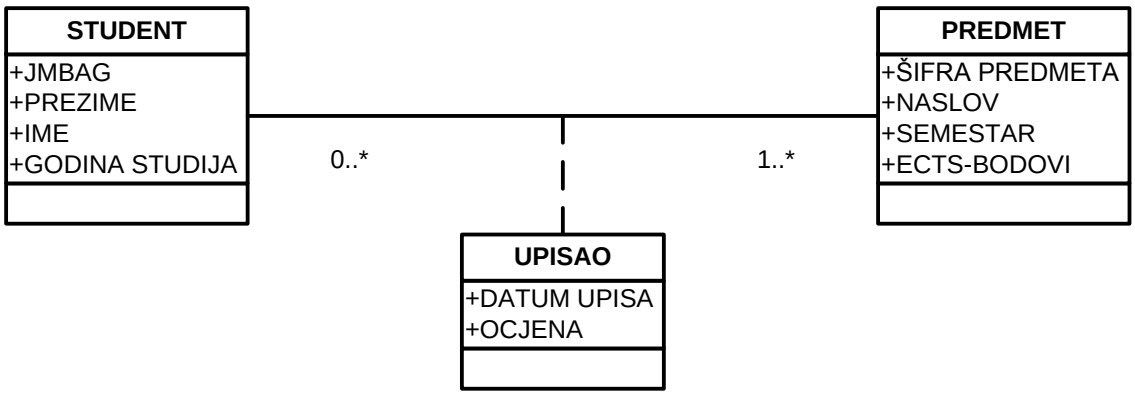
Slika 1.4: Reducirani Chen-ov dijagram.

Tip entiteta STUDENT ima atribute:
JMBAG, PREZIME, IME, GODINA STUDIJA

Tip entiteta PREDMET ima atribute:
ŠIFRA PREDMETA, NASLOV, SEMESTAR, ECTS-BODOVI.

Veza UPISAO ima atribute:
DATUM UPISA, OCJENA.

Slika 1.5: Popratni tekst uz reducirani Chen-ov dijagram.



Slika 1.6: UML-ov *class* dijagram.

Što se tiče dokumentacije na logičkoj razini, rekli smo da ona opisuje logičku shemu baze. U slučaju relacijske baze logička shema je skup relacija (tablica) građenih od atributa (stupaca). Zato se ona također naziva i relacijska shema. Opisat ćemo dva bitno različita predloška za prikaz relacijske sheme koji su danas najčešće u uporabi.

- **Tekstualni prikaz relacijske sheme.** Tradicionalno se rabi u knjigama i člancima o relacijskim bazama podataka. Građa jedne relacije prikazuje se jednim retkom teksta, koji sadrži najprije ime relacije, a zatim okrugle zagrade unutar kojih su nanizana imena atributa odvojena zarezima. Građa cijele baze prikazana je nizom ovakvih redaka: koliko relacija toliko redaka. Poželjno je unutar redaka rabiti bogatu tipografiju, na primjer podcrtavanje istaknutih atributa koji čine takozvani primarni ključ relacije. Također je poželjno priložiti takozvani rječnik podataka: popis svih atributa koji se pojavljuju s neformalnim opisom njihovog značenja i tipa vrijednosti koje mogu poprimiti.
- **Grafički prikaz relacijske sheme.** Pojavio se u softverskim paketima za rad s podacima na osobnim računalima poput MS Access. Također ga rabe neki današnji DBMS-i poput MS SQL Server. Građa baze opisuje se dijagramom koji se sastoji od pravokutnika i strelica. Jedan pravokutnik prikazuje jednu relaciju, te sadrži ime relacije i imena atributa jedno ispod drugog. Ključ je označen odgovarajućom ikonom ili kraticom. Strelice označavaju takozvani referencijalni integritet, dakle one spajaju atribut u jednoj relaciji

koji je ujedno ključ u drugoj relaciji. Uz ovakav dijagram opet je poželjno priložiti rječnik podataka.

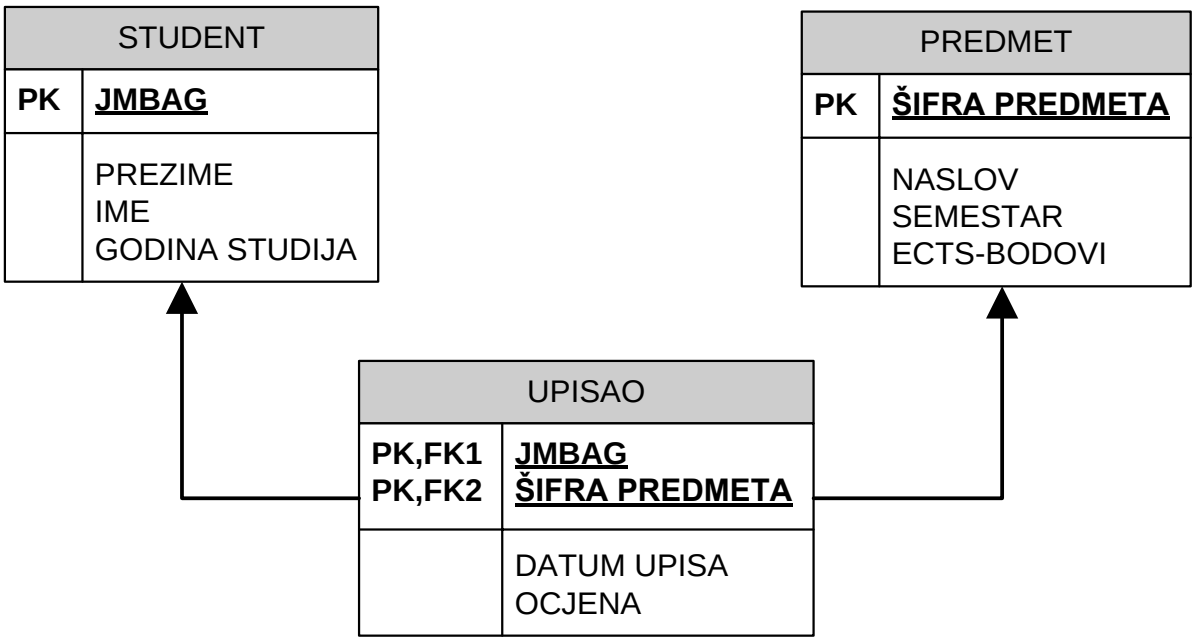
Sljedeće dvije slike prikazuju relacijsku shemu jedne te iste baze prikazanu na dva načina, uporabom opisanih dvaju obrazaca. Riječ je opet o bazi o studentima, predmetima i upisima koju smo upoznali na prethodnim slikama. Slika 1.7 sadrži tekstualni prikaz relacijske sheme, a Slika 1.8 grafički prikaz. U oba slučaja mogli bismo kao nadopunu dodati rječnik podataka koji se nalazi na idućoj Slici 1.9.

STUDENT(JMBAG, PREZIME, IME, GODINA STUDIJA)

PREDMET (ŠIFRA PREDMETA, NASLOV, SEMESTAR, ECTS-BODOVI)

UPISAO (JMBAG, ŠIFRA PREDMETA, DATUM UPISA, OCJENA)

Slika 1.7: Tekstualni prikaz relacijske sheme.



Slika 1.8: Grafički prikaz relacijske sheme.

Od dva spomenuta predloška za prikaz relacijske sheme, mi ćemo u nastavku ovog priručnika dati prednost onom prvom, dakle tekstualnom prikazu. Naime, vjerujemo da je grafički prikaz unatoč svojoj popularnosti donekle nekorektan sa stanovišta relacijskog modela, budući da atribute (stupce) relacija prikazuje kao da su redci.

IME PODATKA	TIP	OPIS
JMBAG	Niz od točno 10 znamenki	Šifra koja jednoznačno određuje studenta
PREZIME	Niz znakova	Prezime studenta
IME	Niz znakova	Ime studenta
GODINA STUDIJA	Cijeli broj između 1 i 5	Godina koju je student upisao
ŠIFRA PREDMETA	Niz od točno 5 znamenki	Šifra koja jednoznačno određuje predmet
NASLOV	Niz znakova	Naslov predmeta
SEMESTAR	„zimski“ ili „ljetni“	Semestar u kojem se predmet predaje
ECTS-BODOVI	Mali cijeli broj	Bodovi koje student dobiva ako položi predmet
DATUM UPISA	Datum	Datum kad je određeni student upisao određeni predmet
OCJENA	Cijeli broj između 2 i 5	Ocjena koju je određeni student dobio iz određenog predmeta

Slika 1.9: Rječnik podataka kao prilog relacijskoj shemi.

Kao što smo već prije spominjali, projektna dokumentacija na fizičkoj razini sadrži fizičku shemu baze. U slučaju uporabe DBMS-a zasnovanog na jeziku SQL, fizička shema zapravo je niz naredbi. Te naredbe su u pravilu zapisane u SQL-u, no dijelom mogu biti zadane i u nekom dodatnom komandnom jeziku kojeg razumije dotični DBMS. U skladu s time, postoji samo jedan predložak za dokumentaciju na fizičkoj razini, a to je tekst sastavljen od ASCII znakova. Ipak, sam sadržaj tog teksta može se razlikovati ovisno o DBMS-u, jer se svaki DBMS koristi donekle različitom sintaksom SQL-a te raspolaže drukčijim komandnim jezikom.

Sljedeća Slika 1.10 prikazuje fizičku shemu naše baze o studentima, predmetima i upisima. Koristi se inačica sintakse SQL-a iz DBMS-a MySQL. U slučaju nekog drugog DBMS-a naredbe bi se donekle razlikovale.

```
CREATE TABLE STUDENT
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   GODINA_STUDIJA ENUM('1','2','3','4','5'),
   PRIMARY KEY(JMBAG))
   ENGINE=INNODB;

CREATE TABLE PREDMET
  (SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   NASLOV CHAR(80),
   SEMESTAR ENUM('Z','L'),
   ECTS_BODOVI NUMERIC(2) UNSIGNED,
   PRIMARY KEY(SIFRA_PREDMETA))
   ENGINE=INNODB;

CREATE TABLE UPISAO
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   DATUM_UPISA DATE,
   OCJENA ENUM('2','3','4','5'),
   PRIMARY KEY(JMBAG,SIFRA_PREDMETA),
   INDEX R_JMBAG_IND (JMBAG),
   INDEX R_SP_IND (SIFRA_PREDMETA),
   FOREIGN KEY (JMBAG) REFERENCES STUDENT(JMBAG),
   FOREIGN KEY (SIFRA_PREDMETA)
     REFERENCES PREDMET(SIFRA_PREDMETA))
   ENGINE=INNODB;
```

Slika 1.10: Fizička shema zapisana u jeziku SQL.

U nastavku ovog priručnika, gotovo sve fizičke sheme i naredbe u SQL-u pisat ćemo u skladu sa sintaksom kojom se koristi MySQL. To je zato što ćemo se istim DBMS-om koristiti i za izvođenje primjera i vježbi.

1.3.3. Uporaba CASE alata i drugih vrsta softvera

Budući da je projektna dokumentacija u osnovi tekstualni dokument s umetnutim dijagramima, osnovni alat za izradu te dokumentacije je tekst procesor poput MS Word. Ipak, za izradu dijagrama koje ćemo umetnuti u tekst potreban nam je dodatni alat, i on se može odabrati na dva načina.

- **Specijalizirani CASE alat.** Riječ je o programskom paketu koji služi softverskim inženjerima za razvoj softvera u skladu s nekom određenom razvojnom metodom. Podržano je crtanje svih dijagrama koje ta metoda propisuje, te je automatizirana izrada odgovarajuće razvojne dokumentacije. Većina današnjih CASE alata daje podršku za aktivnosti analize zahtjeva i oblikovanja, te je prilagođena objektno-orijentiranim metodama razvoja softvera. U skladu s time, takvi CASE alati prvenstveno omogućuju

crtanje UML dijagrama, a neki imaju uključene i dodatne dijagrame koji su specifični za oblikovanje baze podataka. Kao primjer kvalitetnog CASE alata koji je pogodan za oblikovanje baza podataka navodimo Visual Paradigm. Osim svih standardnih UML dijagrama, taj paket podržava i crtanje jedne inačice Chen-ovih dijagrama za konceptualnu shemu baze podataka.

- **Općeniti alat za crtanje dijagrama**. Riječ je o paketu opće namjene za crtanje raznih vrsta dijagrama, koji sadrži razne grafičke predloške bez obzira na njihovo značenje. Neki od takvih paketa imaju uključene i elemente koji se pojavljuju na dijagramima vezanim uz baze podataka. Kao primjer općenitog alata za crtanje dijagrama koji je pogodan za naše svrhe navodimo MS Visio. U njemu već postoje predlošci za većinu UML dijagrama i za grafički prikaz logičke sheme baze podataka, te također i elementi poput pravokutnika, mjehurića u rombova koji mogu poslužiti za sastavljanje Chen-ovih dijagrama.

Obje spomenute vrste softvera imaju svoje prednosti i nedostatke, što ih čini pogodnijim u jednim situacijama, a manje pogodnim u drugim.

- Prednost CASE alata je da on „razumije“ sintaksu i semantiku dijagrama. Zato on može korisniku ispravljati greške kod crtanja. Također, bolji CASE alati nastoje automatizirati dio projektantskog posla, na primjer na osnovu grafičkog prikaza relacijske sheme oni mogu automatski generirati naredbe u SQL-u za stvaranje tablica.
- Nedostatak CASE alata je da je on kompliciraniji, zahtijeva više vremena za savladavanje, te košta više novaca.
- Prednost općenitog alata za crtanje je da on ima jednostavno sučelje poput uobičajenih uredskih paketa, pa se korisnici ne moraju posebno pripremati da bi radili s njime. Također, dijagrami koje oni proizvode mogu biti zapisani u standardnim grafičkim formatima, tako da se lako mogu uklopiti u druge dokumente.
- Nedostatak općenitog alata za crtanje je da on samo editor slika koji ne razumije smisao dijagrama za baze podataka. Zato nam on ne može baš u velikoj mjeri kontrolirati ili automatizirati posao oblikovanja.

Uzevši u obzir nabrojane prednosti i mane, zaključujemo da su CASE alati pogodniji za veće projekte na kojima rade iskusniji projektanti. S druge strane, općeniti alati za crtanje bolji su za male projekte i osobe koje se samo povremeno bave bazama podataka.

1.4. Zadaci za vježbu

Zadatak 1.1. Kako to da od 90-tih godina 20. stoljeća do danas objektne baze podataka nisu uspjele istisnuti iz uporabe relacijske baze? Pritom su u istom razdoblju objektno-orijentirani programski jezici poput C++, Java ili C# istisnuli klasične programske jezike poput C ili COBOL. Imate li kakvo objašnjenje?

Zadatak 1.2. Objasnite zašto se mogućnost istovremenog rada više korisnika s istim podacima ističe kao poseban cilj kojeg bi tehnologija baza podataka trebala ostvariti. Opišite što bi se sve loše moglo desiti kad bi više korisnika nekontrolirano pristupalo istim podacima u isto vrijeme.

Zadatak 1.3. Objasnite zašto se mogućnost oporavka baze nakon kvara ističe kao važan cilj kojeg bi tehnologija baza podataka trebala ostvariti. Što mislite, na koji način DBMS obavlja oporavak baze? Kojim se pomoćnim strukturama podataka on pritom koristi?

Zadatak 1.4. Precrtajte Sliku 1.6 (UML *class* dijagram), služeći se najprije alatom MS Paint, a zatim MS Visio. Koji alat vam se čini spretniji? Jeste li uočili neke prednosti ili mane jednog alata u odnosu na drugi?

Zadatak 1.5. Napišite specifikaciju za neku jednostavniju bazu podataka iz vašeg područja interesa.

Zadatak 1.6. Pod pretpostavkom da znate programirati, napišite program u COBOL-u ili C-u koji podatke o studentima (JMBAG, PREZIME, IME, GODINA_STUDIJA) izravno pohranjuje u datoteku, te ih odatle može i pročitati. Jamči li takav način pohranjivanja podataka fizičku odnosno logičku nezavisnost podataka? Obrazložite odgovor.

2. Konceptualno oblikovanje baze podataka

U ovom poglavlju opisujemo prvu fazu oblikovanja baze podataka, a to je konceptualno oblikovanje. Glavni cilj te faze je stvoriti *konceptualnu shemu* baze, sastavljenu od entiteta, veza i atributa.

Konceptualna shema daje zoran i jezgrovit prikaz baze koji je oslobođen tehničkih detalja. Moglo bi se reći da ta shema zapravo u prvom redu opisuje stvarni svijet o kojem želimo bilježiti podatke, a tek onda na posredan način i samu bazu. U skladu s poslovicom da slika govori tisuću riječi, opis se najviše oslanja na dijagrame.

Važno svojstvo konceptualne sheme je da je ona razumljiva ljudima svih struka, te da ona može služiti kao sredstvo za komunikaciju projektanata i korisnika. U toj komunikaciji korisnici nastoje utvrditi jesu li projektanti prepoznali sve poslovne procese, uključili sve potrebne podatke, te ispravno shvatili odnose među tim podacima.

Konceptualna shema ne može se automatski implementirati pomoću današnjih DBMS-a, u prvom redu zato što joj nedostaju brojni detalji. Ipak, to ne umanjuje njezinu uporabljivost, budući da postoje jasna pravila koja kažu kako se ona dalje pretvara u relacijsku shemu, te koje daljnje informacije joj treba dodati tijekom te pretvorbe.

2.1. Entiteti, atributi, veze

Modeliranje entiteta i veza zahtijeva da svijet promatramo preko tri kategorije:

- **entiteti**: stvari, bića, pojave ili događaji koji su nam od interesa;
- **veze**: odnosi među entitetima koji su nam od interesa;
- **atributi**: svojstva entiteta ili veza koja su nam od interesa.

U nastavku ćemo podrobnije opisati sva tri pojma.

2.1.1. Entiteti i njihovi atributi

Entitet je nešto o čemu želimo spremati podatke, nešto što je u stanju postojati ili ne postojati, te se može identificirati. Entitet može biti stvar ili biće, na primjer: KUĆA, FAKULTET, STUDENT, PREDMET (na fakultetu), NASTAVNIK, AUTO, …, odnosno događaj ili pojava, na primjer: NOGOMETNA UTAKMICA, SERVISIRANJE AUTA, POLAGANJE ISPITA, ….

Entitet je opisan *atributima*. Na primjer, atributi KUĆE su: ULICA, KUĆNI BROJ, BROJ KATOVA, BOJA FASADE, …. , atributi STUDENTA su JMBAG (jedinstveni matični broj akademskog građanina), PREZIME, IME, GODINA STUDIJA, …, a atributi PREDMETA koji se predaje na fakultetu su ŠIFRA PREDMETA, NASLOV, SEMESTAR u kojem se predaje, ECTS-BODOVI koji određuju njegovu težinu, ….

Ukoliko neki atribut i sam zahtijeva svoje atribute, tada ga radije treba smatrati novim entitetom. Na primjer za entitet AUTO mogli bismo uvesti atribut MODEL tog auta. No ako za opis MODELA trebamo dodatne atribute, na primjer KATEGORIJA kojoj taj model

pripada, GODINA kad se taj model pojavio na tržištu, tada MODEL moramo smatrati entitetom, a odnos između AUTA i njegovog MODELA trebamo tumačiti kao vezu između dva entiteta. Isto pravilo vrijedi i ako atribut može istovremeno poprimiti više vrijednosti. Na primjer, za entitet SERVISIRANJE AUTA, atribut KVAR je zapravo lista vrijednosti, jer se na jednom servisiranju može popraviti više kvarova. Tada opet KVAR moramo smatrati entitetom, a niz KVAROVA popravljenih na istom SERVISIRANJU AUTA dobiva se uspostavljanjem veze između tih dvaju entiteta.

Ime entiteta zajedno sa pripadnim popisom atributa zapravo određuje *tip* entiteta. Za zadani tip entiteta može postojati cijeli skup *primjeraka* (pojava) entiteta tog tipa, od kojih je svaki opisan donekle drukčijim vrijednostima atributa. Na primjer, STUDENT je tip čiji primjerci su konkretni studenti Petar Petrović, Marko Marković, Dragica Horvat, …. Svaki od tih konkretnih studenata ima drukčiju kombinaciju vrijednosti za JMBAG, PREZIME, IME, GODINU STUDIJA. Razlika između tipa i primjerka entiteta slična je razlici između općeg i posebnog broja u matematici, ili razlici između klase i objekta u objektno-orijentiranim programskim jezicima.

Kandidat za ključ je atribut, ili skup atributa, čije vrijednosti jednoznačno određuju primjerak entiteta zadanog tipa. Dakle, ne mogu postojati dva različita primjerka entiteta istog tipa s istim vrijednostima kandidata za ključ. Na primjer, za tip entiteta AUTO, kandidat za ključ je atribut REGISTARSKA OZNAKA. Za tip entiteta SERVISIRANJE AUTA teško je naći jedan atribut koji bi jednoznačno određivao primjerak tog entiteta, no zato kombinacija atributa REGISTARSKA OZNAKA (auta na servisu) i DATUM (kad je servisiranje obavljeno) predstavlja kandidat za ključ.

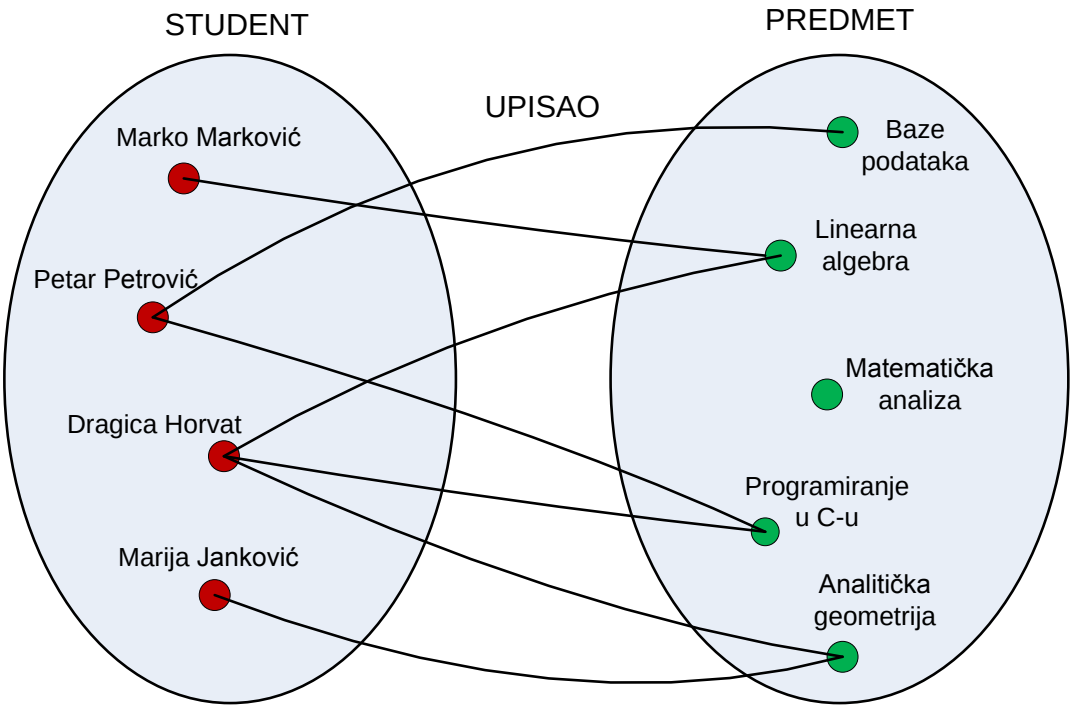
Ukoliko jedan tip entiteta ima više kandidata za ključ, tada biramo jednog od njih koji nam se čini najpogodnijim da služi za identifikaciju, te ga proglašavamo *primarnim* ključem. Na primjer, za tip entiteta STUDENT, dobri kandidati za ključ su JMBAG, OIB i JMBG. U običnom životu služimo se i kombinacijom atributa PREZIME i IME, no strogo govoreći ta kombinacija nije pouzdan kandidat za ključ jer se lako može desiti da dva studenta imaju isto ime i prezime. Od ponuđenih kandidata za ključ za STUDENTA, kao primarni ključ možemo odabrati JMBAG, zato što je on uvriježen u akademskoj zajednici. Odabir primarnog ključa važna je odluka koja povlači konkretne posljedice kod kasnije implementacije baze.

Unutar jedne konceptualne sheme, tipovi entiteta moraju imati različita imena. Također, svi atributi koji opisuju isti entitet moraju imati različita imena. Dopušta se da dva entiteta imaju atribute s istim imenom, no tada se podrazumijeva da su to ustvari atributi s istim značenjem i istim tipom vrijednosti. Na primjer, oba entiteta STUDENT i NASTAVNIK mogu imati atribut PREZIME, zato jer je to atribut koji opisuje bilo koju osobu bez obzira je li ona student ili nastavnik.

2.1.2. Veze i njihovi atributi

Uspostavljanjem *veze* između dvaju ili više entiteta izražavamo činjenicu da se ti entiteti nalaze u nekom odnosu. Veza se uvijek *definira na razini tipova* entiteta, no *realizira se povezivanjem pojedinih primjeraka* entiteta dotičnih tipova. Za sada ćemo se ograničiti na takozvane *binarne* veze, koje su najjednostavnije i najčešće u primjenama. Binarna veza uspostavlja se između točno dva tipa entiteta. Stanje binarne veze opisuje se kao skup uređenih parova primjeraka entiteta koji su trenutno povezani.

Kao primjer, promatramo binarnu vezu UPISAO između tipova entiteta STUDENT i PREDMET. Njome se izražava činjenica da studenti upisuju izborne predmete na fakultetu. U svakom trenutku, stanje veze prikazuje se kao skup parova primjeraka tih entiteta, gdje svaki pojedini par označava da je dotični student upisao dotični predmet.



Slika 2.1: Stanje veze, parovi povezanih primjeraka entiteta.

U jednom trenutku, stanje veze UPISAO može izgledati kao na Slici 2.1. Vidimo da postoje 4 primjerka entiteta STUDENT, za koje smo zbog jednostavnosti pretpostavili da ih možemo razlikovati na osnovu IMENA i PREZIMENA. Također vidimo da postoji 5 primjeraka entiteta PREDMET, koje možemo razlikovati na osnovu NASLOVA. Spojnice na slici prikazuju parove povezanih primjeraka entiteta. Dakle, student Marko Marković je upisao predmet Linearna algebra; studentica Dragica Horvat je upisala tri predmeta: Linearnu algebru, Programiranje u C-u i Analitičku geometriju; predmet Matematička analiza nije upisao nitko od studenata. Stanje veze može se vremenom mijenjati, dakle mogu se pojaviti novi parovi, a nestati postojeći. Na primjer, može se desiti da Marko Marković dodatno upiše Matematičku analizu, a da Petar Petrović odustane od Baza podataka.

Osim što povezuje tipove entiteta, veza može imati i svoje atribute, dakle atribute koje ne možemo pripisati ni jednom od tih tipova. Na primjer, veza UPISAO može imati atribut DATUM UPISA. Zaista, DATUM UPISA ne možemo smatrati atributom entiteta STUDENT zato jer isti student može upisati razne predmete na razne datume. Slično, DATUM UPISA ne može biti ni atribut od PREDMET jer isti predmet može biti upisan od raznih studenata na razne datume. Konkretna vrijednost DATUMA UPISA zapravo se pridružuje uz konkretni par primjeraka za STUDENT i PREDMET koji su povezani. Na primjer, možemo zabilježiti da je Marko Marković upisao Linearnu algebru 3. rujna 2010, a Marija Janković je upisala Analitičku geometriju 5. rujna 2010.

Slično kao tipovi entiteta, i veze unutar iste sheme moraju imati različita imena. Također, svi atributi koji pripadaju istoj vezi moraju imati različita imena. Opet se dopušta da dvije veze ili entitet i veza imaju atribute s istim imenom, no tada se podrazumijeva da su to ustvari atributi s istim značenjem i istim tipom vrijednosti.

2.1.3. Funkcionalnost veze, obaveznost članstva, kardinalnost

Načini na koji veza može povezati primjerke entiteta određeni su svojstvima funkcionalnosti, obaveznosti članstva, odnosno kardinalnosti. Poznavanje tih svojstava važno je da bi se veza u idućoj fazi oblikovanja ispravno prikazala unutar relacijske sheme. Mi ćemo navedena svojstva za sada definirati samo za slučaj binarne veze, makar se ona mogu primijeniti i u općenitijem slučaju.

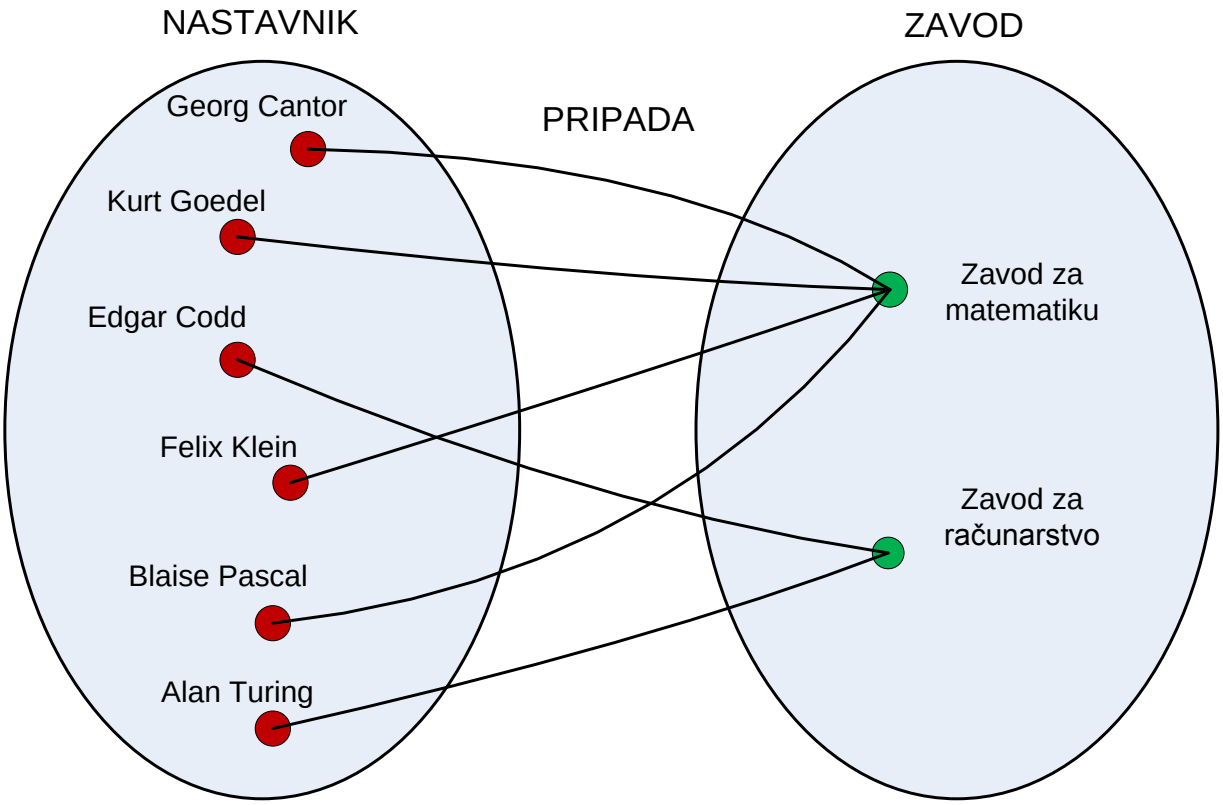
Promatramo vezu između tipova entiteta E_1 i E_2. *Funkcionalnost* te veze je svojstvo koje kaže je li za odabrani primjerak entiteta jednog tipa moguće jednoznačno odrediti povezani primjerak entiteta drugog tipa. Drugim riječima, funkcionalnost je svojstvo koje kaže može li se veza interpretirati kao preslikavanje (funkcija) iz skupa primjeraka entiteta jednog tipa u skup primjeraka entiteta drugog tipa. S obzirom da se ista veza može promatrati u dva smjera, od E_1 do E_2 i obratno, postoje 4 vrste funkcionalnosti koje su opisane u Tabeli 2.1.

OZNAKA	**NAZIV**	**OPIS**
1:1	Jedan-naprama-jedan	Jedan primjerak od E_1 može biti povezan najviše s jednim primjerkom od E_2. Također, jedan primjerak od E_2 može biti povezan najviše s jednim primjerkom od E_1.
1:M	Jedan-naprama-mnogo	Jedan primjerak od E_1 može biti povezan s više primjeraka od E_2. Istovremeno, jedan primjerak od E_2 može biti povezan najviše s jednim primjerkom od E_1.
M:1	Mnogo-naprama-jedan	Jedan primjerak od E_1 može biti povezan najviše s jednim primjerkom od E_2. Istovremeno, jedan primjerak od E_2 može biti povezan s više primjeraka od E_1.
M:M	Mnogo-naprama-mnogo	Jedan primjerak od E_1 može biti povezan s više primjeraka od E_2. Također, jedan primjerak od E_2 može biti povezan s više primjeraka od E_1.

Tabela 2.1: Vrste funkcionalnosti za vezu između tipova entiteta E_1 i E_2.

Prije spomenuta veza UPISAO između tipova entiteta STUDENT i PREDMET ima funkcionalnost M:M. Zaista, jedan student može upisati više predmeta, a jedan predmet može biti upisan od više studenata. To se lijepo vidi na Slici 2.1.

Kao primjer za funkcionalnost M:1 navodimo vezu PRIPADA između tipova entiteta NASTAVNIK i ZAVOD (organizacijska jedinica unutar fakulteta). Zaista, svaki nastavnik pripada samo jednom zavodu, no jedan zavod može zapošljavati mnogo nastavnika. Jedno stanje veze ilustrirano je Slikom 2.2. Zbog jednostavnosti smo pretpostavili da nastavnike možemo razlikovati na osnovu imena i prezimena. Ista veza ako se gleda u suprotnom smjeru može služiti kao primjer za funkcionalnost 1:M.

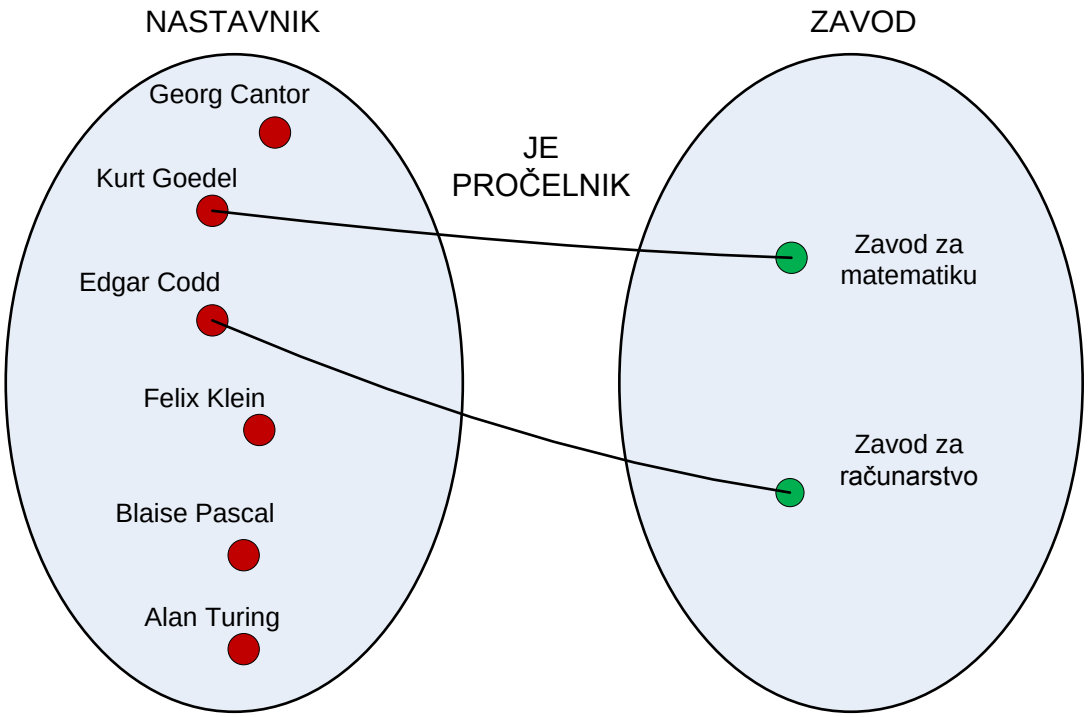


Slika 2.2: Veza s funkcionalnošću M:1.

Kao primjer za funkcionalnost 1:1 navodimo vezu JE PROČELNIK između tipova entiteta NASTAVNIK i ZAVOD. Jedan nastavnik može biti pročelnik najviše jednog zavoda (onog kojem inače pripada), a jedan zavod može imati samo jednog pročelnika. Slika 2.3 prikazuje jedno stanje te veze.

Opet promatramo vezu između tipova entiteta E_1 i E_2. Kažemo da E_1 ima *obavezno članstvo* u toj vezi ako svaki primjerak od E_1 mora sudjelovati u vezi, dakle mora biti povezan barem s jednim primjerkom od E_2. Analogno se definira i obaveznost članstva za E_2.

U prije spomenutoj vezi JE PROČELNIK, tip entiteta NASTAVNIK nema obavezno članstvo jer ne mora svaki nastavnik obavljati dužnost pročelnika. S druge strane, možemo zahtijevati da ZAVOD ima obavezno članstvo, dakle tražimo da svaki zavod u svakom trenutku ima svog pročelnika. To se opet vidi na Slici 2.3.



Slika 2.3: Veza s funkcionalnošću 1:1.

Svojstva funkcionalnosti i obaveznosti članstva mogu se otprilike izraziti samo jednim svojstvom koje se zove *kardinalnost*. I dalje promatramo vezu između tipova entiteta E_1 i E_2. Kardinalnost te veze u smjeru od E_1 do E_2 definira se kao broj primjeraka od E_2 koji istovremeno mogu biti povezani s odabranim primjerkom od E_1. Kardinalnost u smjeru od E_2 do E_1 definira se analogno. To znači da za svaku vezu utvrđujemo dvije kardinalnosti, za jedan i za drugi smjer.

Kardinalnost je obično nemoguće sasvim točno izraziti, pa umjesto točnog broja navodimo interval u obliku donje i gornje granice. Dvije granice označavaju se oznakama 0, 1 ili M, te se odvajaju zarezom. Uobičajene kardinalnosti navedene su i opisane u Tabeli 2.2.

OZNAKA	**OPIS**
0,1	Jedan primjerak od E_1 može biti povezan s nijednim ili najviše s jednim primjerkom od E_2.
1,1	Jedan primjerak od E_1 mora biti povezan s točno jednim primjerkom od E_2.
0,M	Jedan primjerak od E_1 može biti povezan s nijednim, s jednim ili s više primjeraka od E_2.
1,M	Jedan primjerak od E_1 mora biti povezan s najmanje jednim, no možda i s više primjeraka od E_2.

Tabela 2.2: Kardinalnost veze promatrane u smjeru od tipa E_1 do tipa E_2.

Kao primjer, navedimo da veza JE PROČELNIK promatrana u smjeru od NASTAVNIK do ZAVOD ima kardinalnost 0,1. Kardinalnost iste veze promatrane u suprotnom smjeru je 1,1. Veza UPISAO u smjeru od PREDMET do STUDENT ima kardinalnost 0,M jer dopuštamo da neki predmeti ostanu neupisani. No mogli bismo zahtijevati da kardinalnost iste veze u suprotnom smjeru bude 1,M, čime tražimo od svakog studenta da upiše bar jedan predmet. Sve se ovo može provjeriti na slikama 2.1 i 2.3.

Očito je da kod binarnih veza donja granica za kardinalnost u smjeru od E_1 do E_2 zapravo određuje obaveznost članstva za E_1. Zaista, ako je ta donja granica 0, onda primjerak od E_1 ne mora biti povezan ni s jednim primjerkom od E_2, pa je članstvo neobavezno. Slično, ako je donja granica 1 ili više, tada bilo koji primjerak od E_1 mora biti povezan barem s jednim primjerkom od E_2 pa je članstvo obavezno. S druge strane, gornja granica za kardinalnost u smjeru od E_1 od E_2 određuje jedan dio funkcionalnosti veze. Gledanjem kardinalnosti iste veze ali u smjeru od E_2 do E_1 otkrivamo obaveznost članstva za E_2, te drugi dio funkcionalnosti. U tom smislu, svojstvo kardinalnosti kod binarnih veza može služiti kao zamjena za svojstva funkcionalnosti i obaveznosti članstva. Kod veza koje nisu binarne stvar se ipak komplicira i korisno je navoditi sva tri svojstva.

2.2. Oblikovanje konceptualne sheme

Nakon što smo se u prethodnom potpoglavlju upoznali s elementima konceptualne sheme, u ovom potpoglavlju bavimo se samim postupkom oblikovanja te sheme. Dakle pokušavamo opisati korake kojima se na osnovu specifikacije dolazi do projektne dokumentacije na konceptualnoj razini. Opis koraka treba shvatiti uvjetno, naime treba biti svjestan da se postupak oblikovanja ne može do kraja opisati ni definirati. U oblikovanju uvijek ostaje mjesta za dosjetljivost, improvizaciju i kreativnost.

2.2.1. Otkrivanje entiteta, veza i atributa

Prvi korak u oblikovanju konceptualne sheme je otkrivanje samih elemenata od kojih se ta shema sastoji, a to su entiteti, veze i atributi. U pravilu, elementi sheme trebaju se prepoznati čitanjem specifikacije. Analiziraju se rečenice iz specifikacije i uočavaju imenice (subjekti, objekti) i glagoli (predikati).

- Imenice upućuju na entitete i atribute.
- Glagoli upućuju na veze.

Naravno, ne mora svaka riječ iz specifikacije predstavljati element sheme. Na projektantu je da odluči što je važno a što se može ispustiti.

U prepoznavanju elemenata sheme, projektant se često susreće s dilemom treba li neku značajnu imenicu iz specifikacije shvatiti kao entitet ili kao atribut. Dilema se rješava odgovaranjem na sljedeća pitanja.

- Ima li pojam označen imenicom neka svoja dodatna svojstva koja treba pamtiti? Ako da, onda je to entitet.
- Inače, je li taj pojam svojstvo koje može poprimiti više vrijednosti kad njime opisujemo neki predmet, osobu ili pojavu? Ako da, onda je to opet entitet.
- Inače je taj pojam atribut.

Nakon što smo otkrili entitete, veze i atribute, potrebno je za svaki entitet utvrditi koji atributi ga opisuju. Ne treba zaboraviti mogućnost da neki atributi pripadaju vezi između entiteta, a ne pojedinim entitetima. Također, za svaku vezu treba odrediti njezinu funkcionalnost, obaveznosti članstva, te kardinalnosti. Poželjno je da za svaki atribut imamo neku približnu predodžbu o tipu vrijednosti koje on može poprimiti, makar se u ovoj fazi još ne zahtijeva da taj tip bude točno određen. Dalje, za svaki entitet treba odabrati primarni ključ.

Postupak otkrivanja entiteta, veza i atributa može zapeti ukoliko je specifikacija nejasna ili nepotpuna. U tom slučaju projektant treba dodatno razgovarati s korisnicima da bi otklonio nejasnoće. Također, projektant može po vlastitom nahođenju dodavati umjetne atribute koji služe za identifikaciju ili klasifikaciju (šifre, oznake, …), ili atribute za koje je očito da nedostaju.

Kao primjer, zamislimo da trebamo oblikovati konceptualnu shemu baze podataka fakulteta. U specifikaciji pišu sljedeće rečenice.

> Studenti svake akademske godine upisuju godinu studija. Pritom biraju neke od ponuđenih izbornih predmeta, tako da zbroj njihovih ECTS-bodova u svakom semestru bude barem 30. Predmeti moraju biti izabrani najkasnije do 15. rujna. Fakultet je organiziran u zavode. Svaki nastavnik je član točno jednog zavoda. Nastavnici iz istog zavoda između sebe biraju pročelnika. Da bi bolje upravljao ljudskim resursima, pročelnik zavoda može mijenjati plaće svojih nastavnika. Jedna od zadaća zavoda je da se brine o nastavi. Svake akademske godine zavod nudi nekoliko izbornih predmeta za studente i osigurava nastavnike koji će predavati te predmete. Pritom svaki predmet ima samo jednog nastavnika. Na kraju akademske godine fakultet pohvaljuje studente koji su postigli najbolje ocjene iz upisanih predmeta.

Analizom ovih rečenica otkrivamo sljedeće entitete, veze i atribute.

- Tipovi entiteta su: STUDENT, PREDMET, NASTAVNIK, ZAVOD.
- Veze su: UPISAO između STUDENT i PREDMET, PRIPADA između NASTAVNIK i ZAVOD, JE PROČELNIK između NASTAVNIK i ZAVOD, NUDI između ZAVOD i PREDMET, PREDAJE između NASTAVNIK i PREDMET.
- Atributi su: (studentova) GODINA STUDIJA, SEMESTAR (kad se predmet predaje), ECTS-BODOVI (za predmet), DATUM UPISA (kad je student izabrao i upisao predmet), OCJENA (koju je student dobio iz predmeta).

Dalje za svaki tip entiteta ili vezu treba utvrditi pripadni popis atributa.

- Jasno je da STUDENT ima atribut GODINA STUDIJA, no zdrav razum nam kaže da treba dodati i PREZIME i IME. Da bismo imali pouzdani primarni ključ, dodajemo još i JMBAG.
- PREDMET na osnovu utvrđenog ima atribute SEMESTAR i ECTS-BODOVI. No opet je prirodno da dodamo NASLOV. Također, kao primarni ključ dodajemo umjetni atribut ŠIFRA PREDMETA.
- NASTAVNIK za sada ima atribut PLAĆA. Po vlastitom nahođenju dodajemo OIB, PREZIME, IME, te BROJ SOBE (u kojoj ima ured i gdje ga studenti mogu naći). Očito, OIB može poslužiti kao primarni ključ.
- ZAVOD za sada nema atributa. Dodajemo IME ZAVODA koje je ujedno i primarni ključ, te OPIS DJELATNOSTI.
- Veza UPISAO ima atribute DATUM UPISA i OCJENA. Ostale veze nemaju atributa.

Dalje za svaku vezu određujemo njezinu kardinalnost u jednom i u drugom smjeru. Budući da su sve naše veze binarne, time je odmah određena i njihova funkcionalnost, te obaveznost članstva entiteta u njima.

- UPISAO u smjeru od STUDENT do PREDMET ima kardinalnost 1,M, a u obratnom smjeru 0,M.
- PRIPADA u smjeru od NASTAVNIK do ZAVOD je 1,1, a obratno je 1,M.
- JE PROČELNIK u smjeru od NASTAVNIK do ZAVOD je 0,1, a obratno je 1,1.
- NUDI u smjeru od ZAVOD do PREDMET je 0,M, a obratno je 1,1.
- PREDAJE u smjeru od NASTAVNIK do PREDMET je 0,M, a obratno je 1,1.

Primijetimo da ova jednostavna shema opisuje samo trenutno stanje na našem fakultetu, a ne pamti „povijest“. Dakle za studenta se bilježi koje predmete je on upisao ove akademske godine, a ne vidi se što je imao upisano prošle godine. Isto tako, za zavod se bilježi tko je njegov sadašnji pročelnik, a ne vidi se tko je bio prethodni pročelnik, i tako dalje. Baza koja bi pamtila povijest događaja imala bi složeniju shemu.

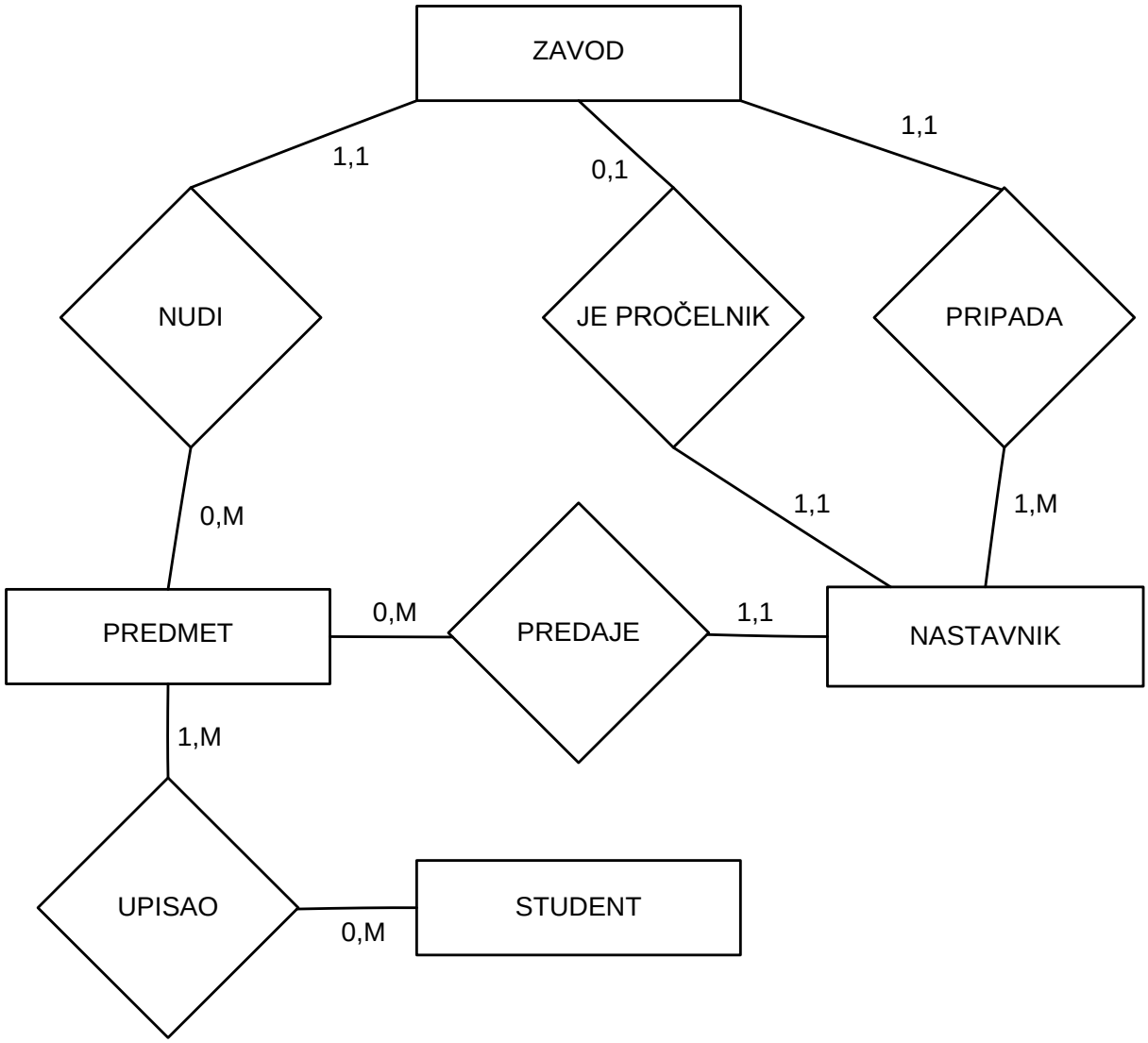
2.2.2. Crtanje dijagrama

Nakon što smo otkrili sve entitete, veze i atribute, idući korak u oblikovanju konceptualne shema je da se ti elementi povežu i prikažu u obliku dijagrama. Služimo se reduciranim Chen-ovim dijagramom, gdje su tipovi entiteta nacrtani kao pravokutnici, a veze kao rombovi. Imena tipova entiteta odnosno veza upisana su u odgovarajuće pravokutnike odnosno rombove. Da bi se znalo između kojih tipova entiteta je uspostavljena određena veza, odgovarajući romb je povezan spojnicama s odgovarajućim pravokutnicima. Uz spojnice su također upisane oznake kardinalnosti veze, i to tako da kardinalnost u smjeru od jednog do drugog tipa entiteta piše bliže drugom tipu.

Primjer dijagrama nacrtanog prema navedenim pravilima vidi se na Slici 2.4. Riječ je o prikazu konceptualne sheme za bazu podataka o fakultetu, čije elemente smo otkrili u prethodnom odjeljku.

Nakon što je nacrtao dijagram, projektant svoj crtež svakako treba pokazati korisnicima, te ga treba analizirati zajedno s njima. Na taj način lagano se pronalaze eventualni propusti u konceptualnoj shemi, dakle nedostatak nekog entiteta ili veze, krivo postavljena veza, netočna kardinalnost i slično. Ako postoje propusti, projektant se vraća na korak otkrivanja veza i atributa, ponovo crta dijagram i opet ga pokazuje korisnicima. Ciklus oblikovanja konceptualne sheme može se iterirati više puta, sve dok se ne dođe do dijagrama na kojeg korisnici više nemaju primjedbi.

Primijetimo da reducirani Chen-ov dijagram kakvim se mi koristimo opisuje samo entitete i veze, a ne sadrži informacije o atributima. Ovo ispuštanje informacija radi se zato da bi dijagram bio jednostavan i pregledan. Naime, atributa obično ima mnogo, pa bi njihovo ucrtavanje stvorilo veliku gužvu. Ipak, to znači da konceptualna shema nije u potpunosti opisana samim dijagramom, te da se nedostajuća informacija mora dostaviti u tekstu koji prati dijagram.



Slika 2.4: Dijagram s entitetima i vezama za bazu podataka o fakultetu.

2.2.3. Sastavljanje teksta koji prati dijagram

Zadnji korak u oblikovanju konceptualne sheme je sastavljanje teksta koji prati dijagram. Taj tekst treba dati one informacije o konceptualnoj shemi koje se ne vide na samom dijagramu. U prvom redu, tu mora biti uključen popis atributa za svaki tip entiteta i svaku vezu.

Za naš primjer baze podataka o fakultetu, tekst koji prati dijagram sa Slike 2.4 mogao bi izgledati kao što je prikazano na Slici 2.5. Znači za svaki entitet i vezu navodimo sve atribute. Također, za svaki tip entiteta podvlačenjem označavamo primarni ključ.

Osim ovih najnužnijih informacija, tekst koji prati dijagram može po potrebi uključiti i dodatne sadržaje, na primjer projektantovo objašnjenje zašto je uveo neki atribut kojeg nije bilo u specifikaciji, zašto je odabrao primarni ključ na određeni način, zašto je pretpostavio da neka veza ima baš neku određenu kardinalnost, i tako dalje.

Tip entiteta STUDENT ima atribute:
JMBAG, PREZIME, IME, GODINA STUDIJA.

Tip entiteta PREDMET ima atribute:
ŠIFRA PREDMETA, NASLOV, SEMESTAR, ECTS-BODOVI.

Tip entiteta NASTAVNIK ima atribute:
OIB, PREZIME, IME, BROJ SOBE, PLAĆA.

Tip entiteta ZAVOD ima atribute:
IME ZAVODA, OPIS DJELATNOSTI.

Veza UPISAO ima atribute:
DATUM UPISA, OCJENA.

Ostale veze nemaju atribute.

Slika 2.5: Popratni tekst uz dijagram sa Slike 2.4.

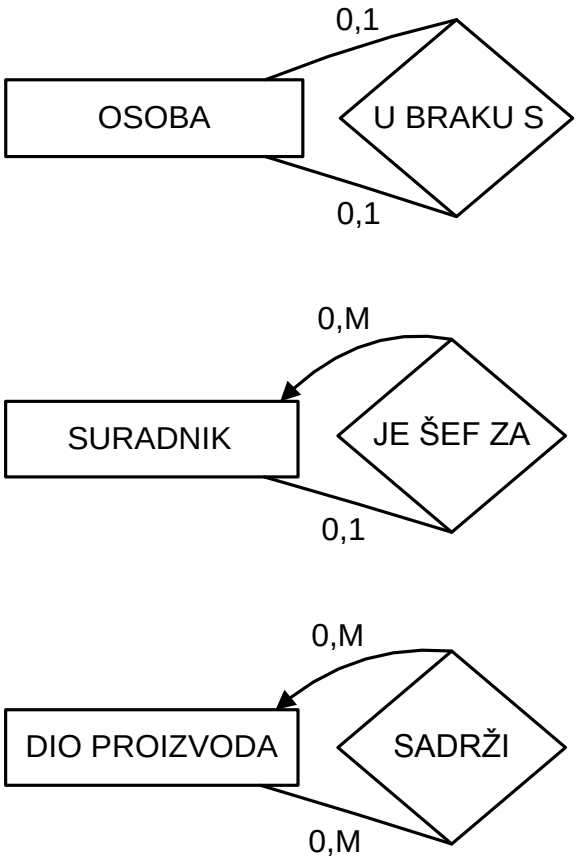
Ako to želimo, u tekst možemo uključiti i neku vrstu rječnika podataka, gdje je za svaki atribut otprilike objašnjeno što on znači i koji tip vrijednosti on može poprimiti. Ipak, takav rječnik nije obavezan na konceptualnoj razini oblikovanja, već se ostavlja za logičku razinu. Na konceptualnoj razini zapravo je dovoljno da atributi imaju smislene nazive iz kojih se naslućuje njihovo značenje i tip. Na primjer, ako se atribut zove PREZIME, tada je jasno što on znači te da on može poprimiti vrijednosti koje odgovaraju ljudskim prezimenima, ako se zove ŠIFRA PREDMETA, onda je to očito podatak koji služi a identifikaciju predmeta i vjerojatno izgleda kao kombinacija znakova s propisanom duljinom i obrascem, ako se zove ECTS-BODOVI, onda bi to trebao biti cijeli broj, i tako dalje.

2.3. Složenije veze

U dosadašnjim primjerima pojavljivale su se samo binarne veze, dakle takve koje povezuju dva različita tipa entiteta. Ta vrsta odnosa među entitetima je najčešća, a i najpoželjnija zbog svoje jednostavnosti. No postoje situacije kad nam binarne veze nisu dovoljne. U nastavku ćemo opisati tri vrste složenijih veza, te ćemo objasniti pravila za njihovo crtanje na dijagramima.

2.3.1. Prikaz involuirane veze

Involuirana veza povezuje jedan tip entiteta s tim istim tipom. Njezino stanje opisuje se kao skup uređenih parova primjeraka entiteta istog tipa koji su povezani. Funkcionalnost takve veze opet može biti 1:1, 1:M, odnosno M:M. Slika 2.6 sadrži primjere za involuirane veze s različitim funkcionalnostima.



Slika 2.6: Primjeri za involuirane veze.

Prvi dijagram na Slici 2.6 prikazuje vezu U BRAKU S, koja povezuje tip entiteta OSOBA sa samim sobom. Riječ je o bračnoj vezi između osoba, dakle povezani su oni primjerci entiteta OSOBA koji su u braku. Funkcionalnost veze je 1:1, naime pretpostavlja se da su prošli brakovi zaboravljeni, a poligamija zabranjena. Članstvo entiteta u toj vezi je, naravno, neobavezno.

Drugi dijagram na Slici 2.6 prikazuje vezu JE ŠEF ZA, koja povezuje tip entiteta SURADNIK sa samim sobom. Veza prikazuje odnos suradnika u nekom poduzeću. Bilježi se tko je kome šef. Funkcionalnost je 1:M jer jedan šef može imati više podređenih suradnika, a jedan suradnik ima najviše jednog neposrednog šefa. Dijagram ima ucrtanu strelicu koja pokazuje smjer tumačenja veze (jedan je šef za više suradnika, a ne obratno). Članstvo entiteta u vezi je obavezno jer skoro svaki suradnik ima svog šefa, a onaj koji nema šefa (glavni direktor) je šef drugima.

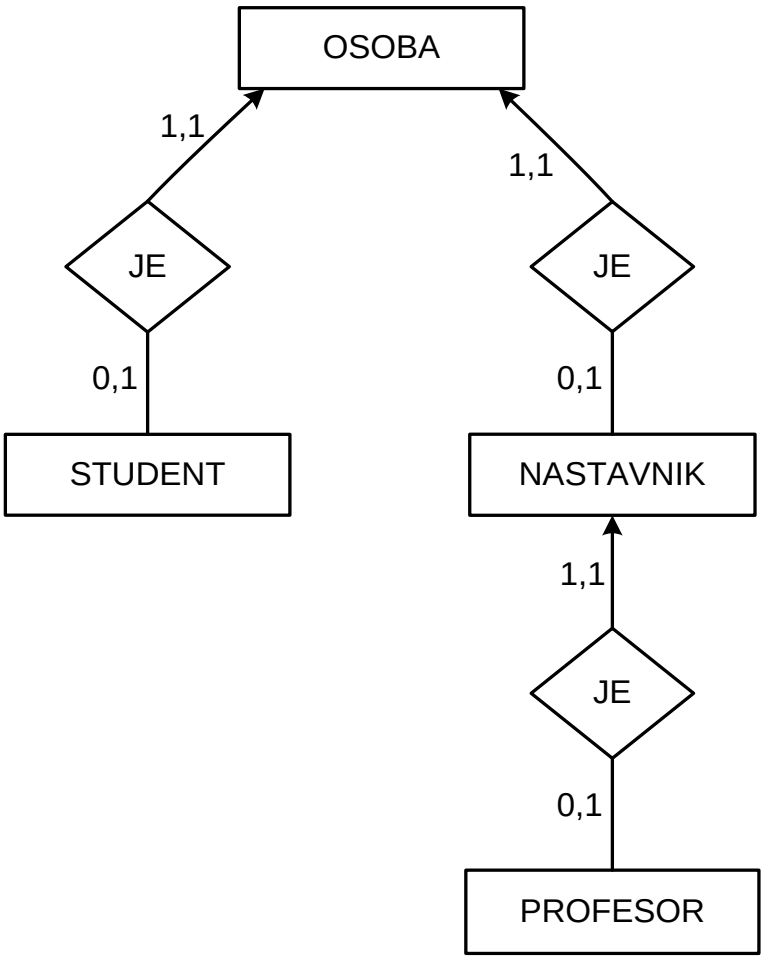
Treći dijagram na Slici 2.6 prikazuje vezu SADRŽI koja povezuje entitet DIO PROIZVODA sa samim sobom. Primjerci entiteta opisuju dijelove proizvoda koji se proizvode u nekoj tvornici. Veza prikazuje odnos između složenijih i jednostavnijih dijelova, dakle da jedan složeniji dio sadrži više jednostavnijih, te da se isti jednostavniji dio pojavljuje se u više složenijih. Funkcionalnost veze je očito M:M. Članstvo entiteta u vezi je najvjerojatnije obavezno, jer ako je dio jednostavan onda se nekamo ugrađuje, a ako je složen onda se od nečega sastoji.

2.3.2. Prikaz pod-tipova i nad-tipova entiteta

Tip entiteta E_1 je *pod-tip* tipa entiteta E_2 ako je svaki primjerak od E_1 također i primjerak od E_2. Pritom E_1 nasljeđuje sve atribute od E_2, no E_1 može imati i dodatne atribute.

Situaciju da je E_1 pod-tip od E_2 crtamo na dijagramu tako da pravokutnik za E_1 smjestimo ispod pravokutnika za E_2, a između crtamo romb koji prikazuje vezu s nazivom JE (engleski IS A). Riječ je o posebnoj vezi s funkcionalnošću 1:1 koja povezuje primjerak entiteta E_1 s njim samim shvaćenim kao primjerkom od E_2. Obavezno se crta strelica prema gore koja naglašava smjer tumačenja veze (primjerak od E_1 *je* primjerak od E_2, a ne obratno).

Primijetimo da je ime veze JE rezervirano, dakle ne bi se smjelo rabiti u druge svrhe osim za povezivanje pod-tipova i nad-tipova. Primijetimo također da se sama veza JE može pojaviti više puta na istom dijagramu ako imamo više parova entiteta koji su u odnosu pod-tip - nad-tip. Time je napravljena iznimka od općenitog pravila da svaka veza unutar sheme mora imati jedinstveno ime; no taj izuzetak ne smeta jer sve pojave veze JE imaju u biti isto značenje.



Slika 2.7: Primjeri za pod-tipove i nad-tipove.

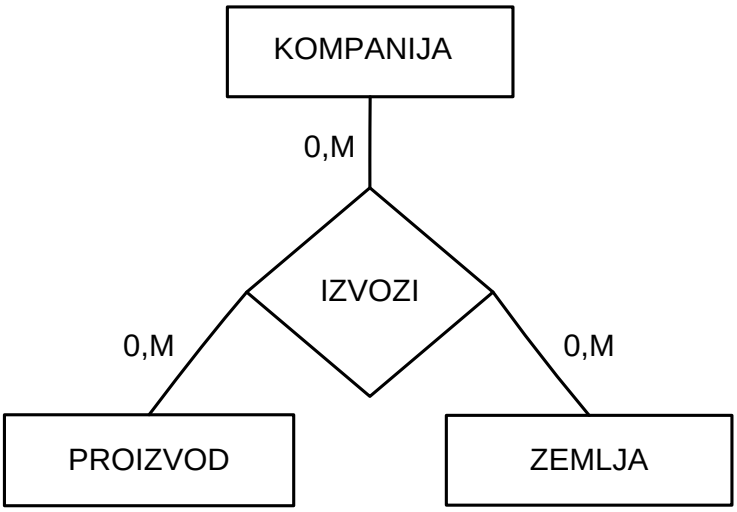
Slika 2.7 sadrži dijagram sheme s pod-tipovima i nad-tipovima. Riječ je o tipovima entiteta za osobe koje se pojavljuju na fakultetu. Najopćenitiji tip je OSOBA; on uključuje atribute koji su primjenjivi za sve osobe, na primjer PREZIME, IME, SPOL, DATUM ROĐENJA … Tipovi STUDENT i NASTAVNIK su pod-tipovi od OSOBA; pritom svaki od njih uključuje neke specifične atribute. Na primjer, STUDENT bi mogao imati atribut GODINA STUDIJA koji nije primjenjiv na NASTAVNIKA a ni na općenitu OSOBU. Slično, NASTAVNIK bi mogao imati svoj specifični atribut DATUM ZAPOŠLJAVANJA.

Dok tip NASTAVNIK uključuje sve asistente, docente i profesore, tip PROFESOR uključuje samo one nastavnike koji imaju zvanje profesora. Vidimo da je PROFESOR pod-tip od NASTAVNIK, a time posredno i pod-tip od OSOBA. PROFESOR nasljeđuje sve atribute NASTAVNIKA, što znači da posredno nasljeđuje i atribute od OSOBE. PROFESOR može imati svoje specifične atribute koji općenito nisu primjenjivi na NASTAVNIKA, a još manje na OSOBU.

2.3.3. Prikaz ternarne veze

Ternarna veza uspostavlja se između tri tipa entiteta. Stanje ternarne veze opisuje se kao skup uređenih trojki primjeraka entiteta koji su trenutno povezani. Svojstva ternarne veze teže je opisati nego kod binarne veze, budući da postoji više vrsta funkcionalnosti, te više kombinacija za obaveznost članstva. Za istu vezu mogu se promatrati tri kardinalnosti, tako da se na jedan od tri načina odaberu dva od tri tipa entiteta, fiksiraju se primjerci entiteta odabranih dvaju tipova, te se gleda broj primjeraka trećeg tipa koji su u vezi s dva fiksirana primjerka.

Ternarna veza prikazuje se na dijagramu slično kao binarna, dakle opet kao romb s upisanim imenom veze. Razlika je da sad iz romba izlaze tri spojnice prema odgovarajućim pravokutnicima, a ne dvije. Uz spojnice su opet upisane oznake kardinalnosti veze, i to tako da kardinalnost u smjeru od odabrana dva tipa prema trećem tipu piše blizu trećem tipu.



Slika 2.8: Primjer ternarne veze.

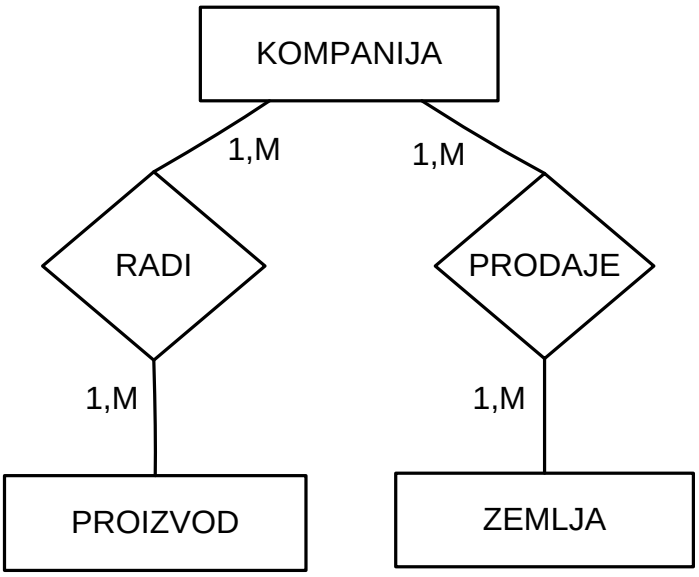
Primjer ternarne veze sa Slike 2.4 odnosi se na podatke o kompanijama, proizvodima koje one proizvode, te zemljama u koje one izvoze svoje proizvode. Stanje te veze bilježi se kao skup

uređenih trojki sastavljenih od primjeraka triju tipova. Pritom jedna trojka znači da dotična kompanija proizvodi dotični proizvod i izvozi ga u dotičnu zemlju.

Sva tri entiteta u vezi na Slici 2.4 imaju obavezno članstvo, jer na primjer svaka kompanija nešto proizvodi i to što proizvodi nekamo izvozi. Funkcionalnost veze je mnogo-naprama-mnogo-naprama-mnogo, dakle M:M:M, jer na primjer za zadani par (kompanija, proizvod) može postojati mnogo zemalja u koje ta kompanija izvozi taj proizvod. Moguće je također da za zadani par (kompanija, proizvod) dotična kompanija ne proizvodi dotični proizvod, tako da je kardinalnost oblika 0,M.

Ternarnu vezu uvodimo samo onda kad se ona ne može rastaviti na dvije binarne. Uzmimo da u primjeru sa Slike 2.8 vrijedi pravilo: *ako kompanija izvozi u neku zemlju, tada ona odmah izvozi sve svoje proizvode u tu zemlju*. Uz ovo pravilo, razmatrana ternarna veza može se zamijeniti s dvije binarne, kao što je prikazano na Slici 2.9. Pritom veza RADI bilježi radi li neka određena kompanija neki određeni proizvod, a PRODAJE bilježi pojavljuje li se neka kompanija na tržištu neke zemlje.

Iz dviju binarnih veza RADI i PRODAJE moguće je reproducirati polaznu ternarnu vezu IZVOZI. Naime, da bismo provjerili je li zadana trojka primjeraka kompanije, proizvoda i zemlje povezana vezom IZVOZI, dovoljno je provjeriti je li odgovarajući par kompanije i proizvoda povezan vezom RADI, te je li istovremeno odgovarajući par kompanije i zemlje povezan vezom PRODAJE. Naglašavamo da je ova zamjena veza ispravna samo od uvjetom da *zaista* vrijedi gore spomenuto pravilo. Ako pravilo ne vrijedi, tada veza mora ostati ternarna.



Slika 2.9: Rastavljanje ternarne veze na dvije binarne.

Slično kao ternarne veze, koje povezuju tri tipa entiteta, mogli bismo promatrati i veze koje povezuju četiri ili pet ili još više tipova entiteta. One bi očito bile još složenije od ternarnih. Za takve veze još bi u većoj mjeri vrijedio savjet da ih po mogućnosti treba rastaviti na nekoliko jednostavnijih veza.

2.4. Zadaci za vježbu

Zadatak 2.1. Očito je da već u specifikaciji iz Odjeljka 2.2.1, pa shodno tome i u konceptualnoj shemi sa Slike 2.4 i 2.5, nedostaju mnogi važni podaci o fakultetu. Predložite nadopunu te sheme: koje entitete, veze i atribute bi po vašem mišljenju trebalo dodati?

Zadatak 2.2. Oblikujte konceptualnu shemu za bazu podataka o knjižnici. Predvidite da ta baza mora pohranjivati podatke o knjigama u knjižnici, članovima knjižnice, te zaposlenicima knjižnice. Također, moraju se evidentirati posudbe knjiga članovima.

Zadatak 2.3. Na osnovu specifikacije koju ste dobili rješavanjem Zadatka 1.5 oblikujte konceptualnu shemu za bazu podataka iz vašeg područja interesa.

Zadatak 2.4. Pročitajte specifikaciju baze podataka o bolnici koja se nalazi na početku Priloga 1. Na osnovu te specifikacije i bez čitanja ostatka priloga sami oblikujte odgovarajuću konceptualnu shemu. Usporedite vaše rješenje s onim u prilogu.

Zadatak 2.5. Pročitajte specifikaciju baze podataka o znanstvenoj konferenciji koja se nalazi na početku Priloga 2. Na osnovu te specifikacije i bez čitanja ostatka priloga sami oblikujte odgovarajuću konceptualnu shemu. Usporedite vaše rješenje s onim u prilogu.

3. Relacijski model - logičko oblikovanje baze podataka

U ovom poglavlju počinjemo govoriti o drugoj fazi oblikovanja baze podataka, a to je logičko oblikovanje. Glavni cilj te faze je stvoriti *relacijsku shemu* baze, dakle shemu koja opisuje logičku strukturu baze u skladu s pravilima relacijskog modela podataka.

Relacijska shema manje je razumljiva korisnicima od konceptualne, budući da su u njoj i entiteti i veze među entitetima pretvoreni u relacije, pa je teško razlikovati jedno od drugog. Ipak, važno svojstvo relacijske sheme je da se ona može više-manje izravno implementirati pomoću današnjih DBMS-a. Zahvaljujući današnjem softveru, od relacijske sheme do njezine konačne implementacije vrlo je kratak put.

U ovom poglavlju najprije ćemo detaljnije opisati svojstva relacijskog modela, te način kako se zapisuje relacijska shema. Zatim ćemo izložiti pravila kojima se konceptualna shema baze dobivena u prethodnom Poglavlju 2 pretvara u relacijsku shemu. Tako dobivena relacijska shema obično još nije u svom konačnom obliku, naime ona se dalje podvrgava postupku dotjerivanja (normalizacije) koji će biti opisan u sljedećem Poglavlju 4.

3.1. Općenito o relacijskom modelu

Relacijski model bio je teoretski zasnovan još krajem 60-tih godina 20. stoljeća, u radovima Edgara Codd-a. Model se dugo pojavljivao samo u akademskim raspravama i knjigama. Prve realizacije na računalu bile su suviše spore i neefikasne. Zahvaljujući intenzivnom istraživanju, te napretku samih računala, efikasnost relacijskih baza postepeno se poboljšavala. Sredinom 80-tih godina 20. stoljeća relacijski model je postao prevladavajući. I danas se ogromna većina DBMS-ova koristi baš tim modelom.

3.1.1. Relacija, atribut, n-torka

Relacijski model zahtijeva da se baza podataka sastoji od skupa pravokutnih tablica - takozvanih *relacija*. Svaka relacija ima svoje ime po kojem je razlikujemo od ostalih u istoj bazi. Jedan stupac relacije obično sadrži vrijednost jednog atributa (za entitet ili vezu) - zato stupac poistovjećujemo s *atributom* i obratno. Atribut ima svoje ime po kojem ga razlikujemo od ostalih u istoj relaciji. Dopušta se da dvije relacije imaju atribute s istim imenom, no tada se podrazumijeva da su to ustvari atributi s istim značenjem. Vrijednosti jednog atributa su podaci iste vrste. Dakle, definiran je *tip* ili skup dozvoljenih vrijednosti za atribut, koji se također zove i zove *domena* atributa. Vrijednost atributa mora biti jednostruka i jednostavna (ne ponavlja se, ne da se rastaviti na dijelove). Pod nekim uvjetima toleriramo situaciju da vrijednost atributa nedostaje (nije upisana). Jedan redak relacije obično predstavlja jedan primjerak entiteta, ili bilježi vezu između dva ili više primjeraka. Redak nazivamo *n-torka*. U jednoj relaciji ne smiju postojati dvije jednake n-torke, naime relaciju tumačimo kao skup n-torki. Broj atributa se zove *stupanj* relacije, a broj n-torki je *kardinalnost* relacije.

Kao primjer, na Slici 3.1 prikazana je relacija STUDENT, s atributima JMBAG, PREZIME, IME, GODINA STUDIJA. Relacija sadrži podatke o studentima koji su upisani na fakultetu.

STUDENT

JMBAG	PREZIME	IME	GODINA STUDIJA
0036398757	Marković	Marko	1
1191203289	Petrović	Petar	2
1192130031	Horvat	Dragica	2
1191205897	Janković	Marija	1
0165043021	Kolar	Ivan	3
0036448430	Grubišić	Katica	3
0246022858	Vuković	Janko	1

Slika 3.1: Relacija s podacima o studentima.

Relacija ne propisuje nikakav redoslijed svojih n-torki i atributa. Dakle, permutiranjem redaka i stupaca tablice dobivamo drukčiji zapis iste relacije.

Uvedena terminologija potječe iz matematike. U komercijalnim DBMS-ovima i pripadnoj dokumentaciji, umjesto „matematičkih“ termina (relacija, n-torka, atribut), češće se koriste neposredni termini (tablica, redak, stupac). Jezik SQL relaciju naziva *table*, njezin atribut *column*, a n-torku *row*.

3.1.2. Kandidati za ključ, primarni ključ

Ključ K relacije *R* je podskup skupa atributa od *R* sa sljedećim svojstvima:
1. Vrijednosti atributa iz *K* jednoznačno određuju n-torku u *R*. Dakle ne mogu u *R* postojati dvije n-torke s istim vrijednostima atributa iz *K*.
2. Ako izbacimo iz *K* bilo koji atribut, tada se narušava svojstvo 1.

Ova svojstva su „vremenski neovisna“, u smislu da vrijede u svakom trenutku bez obzira na povremene unose, promjene i brisanja n-torki.

Na primjer, u relaciji o studentima sa Slike 3.1 atribut JMBAG čini ključ. Kombinacija IMENA i PREZIMENA vjerojatno nije ključ jer se mogu pojaviti osobe s istim imenom i prezimenom.

Budući da su sve n-torke u *R* međusobno različite, *K* uvijek postoji. Naime, skup svih atributa zadovoljava svojstvo 1. Izbacivanjem suvišnih atributa doći ćemo do podskupa koji zadovoljava i svojstvo 2. Dešava se da relacija ima više kandidata za ključ. Tada jedan on njih proglašavamo *primarnim ključem*. Atributi koji sastavljaju primarni ključ zovu se *primarni atributi*. Vrijednost primarnog atributa ne smije ni u jednoj n-torki ostati neupisana.

3.1.3. Relacijska shema, načini njezina zapisivanja

Građu relacije kratko opisujemo takozvanom *shemom relacije*. To je redak koji se sastoji od imena relacije, te popisa imena atributa odvojenih zarezima i zatvorenih u zagrade. Primarni atributi su podvučeni. Na primjer, za relaciju o studentima sa Slike 3.1, shema izgleda ovako:

STUDENT (JMBAG, PREZIME, IME, GODINA STUDIJA) .

Relacijska shema cijele baze zapisuje se tako da se nanižu sheme za sve relacije od kojih se ta baza sastoji. Dakle, shema baze ima onoliko redaka koliko u njoj ima relacija. Na primjer, za bazu podataka o fakultetu opisanu konceptualnom shemom na Slikama 2.4 i 2.5, relacijska shema izgleda kao na Slici 3.5.

Opisani prikaz relacijske sheme vrlo je koncizan i pregledan, no u njemu nedostaju informacije o tipovima atributa. Zato je potrebno da se shema nadopuni rječnikom podataka, dakle popisom svih atributa, s pripadnim tipovima vrijednosti i neformalnim opisom. Tipovi ne moraju biti definirani onako kako će to zahtijevati fizička shema, nego onako kako to prirodno zahtijevaju sami podaci. Za relacijsku shemu sa Slike 3.5, pripadni rječnik podataka mogao bi izgledati kao što se vidi na Slici 3.6.

3.2. Pretvaranje konceptualne sheme u relacijsku shemu

U nastavku objašnjavamo kako se pojedini elementi konceptualne sheme pretvaraju u relacije. Na taj način pokazat ćemo kako se iz cijele konceptualne sheme dobiva relacijska shema.

3.2.1. Pretvorba entiteta i atributa

Svaki tip entiteta prikazuje se jednom relacijom. Atributi tipa postaju atributi relacije. Jedan primjerak entiteta prikazan je jednom n-torkom. Primarni ključ entiteta postaje primarni ključ relacije. Na primjer, tip entiteta PREDMET iz fakultetske baze podataka sa Slika 2.4 i 2.5 prikazuje se relacijom

PREDMET (ŠIFRA PREDMETA, NASLOV, SEMESTAR, ECTS-BODOVI) .

Doduše, sudjelovanje entiteta u vezama može zahtijevati da se u relaciju dodaju još neki atributi koji nisu postojali u odgovarajućem tipu entiteta. No o tome ćemo govoriti u idućim odjeljcima.

Ako pravilo o pretvorbi entiteta primijenimo na cijelu konceptualnu shemu sa Slika 2.4 i 2.5, dobivamo relacijsku shemu prikazanu na Slici 3.2. To je za sada krnja shema, jer se iz nje ne mogu prepoznati veze među entitetima.

STUDENT (JMBAG, PREZIME, IME, GODINA STUDIJA)

PREDMET (ŠIFRA PREDMETA, NASLOV, SEMESTAR, ECTS-BODOVI)

NASTAVNIK (OIB, PREZIME, IME, BROJ SOBE, PLAĆA)

ZAVOD (IME ZAVODA, OPIS DJELATNOSTI)

Slika 3.2: Pretvorba entiteta iz baze podataka o fakultetu u relacije.

3.2.2. Pretvorba veza jedan-naprama-mnogo

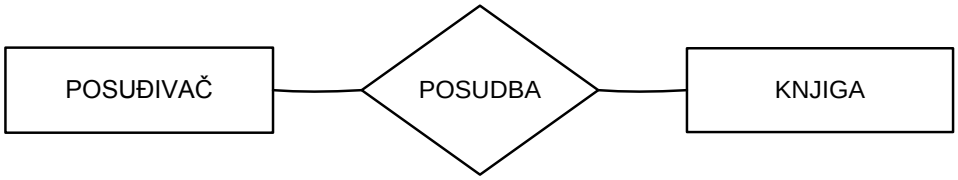
Ako tip entiteta E_1 ima obavezno članstvo u vezi s tipom E_2 koja ima funkcionalnost M:1, tada relacija za E_1 treba uključiti primarne atribute od E_2. Na primjer, ako u konceptualnoj shemi sa Slike 2.4 svaki predmet mora biti ponuđen od nekog zavoda, tada se veza NUDI svodi na to da u relaciju PREDMET ubacimo ključ relacije ZAVOD:

PREDMET (ŠIFRA PREDMETA, IME ZAVODA, NASLOV, SEMESTAR,
ECTS-BODOVI) .

Ključ jedne relacije koji je prepisan u drugu relaciju zove se *strani ključ* (u toj drugoj relaciji).

Lako se vidi da uvođenje stranog ključa zaista omogućuje da se polazna veza reproducira u oba smjera. Na primjer, da bismo za zadani predmet ustanovili koji zavod ga nudi, dovoljno je pogledati odgovarajuću n-torku iz relacije PREDMET, te iz nje pročitati vrijednost atributa IME ZAVODA. Obratno, da bismo za zadani zavod pronašli koje sve predmete on nudi, moramo pretražiti relaciju PREDMET i izdvojiti sve n-torke sa zadanom vrijednošću za IME ZAVODA – svaka od njih opisuje jedan od traženih predmeta.

Ako tip entiteta E_1 nema obavezno članstvo u M:1 vezi s tipom E_2, tada vezu možemo prikazati na prethodni način, dakle uvođenjem stranog ključa, ili uvođenjem nove relacije čiji atributi su primarni atributi od E_1 i E_2.



Slika 3.3: Konceptualna shema baze podataka o knjižnici.

Kao primjer, promotrimo vezu na Slici 3.3 koja prikazuje posuđivanje knjiga u knjižnici. Prvo rješenja za prikaz te veze i pripadnih entiteta izgleda ovako:

POSUĐIVAČ (BROJ ISKAZNICE, PREZIME, IME, ADRESA, …)
KNJIGA (INVENTARSKI BROJ, BROJ ISKAZNICE, NASLOV, …) .

Relacije POSUĐIVAČ i KNJIGA odgovaraju samim tipovima entiteta. Kao primarne ključeve uveli smo atribute BROJ ISKAZNICE (posuđivača) odnosno INVENTARSKI BROJ (knjige). Da bismo prikazali vezu posuđivanja, u relaciju KNJIGA dodali smo kao strani ključ BROJ ISKAZNICE osobe koja je posudila knjigu. Slično kao prije, taj strani ključ omogućuje da se veza POSUDBA reproducira u oba smjera. Primijetimo da će vrijednost atributa BROJ ISKAZNICE ostati prazna u mnogim n-torkama relacije KNJIGA, to jest za sve knjige koje trenutno nisu posuđene.

Drugo rješenje za prikaz iste veze POSUDBA zahtijeva tri relacije, gdje treća relacija služi za prikaz same veze:

POSUĐIVAČ (BROJ ISKAZNICE, PREZIME, IME, ADRESA, …)
KNJIGA (INVENTARSKI BROJ, NASLOV, …)
POSUDBA (INVENTARSKI BROJ, BROJ ISKAZNICE) .

Samo one knjige koje su trenutno posuđene predstavljene su n-torkom u relaciji POSUDBA. Budući da jedna knjiga može biti posuđena samo jednom posuđivaču, ključ u relaciji POSUDBA isti je kao u KNJIGA.

Slično kao strani ključ, posebna relacija za prikaz veze opet omogućuje da se ta veza reproducira u oba smjera. Na primjer, da bismo ustanovili status neke određene knjige, u relaciji POSUDBA tražimo n-torku s odgovarajućom vrijednošću INVENTARSKOG BROJA – ako takvu n-torku ne nađemo tada knjiga nije posuđena, inače je posuđena, i to posuđivaču s upisanom vrijednošću BROJA ISKAZNICE. Obratno, da bismo pronašli koje sve knjige je posudio određeni posuđivač, pretražujemo relaciju POSUDBA i izdvajamo sve n-torke sa odgovarajućom vrijednošću za BROJ ISKAZNICE – svaka od tih n-torki sadrži INVENTARSKI BROJ jedne od posuđenih knjiga.

Glavna prednost uvođenja posebne relacije za prikaz veze umjesto stranog ključa je u tome da na taj način nećemo imati praznih vrijednosti atributa. Posebna relacija za prikaz veze je pogotovo preporučljiva ako veza ima svoje atribute. Na primjer u relaciju POSUDBA mogli bismo uvesti atribut DATUM VRAĆANJA.

Izložena pravila za prikaz veze s funkcionalnošću M:1 analogno se primjenjuju i na veze s funkcionalnošću 1:M i 1:1. Ako ta pravila primijenimo na našu bazu podataka o fakultetu, dakle na sve M:1, 1:M i 1:1 veze sa Slike 2.4, tada se relacijska shema sa Slike 3.2 pretvara u shemu prikazanu na Slici 3.4.

STUDENT (JMBAG, PREZIME, IME, GODINA STUDIJA)

PREDMET (ŠIFRA PREDMETA, NASLOV, IME ZAVODA, OIB NASTAVNIKA, SEMESTAR, ECTS-BODOVI).

NASTAVNIK (OIB, PREZIME, IME, IME ZAVODA, BROJ SOBE, PLAĆA)

ZAVOD (IME ZAVODA, OIB PROČELNIKA, OPIS DJELATNOSTI)

Slika 3.4: Pretvorba veza jedan-naprama-mnogo za bazu o fakultetu.

Na Slici 3.4 sve novosti u odnosu na Sliku 3.2 označene su crvenom bojom. Vidimo da su u relacije koje odgovaraju entitetima dodani strani ključevi koji omogućuju reproduciranje svih M:1, 1:M i 1:1 veza. Atribut IME ZAVODA u relaciji PREDMET određuje zavod koji nudi dotični predmet. Slično, atribut IME ZAVODA unutar relacije NASTAVNIK bilježi kojem zavodu pripada taj nastavnik. Atribut OIB NASTAVNIKA u relaciji PREDMET određuje nastavnika koji predaje taj predmet. Atribut OIB PROČELNIKA unutar relacije ZAVOD odnosi se na pročelnika dotičnog zavoda. Ipak, dobivena shema još uvijek ne bilježi M:M vezu UPISAO.

3.2.3. Pretvorba veza mnogo-naprama-mnogo

Veza s funkcionalnošću M:M uvijek se prikazuje posebnom relacijom, koja se sastoji od primarnih atributa za oba tipa entiteta zajedno s eventualnim atributima veze. Na primjer, veza UPISAO iz fakultetske baze sa Slike 2.4 prikazuje se relacijom:

UPISAO (JMBAG, ŠIFRA PREDMETA, DATUM UPISA, OCJENA) .

Činjenica da je jedan student upisao jedan predmet prikazuje se jednom n-torkom u relaciji UPISAO. Ključ za UPISAO je očito složen, to jest sastoji se od kombinacije atributa JMBAG i ŠIFRA PREDMETA. Naime, budući da isti student može upisati više predmeta, a isti predmet može biti upisan od više studenata, ni jedan od tih atributa sam nije dovoljan da jednoznačno odredi n-torku.

Lako se vidi da opisana relacija zaista omogućuje da se pripadna M:M veza reproducira u oba smjera. Na primjer, da bismo ustanovili koje je sve predmete upisao zadani student, pretražujemo relaciju UPISAO i izdvajamo sve n-torke s odgovarajućom vrijednošću za JMBAG – svaka od izdvojenih n-torki sadrži ŠIFRU PREDMETA kojeg je taj student upisao. Obratno, da bismo pronašli sve studente koji su upisali zadani predmet, pretražujemo relaciju UPISAO ali tako da izdvojimo n-torke s odgovarajućom vrijednošću za ŠIFRU PREDMETA – svaka od izdvojenih n-torki otkriva JMBAG jednog od traženih studenata.

STUDENT (JMBAG, PREZIME, IME, GODINA STUDIJA)

PREDMET (ŠIFRA PREDMETA, NASLOV, IME ZAVODA, OIB NASTAVNIKA, SEMESTAR, ECTS-BODOVI).

NASTAVNIK (OIB, PREZIME, IME, IME ZAVODA, BROJ SOBE, PLAĆA)

ZAVOD (IME ZAVODA, OIB PROČELNIKA, OPIS DJELATNOSTI)

UPISAO (JMBAG, ŠIFRA PREDMETA, DATUM UPISA, OCJENA)

Slika 3.5: Relacijska shema za bazu podataka o fakultetu.

Ubacivanjem relacije UPISAO u shemu sa Slike 3.4, dobivamo shemu prikazanu na Slici 3.5. To je napokon cjelovita relacijska shema za našu bazu podataka o fakultetu, koja je ekvivalentna konceptualnoj shemi budući da sadrži sve entitete, veze i atribute kao na Slikama 2.4 i 2.5.

3.2.4. Sastavljanje rječnika podataka

Rekli smo da relacijska shema na koncizan način opisuje logičku strukturu baze u skladu s relacijskim modelom. Zaista, iz te sheme se vidi od kojih se relacija sastoji cijela baza, te od kojih se atributa sastoji svaka pojedina relacija. Ipak, uočili smo da se iz sheme ne vide tipovi tributa. Također, koji put se može desiti da iz imena pojedinih atributa nije lako odrediti njihovo značenje.

IME ATRIBUTA	TIP	OPIS
JMBAG	Niz od točno 10 znamenki	Šifra koja jednoznačno određuje studenta
PREZIME	Niz znakova	Prezime osobe
IME	Niz znakova	Ime osobe
GODINA STUDIJA	Cijeli broj između 1 i 5	Godina koju je student upisao
ŠIFRA PREDMETA	Niz od točno 5 znamenki	Šifra koja jednoznačno određuje predmet
NASLOV	Niz znakova	Naslov predmeta
SEMESTAR	„zimski“ ili „ljetni“	Semestar u kojem se predmet predaje
ECTS-BODOVI	Mali cijeli broj	Bodovi koje student dobiva ako položi predmet
OIB	Niz od točno 11 znamenki	Šifra koja jednoznačno određuje osobu
BROJ SOBE	Niz od točno 3 znamenke	Određuje sobu u kojoj sjedi nastavnik
PLAĆA	Cijeli broj	Neto plaća osobe u kunama
IME ZAVODA	Niz znakova	Jednoznačno određuje zavod unutar fakulteta
OPIS DJELATNOSTI	Niz znakova	Tekst koji opisuje djelatnost zavoda
DATUM UPISA	Datum	Datum kad je određeni student upisao određeni predmet
OCJENA	Cijeli broj između 2 i 5	Ocjena koju je određeni student dobio iz određenog predmeta

Slika 3.6: Rječnik podataka za bazu podataka o fakultetu.

Ovi nedostaci informacije nadoknađuju se tako da se sastavi rječnik podataka, te da se on priloži uz shemu. *Rječnik podataka* je tabela u kojoj su popisani svi atributi, te je za svakog od njih definiran tip i opisano značenje.

Rječnik podataka stvaramo tako da prođemo svim relacijama iz sheme, te upišemo u tabelu sve atribute na koje naiđemo, s time da isti atribut upišemo samo jednom. Zatim atributima u tabeli na što razumljiviji način opišemo njihovo značenje, te im odredimo tip.

Kod određivanja tipova treba uzeti u obzir da današnji DBMS-i očekuju da će atributi biti cijeli ili realni brojevi, ili znakovi, ili nizovi znakova, eventualno datumi ili novčani iznosi. Znači, za svaki atribut treba odrediti tip u jednom od ovih oblika, s time da taj tip možemo preciznije podesiti uvođenjem raznih ograničenja.

Primjenom opisanog postupka na našu bazu podataka o fakultetu dobivamo rječnik podataka koji je prikazan na Slici 3.6. Taj rječnik služi kao nadopuna relacijske sheme sa Slike 3.5. Budući da se još uvijek bavimo logičkom razinom oblikovanja, ograničenja vezana uz tip biramo na najprirodniji način, bez obzira što DBMS u konačnoj implementaciji možda neće moći uvažiti neka od njih ili će nametnuti neka svoja. Na primjer, tip atributa REGISTARSKA OZNAKA (automobila) definiramo kao niz od 8 znakova gdje su prva dva znaka slova, iduća četiri znamenke, a zadnja dva opet slova, makar DBMS možda neće biti u stanju obavljati tako detaljnu kontrolu znak-po-znak. Slično, tip atributa PREZIME definiramo kao niz znakova proizvoljne duljine, makar će DBMS sigurno nametnuti neku gornju ogradu na duljinu.

3.3. Pretvaranje složenijih veza u relacije

Dosad izložena pravila obično su dovoljna za pretvorbu konceptualne sheme u relacijsku. Točnije, ona su dovoljna kod jednostavnijih konceptualnih shema koje sadrže samo binarne veze. No ako se pojavljuju i složenije veze, tada su nam potrebna dodatna pravila. U ovom potpoglavlju opisujemo kako se svaka od složenijih vrsta veza može pretvoriti u relacije.

3.3.1. Pretvorba involuiranih veza

Involuirane veze prikazuju se pomoću relacija slično kao binarne veze. Poslužit ćemo se primjerima sa Slike 2.6.

Tip entiteta OSOBA i 1:1 vezu U BRAKU S najbolje je (zbog neobaveznosti) prikazati pomoću dvije relacije:

OSOBA (<u>OIB</u>, PREZIME, IME, ADRESA, …)
BRAK (<u>OIB MUŽA</u>, OIB ŽENE, DATUM VJENČANJA) .

Prva relacija OSOBA odgovara samom tipu entiteta za osobe, dakle jedna njezina n-torka opisuje jednu osobu. Kao primarni ključ za identificiranje osoba odabrali smo OIB. Relacija BRAK zapisuje bračnu vezu, dakle jedna njezina n-torka bilježi par od jedne muške i jedne ženske osobe koje su u braku. Kao primarni ključ u relaciji BRAK odabrali OIB MUŽA. Zaista, budući da su prošli brakovi zaboravljeni a poligamija zabranjena, identifikator muškog supružnika jednoznačno određuje cijeli brak. No isto tako smo kao primarni ključ mogli odabrati i OIB ŽENE.

Lako se vidi da relacija BRAK zaista omogućuje da se polazna veza U BRAKU S reproducira u oba smjera. Na primjer, da bismo za zadanu mušku osobu utvrdili bračni status, u relaciji BRAK tražimo n-torku sa zadanim OIB-om MUŽA - ako takve n-torke nema, tad je dotični muškarac neoženjen, inače iz n-torke čitamo OIB njegove ŽENE. Sasvim analogno postupamo da bismo pronašli bračni status i eventualnog bračnog druga zadane ženske osobe.

Tip entiteta SURADNIK i 1:M vezu JE ŠEF ZA prikazujemo jednom relacijom:

SURADNIK (MATIČNI BROJ, MATIČNI BROJ ŠEFA, PREZIME, IME, …) .

Riječ je o relaciji koja odgovara samom tipu entiteta za suradnike. No, da bismo prikazali vezu između šefova i suradnika, u istu relaciju smo dodali novi atribut, ustvari strani ključ, MATIČNI BROJ ŠEFA. Taj novi atribut je iste vrste kao MATIČNI BROJ, ali se odnosi na šefa dotičnog suradnika. Ovo neće uzrokovati mnogo praznih vrijednosti atributa, budući da većina suradnika ima šefa.

Ubacivanje atributa MATIČNI BROJA ŠEFA očigledno omogućuje da se veza JE ŠEF ZA reproducira u oba smjera. Na primjer, da bismo za zadanog suradnika ustanovili tko je njemu šef, iz odgovarajuće n-torke relacije SURADNIK čitamo MATIČNI BROJ ŠEFA. Obratno, da bismo za zadanog šefa ustanovili koji su sve suradnici njemu podređeni, pretražujemo relaciju SURADNIK i izdvajamo sve n-torke s odgovarajućom vrijednošću za MATIČNI BROJ ŠEFA – svaka izdvojena n-torka opisuje jednog podređenog suradnika.

Tip entiteta DIO PROIZVODA i M:M vezu SADRŽI moramo prikazati pomoću dvije relacije:

DIO PROIZVODA (BROJ DIJELA, IME DIJELA, OPIS, …)
SADRŽI (BROJ DIJELA SLOŽENOG, BROJ DIJELA JEDNOSTAVNOG, KOLIČINA).

Prva relacija DIO PROIZVODA odgovara samom tipu entiteta za dijelove proizvoda. Kao primarni ključ za identificiranje dijelova uveli smo atribut BROJ DIJELA. Relacija SADRŽI zapisuje vezu između složenih i jednostavnih dijelova, dakle jedna njezina n-torka bilježi par od jednog složenog i jednog jednostavnog dijela, takvih da taj složeni dio sadrži u sebi taj jednostavni dio. Atributi BROJ DIJELA SLOŽENOG i BROJ DIJELA JEDNOSTAVNOG iste su vrste kao atribut BROJ DIJELA, ali se odnose na složeni odnosno jednostavni dio u paru. Dodali smo i atribut KOLIČINA koji kaže koliko komada tih jednostavnih dijelova ulazi u taj složeni dio. Primarni ključ u relaciji SADRŽI mora biti sastavljen od oba broja dijela, i to zato što jedan složeni dio može sadržavati više jednostavnih, a jedan jednostavni dio se može pojaviti u više složenih.

Lako se vidi da opisana relacija zaista omogućuje da se pripadna M:M veza reproducira u oba smjera. Na primjer, da bismo ustanovili koje sve jednostavne dijelove sadrži zadani složeni dio, pretražujemo relaciju SADRŽI i izdvajamo sve n-torke s fiksiranom vrijednošću za BROJ DIJELA SLOŽENOG – svaka od izdvojenih n-torki sadrži BROJ DIJELA JEDNOSTAVNOG za jedan od uključenih jednostavnih dijelova. Obratno, da bismo pronašli sve složene dijelove koji sadrže zadani jednostavni dio, pretražujemo relaciju SADRŽI ali tako da izdvojimo n-torke s fiksiranom vrijednošću za BROJ DIJELA JEDNOSTAVNOG – svaka od izdvojenih n-torki otkriva BROJ DIJELA SLOŽENOG za jedan od traženih složenih dijelova.

3.3.2. Pretvorba pod-tipova i nad-tipova

Pod-tip nekog tipa entiteta prikazuje se posebnom relacijom koja sadrži primarne atribute nadređenog tipa i atribute specifične za taj pod-tip. Na primjer, hijerarhija tipova za osobe sa Slike 2.7 prikazuje se sljedećim relacijama.

OSOBA (OIB, … atributi zajednički za sve tipove osoba …)
STUDENT (OIB, … atributi specifični za studente …)
NASTAVNIK (OIB, … atributi specifični za nastavnike …)
PROFESOR (OIB, … atributi specifični za profesore …) .

Dakle svaka od ovih relacija odgovara jednom od tipova entiteta. Kao primarni ključ za identificiranje osobe uveli smo OIB. Veza JE između pod-tipova i nad-tipova uspostavlja se na osnovu pojavljivanja iste vrijednosti OIB u raznim relacijama. Dakle n-torke u različitim relacijama s istom vrijednošću OIB-a odnose se na istu osobu.

Primijetimo da relacijski model zapravo nije pogodan za prikaz pod-tipova i nad-tipova. Naime, osnovna ideja relacijskog modela je jednostavnost i jednoobraznost strukture. To znači da se relacijska baza mora sastojati isključivo od relacija (tablica), a između tih relacija ne može postojati nikakva struktura, pa tako ni hijerarhija. Predloženi način prikaza zapravo omogućuje da se odnos pod-tipova i nad-tipova po potrebi reproducira unutar *aplikacija*. Pritom sama baza, odnosno njezina logička struktura, nije svjesna hijerarhije tipova.

Odnosi pod-tipova i nad-tipova entiteta, te nasljeđivanje atributa, puno prirodnije i izravnije bi se trebali moći realizirati u objektnom modelu za baze podataka. No objektne baze za sada još nisu u širokoj uporabi. U međuvremenu, moramo se zadovoljiti ovakvim polovičnim rješenjem.

3.3.3. Pretvorba ternarnih veza

Ternarna veza gotovo uvijek se prikazuje posebnom relacijom, koja sadrži primarne atribute svih triju tipova entiteta zajedno s eventualnim atributima veze. Za primjer sa Slike 2.8 imamo relacijsku shemu koja se sastoji od četiri relacije:

KOMPANIJA (IME KOMPANIJE, …)
PROIZVOD (IME PROIZVODA, …)
ZEMLJA (IME ZEMLJE, …)
IZVOZI (IME KOMPANIJE, IME PROIZVODA, IME ZEMLJE, …) .

Prve tri relacije odgovaraju samim tipovima entiteta, dakle u ovom slučaju kompanijama, proizvodima i zemljama. Kao primarne ključeve u te tri relacije uveli smo atribute IME KOMPANIJE, IME PROIZVODA, odnosno IME ZEMLJE. Četvrta relacija IZVOZI prikazuje promatranu ternarnu vezu, dakle jedna njezina n-torka izražava činjenicu da se određeni proizvod određene kompanije izvozi u određenu zemlju. Primarni ključ u IZVOZI je trojka atributa IME KOMPANIJE, IME PROIZVODA i IME ZEMLJE. Naime, ni jedna dva od ta tri atributa ne određuju n-torku, jer na primjer za zadanu kompaniju i zadani proizvod može biti više zemalja u koje ta kompanija izvozi taj proizvod, i tako dalje. Kod ternarnih veza čija funkcionalnost nije M:M:M broj primarnih atributa može biti manji.

Slično kao u prethodnim slučajevima, i ovdje se lako vidi da uvedena relacija za prikaz veze zaista omogućuje da se ta veza reproducira u svim smjerovima. Na primjer, da bismo za zadanu kompaniju i zadani proizvod ustanovili u koje sve zemlje ta kompanija izvozi taj proizvod, pretražujemo relaciju IZVOZI i izdvajamo sve n-torke s fiksiranim vrijednostima za IME KOMPANIJE i IME PROIZVODA – svaka od izdvojenih n-torki otkriva IME ZEMLJE za jednu od traženih zemalja. Analognim pretraživanjima mogli bismo za zadanu kompaniju i zemlju pronaći popis proizvoda koje ta kompanija izvozi u tu zemlju, odnosno za zadani proizvod i zemlju popis kompanija koje taj proizvod izvoze u tu zemlju.

3.4. Zadaci za vježbu

Zadatak 3.1. Rješavanjem Zadatka 2.1 dobili ste nadopunjenu konceptualnu shemu baze podataka o fakultetu. Sad tu nadopunjenu konceptualnu shemu pretvorite u (nadopunjenu) relacijsku shemu. Također sastavite odgovarajući rječnik podataka. Koje nove relacije ili novi atributi su se pojavili u odnosu na Slike 3.5 i 3.6?

Zadatak 3.2. Rješavanjem Zadatka 2.2 dobili ste konceptualnu shemu za bazu podataka o knjižnici. Sad tu konceptualnu shemu pretvorite u relacijsku shemu. Također sastavite odgovarajući rječnik podataka.

Zadatak 3.3. Na osnovu konceptualne sheme koju ste dobili rješavanjem Zadatka 2.3, oblikujte relacijsku shemu i rječnik podataka za bazu podataka iz vašeg područja interesa.

Zadatak 3.4. U Prilogu 1 pronađite konceptualnu shemu baze podataka o bolnici. Na osnovu te sheme i bez čitanja ostatka priloga sami oblikujte odgovarajuću relacijsku shemu i rječnik podataka. Usporedite vaše rješenje s onim u prilogu.

Zadatak 3.5. U Prilogu 2 pronađite konceptualnu shemu baze podataka o znanstvenoj konferenciji. Na osnovu te sheme i bez čitanja ostatka priloga sami oblikujte odgovarajuću relacijsku shemu i rječnik podataka. Usporedite vaše rješenje s onim u prilogu.

4. Normalizacija - nastavak logičkog oblikovanja baze podataka

U ovom poglavlju nastavljamo govoriti o drugoj fazi oblikovanja baze podataka, dakle opet govorimo o logičkom oblikovanju. Opisat ćemo postupak daljnjeg dotjerivanja polazne relacijske sheme dobivene primjenom pravila iz prethodnog poglavlja. Dotjerivanje je potrebno zato što polazna relacijska shema može sadržavati određene nepravilnosti. Te nepravilnosti treba otkloniti prije nego što krenemo u fizičko oblikovanje. Sam postupak dotjerivanja zove se *normalizacija*.

U daljnjem tekstu najprije opisujemo jednostavniji i u praksi više rabljeni dio normalizacije, a to je prevođenje u prvu, drugu i treću normalnu formu. Zatim se bavimo složenijim dijelom normalizacije koji je vezan uz Boyce-Codd-ovu i četvrtu normalnu formu. Na kraju raspravljamo o tome zašto je normalizacija uopće potrebna, te možemo li od nje odustati.

4.1. Prva, druga i treća normalna forma

Teorija normalizacije zasnovana je na pojmu *normalnih formi*. Svaka normalna forma predstavlja određeni „zahtjev na kvalitetu“ kojeg bi relacija trebala zadovoljavati. Što je normalna forma viša, njezini zahtjevi su stroži. Relacije dobivene postupkom iz Poglavlja 2 i 3 morale bi u najmanju ruku biti u prvoj normalnoj formi. No Edgar Codd je u svojim radovima iz ranih 1970-tih godina definirao daljnja mjerila kvalitete koja su izražena drugom i trećom normalnom formom.

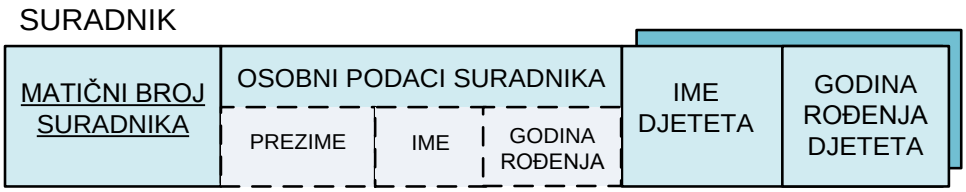
4.1.1. Pod-zapisi, ponavljajuće skupine, prevođenje podataka u prvu normalnu formu

Kad smo govorili o relacijskom modelu za bazu podataka, naglasili smo da vrijednost atributa unutar relacije (sadržaj jedne kućice unutar tablice) mora biti *jednostruka* i *jednostavna*. Dakle u jednu kućicu ne može se upisati više vrijednosti, nego najviše jedna vrijednost. Također, ta upisana vrijednost ne smije biti složena, nego takva da ju sama baza smatra nedjeljivom.

Ovo svojstvo jednostrukosti i jednostavnosti zove se jednim imenom svojstvo *prve normalne forme* (oznaka: 1NF). Dakle, kažemo da relacija jeste u 1NF, jer su vrijednosti njezinih atributa jednostruke i nedjeljive. Primijetimo da 1NF zapravo ne predstavlja nikakav posebni zahtjev na relaciju, budući da je to svojstvo već ugrađeno u sam relacijski model i definiciju relacije. Dakle, u relacijskoj bazi podataka niti ne može postojati relacija koja ne bi već otpočetka bila u 1NF. Pojam 1NF zapravo je izmišljen zbog drugih modela podataka gdje podaci ne moraju biti normalizirani čak ni u tom najjednostavnijem smislu.

Ukoliko bazu podataka oblikujemo na način opisan u Poglavlju 2, tada vjerojatno nećemo imati prilike susresti se s nenormaliziranim podacima. Naime, oblikovanjem konceptualne sheme dobit ćemo entitete i veze čiji atributi već imaju jednostruke i jednostavne vrijednosti (inače bismo morali uvesti više entiteta i nove veze među njima, ili više atributa). Pretvorbom takvih entiteta i veza po pravilima iz Poglavlja 3 dobit ćemo korektne relacije, dakle one u 1NF.

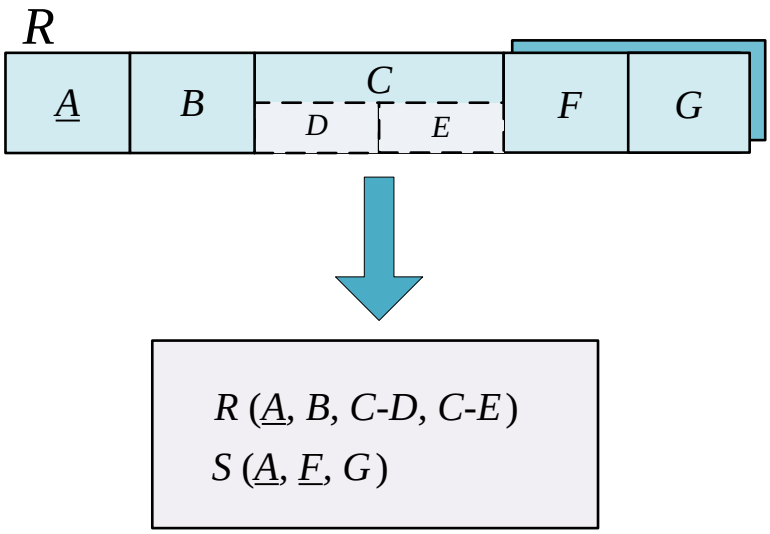
Znanje o 1NF potrebno nam je prvenstveno u situaciji kad neke podatke već imamo pohranjene u nekom ne-relacijskom obliku, te ih želimo izravno prebaciti u relacijsku bazu. Zapisi (slogovi) u ne-relacijskim kolekcijama podataka mogu biti nenormalizirani, naime oni mogu sadržavati takozvane *pod-zapise* ili *ponavljajuće skupine*. Na primjer, u nekom poduzeću mogli bismo imati datoteku s podacima o suradnicima. Zapis o jednom suradniku mogao bi imati oblik prikazan Slikom 4.1. Vidimo da se tu pojavio pod-zapis s osobnim podacima samog suradnika, te ponavljajuća skupina koja se ponavlja za svako dijete tog suradnika. Podaci očito nisu u 1NF.



Slika 4.1: Nenormalizirani zapis o suradniku.

Da bismo nenormalizirane podatke pohranili u relacijskoj bazi, moramo ih prevesti u 1NF. Dakle, uvođenjem dovoljnog broja relacija i atributa moramo eliminirati sve ponavljajuće skupine i pod-zapise. Postupak prevođenja izgleda otprilike ovako.

- Uvodi se jedna osnovna relacija, te onoliko pomoćnih relacija koliko ima ponavljajućih skupina.
- Fiksni dio zapisa prikazuje se kao jedna n-torka u osnovnoj relaciji, s time da je svaki pod-zapis rastavljen na nekoliko zasebnih atributa.
- Svaka pojava ponavljajuće skupine unutar zapisa prikazuje se kao zasebna n-torka u odgovarajućoj pomoćnoj relaciji. Zbog čuvanja veze s polaznim zapisom u tu n-torku se prepisuje i neki od identifikacijskih podataka iz fiksnog dijela zapisa.



Slika 4.2: Shematski prikaz prevođenja u 1NF.

Postupak prevođenja u 1NF shematski je prikazan Slikom 4.2. Na toj slici vidi se pretvorba jednog pod-zapisa i jedne ponavljajuće skupine. Ako imamo više pod-zapisa ili više ponavljajućih skupina, tada se zahvati sa Slike 4.2 moraju iterirati.

Ako ovaj postupak prevođenja u 1NF primijenimo na naše zapise o suradnicima sa Slike 4.1, dobit ćemo sljedeće dvije relacije.

SURADNIK (MATIČNI BROJ SURADNIKA, PREZIME SURADNIKA,
IME SURADNIKA, GODINA ROĐENJA SURADNIKA)
DIJETE (MATIČNI BROJ SURADNIKA, IME DJETETA,
GODINA ROĐENJA DJETETA) .

Na primjer, podaci o suradniku koji ima troje djece bit će prikazani jednom n-torkom u relaciji SURADNIK i trima n-torkama u relaciji DIJETE (po jedna za svako dijete). Veza između te četiri n-torke uspostavlja se na osnovu iste vrijednosti MATIČNOG BROJA SURADNIKA.

4.1.2. Funkcionalne ovisnosti između atributa ili skupina atributa

Većina normalnih formi zasnovana je na pojmu funkcionalne ovisnosti između atributa ili skupina atributa. Zato, prije nego što počnemo govoriti o drugoj, trećoj ili višim normalnim formama, moramo naučiti što je to funkcionalna ovisnost.

Za zadanu relaciju *R*, atribut *B* od *R* je *funkcionalno ovisan* o atributu *A* od *R* (oznaka: $A \rightarrow B$) ako vrijednost od *A* jednoznačno određuje vrijednost od *B*. Dakle ako u isto vrijeme postoje u *R* dvije n-torke s jednakom vrijednošću od *A*, tada te n-torke moraju imati jednaku vrijednost od *B*. Analogna definicija primjenjuje se i za slučaj kad su *A* i *B* složeni atributi (dakle skupine atributa).

Kao primjer za postojanje funkcionalnih ovisnosti, promotrimo sljedeću (namjerno loše oblikovanu) relaciju:

UPISAO (JMBAG, ŠIFRA PREDMETA, NASLOV PREDMETA, OIB NASTAVNIKA,
BROJ SOBE NASTAVNIKA, OCJENA) .

Riječ je opet o relaciji koja bilježi da su studenti upisali predmete na fakultetu i dobili iz njih odgovarajuće ocjene. Uzimamo da JMBAG jednoznačno određuje studenta, a ŠIFRA PREDMETA jednoznačno određuje predmet. Pretpostavljamo da svaki predmet ima jednog nastavnika, a svaki nastavnik jednu sobu. Također smatramo da OIB NASTAVNIKA jednoznačno određuje nastavnika. Vidimo da u relaciji postoji velik broj funkcionalnih ovisnosti. Navodimo neke od njih:

(JMBAG, ŠIFRA PREDMETA) → OCJENA ,
ŠIFRA PREDMETA → NASLOV PREDMETA ,
ŠIFRA PREDMETA → OIB NASTAVNIKA ,
ŠIFRA PREDMETA → BROJ SOBE NASTAVNIKA ,
OIB NASTAVNIKA → BROJ SOBE NASTAVNIKA .

U slučaju funkcionalne ovisnosti o skupini atributa, uvodi se još i dodatni pojam potpune funkcionalne ovisnosti. Za zadanu relaciju *R*, atribut *B* od *R* je *potpuno* funkcionalno ovisan o (složenom) atributu *A* od *R* ako vrijedi: *B* je funkcionalno ovisan o *A*, no *B* nije funkcionalno ovisan ni o jednom pravom podskupu od *A*.

Važna vrsta funkcionalne ovisnosti koju redovito susrećemo u svakoj relaciji je ovisnost atributa o ključu. Očito, svaki atribut relacije je funkcionalno ovisan o ključu, no ta ovisnost ne mora biti potpuna. Na primjer, u prethodnoj relaciji UPISAO, atribut OCJENA je potpuno funkcionalno ovisan o primarnom ključu (JMBAG, ŠIFRA PREDMETA). S druge strane, NASLOV PREDMETA, OIB NASTAVNIKA, te BROJ SOBE NASTAVNIKA nisu potpuno funkcionalno ovisni o primarnom ključu, budući da su ovisni samo o ŠIFRA PREDMETA, a ne i o JMBAG-u.

4.1.3. Parcijalne ovisnosti, prevođenje relacije u drugu normalnu formu

Vidjeli smo da u prethodnoj relaciji UPISAO postoje atributi NASLOV PREDMETA , OIB NASTAVNIKA i BROJ SOBE NASTAVNIKA koji nisu potpuno funkcionalno ovisni o primarnom ključu. Za njih kažemo da su *parcijalno ovisni* o tom ključu.

Parcijalna ovisnost smatra se nepoželjnim svojstvom. Naime ona može uzrokovati teškoće kod manipuliranja s podacima, kao što će biti pokazano u Odjeljku 4.3.1. Da bi se parcijalne ovisnosti mogle „proskribirati“, uvodi se pojam druge normalne forme.

Relacija je u *drugoj normalnoj formi* (oznaka: 2NF) ako je svaki njezin ne-primarni atribut potpuno funkcionalno ovisan o primarnom ključu. Drugim riječima, relacija je u 2NF ako u njoj nema parcijalnih ovisnosti atributa o primarnom ključu.

Primijetimo da definicija 2NF zaista ima smisla jedino ako je primarni ključ relacije složen. Relacija s jednostavnim ključem (dakle ključem koji se sastoji samo od jednog atributa) automatski je u 2NF.

Relacija UPISAO nije u 2NF jer, kao što smo već rekli, u njoj postoje parcijalne ovisnosti:

ŠIFRA PREDMETA → NASLOV PREDMETA ,
ŠIFRA PREDMETA → OIB NASTAVNIKA ,
ŠIFRA PREDMETA → BROJ SOBE NASTAVNIKA .

U skladu s prethodno rečenim, relacija koja nije u 2NF loše je oblikovana, te se preporuča da se ona prevede u 2NF. Postupak prevođenja svodi se na razbijanje polazne relacije u barem dvije, tako da se prekinu nepoželjne parcijalne ovisnosti, no pritom se sačuvaju svi semantički odnosi među podacima. Preciznije:

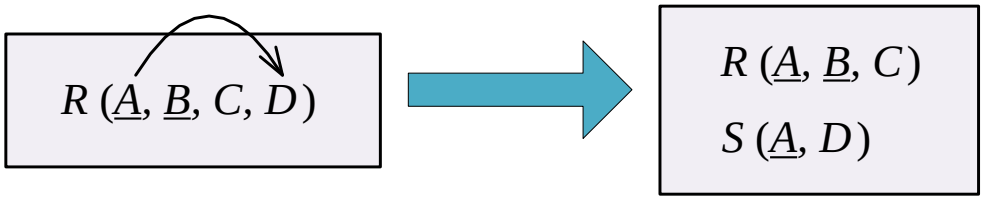
- Uz polaznu relaciju dodaje se onoliko novih relacija koliko ima različitih dijelova primarnog ključa koji sudjeluju u parcijalnim ovisnostima.
- Iz polazne relacije *izbacuju se i prebacuju* u nove svi oni atributi koji su parcijalno ovisni o ključu.
- Pritom u jednu novu relaciju idu oni atributi koji su ovisni o istom dijelu primarnog ključa.
- Uz prebačene atribute, u novu relaciju *prepisuje se* i odgovarajući dio primarnog ključa, te on postaje ključ u toj novoj relaciji.

U našem konkretnom primjeru, postupak prevođenja u 2NF razbit će polaznu relaciju UPISAO u sljedeće dvije relacije:

UPISAO (JMBAG, ŠIFRA PREDMETA, OCJENA)
PREDMET (ŠIFRA PREDMETA, NASLOV PREDMETA, OIB NASTAVNIKA,
BROJ SOBE NASTAVNIKA) .

Zaista, u polaznoj verziji UPISAO sve parcijalne ovisnosti kreću iz istog dijela ključa, a to je ŠIFRA PREDMETA. Zato postupak prevođenja stvara samo jednu novu relaciju, te su u nju prebačeni svi parcijalno-ovisni atributi zajedno sa ŠIFROM PREDMETA. Vidimo da ŠIFRA PREDMETA postaje ključ u novoj relaciji. Ta nova relacija zapravo opisuje predmet na fakultetu, pa je zato dobila ime PREDMET.

Nakon opisanog prevođenja, obje relacije su u 2NF. Naime, u preostaloj verziji UPISAO više nema parcijalnih ovisnosti pa ona zadovoljava definiciju 2NF. Također, PREDMET ima jednostavan ključ pa je automatski u 2NF.



Slika 4.3: Shematski prikaz prevođenja u 2NF.

Postupak prevođenja u 2NF shematski je prikazan Slikom 4.3. Na toj slici pretpostavljeno je da polazna relacija ima samo jednu parcijalnu ovisnost. Ako imamo više parcijalnih ovisnosti iz istog dijela ključa, ili parcijalne ovisnosti iz raznih dijelova ključa, tada se zahvati sa Slike 4.3 moraju iterirati.

4.1.4. Tranzitivne ovisnosti, prevođenje relacije u treću normalnu formu

Primijetimo da u prethodnoj relaciji PREDMET postoji sljedeći niz funkcionalnih ovisnosti:

ŠIFRA PREDMETA → OIB NASTAVNIKA → BROJ SOBE NASTAVNIKA .

Ovakav niz, pod uvjetom da srednji atribut nije kandidat za ključ, nazivamo *tranzitivna* ovisnost. U našem slučaju, BROJ SOBE NASTAVNIKA je tranzitivno ovisan o ŠIFRI PREDMETA, posredstvom OIB NASTAVNIKA. Pritom OIB NASTAVNIKA zaista nije kandidat za ključ u relaciji PREDMET budući da isti nastavnik može predavati više predmeta.

Slično kao parcijalna ovisnost, i tranzitivna ovisnost se smatra nepoželjnim svojstvom. Naime i ona može uzrokovati slične teškoće kod manipuliranja s podacima, što će također biti pokazano u Odjeljku 4.3.1. Da bi se i tranzitivne ovisnosti mogle „proskribirati“, uvodi se pojam treće normalne forme.

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

Prije navedena relacija PREDMET nije u 3NF jer imamo funkcionalnu ovisnost

OIB NASTAVNIKA → BROJ SOBE NASTAVNIKA ,

a pritom OIB NASTAVNIKA nije ključ, a BROJ SOBE NASTAVNIKA nije primarni atribut.

Opet u skladu s prethodno rečenim, relacija koja nije u 3NF loše je oblikovana, te se preporuča da se ona prevede u 3NF. Postupak prevođenja u 3NF sličan je postupku za 2NF. Dakle polazna relacija se razbija u barem dvije, tako da se prekinu nepoželjne tranzitivne ovisnosti, a da pritom ne dođe do nikakvog gubitka informacije. Preciznije:

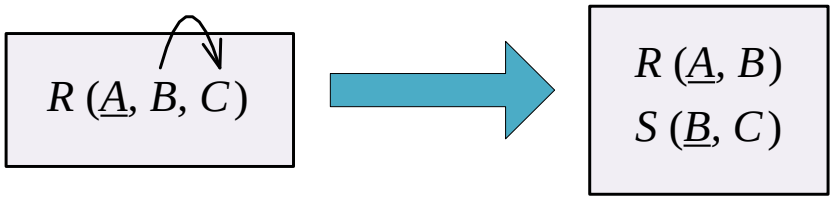
- Uz polaznu relaciju dodaje se onoliko novih relacija koliko ima različitih atributa koji se pojavljuju kao srednji atributi u tranzitivnim ovisnostima.
- Iz polazne relacije *izbacuju se i prebacuju* u nove svi oni atributi koji su tranzitivno ovisni.
- Pritom u jednu novu relaciju idu oni atributi u čijim tranzitivnim ovisnostima se pojavljuje isti srednji atribut.
- Uz prebačene atribute, u novu relaciju *prepisuje se* i odgovarajući srednji atribut, te on postaje ključ u toj novoj relaciji.

U našem konkretnom primjeru, postupak prevođenja u 3NF razbija polaznu relaciju PREDMET u sljedeće dvije relacije:

PREDMET (ŠIFRA PREDMETA, NASLOV PREDMETA,
OIB NASTAVNIKA)
NASTAVNIK (OIB NASTAVNIKA, BROJ SOBE NASTAVNIKA) .

Naime, u polaznoj verziji od PREDMET postojala je samo jedna tranzitivna ovisnost sa srednjim atributom OIB NASTAVNIKA. Zato postupak prevođenja stvara samo jednu novu relaciju, te je u nju prebačen tranzitivno-ovisni atribut BROJ SOBE NASTAVNIKA zajedno s OIB NASTAVNIKA. Vidimo da OIB NASTAVNIKA postaje ključ u novoj relaciji, te da nova relacija zapravo opisuje nastavnika i u skladu s time dobiva ime NASTAVNIK.

Nakon opisanog prevođenja, obje relacije su u 3NF. Naime, u preostaloj verziji PREDMET više nema tranzitivnih ovisnosti pa ona zadovoljava definiciju 3NF. Također, NASTAVNIK ima samo dva atributa pa je sigurno u 3NF.



Slika 4.4: Shematski prikaz prevođenja u 3NF.

Postupak prevođenja u 3NF shematski je prikazan Slikom 4.4. Na toj slici pretpostavljeno je da polazna relacija ima samo jednu tranzitivnu ovisnost. Ako imamo više tranzitivnih ovisnosti s istim srednjim atributom, ili tranzitivne ovisnosti s raznim srednjim atributima, tada se zahvati sa Slike 4.4 moraju iterirati.

4.2. Boyce-Codd-ova i četvrta normalna forma

Makar su za svakodnevne potrebe obično dovoljne i prve tri normalne forme, teorija normalizacije nije se zaustavila na tome. U svojim kasnijim radovima iz druge polovice 70-tih godina 20. stoljeća, Edgar Codd je definirao pojačanu varijantu 2NF i 3NF koja se zove Boyce-Codd-ova normalna forma. Ronald Fagin je 1977. i 1979. godine uveo četvrtu i petu normalnu formu. Kasniji autori uveli su i šestu normalnu formu.

U praksi je lako naići na relacije koje odstupaju od 2NF, 3NF, no vrlo rijetko se susreću relacije u 3NF koje nisu u višim normalnim formama. Zato su te više normalne forme prvenstveno od teorijskog značaja. Mi ćemo ipak opisati Boyce-Coddovu i četvrtu normalnu formu, no nećemo se baviti ni petom ni šestom formom. Boyce-Coddova normalna forma je posebno zanimljiva jer ona može poslužiti kao efikasna zamjena za 2NF i 3NF.

4.2.1. Determinante, prevođenje relacije u Boyce-Coddovu normalnu formu

Definicija Boyce-Codd-ove normalne forme zasnovana je na pojmu determinante. Zato najprije moramo objasniti taj pojam, a tek nakon toga možemo izreći definiciju same normalne forme.

Determinanta je atribut (ili kombinacija atributa) unutar neke relacije o kojem je neki drugi atribut unutar iste relacije potpuno funkcionalno ovisan. Relacija je u *Boyce-Codd-ovoj normalnoj formi* (oznaka: BCNF) ako je svaka njezina determinanta ujedno i kandidat za ključ.

Kao primjer relacije koja nije u BCNF može nam poslužiti ona ista koju smo promatrali kad smo govorili o 2NF i 3NF, dakle:

UPISAO (JMBAG, ŠIFRA PREDMETA, NASLOV PREDMETA, OIB NASTAVNIKA,
BROJ SOBE NASTAVNIKA, OCJENA) .

Ponovo uočavamo sljedeće funkcionalne ovisnosti:

ŠIFRA PREDMETA → NASLOV PREDMETA ,
ŠIFRA PREDMETA → OIB NASTAVNIKA ,
OIB NASTAVNIKA → BROJ SOBE NASTAVNIKA .

Vidimo da postoje dvije determinante, ŠIFRA PREDMETA i OIB NASTAVNIKA, i pritom nijedna od njih nije kandidat za ključ. Dakle relacija nije u BCNF.

Slično kao prije, relacija koja nije u BCNF smatra se loše oblikovanom, te mogućim izvorom poteškoća. Preporuča se da se ona prevede u BCNF. Postupak prevođenja analogan je onom

za 2NF ili 3NF. Dakle polazna relacija se na pogodan način razbija u nekoliko manjih, tako da se prekinu nepoželjne ovisnosti o determinanti koja nije kandidat za ključ. Detaljnije:

- Uz polaznu relaciju dodaje se onoliko novih relacija koliko ima različitih takvih determinanti.
- Iz polazne relacije *izbacuju se i prebacuju* u nove svi oni atributi koji su ovisni o nekoj tih determinanti.
- Pritom u jednu novu relaciju idu oni atributi koji su ovisni o istoj determinanti.
- Uz prebačene atribute, u novu relaciju *prepisuje se* i sama determinanta, te ona postaje ključ u toj novoj relaciji.

Ako ovaj postupak prevođenja u BCNF primijenimo na našu polaznu relaciju UPISAO, ona se zbog postojanja dviju determinanti koje nisu kandidat za ključ razbija na ukupno tri relacije koje izgledaju ovako:

UPISAO (JMBAG, ŠIFRA PREDMETA, OCJENA)
PREDMET (ŠIFRA PREDMETA, NASLOV PREDMETA,
OIB NASTAVNIKA)
NASTAVNIK (OIB NASTAVNIKA, BROJ SOBE NASTAVNIKA) .

Vidimo da smo sad jednim prevođenjem dobili ono isto rješenje za koje su prije bila potrebna dva prevođenja, najprije u 2NF, zatim u 3NF. U tom smislu, BCNF predstavlja efikasnu i kompaktnu zamjenu za 2NF i 3NF.

4.2.2. Odnos Boyce-Codd-ove prema drugoj i trećoj normalnoj formi

Upravo smo vidjeli da između BCNF, 2NF i 3NF postoji bliska veza. Razlog je u tome što pojam determinante poopćuje razne oblike funkcionalnih ovisnosti koje smo prije razmatrali. Zaista: i parcijalne i tranzitivne ovisnosti zapravo određuju neku vrstu determinante koja nije kandidat za ključ. Zato BCNF uključuje u sebi sve zahtjeve koje postavljaju 2NF i 3NF. Drugim riječima, ako relacija nije u 2NF ili 3NF, ona ne može biti ni u BCNF. Ili obratno, relacija koja jeste u BCNF mora nužno biti u 2NF i 3NF.

Na osnovu ovih primjedbi moglo bi se pomisliti da je BCNF zapravo ekvivalentna 2NF i 3NF, te da je riječ samo o drukčijoj formulaciji istih svojstava. No pokazuje se da BCNF ipak postavlja na relaciju malo jači zahtjev od 2NF i 3NF. Dakle, makar relacija koja je u BCNF nužno mora biti u 2NF i 3NF, može se desiti da relacija koja jest u 2NF i 3NF ipak nije u BCNF. Drukčije rečeno, BCNF se može smatrati „trećom-i-pol normalnom formom“.

Primjeri relacija koje jesu u 2NF i 3NF no nisu u BCNF zapravo su vrlo rijetki. U nastavku ćemo izložiti jedan takav primjer – on se zasniva na postojanju dva kandidata za ključ koja su oba složena i preklapaju se u jednom atributu.

Naš primjer odnosi se na fakultet gdje jedan predmet predaje više nastavnika, ali svaki nastavnik predaje samo jedan predmet. Svaki student upisuje više predmeta, no ima samo jednog nastavnika za zadani predmet. Situacija je slična onoj na glazbenoj akademiji, gdje postoji više nastavnika violine, ili više nastavnika klavira, a svaki student uči jedan instrument „u klasi“ samo jednog nastavnika. Sve skupa može se opisati sljedećom relacijom, gdje je opet pretpostavljeno da JMBAG jednoznačno određuje studenta, OIB NASTAVNIKA određuje nastavnika, a ŠIFRA PREDMETA predmet:

UPISAO (JMBAG, OIB NASTAVNIKA, ŠIFRA PREDMETA) .

Uz ovako određeni primarni ključ, relacija nije ni u 2NF, jer postoji parcijalna ovisnost

OIB NASTAVNIKA → ŠIFRA PREDMETA .

No mi možemo drukčije izabrati primarni ključ. Ista relacija tada izgleda ovako:

UPISAO (JMBAG, ŠIFRA PREDMETA, OIB NASTAVNIKA) .

Sad je relacija u 2NF i 3NF, no ne i u BCNF jer i dalje postoji ovisnost:

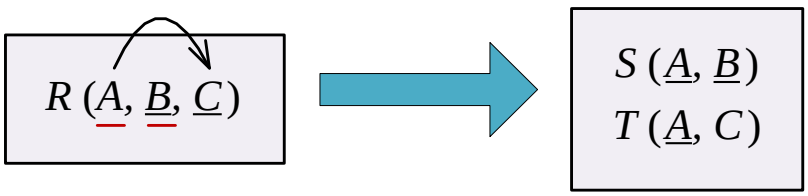
OIB NASTAVNIKA → ŠIFRA PREDMETA .

Pritom determinanta OIB NASTAVNIKA nije kandidat za ključ jer može postojati više studenata koji su upisali nastavnikov predmet u njegovoj klasi. Znači zaista imamo primjer relacije koja je u 2NF i 3NF no nije u BCNF.

Ako na zadnju verziju relacije UPISAO primijenimo naš postupak prevođenja u BCNF, ona se raspada na sljedeće dvije relacije:

KLASA (JMBAG, OIB NASTAVNIKA)
PREDAJE (OIB NASTAVNIKA, ŠIFRA PREDMETA) .

Obje novonastale relacije su u BCNF. Pritom smo im dali imena koja najbolje opisuju njihovo značenje. Iz prve od njih vidimo u čijoj klasi je određeni student upisao određeni (to jest nastavnikov) predmet, a iz druge vidimo koji to predmet predaje određeni nastavnik.



Slika 4.5: Shematski prikaz prevođenja u BCNF.

Postupak prevođenja u BCNF za slučaj relacije koja jeste u 2NF i 3NF shematski je prikazan Slikom 4.5. Kao u našem primjeru, na slici je pretpostavljeno da polazna relacija ima dva kandidata za ključ koja su oba složena i preklapaju se u jednom atributu.

4.2.3. Višeznačne ovisnosti, prevođenje relacije u četvrtu normalnu formu

Četvrtu normalnu formu najlakše je opisati pomoću primjera. Promatrajmo opet relaciju IZVOZI koja nam je poznata iz prošlog poglavlja i koja prikazuje vezu između kompanija, proizvoda i zemalja:

IZVOZI (IME KOMPANIJE, IME PROIZVODA, IME ZEMLJE).

Jedna n-torka relacije IZVOZI izražava činjenicu da zadana kompanija svoj zadani proizvod izvozi u zadanu zemlju. Atributi IME KOMPANIJE, IME PROIZVODA, odnosno IME ZEMLJE jednoznačno određuju primjerke odgovarajućih entiteta. Lagano je provjeriti da je relacija u BCNF. U jednom trenutku ona može izgledati kao na Slici 4.6.

IZVOZI

IME KOMPANIJE	IME PROIZVODA	IME ZEMLJE
IBM	Desktop	Francuska
IBM	Desktop	Italija
IBM	Desktop	Velika Britanija
IBM	Mainframe	Francuska
IBM	Mainframe	Italija
IBM	Mainframe	Velika Britanija
HP	Desktop	Francuska
HP	Desktop	Španjolska
HP	Desktop	Irska
HP	Server	Francuska
HP	Server	Španjolska
HP	Server	Irska
Fujitsu	Mainframe	Italija
Fujitsu	Mainframe	Francuska

Slika 4.6: Primjer relacije koja nije u četvrtoj normalnoj formi.

Na osnovu podataka na Slici 4.6 stječe se dojam da vrijedi pravilo: *čim kompanija izvozi u neku zemlju, ona odmah izvozi sve svoje proizvode u tu zemlju*. Ako prihvatimo da vrijedi takvo pravilo, tada nam postaje očito da relacija IZVOZI mora sadržavati veliku dozu redundancije.

Uočena redundancija će se eliminirati ako zamijenimo polaznu relaciju IZVOZI s dvije manje relacije RADI i PRODAJE:

RADI (IME KOMPANIJE, IME PROIZVODA)
PRODAJE (IME KOMPANIJE, IME ZEMLJE) .

Podacima s prethodne Slike 4.6 tada odgovaraju podaci na Slici 4.7.

Dosadašnja pravila normalizacije ne pomažu nam da eliminiramo redundanciju u relaciji IZVOZI. Pokazuje se da je to zato što redundancija nije bila uzrokovana funkcionalnim ovisnostima, već takozvanim višeznačnim ovisnostima. U nastavku slijedi odgovarajuća definicija.

RADI

IME KOMPANIJE	IME PROIZVODA
IBM	Desktop
IBM	Mainframe
HP	Desktop
HP	Server
Fujitsu	Mainframe

PRODAJE

IME KOMPANIJE	IME ZEMLJE
IBM	Francuska
IBM	Italija
IBM	Velika Britanija
HP	Francuska
HP	Španjolska
HP	Irska
Fujitsu	Italija
Fujitsu	Francuska

Slika 4.7: Prevođenje u četvrtu normalnu formu.

Zadana je relacija s tri atributa: $R(A, B, C)$. Višeznačna ovisnost od A do B (oznaka: $A \rightarrow\rightarrow B$) vrijedi ako skup B-vrijednosti koje se u R pojavljuju uz zadani par (A-vrijednost, C-vrijednost) ovisi samo o A-vrijednosti, a ne i o C-vrijednosti. Analogna definicija primjenjuje se i kad su A, B i C složeni atributi (dakle skupine atributa).

U našem primjeru, skup proizvoda koje zadana kompanija izvozi u zadanu zemlju ovisi samo o kompaniji a ne o zemlji. Slično, skup zemalja u koje zadana kompanija izvozi zadani proizvod ovisi samo o kompaniji a ne i o proizvodu. Zato kod nas vrijede sljedeće višeznačne ovisnosti:

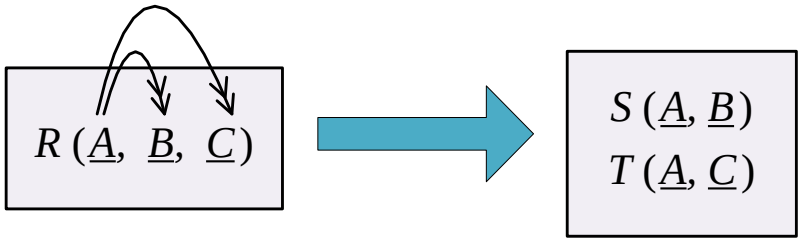
IME KOMPANIJE $\rightarrow\rightarrow$ IME PROIZVODA ,
IME KOMPANIJE $\rightarrow\rightarrow$ IME ZEMLJE .

Nije slučajno da su se odmah pojavile dvije višeznačne ovisnosti. Takve ovisnosti zapravo se uvijek pojavljuju u paru. Naime, može se matematički dokazati da čim u relaciji $R(A, B, C)$ postoji ovisnost $A \rightarrow\rightarrow B$, nužno mora vrijediti i ovisnost $A \rightarrow\rightarrow C$.

Iz upravo objašnjenih razloga, višeznačna ovisnost smatra se nepoželjnom pojavom. Da bi se takva ovisnost mogla eliminirati, uvodi se pojam četvrte normalne forme.

Relacija R je u *četvrtoj normalnoj formi* (oznaka: 4NF) ako vrijedi: kad god postoji višeznačna ovisnost u R, na primjer $A \rightarrow\rightarrow B$, tada su svi atributi od R funkcionalno ovisni o A. Ekvivalentno, R je u 4NF ako je u BCNF i sve višeznačne ovisnosti u R su zapravo funkcionalne ovisnosti.

U našoj relaciji IZVOZI, ni jedna od uočenih višeznačnih ovisnosti nije funkcionalna ovisnost. Znači, IZVOZI nije u 4NF i zato je treba rastaviti na RADI i PRODAJE. Te dvije jednostavnije relacije sigurno jesu u 4NF budući da svaka od njih ima po dva atributa.



Slika 4.8: Shematski prikaz prevođenja u 4NF.

Primjer s relacijom IZVOZI otkriva nam i općenitu postupak za prevođenje relacije u 4NF. Polazna relacija uvijek ima tri atributa (ili tri skupine atributa) i dvije višeznačne ovisnosti. Postupak izgleda ovako.

- Polaznu relaciju pretvaramo u dvije manje relacije od po dva atributa (odnosno dvije skupine atributa), tako da u njima nema višeznačnih ovisnosti.
- U svaku od novih relacija stavljamo po dva atributa (odnosno dvije skupine atributa) koji u polaznoj relaciji sudjeluju u istoj višeznačnoj ovisnosti.

Sve je to shematski prikazano Slikom 4.8.

4.3. Potreba za normalizacijom

Normalizacija je u prvom redu potrebna zato jer se njome izbjegavaju teškoće koje bi nastupile ako bi radili s nenormaliziranim podacima. Normalizacija je korisna i zato jer se njome naknadno otkrivaju i ispravljaju greške u oblikovanju entiteta, veza i atributa. Od normalizacije možemo odustati samo u nekim rijetkim situacijama, no i tada moramo biti svjesni eventualnih loših posljedica takve odluke. U nastavku ovog potpoglavlja detaljnije ćemo raspraviti o ovim tvrdnjama.

4.3.1. Teškoće u radu s nenormaliziranim podacima

Ako relacije nisu normalizirane, dolazi do teškoća kod unosa, promjene i brisanja podataka. Bez obzira o kojoj normalnoj formi je riječ, teškoće su otprilike slične. Ilustrirat ćemo to na primjerima relacija iz prethodnih odjeljaka.

Promatrajmo opet polaznu verziju relacije UPISAO iz Odjeljka 4.1.3. koja nije u 2NF. Primijetimo da ta relacija, osim što bilježi koji student je upisao koji predmet, također govori i o nastavnicima i predmetima:

UPISAO (JMBAG, ŠIFRA PREDMETA, NASLOV PREDMETA, OIB NASTAVNIKA,
BROJ SOBE NASTAVNIKA, OCJENA) .

Do teškoća u radu s UPISAO dolazi onda kad preko nje pokušavamo upravljati podacima o nastavnicima ili predmetima. Na primjer:

- Ako u bazu želimo unijeti podatke o novom predmetu, to ne možemo učiniti sve dok bar jedan student ne upiše taj predmet (naime ne smijemo imati praznu vrijednost primarnog atributa JMBAG). Slično, ako želimo unijeti podatke o novom nastavniku i njegovoj sobi, to ne možemo učiniti dok tog nastavnika ne zadužimo s bar jednim predmetom i dok bar jedan student ne upiše taj predmet.
- Ako želimo promijeniti naslov nekog postojećeg predmeta, tada moramo naći sve n-torke koje sadrže odgovarajuću vrijednost za ŠIFRU PREDMETA, te promijeniti vrijednost za NASLOV PREDMETA u svim takvim n-torkama. Bit će onoliko promjena koliko ima studenata koji su upisali taj predmet. Ako zaboravimo izvršiti neku od promjena, imat ćemo kontradiktorne podatke.
- Pretpostavimo da svi studenti koji su upisali neki predmet naknadno odustanu od tog predmeta. Ako shodno tome pobrišemo odgovarajuće n-torke, iz baze će nestati svi podaci o tom predmetu.

Promatrajmo dalje relaciju PREDMET iz Odjeljka 4.1.4. koja nije u 3NF. Primijetimo da ta relacija, osim o predmetima, govori i o nastavnicima:

PREDMET (ŠIFRA PREDMETA, NASLOV PREDMETA, OIB NASTAVNIKA,
BROJ SOBE NASTAVNIKA) .

Do teškoća u radu s PREDMET dolazi baš onda kad preko nje pokušavamo mijenjati podacima o nastavnicima. Na primjer:
- Ne možemo unijeti podatke o novom nastavniku i njegovoj sobi, sve dok ga nismo zadužili s bar jednim predmetom.
- Da bismo promijenili broj sobe određenog nastavnika, moramo izvršiti promjenu u svakoj n-torki koja odgovara nekom predmetu kojeg predaje taj nastavnik. Ako zaboravimo neki od tih predmeta, imat ćemo kontradiktorne podatke.
- Ako nastavnik privremeno ne predaje ni jedan predmet, tada iz baze nestaju svi podaci o njemu i njegovoj sobi.

Promatrajmo zatim relaciju UPISAO iz Odjeljka 4.2.2, koja se odnosi na glazbenu akademiju i koja nije u BCNF. Primijetimo da ta relacija, osim o tome koji student je upisao koji predmet, govori i o samim predmetima i nastavnicima:

UPISAO (JMBAG, ŠIFRA PREDMETA, OIB NASTAVNIKA) .

Do teškoća u radu s ovom verzijom relacije UPISAO doći će opet onda kad preko nje počnemo raditi s podacima o predmetima i nastavnicima. Na primjer:
- Ne možemo evidentirati činjenicu da zadani nastavnik predaje zadani predmet, sve dok bar jedan student ne upiše taj predmet baš kod tog nastavnika.
- Veza nastavnika i predmeta zapisana je s velikom redundancijom, onoliko puta koliko ima studenata u klasi tog nastavnika, što otežava ažuriranje.
- Ako svi studenti u klasi nekog nastavnika odustanu od sudjelovanja u klasi, briše se evidencija da taj nastavnik predaje taj predmet.

Promatrajmo na kraju relaciju IZVOZI iz Odjeljka 4.2.3 koja prikazuje vezu između kompanija, proizvoda i zemalja. Ta relacija nije u 4NF zbog pretpostavljenog pravila da čim kompanija izvozi u neku zemlju, ona odmah izvozi sve svoje proizvode u tu zemlju:

IZVOZI (IME KOMPANIJE, IME PROIZVODA, IME ZEMLJE) .

Do teškoća u radu s relacijom IZVOZI dolazi zbog toga što ona s velikom redundancijom opisuje dvije nezavisne veze između triju entiteta. Na primjer:

- Da bismo evidentirali da neka kompanija ima novi proizvod, morat ćemo unijeti onoliko n-torki koliko ima zemalja u koje ta kompanija izvozi svoje proizvode.
- Da bismo evidentirali da je neka kompanija počela izvoziti u neku novu zemlju, morat ćemo unijeti onoliko n-torki koliko ima proizvoda te kompanije.

4.3.2. Normalizacija kao ispravak konceptualnih grešaka

U prethodnom odjeljku uočili smo da svi nenormalizirani podaci imaju jednu zajedničku osobinu: oni pokušavaju govoriti o više stvari u isto vrijeme. Zaista, neki od njih pokušavaju istovremeno opisati više tipova entiteta, drugi bilježe vezu između entiteta no istovremeno navode i svojstva samih entiteta, treći opet istovremeno zapisuju više veza. To nije u skladu s postupkom oblikovanja opisanim u Poglavljima 2 i 3. Naime, taj postupak, ukoliko je ispravno proveden, morao bi rezultirati relacijama koje govore o jednom i samo jednom entitetu, ili relacijama koje bilježe jednu i samo jednu vezu.

U skladu s ovom primjedbom, pojavu nenormaliziranih podataka možemo promatrati kao rezultat greške u postupku oblikovanja. Izvor takve greške obično se nalazi već u oblikovanju konceptualne sheme, dakle u pogrešnom prepoznavanju entiteta, veza i atributa. U nastavku ćemo detaljnije analizirati primjere nenormaliziranih podataka iz prethodnih odjeljaka, te ćemo odrediti greške u konceptualnom oblikovanju koje su dovele do njihove pojave.

Zapis SURADNIK iz Odjeljka 4.1.1 koji nije ni u 1NF očito nije mogao nastati ispravnom primjenom postupka oblikovanja entiteta, atributa i veza. Naime, tu se krši pravilo da vrijednosti atributa za entitet koji odgovara suradniku moraju biti jednostavne i jednostruke. Da smo poštovali to pravilo, pod-zapis s osobnim podacima suradnika odmah bismo razbili u više jednostavnih atributa, a ponavljajuću skupinu o djetetu bismo proglasili zasebnim entitetom, te bismo uveli vezu između suradnika i djece. Kad bismo takvu ispravnu konceptualnu shemu pretvorili u relacije, odmah bismo dobili one iste relacije koje smo u Odjeljku 4.1.1 dobili prevođenjem u 1NF.

Relacija UPISAO iz Odjeljka 4.1.3 koja nije u 2NF nastala je zato što se događaj upisivanja predmeta od strane studenta pogrešno interpretirao kao zasebni tip entiteta s vlastitim atributima. Umjesto toga, trebalo je uočiti da kao tipove entiteta zapravo imamo studente i predmete, te da je upisivanje samo veza između tih tipova. Također, trebalo je uočiti da postoji i tip entiteta za nastavnike, te da predavanje predmeta od strane nastavnika predstavlja vezu između predmeta i nastavnika.

Postupak prevođenja iz Odjeljaka 4.1.3 i 4.1.4, najprije u 2NF, zatim u 3NF, postepeno ispravlja te greške. Naime, tim postupkom relacija UPISAO reducira se u oblik koji služi isključivo za bilježenje veze između studenata i predmeta. Također nastaju nove relacije PREDMET i NASTAVNIK koje odgovaraju istoimenim entitetima. Veza između predmeta i nastavnika ispravno se realizira preko stranog ključa u relaciji PREDMET. Doduše, ne dobivamo posebnu relaciju za studente, no to je zato što u polaznim podacima nismo imali nikakvih atributa za studente osim JMBAG.

Relacija UPISAO iz Odjeljka 4.2.2, koja jeste u 2NF i 3NF no nije u BCNF, nastala je zato što smo odnose između nastavnika, predmeta i studenata na glazbenoj akademiji pogrešno tumačili kao ternarnu vezu. Zapravo se radilo o dvije nezavisne binarne veze. Da smo odmah uočili te binarne veze, podaci bi umjesto relacijom UPISAO odmah bili prikazani relacijama KLASA i PREDAJE koje smo mi dobili prevođenjem u BCNF.

Relacija IZVOZI iz Odjeljka 4.2.3, koja nije u 4NF, nastala je opet zato što smo odnose između kompanija, proizvoda i zemalja pogrešno prikazali jednom ternarnom vezom umjesto dvjema binarnim vezama. Da smo krenuli od tih binarnih veza, odmah bi nastale one iste relacije RADI i PRODAJE koje smo mi dobili kao rezultat prevođenja u 4NF.

Na osnovu svega izloženog, zaključujemo da pravila normalizacije nisu ništa drugo nego formalni opis intuitivno prihvatljivih principa o zdravom i prirodnom oblikovanju entiteta, veza i atributa. Ukoliko, služeći se projektantskom vještinom i intuicijom, oblikovanje uspijemo provesti na ispravan način, tada ćemo odmah dobiti relacije u visokim normalnim formama i normalizacija neće imati što raditi.

To naravno ne znači da je postupak normalizacije nepotreban, baš naprotiv. Naime, greške su uvijek moguće i mi nikad ne znamo je li do njih došlo ili nije. Normalizacija predstavlja mehanizam naknadnog prepoznavanja i ispravljanja grešaka. Svođenjem u normalnu formu pogrešno oblikovane relacije ponovo se vraćaju u onaj oblik koje bi imale da grešaka nije bilo. Zahvaljujući normalizaciji, postupak oblikovanja postaje samokorigirajući proces, gdje se greške počinjene u ranijim fazama uspješno ispravljaju u kasnijim fazama.

4.3.3. Razlozi kad se ipak može odustati od normalizacije

Već smo napomenuli da je za većinu praktičnih primjera dovoljno relacije normalizirati do 3NF. No postoje razlozi zbog kojih iznimno možemo odustati čak i od takve skromne normalizacije. Navest ćemo dva moguća razloga.

- **Složeni atribut**. Dešava se da nekoliko atributa u relaciji čine cjelinu koja se u aplikacijama nikad ne rastavlja na sastavne dijelove. Na primjer, promatrajmo sljedeću relaciju:

  KUPAC (OIB KUPCA, PREZIME, IME, POŠTANSKI BROJ, IME GRADA, ULICA I KUĆNI BROJ) .

  Strogo govoreći, IME GRADA je funkcionalno ovisno o POŠTANSKOM BROJU, pa relacija nije u 3NF. No mi znamo da POŠTANSKI BROJ, IME GRADA, te ULICA I KUĆNI BROJ čine cjelinu koja se zove adresa. Budući da se podaci iz adrese rabe i ažuriraju „u paketu“, ne može doći do prije spominjanih teškoća. Nije preporučljivo razbijati ovu relaciju na dvije.
- **Efikasna uporaba podataka**. Normalizacijom se velike relacije razbijaju na mnogo manjih. U aplikacijama je često potrebno podatke iz malih relacija ponovo sastavljati u veće nenormalizirane n-torke. Uspostavljanje veza među podacima u manjim relacijama traje znatno dulje nego čitanje podataka koji su već povezani i upisani u jednu veliku n-torku. Ako imamo aplikacije kod kojih se zahtijeva izuzetno velika brzina odziva, tada je bolje da su podaci za njih već pripremljeni u nenormaliziranom obliku.

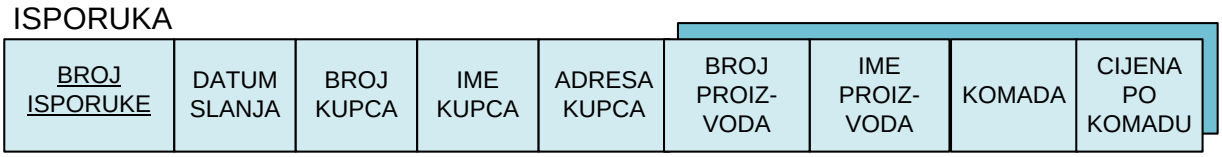
Projektant baze podataka treba procijeniti kada treba provesti normalizaciju do kraja a kada ne. Za tu procjenu je važno razumijevanje značenja podataka i načina kako će se oni zaista rabiti. Drugim riječima, treba utvrditi mogu li se u tom konkretnom slučaju zaista desiti teškoće u radu s nenormaliziranim podacima koje smo opisali u Odjeljku 4.3.1. Također, treba predvidjeti hoće li se zahtijevati izuzetno velika brzina kod pronalaženja podataka.

4.4. Zadaci za vježbu

Zadatak 4.1. Rješavanjem Zadatka 3.1 dobili ste nadopunjenu relacijsku shemu baze podataka o fakultetu. Normalizirajte tu shemu tako da sve relacije budu barem u 3NF.

Zadatak 4.2. Rješavanjem Zadatka 3.2 dobili ste relacijsku shemu za bazu podataka o knjižnici. Normalizirajte tu shemu tako da sve relacije budu barem u 3NF.

Zadatak 4.3. Tvornica isporučuje svoje proizvode kupcima. Jedna isporuka šalje se jednom kupcu i može sadržavati više komada raznih proizvoda. Situacija je prikazana ne-normaliziranim zapisom na Slici 4.9. Zamijenite zapis relacijama u 3NF.



Slika 4.9: Nenormalizirani zapis o isporuci proizvoda kupcu.

Zadatak 4.4. Tvornica sklapa proizvode od dijelova, a te dijelove kupuje od raznih dobavljača. Isti dio može se dobiti od raznih dobavljača po raznim cijenama, a isti dobavljač nudi razne dijelove. Situacija je opisana sljedećom relacijom:

CJENIK (BROJ DIJELA, BROJ DOBAVLJAČA, IME DOBAVLJAČA,
ADRESA DOBAVLJAČA, CIJENA).

Prevedite tu relaciju u 3NF ako je to potrebno.

Zadatak 4.5. Suradnici neke ustanove rade na raznim projektima. Pritom jedan suradnik radi na točno jednom projektu. Situacija je opisana sljedećom relacijom:

SURADNIK (MATIČNI BROJ, PREZIME I IME, PLAĆA, BROJ PROJEKTA,
ROK ZAVRŠETKA PROJEKTA).

Prevedite tu relaciju u 3NF ako je to potrebno.

Zadatak 4.6. Na fakultetu se nastava iz jednog predmeta održava uvijek u istoj predavaonici, no u nekoliko termina tjedno. Situacija je opisana sljedećom relacijom:

RASPORED (BROJ PREDAVAONICE, TERMIN, ŠIFRA PREDMETA).

Prevedite tu relaciju u BCNF ako je to potrebno.

Zadatak 4.7. Studenti upisuju izborne predmete iz matematike, te izborne predmete iz računarstva. Ne postavljaju se nikakvi uvjeti na izbor jednih u odnosu na druge. Situacija je opisana sljedećom relacijom:

UPISAO (JMBAG, ŠIFRA M-PREDMETA, ŠIFRA R-PREDMETA).

Prevedite tu relaciju u 4NF ako je to potrebno.

Zadatak 4.8. Rješavanjem Zadatka 3.3 dobili ste relacijsku shemu za bazu podataka iz vašeg područja interesa. Normalizirajte tu shemu tako da sve relacije budu barem u 3NF.

5. Postavljanje upita u relacijskim bazama podataka

U prethodnim poglavljima pratili smo postupak oblikovanja baze podataka. Govorili smo o konceptualnom i logičkom oblikovanju uključujući i normalizaciju. Usput smo se također upoznali s relacijskim modelom za bazu podataka. Pritom smo se ograničili na statički aspekt tog modela, dakle bavili smo se logičkom strukturom relacijske baze.

U ovom poglavlju nakratko ćemo prekinuti s praćenjem postupka oblikovanja da bismo se osvrnuli na još neka važna svojstva relacijskih baza. Sad će nam u središtu interesa biti dinamički aspekt relacijskog modela, koji se očituje u postojanju fleksibilnih jezika za postavljanje upita. Vidjet ćemo da se, u skladu s matematičkim temeljima samog relacijskog modela, ti jezici mogu precizno matematički opisati. Postavljanje upita može se interpretirati kao primjena matematičkih operacija kojima se iz postojećih relacija dinamički stvaraju nove virtualne relacije.

Jezici za postavljanje upita u relacijskim bazama podataka proučavaju se jednako dugo kao i same relacijske baze. U svojim ključnim radovima, Edgar Codd takve jezike smatra sastavnim dijelom relacijskog modela. Zaista, relacijski model ne bi vrijedio mnogo bez svoje dinamičke komponente. Baza koju je normalizacija razbila u velik broj malih relacija ne bi bila osobito pogodna za aplikacije kad ne bi postojao fleksibilan i brz način pronalaženja i pregrupiranja podataka.

Ovo poglavlje podijeljeno je u tri potpoglavlja. Prva dva potpoglavlja bave se relacijskom algebrom i relacijskim računom – riječ je o upitnim jezicima zasnovanima na algebri odnosno predikatnom računu koji se doduše ne rabe izravno u praksi no važni su za teoriju. Treće poglavlje predstavlja danas najrašireniji jezik za rad s relacijskim bazama, a to je SQL. Makar se to na prvi pogled možda ne vidi, SQL je zapravo utemeljen na jednoj vrsti relacijskog računa, a također on s lakoćom može simulirati sve operacije iz relacijske algebre.

5.1. Relacijska algebra

Relacijsku algebru uveo je Edgar Codd u svojim radovima iz 70-tih godina 20. stoljeća. Budući da je riječ je o matematičkoj notaciji, a ne o praktičnom jeziku kojeg bi korisnici neposredno rabili, sintaksa u pojedinim knjigama ili člancima se donekle razlikuje. Mi ćemo se koristiti sintaksom s ključnim riječima nalik na programske jezike.

Relacijska algebra svodi se na izvrednjavanje algebarskih izraza. Ti izrazi građeni su od unarnih i binarnih operacija, operanada, te zagrada. Pritom je riječ o algebarskim operacijama čiji operandi su relacije, a rezultati opet relacije. Svaki algebarski izraz predstavlja jedan upit u bazu, a njegova vrijednost predstavlja odgovor na upit. Dakle odgovor na upit izražava se kao nova (virtualna) relacija dobivena iz postojećih relacija primjenom algebarskih operacija.

U ovom i idućim potpoglavljima, za zadavanje i praćenje primjera upita služit će nam baza podataka o fakultetu opisana relacijskom shemom sa Slike 3.5 Pretpostavljat ćemo da trenutno stanje te baze izgleda kao na Slici 5.1.

STUDENT

JMBAG	PREZIME	IME	GODINA_STUDIJA
0036398757	Marković	Marko	1
1191203289	Petrović	Petar	2
1192130031	Horvat	Dragica	2
1191205897	Janković	Marija	1
0165043021	Kolar	Ivan	3
0036448430	Grubišić	Katica	3
0246022858	Vuković	Janko	1

ZAVOD

IME_ZAVODA	OIB_PROCELNIKA	OPIS_DJELATNOSTI
Zavod za matematiku	25810043761	Ovaj zavod bavi se svim područjima teorijske i primijenjene matematike ...
Zavod za računarstvo	33571209458	Ovaj zavod bavi se računarskom znanošću i softverskim inženjerstvom ...

NASTAVNIK

OIB	PREZIME	IME	IME_ZAVODA	BROJ_SOBE	PLACA
13257600947	Cantor	Georg	Zavod za matematiku	102	12000
25810043761	Goedel	Kurt	Zavod za matematiku	305	14000
33571209458	Codd	Edgar	Zavod za računarstvo	127	16000
44102179316	Klein	Felix	Zavod za matematiku	252	12000
50076128203	Pascal	Blaise	Zavod za matematiku	101	11000
67741205512	Turing	Alan	Zavod za računarstvo	315	13000

PREDMET

SIFRA_PREDMETA	NASLOV	IME_ZAVODA	OIB_NASTAVNIKA	SEMESTAR	ECTS_BODOVI
56001	Baze podataka	Zavod za računarstvo	33571209458	L	5
72001	Linearna algebra	Zavod za matematiku	44102179316	Z	6
72005	Matematička analiza	Zavod za matematiku	25810043761	L	6
56002	Programiranje u C-u	Zavod za računarstvo	67741205512	Z	5
72009	Analitička geometrija	Zavod za matematiku	44102179316	L	5

UPISAO

JMBAG	SIFRA_PREDMETA	DATUM_UPISA	OCJENA
0036398757	72001	2010-09-05	
1191203289	56001	2009-10-03	2
1191203289	56002	2009-10-03	2
1192130031	72001	2009-09-15	5
1192130031	56002	2009-09-15	4
1192130031	72009	2009-09-15	2
1191205897	72009	2010-09-05	
0165043021	56001	2008-10-03	3
0165043021	72001	2008-10-03	4
0165043021	56002	2009-06-10	
0036448430	56001	2009-09-04	5
0246022858	56001	2010-09-03	
0246022858	72001	2010-09-03	
0246022858	72005	2010-09-20	
0246022858	56002	2010-09-25	
0246022858	72009	2010-09-25	

Slika 5.1: Stanje baze podataka o fakultetu.

Primjećujemo da u odnosu na shemu sa Slike 3.5 baza na Slici 5.1 ima neznatno promijenjena imena atributa. Naime, izbjegnuta je uporaba domaćih slova ili bjelina unutar imena. To je napravljeno zbog prilagodbe sintaktičkim pravilima stvarnih jezika poput SQL.

U nastavku ovog potpoglavlja opisujemo redom pojedine algebarske operacije koje se pojavljuju u relacijskoj algebri. Operacije su s obzirom na svoje međusobne sličnosti i srodnosti razvrstane u nekoliko odjeljaka. Za svaku operaciju dajemo primjere njihove uporabe.

5.1.1. Skupovne operacije

Sjetimo se da su relacije ustvari skupovi n-torki. Zato na njih možemo primjenjivati uobičajene skupovne operacije, kao što su *unija*, *presjek* i *razlika*. Neka *R* i *S* označavaju relacije. Tada je:

- *R* union *S* ... unija od *R* i *S*, dakle skup n-torki koje su u *R* ili u *S* (ili u obje relacije).
- *R* intersect *S* ... ... presjek od *R* i *S*, dakle skup n-torki koje su u *R* i također u *S*.
- *R* minus *S* ... razlika od *R* i *S*, dakle skup n-torki koje su u *R* no nisu u *S*.

Da bi se ove operacije mogle primijeniti, relacije *R* i *S* moraju biti *kompatibilne*, to jest jednako građene, dakle moraju imati isti stupanj i iste atribute (ista imena i tipove).

Kao primjer, promatrajmo relaciju NOVI_STUDENT, građenu jednako kao STUDENT, u kojoj se nalaze n-torke popisane na Slici 5.2. Možemo zamišljati da ta relacija bilježi studente nekog drugog fakulteta, s time da među njima ima i onih koji su također upisali i naš fakultet. Tada primjenom skupovnih operacija i u skladu sa Slikom 5.1 dobivamo rezultate prikazane na Slici 5.3.

NOVI_STUDENT

JMBAG	PREZIME	IME	GODINA_STUDIJA
1191205336	Drašković	Janko	2
1192130031	Horvat	Dragica	2
1191621335	Iveković	Ivka	1
0165043021	Kolar	Ivan	3
0248022869	Zalar	Mladen	1

Slika 5.2: Relacija koja je kompatibilna s relacijom STUDENT iz fakultetske baze.

U stvarnosti se zapravo rijetko susreće situacija iz prethodnog primjera gdje smo skupovne operacije mogli primijeniti na relacije koje zaista postoje u bazi. Znatno je češća situacija kad te operacije primjenjujemo na virtualne relacije koje su nastale kao odgovor na druge upite.

Dakle skupovne operacije prvenstveno služe zato da se pomoću njih jednostavniji upiti kombiniraju u složenije. Zaista, ako su *R* i *S* virtualne relacije koje sadrže odgovore na dva jednostavnija upita, tada:

- *R* union *S* sadrži podatke koji zadovoljavaju kriterije barem jednog od tih jednostavnijih upita,
- *R* intersect *S* podatke koji istovremeno zadovoljavaju oba upita,
- *R* minus *S* podatke koji odgovaraju na prvi upit no ne odgovaraju na drugi.

STUDENT union NOVI_STUDENT

JMBAG	PREZIME	IME	GODINA_STUDIJA
0036398757	Marković	Marko	1
1191203289	Petrović	Petar	2
1192130031	Horvat	Dragica	2
1191205897	Janković	Marija	1
0165043021	Kolar	Ivan	3
0036448430	Grubišić	Katica	3
0246022858	Vuković	Janko	1
1191205336	Drašković	Janko	2
1191621335	Iveković	Ivka	1
0248022869	Zalar	Mladen	1

STUDENT intersect NOVI_STUDENT

JMBAG	PREZIME	IME	GODINA_STUDIJA
1192130031	Horvat	Dragica	2
0165043021	Kolar	Ivan	3

STUDENT minus NOVI_STUDENT

JMBAG	PREZIME	IME	GODINA_STUDIJA
0036398757	Marković	Marko	1
1191203289	Petrović	Petar	2
1191205897	Janković	Marija	1
0036448430	Grubišić	Katica	3
0246022858	Vuković	Janko	1

Slika 5.3: Rezultati skupovnih operacija.

Primijetimo da uvijek vrijedi:

R intersect $S = R$ minus (R minus S).

Znači, nisu nužne sve tri skupovne operacije, jer bismo jednu od njih mogli zamijeniti s druge dvije. Ipak, zbog udobnosti, koristimo se svim trima operacijama.

5.1.2. Selekcija i projekcija

Selekcija je unarni operator koji izvlači uz relacije one n-torke koje zadovoljavaju zadani Booleovski uvjet. Selekcija na relaciji R u skladu s Booleovskim uvjetom $\mathcal{B}$ označavamo s R where $\mathcal{B}$. Uvjet $\mathcal{B}$ je formula koja se sastoji od:

- konstanti ili atributa,
- operatora za uspoređivanje $=$, $<$, $>$, $\leq$, $\geq$, $\neq$,
- logičkih operatora and, or, not.

U nastavku navodimo nekoliko primjera od kojih svaki sadrži jedan upit sa selekcijom i odgovarajući izraz u relacijskoj algebri. Izračunate vrijednosti izraza, dakle odgovori na upite, vide se na Slici 5.4.

Upit 1: pronađi sve studente na prvoj godini.

ODGOVOR1 := STUDENT where GODINA_STUDIJA = 1.

Upit2: pronađi sve predmete s naslovom Baze podataka.

ODGOVOR2 := PREDMET where NASLOV = 'Baze podataka'.

Upit 3: pronađi one studente koji su iz predmeta sa šifrom 56001 dobili ocjenu veću od 2.

ODGOVOR3 := UPISAO where ((SIFRA_KOLEGIJA = 56001) and (OCJENA > 2)).

ODGOVOR1

JMBAG	PREZIME	IME	GODINA_STUDIJA
0036398757	Marković	Marko	1
1191205897	Janković	Marija	1
0246022858	Vuković	Janko	1

ODGOVOR2

SIFRA_ PREDMETA	NASLOV	IME_ZAVODA	OIB_ NASTAVNIKA	SEMESTAR	ECTS_ BODOVI
56001	Baze podataka	Zavod za računarstvo	33571209458	L	5

ODGOVOR3

JMBAG	SIFRA_PREDMETA	DATUM_UPISA	OCJENA
0165043021	56001	2008-10-03	3
0036448430	56001	2009-09-04	5

Slika 5.4: Odgovori na upite sa selekcijom.

Projekcija je unarni operator koji iz relacije izvlači zadane atribute, s time da se u rezultirajućoj relaciji eliminiraju n-torke duplikati. Projekciju relacije R na njezine atribute A_1, A_2, …, A_m označavat ćemo s $R[A_1,A_2,\ldots, A_m]$. U složenim algebarskim izrazima smatrat ćemo da projekcija ima viši prioritet od ostalih operacija, osim ako nije drukčije označeno zagradama.

U nastavku navodimo dva primjera upita gdje odgovarajući izraz u relacijskoj algebri sadrži projekciju. Izračunate vrijednosti izraza, dakle odgovori na upite, vide se na Slici 5.5.

Upit4: pronađi brojeve soba svih nastavnika.

ODGOVOR4 := NASTAVNIK [BROJ_SOBE].

Upit5: pronađi OIB nastavnika koji predaje predmet sa šifrom 72009.

ODGOVOR5 :=
(PREDMET where SIFRA_PREDMETA = 72009) [OIB_NASTAVNIKA].

ODGOVOR4

BROJ_SOBE
102
305
127
252
101
315

ODGOVOR5

OIB_NASTAVNIKA
44102179316

Slika 5.5: Odgovori na upite sa projekcijom.

Primijetimo da su selekcija i projekcija komplementarne operacije, jer jedna iz relacije izvlači retke, a druga stupce. Da bismo pronašli točno određene podatke unutar relacije, obično moramo kombinirati i selekciju i projekciju, kao što je to bilo učinjeno u petom upitu.

5.1.3. Kartezijev produkt i dijeljenje

Neka su R i S relacije stupnja n_1 odnosno n_2. Tada algebarski izraz R times S daje *Kartezijev produkt* od R i S, dakle skup svih (n_1+n_2)-torki čijih prvih n_1 komponenti čine n_1-torku u R, a zadnjih n_2 komponenti čine n_2-torku u S. Atribut u R times S ima isto ime kao odgovarajući atribut u R odnosno S, s time da se po potrebi to ime proširuje imenom polazne relacije i točkom (slično kao za komponentu struct-a u jeziku C).

Kartezijev produkt je prva od operacija koje nam omogućuju odgovaranje na složenije upite gdje je potrebno povezivanje podataka iz raznih relacija. Kao prvi primjer uporabe Kartezijevog produkta (Upit 6), ispisat ćemo za svakog studenta sve predmete koje on nije upisao:

```
SVE_KOMBINACIJE := STUDENT[JMBAG] times PREDMET[SIFRA_PREDMETA],
ODGOVOR6 :=
    SVE_KOMBINACIJE minus UPISAO[JMBAG, SIFRA_PREDMETA].
```

Drugi primjer (Upit 7) pronalazi sve parove prezimena studenata koji su na istoj godini studija. Za to nam je potrebno napraviti Kartezijev produkt relacije STUDENT sa samom sobom. Da ne bi došlo do zbrke s imenima atributa, posebnim operatorom aliases uvodimo „pseudonim“ (drugo ime) za relaciju STUDENT, pa zatim sastavljamo odgovor na upit:

```
STUDENT1 aliases STUDENT;
ODGOVOR7 := ( ( STUDENT1 times STUDENT)
     where ( (STUDENT1.GODINA_STUDIJA = STUDENT.GODINA_STUDIJA)
     and (STUDENT1.JMBAG < STUDENT.JMBAG)) )
     [STUDENT1.PREZIME, STUDENT.PREZIME].
```

Odgovori za oba upita prikazani su na Slici 5.6. Kod drugog upita, ograničenjem da JMBAG prvog studenta u paru mora biti manji od JMBAG-a drugog studenta spriječili smo da se u odgovoru isti par studenata pojavljuje dvaput (u dva poretka), te da se student pojavljuje u paru sam sa sobom.

ODGOVOR6

JMBAG	SIFRA_PREDMETA
0036398757	56001
0036398757	72005
0036398757	56002
0036398757	72009
1191203289	72001
1191203289	72005
1191203289	72009
1192130031	56001
1192130031	72005
1191205897	56001
1191205897	72001
1191205897	72005
1191205897	56002
0165043021	72005
0165043021	72009
0036448430	72001
0036448430	72005
0036448430	56002
0036448430	72009

ODGOVOR7

STUDENT1.PREZIME	STUDENT.PREZIME
Marković	Janković
Marković	Vuković
Vuković	Janković
Petrović	Horvat
Grubišić	Kolar

Slika 5.6: Odgovori na upite s Kartezijevim produktom.

Neka je *R* relacija stupnja *n*, a *S* relacija stupnja *m*, i neka se svi atributi od *S* pojavljuju i u *R*. Rezultat *dijeljenja R* sa *S*, oznakom *R* divideby *S*, je skup svih (*n*-*m*)-torki <*x*> takvih da se *n*-torke <*x*,*y*> pojavljuju u *R* za sve *m*-torke <*y*> u *S*. Ovdje *x* i *y* predstavljaju skupinu od jedne ili više vrijednosti atributa. Definicija se bolje može razumjeti na osnovu apstraktnog primjera sa Slike 5.7. Na toj slici isti simboli oblika a_i odnosno b_j označavaju jednake vrijednosti atributa.

R1

A	*B*
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_3	b_1
a_4	b_4

R2

B
b_1

R3

B
b_1
b_2
b_3

R1 divideby *R2*

A
a_1
a_2
a_3

R1 divideby *R3*

A
a_1

Slika 5.7: Apstraktni primjer dijeljenja.

U nastavku slijede još neki primjeri upita koji se odnose na bazu podataka o fakultetu i koji se mogu zapisati pomoću operacije dijeljenja. Izračunate vrijednosti izraza, dakle odgovori na upite, vide se na Slici 5.8.

Upit 8: pronađi JMBAG-ove studenata koji su upisali sve predmete.

```
ODGOVOR8 :=
    UPISAO[JMBAG, SIFRA_PREDMETA] divideby PREDMET[SIFRA_PREDMETA].
```

Upit 9: pronađi JMBAG-ove onih studenata koji su upisali barem one predmete koje je upisao student s JMBAG-om 1191203289.

```
ODGOVOR9 :=
  (UPISAO[JMBAG, SIFRA_PREDMETA] divideby
    ( UPISAO where JMBAG = 1191203289 )[SIFRA_PREDMETA] .
```

ODGOVOR8

JMBAG
0246022858

ODGOVOR9

JMBAG
1191203289
0165043021
0246022858

Slika 5.8: Odgovori na upite s operacijom dijeljenja.

Kao što vidimo, operacija dijeljenja predstavlja način implementiranja univerzalne kvantifikacije u relacijskoj algebri. Dijeljenje se uvodi zbog udobnosti, naime ono nije nužno jer se može izraziti i preko prethodno opisanih operacija. Na primjer, za relacije $R(A,B,C,D)$ i $S(C,D)$ vrijedi:

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

Primijetimo da se dijeljenje donekle može smatrati inverznom operacijom u odnosu na Kartezijev produkt. Naime, ako je R dobivena kao $R = S_1$ times S_2, tada R divideby S_1 daje S_2, a R divideby S_2 daje S_1. No naravno, dijeljenje se može primijeniti i općenitije, dakle i na relacije koje nisu bile dobivene Kartezijevim produktom.

5.1.4. Prirodni spoj i slične operacije

Prirodni spoj (*natural join*) je binarna operacija primjenjiva na dvije relacije R i S koje imaju bar jedan zajednički atribut. R join S sastoji se od svih n-torki dobivenih spajanjem jedne n-torke iz R s jednom n-torkom iz S koja ima iste vrijednosti zajedničkih atributa. U rezultirajućoj relaciji zajednički atribut se pojavljuje samo jednom. Definicija se bolje može razumjeti na osnovu apstraktnog primjera sa Slike 5.9. Na toj slici isti simboli oblika a_i odnosno b_j odnosno c_k označavaju jednake vrijednosti atributa.

Kao što joj ime kaže, prirodni spoj je operacija koja na najprirodniji način uspostavlja veze između podataka u raznim relacijama. Na taj način dobivaju se lijepo oblikovani odgovori na složene upite. To ćemo opet ilustrirati primjerima s našom fakultetskom bazom podataka.

R

A	*B*	*C*
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_2
a_4	b_2	c_3

S

B	*C*	*D*
b_1	c_1	d_1
b_1	c_1	d_2
b_2	c_3	d_3

R join *S*

A	*B*	*C*	*D*
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_2	b_1	c_1	d_2
a_4	b_2	c_3	d_3

Slika 5.9: Apstraktni primjer prirodnog spoja.

Upit 10: pronađi prezimena i imena svih studenata koji su upisali predmet sa šifrom 56001.

```
ODGOVOR10 :=
    ( (UPISAO where SIFRA_PREDMETA = 56001) join STUDENT) [PREZIME, IME].
```

Upit 11: pronađi broj sobe nastavnika koji predaje predmet sa šifrom 72005.

```
ODGOVOR11 :=
    ( (PREDMET where SIFRA_PREDMETA = 72005) join NASTAVNIK)
    [BROJ_SOBE].
```

Upit 12: Pronađi OIB-e nastavnika koji predaju predmete koje je upisao bar jedan student na drugoj godini studija.

```
ODGOVOR12 :=
    ( ( (STUDENT where GODINA_STUDIJA = 2)  join UPISAO) join PREDMET)
    [OIB_NASTAVNIKA] .
```

Odgovori na Upite 10 i 11 prikazani su na Slici 5.10. Detaljno izvrednjavanje Upita 12 vidi se na Slici 5.11.

ODGOVOR10

PREZIME	IME
Petrović	Petar
Kolar	Ivan
Grubišić	Katica
Vuković	Janko

ODGOVOR11

BROJ_SOBE
305

Slika 5.10: Odgovori na Upite 10 i 11 koji rabe prirodni spoj.

Prirodni spoj uvodi se samo zbog udobnosti. Naime, riječ je o operaciji koja bi se uvijek mogla izraziti preko prije uvedenih operacija. Na primjer, za relacije *R*(*A, B, C*) i *S*(*C, D*) vrijedi:

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

STUDENT where GODINA_STUDIJA = 2

JMBAG	PREZIME	IME	GODINA_STUDIJA
1191203289	Petrović	Petar	2
1192130031	Horvat	Dragica	2

(STUDENT where GODINA_STUDIJA = 2) join UPISAO

JMBAG	PREZIME	IME	GODINA_ STUDIJA	SIFRA_ PREDMETA	DATUM_ UPISA	OCJENA
1191203289	Petrović	Petar	2	56001	2009-10-03	2
1191203289	Petrović	Petar	2	56002	2009-10-03	2
1192130031	Horvat	Dragica	2	72001	2009-09-15	5
1192130031	Horvat	Dragica	2	56002	2009-09-15	4
1192130031	Horvat	Dragica	2	72009	2009-09-15	2

((STUDENT where GODINA_STUDIJA = 2) join UPISAO) join PREDMET

JMBAG	....	SIFRA_ PREDMETA	....	OIB_ NASTAVNIKA	...
1191203289	....	56001	....	33571209458	...
1191203289	....	56002	....	67741205512	...
1192130031	....	72001	....	44102179316	...
1192130031	....	56002	....	67741205512	...
1192130031	....	72009	....	44102179316	...

ODGOVOR12

OIB_NASTAVNIKA
33571209458
67741205512
44102179316

Slika 5.11: Izvrednjavanje Upita 12 koji rabi dva prirodna spoja.

Osim prirodnog spoja, u literaturi se također spominje i nešto općenitija operacija koja se zove theta-spoj. Preciznije, neka su *R* i *S* relacije od kojih prva ima atribut *A*, a druga atribut *B*. *Theta-spoj* od *R* i *S* preko *A* i *B*, u oznaci *R* join(*A* θ *B*) *S*, definira se kao skup onih n-torki Kartezijevog produkta *R* sa *S* za koje je predikat *R.A* θ *S.B* istina. Pritom simbol θ predstavlja jedan od operatora za usporedbu: =, <, >, ≠, ≤, ≥ . Dakle, za razliku od prirodnog spoja gdje se n-torke iz različitih relacija spajaju samo na osnovu jednakosti vrijednosti atributa, theta-spoj omogućuje i općenitija spajanja na osnovu nejednakosti. Očito je da se theta-spoj uvijek može izraziti preko Kartezijevog produkta i selekcije.

Vanjski spoj (*outer join*) je još jedna operacija vrlo slična prirodnom spoju. Primjenjiva je pod istim uvjetima i daje kao rezultat relaciju s istom shemom. No sadržaj rezultirajuće relacije *R* outerjoin *S* je nešto bogatiji nego sadržaj od *R* join *S*. Naime, pored svih n-torki iz *R* join *S*, relacija *R* outerjoin *S* sadrži i sve „nesparene“ (nespojene) n-torke iz *R* odnosno *S*. Pritom su te nesparene n-torke na odgovarajući način proširene nedostajućim (null) vrijednostima. Dakle, za isti primjer *R* i *S* koji smo imali Slici 5.9, *R* outerjoin *S* izgleda kao što je prikazano na Slici 5.12.

R

A	*B*	*C*
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_2
a_4	b_2	c_3

S

B	*C*	*D*
b_1	c_1	d_1
b_1	c_1	d_2
b_2	c_3	d_3

R outerjoin *S*

A	*B*	*C*	*D*
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_2	b_1	c_1	d_2
a_4	b_2	c_3	d_3
a_3	b_2	c_2	-

Slika 5.12: Apstraktni primjer vanjskog spoja.

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

Upit 13: pronađi naslove predmeta koje do 31. prosinca 2009. godine nije upisao ni jedan student.

ODGOVOR13 :=
((PREDMET outerjoin (UPISAO where DATUM_UPISA < 2010-01-01))
where (JMBAG is null)) [NASLOV].

Ovdje smo upotrijebili uvjet „JMBAG is null“ koji je istinit za one n-torke gdje JMBAG nema upisanu vrijednost. Odgovor na upit vidi se na Slici 5.13.

PREDMET outerjoin (UPISAO where DATUM_UPISA < 2010-01-01)

SIFRA_PREDMETA	NASLOV	...	JMBAG	DATUM_UPISA	...
56001	Baze podataka	...	1191203289	2009-10-03	...
	Baze podataka	...	0165043021	2008-10-03	
	Baze podataka	...	0036448430	2009-09-04	
72001	Linearna algebra	...	1192130031	2009-09-15	...
	Linearna algebra	...	0165043021	2008-10-03	
56002	Programiranje u C-u	...	1191203289	2009-10-03	...
	Programiranje u C-u	...	1192130031	2009-09-15	
	Programiranje u C-u	...	0165043021	2009-06-10	
72009	Analitička geometrija	...	1192130031	2009-09-15	...
72005	Matematička analiza	...	-	-	-

ODGOVOR13

NASLOV
Matematička analiza

Slika 5.13: Izvrednjavanje Upita 13 koji rabi vanjski spoj.

Vanjski spoj *R* outerjoin *S* kojeg smo mi promatrali također se naziva i *simetrični* ili *dvostrani* vanjski spoj. Naime, u rezultirajuću relaciju on dodaje nesparene n-torke i iz *R* i iz *S*. Moguće je promatrati i *lijevi* odnosno *desni* vanjski spoj koji dodaje nesparene n-torke samo iz *R* odnosno samo iz *S*.

5.2. Relacijski račun

Relacijski račun također je uveo Edgar Codd u svojim radovima iz 70-tih godina 20. stoljeća, i to kao alternativu relacijskoj algebri. Riječ je opet o matematičkoj notaciji, no ovaj put ona je zasnovana na predikatnom računu. Upit se izražava tako da zadamo predikat kojeg tražene n-torke moraju zadovoljavati.

Postoje dvije varijante relacijskog računa: račun orijentiran na n-torke, gdje su osnovni objekti n-torke, te račun orijentiran na domene, gdje su osnovni objekti vrijednosti iz domena za atribute. Te dvije varijante detaljnije ćemo obraditi u sljedeća dva odjeljka ovog potpoglavlja. Zadnji odjeljak unutar potpoglavlja objasnit će odnos relacijskog računa i relacijske algebre.

5.2.1. Račun orijentiran na n-torke

U slučaju *računa orijentiranog na n-torke*, izrazi koji izražavaju upit sastoje se od sljedećih elemenata.

- Varijable n-torke, koje poprimaju vrijednosti iz imenovane relacije. Ako je t varijabla koja „prolazi“ relacijom R, a A je atribut od R, tada $t.A$ označava vrijednost od A unutar t.
- Uvjeti oblika $x\ \theta\ y$ gdje je θ operator za uspoređivanje =, <, >, ≠, ≤ ili ≥ . Bar jedno od x i y mora biti oblika $t.A$, a drugo može biti konstanta. Uvjeti također mogu biti oblika $R(t)$, što znači da je t n-torka u relaciji R.
- Dobro oblikovane formule (WFF). One su građene od logičkih veznika and, or, not, te egzistencijalnog kvantifikatora ∃ i univerzalnog kvantifikatora ∀, u skladu sa sljedećim pravilima.
  - Svaki uvjet je WFF.
  - Ako su f_1 i f_2 WFF, tada su to i f_1 and f_2, f_1 or f_2, i not f_1.
  - Ako je f jedna WFF u kojoj se t pojavljuje kao slobodna varijabla, tada su $\exists t\ (f)$ i $\forall t\ (f)$ također WFF. Pojava varijable u WFF je vezana ukoliko je ta varijabla uvedena kvantifikatorom, inače je slobodna.
  - Ništa drugo nije WFF.

Izraz računa orijentiranog na n-torke je oblika { $t.A, u.B, v.C, \ldots \mid f$ }, gdje su $t, u, v, \ldots$ varijable n-orke, $A, B, C, \ldots$ su atributi odgovarajućih relacija, a f je WFF koja sadrži $t, u, v, \ldots$ kao slobodne varijable.

U nastavku slijede primjeri upita u relacijskom računu orijentiranom na n-torke koji su vezani uz našu fakultetsku bazu podataka. Odgovori su prikazani na Slici 5.14.

Upit 14: pronađi sve šifre predmeta.

{ p.SIFRA_PREDMETA | PREDMET(p) } .

Upit 15: pronađi JMBAG-ove svih studenata na prvoj godini.

{ s.JMBAG | STUDENT(s) and s.GODINA_STUDIJA = 1} .

Upit 16: Pronađi JMBAG-ove i prezimena studenata koji su upisali predmet sa šifrom 56002.

{ *s*.JMBAG, *s*.PREZIME | STUDENT(*s*) and ∃*u* (UPISAO(*u*)
and *u*.JMBAG = *s*.JMBAG and *u*.SIFRA_PREDMETA = 56002) } .

Upit 17: pronađi šifre onih predmeta koje je upisao bar jedan student na drugoj godini.

{ *p*.SIFRA_PREDMETA | PREDMET(*p*) and ∃*u* (UPISAO(*u*) and
u.SIFRA_PREDMETA = *p*.SIFRA_PREDMETA and ∃*s* (STUDENT(*s*) and
s.JMBAG = *u*.JMBAG and *s*.GODINA_STUDIJA = 2)) } .

Upit 18: pronađi prezimena onih studenata koji su upisali sve predmete.

{ *s*.PREZIME | STUDENT (*s*) and i ∀*p* ∃*u* (PREDMET(*p*) and UPISAO(*u*) and
p.SIFRA_PREDMETA = *u*.SIFRA_PREDMETA and *u*.JMBAG = *s*.JMBAG} .

(odgovor na Upit 14)

SIFRA_ PREDMETA
56001
72001
72005
56002
72009

(Odgovor na Upit 15)

JMBAG
0036398757
1191205897
0246022858

(Odgovor na Upit 16)

JMBAG	PREZIME
1191203289	Petrović
1191130031	Horvat
0165043021	Kolar
0246022858	Vuković

(odgovor na Upit 17)

SIFRA_ PREDMETA
56001
56002
72001
72009

(Odgovor na Upit 18)

PREZIME
Vuković

Slika 5.14: Odgovori na upite u relacijskom računu.

5.2.2. Račun orijentiran na domene

Kod računa orijentiranog na domene varijable „prolaze“ domenama a ne relacijama. Također, imamo takozvane uvjete članstva oblika R (A: v_1, B: v_2, C: v_3, ...) gdje su A, B, C, ... atributi od relacije R, a v_1, v_2, v_3, ... su ili varijable ili konstante. Na primjer, uvjet STUDENT (JMBAG: 1191203289, GODINA_STUDIJA: 2) je istina ako i samo ako postoji n-torka u relaciji STUDENT takva da je JMBAG = 1191203289 i GODINA_STUDIJA = 2. Pravila za WFF su ista kao u računu orijentiranom na n-torke.

U nastavku slijede isti primjeri upita koje smo u prethodnom odjeljku riješili pomoću računa orijentiranom na n-torke. No sada su rješenja zapisana pomoću računa orijentiranog na domene. Odgovori na upite izgledaju isto kao na Slici 5.14.

Upit 14: pronađi sve šifre predmeta.

{ *S* | PREDMET (SIFRA_PREDMETA: *S*) } .

Upit 15: pronađi JMBAG-ove svih studenata na prvoj godini.

{ *J* | STUDENT (JMBAG: *J*, GODINA_STUDIJA: 1) } .

Upit 16: Pronađi JMBAG-ove i prezimena studenata koji su upisali predmet sa šifrom 56002.

{ *J*, *P* | STUDENT (JMBAG: *J*, PREZIME: *P*) and
UPISAO (JMBAG: *J*, SIFRA_PREDMETA: 56002) } .

Upit 17: pronađi šifre onih predmeta koje je upisao bar jedan student na drugoj godini.

{ *S* | PREDMET (SIFRA_PREDMETA: *S*) and
∃*J* (UPISAO (SIFRA_PREDMETA: *S*, JMBAG: *J*) and
STUDENT (JMBAG: *J*, GODINA_STUDIJA: 2)) } .

Upit 18: pronađi prezimena onih studenata koji su upisali sve predmete.

{ *P* | ∃*J* (STUDENT (JMBAG: *J*, PREZIME: *P*) and
∀*S* (if PREDMET (SIFRA_PREDMETA: *S*) then
UPISAO (JMBAG: *J*, SIFRA_PREDMETA: *S*))) } .

U zadnjem upitu smo se zbog udobnosti koristili implikacijom if ... then. To je dozvoljeno zato jer se implikacija može izraziti pomoću and, or i not.

5.2.3. Odnos relacijskog računa prema relacijskoj algebri

Srodnost između relacijskog računa i relacijske algebre je u tome što su ti jezici ekvivalentni u smislu izražajnosti. Dakle, svaki upit zapisan u relacijskoj algebri može se zamijeniti ekvivalentnim upitom u relacijskom računu. Također vrijedi i obratno: svaki upit zapisan pomoću relacijskog računa može se prevesti u ekvivalentan izraz u relacijskoj algebri. Pritom je svejedno služimo li se računom orijentiranim na n-torke ili na domene. Postoje doduše i upiti koji se ne mogu zapisati ni u jednom od razmatranih jezika.

Strogi dokaz tvrdnje o izražajnoj ekvivalentnosti relacijskog računa i relacijske algebre može se naći u literaturi. Prvu verziju tog dokaza objavio je već Edgar Codd u svojem članku iz 1972. godine. Štoviše, Codd je u istom radu izložio *redukcijski algoritam* kojim se upit u računu pretvara u izraz u algebri.

Mi se ovdje nećemo baviti strogim dokazivanjem ekvivalentnosti relacijskog računa i relacijske algebre. Umjesto toga, dajemo samo jednu ilustraciju. Za nekoliko najvažnijih algebarskih operacija navodimo ekvivalente u relacijskom računu orijentiranom na n-torke:

R_1 union R_2 ... { t | $R_1(t)$ or $R_2(t)$ } ,

R_1 minus R_2 ... { t | $R_1(t)$ and not $R_2(t)$ } ,

R_1 times R_2 ... { < t, r > | $R_1(t)$ and $R_2(r)$ } ,

R_1 where $\mathcal{B}(A)$ ... { t | $R_1(t)$ and $\mathcal{B}(t.A)$ } ,

$R_1[A]$ ... { $t.A$ | $R_1(t)$ } .

Ovdje < t, r > znači kombinaciju n-torki t i r, R_1 i R_2 su relacije, A je atribut odgovarajuće relacije, a $\mathcal{B}$ je Booleovski uvjet koji se odnosi na atribut A.

Glavna razlika između relacijskog računa i relacijske algebre je u tome što je račun u mnogo većoj mjeri „neproceduralan“ od algebre. Naime, kod postavljanja upita u relacijskoj algebri, korisnik dobrim dijelom konstruira postupak odgovaranja na upit time što bira odgovarajuće operacije i utvrđuje redoslijed njihovog izvršavanja. S druge strane, kod postavljanja upita u relacijskom računu, korisnik samo specificira podatke koje želi dobiti a ne zadaje način kako da se do tih podataka dođe.

Svojstvo neproceduralnosti smatra se poželjnim za neposredne korisnike. Stoga praktični jezici za postavljanje upita više liče na relacijski račun nego na relacijsku algebru. Na primjer, najraširenij današnji jezik SQL uglavnom je zasnovan na računu orijentiranom na n-torke. Drugi poznati jezik QBE (Query by Example) rabi račun orijentiran na domene. No bez obzira kako izgledao jezik za korisnike, DBMS prilikom interpretiranja tog jezika mora odrediti postupak pronalaženja traženih podataka, to jest on mora postavljeni upit nekom vrstom redukcijskog algoritma prevesti u neki oblik relacijske algebre.

Sve u svemu, relacijski račun možemo smatrati matematičkim modelom za način kako bi korisnici trebali postavljati upite. Relacijsku algebru možemo smatrati matematičkim modelom za način kako će DBMS dalje interpretirati i izvrednjavati korisničke upite.

5.3. Jezik SQL

SQL je najraširenij jezik za rad s relacijskom bazom podataka. Sama kratica znači *Structured Query Language*. Jezik je nastao u 70-tim godinama 20. stoljeća u sklopu razvojno-istraživačkog projekta *System R* unutar kompanije IBM. Voditelj tog projekta i glavni dizajner SQL-a bio je Donald Chamberlin. Jezik se postepeno usavršavao, a njegova dotjerana varijanta pojavljuje se u današnjem IBM-ovom relacijskom DBMS-u zvanom DB2.

U širenju SQL-a tijekom 80-tih godina važnu ulogu odigrala je softverska kuća Oracle Corporation: ona je ugradila SQL u svoj DBMS, te ga je time učinila dostupnim i popularnim na svim važnijim računalnim platformama. Drugi tadašnji proizvođači DBMS-a, na primjer Ingres Corporation, Digital Equipment Corporation, Informix Inc, Sybase Inc, bili su pod pritiskom tržišta prisiljeni prihvatiti SQL i odustati od eventualnih vlastitih jezika. Uporabu SQL-a u razvoju web aplikacija tijekom 90-tih godina potakla je kompanija MySQL AB svojim besplatnim DBMS-om. Zbog pojave raznih „dijalekata“, već 1986. godine bio je donesen prvi ISO/ANSI standard za SQL - njegova zadnja verzija objavljena je 2008. godine.

U skladu sa svojim imenom, SQL u prvom redu omogućuje postavljanje upita u relacijskim bazama podataka. No jezik se ne zaustavlja na tome: postoje i naredbe za stvaranje relacija, unos, promjenu i mijenjanje podataka, upravljanje transakcijama, davanje i oduzimanje ovlaštenja korisnicima. Također postoje i brojne funkcije za računanje s podacima. U ovom potpoglavlju bavit ćemo se isključivo postavljanjem upita. Ostale naredbe ukratko će se obraditi u potpoglavljima 6.2, 7.1 i 7.2.

Upitni dio SQL-a uglavnom je zasnovan na relacijskom računu, s time da je matematička notacija zamijenjena ključnim riječima nalik na govorni engleski jezik. No lagano se realiziraju i sve operacije iz relacijske algebre. Zahvaljujući takvim svojstvima, SQL je u izražajnom smislu ekvivalentan relacijskoj algebri odnosno relacijskom računu, dakle svaki SQL upit može se prevesti u algebru ili račun, te obratno, svaki upit zapisan u algebri ili računu može se izraziti i u SQL-u. Spomenuta ekvivalencija je važna za interpretiranje SQL upita, naime DBMS odgovara na upit tako da ga interno prevede u neku vrstu algebarskog izraza kojeg dalje izvrednjava.

Nastavak ovog potpoglavlja sastoji se od tri odjeljka. U prvom od njih dajemo primjere jednostavnih upita u SQL-u koji pretražuju jednu relaciju. Drugi odjeljak bavi se složenijim upitima gdje sudjeluje više relacija ili se pojavljuju ugniježđeni pod-upiti. Treći odjeljak obrađuje grupirajuće upite, dakle upite koji umjesto podataka pohranjenih u bazi ispisuju podatke dobivene grupiranjem i daljnjom obradom pohranjenih podataka.

5.3.1. Jednostavni upiti

Svi upiti u SQL-u, pa tako i oni najjednostavniji, postavljaju se fleksibilnom naredbom SELECT. Naglasimo da ta naredba nije isto što i operacija selekcije iz relacijske algebre, mada može obavljati i njezinu zadaću. Rezultat izvođenja naredbe SELECT shvaća se kao nova, bezimena i privremena relacija koja ispisuje na zaslonu ili prosljeđuje u aplikacijski program.

U nastavku slijede primjeri jednostavnih SQL upita koji su vezani uz našu fakultetsku bazu podataka sa Slike 5.1. Odgovori su prikazani na Slici 5.15.

Upit 19: pronađi JMBAG-ove, prezimena i imena svih studenata na drugoj godini studija.

```
SELECT JMBAG, PREZIME, IME FROM STUDENT
WHERE GODINA_STUDIJA = 2;
```

Upit 20: ispiši OIB-e nastavnika koji predaju bar jedan predmet.

```
SELECT DISTINCT OIB_NASTAVNIKA FROM PREDMET;
```

Upit 21: ispiši sve podatke o nastavnicima u obliku rang liste s obzirom na njihove plaće.

```
SELECT * FROM NASTAVNIK ORDER BY PLACA DESC;
```

Upit 22: pronađi JMBAG-ove onih studenata koji su iz predmeta sa šifrom 56001 dobili ocjenu veću od 2.

```
SELECT JMBAG FROM UPISAO
WHERE SIFRA_PREDMETA = 56001
AND OCJENA > 2;
```

ili

```
SELECT JMBAG FROM UPISAO
WHERE SIFRA_PREDMETA =
56001 AND OCJENA IN (3, 4, 5);
```

Upit 23: ispiši imena studenata koja počinju na slovo 'M'.

```
SELECT IME FROM STUDENT
WHERE IME BETWEEN 'B' AND 'C';
```

ili

```
SELECT IME FROM STUDENT
WHERE IME LIKE 'B%';
```

Upit 24: fakultet je odlučio svim nastavnicima udvostručiti plaću. Ispišite prezimena svih nastavnika zajedno s uvećanim plaćama, sortirano po prezimenima.

```
SELECT PREZIME, PLACA*2 FROM NASTAVNIK ORDER BY PREZIME;
```

(Odgovor na Upit 19)

JMBAG	PREZIME	IME
1191203289	Petrović	Petar
1192130031	Horvat	Dragica

(Odgovor na Upit 20)

OIB_NASTAVNIKA
33571209458
44102179316
25810043761
67741205512

(Odgovor na Upit 21)

OIB	PREZIME	IME	IME_ZAVODA	BROJ_SOBE	PLACA
33571209458	Codd	Edgar	Zavod za računarstvo	127	16000
25810043761	Goedel	Kurt	Zavod za matematiku	305	14000
67741205512	Turing	Alan	Zavod za računarstvo	315	13000
13257600947	Cantor	Georg	Zavod za matematiku	102	12000
44102179316	Klein	Felix	Zavod za matematiku	252	12000
50076128203	Pascal	Blaise	Zavod za matematiku	101	11000

(Odgovor na Upit 22)

JMBAG
0165043021
0036448430

(Odgovor na Upit 23)

IME
Marko
Marija

(Odgovor na Upit 24)

PREZIME	PLACA * 2
Cantor	24000
Codd	32000
Goedel	28000
Klein	24000
Pascal	22000
Turing	26000

Slika 5.15: Odgovori na jednostavne upite u SQL-u.

Kao što smo vidjeli, u naredbi za postavljanje upita obavezno se pojavljuju ključne riječi SELECT i FROM, a po potrebi i WHERE i ORDER BY. Naredba se može protezati kroz više redaka, no mora završiti s točka-zarezom. Između riječi SELECT i FROM navode se imena atributa koje želimo ispisati, iza FROM slijede imena relacija iz kojih se čitaju podaci, a iza WHERE navode se uvjeti koje podaci moraju zadovoljiti. ORDER BY omogućuje da se zadaje način kako će ispis biti sortiran.

5.3.2. Složeniji upiti

Upiti u SQL-u mogu poprimiti puno složenije oblike od onih koje smo vidjeli u prethodnom odjeljku. Kao prvo, dozvoljeno je odjednom rabiti više relacija, te kombinirati podatke iz njih. Također, jedna SELECT naredba može se ugnijezditi unutar druge, tako da rezultat prve naredbe služi kao dio uvjeta u drugoj.

U nastavku slijedi nekoliko primjera složenijih SQL upita koji su opet vezani uz našu fakultetsku bazu podataka sa Slike 5.1. Odgovori su prikazani na Slici 5.16.

Upit 25: pronađi JMBAG-ove i prezimena studenata koji su upisali predmet sa šifrom 72009.

```
SELECT STUDENT.JMBAG, PREZIME
FROM STUDENT, UPISAO
WHERE STUDENT.JMBAG = UPISAO.JMBAG
AND SIFRA_PREDMETA = 72009;
```

ili

```
SELECT JMBAG, PREZIME
FROM STUDENT
WHERE JMBAG IN
    (SELECT JMBAG
     FROM UPISAO
     WHERE SIFRA_PREDMETA = 72009);
```

Prvo rješenje zasnovano je na jednostrukoj SELECT naredbi koja simultano rabi dvije relacije. U takvom slučaju zapravo se stvara Kartezijev produkt, to jest naredba proizvodi sve kombinacije n-torki iz prve relacije s n-torkama iz druge relacije. No uvjet iza WHERE izbacit će one kombinacije n-torki koje nas ne zanimaju. Primijetimo da smo ime prvog atributa iza ključne riječi SELECT morali proširiti s imenom relacije kao prefiksom – to je zato da bismo izbjegli dvoznačnost, naime isto ime atributa pojavljuje se u obje relacije. Drugo rješenje za isti upit rabi ugniježđenu SELECT naredbu koja pronalazi popis JMBAG-ova studenata koji su upisali traženi predmet. Operator IN provjerava nalazi li se zadana vrijednost podataka u zadanom skupu vrijednosti.

Upit 26: pronađi JMBAG-ove i prezimena studenata koji su upisali barem jedan predmet koji predaje nastavnik s OIB-om 44102179316.

```
SELECT STUDENT.JMBAG, PREZIME
FROM STUDENT, UPISAO, PREDMET
WHERE STUDENT.JMBAG = UPISAO.JMBAG
AND UPISAO.SIFRA_PREDMETA =
   PREDMET.SIFRA_PREDMETA
AND OIB_NASTAVNIKA = 44102179316;
```

ili

```
SELECT JMBAG, PREZIME
FROM STUDENT
WHERE JMBAG IN
   (SELECT JMBAG
    FROM UPISAO
    WHERE SIFRA_PREDMETA IN
      (SELECT SIFRA_PREDMETA
       FROM PREDMET WHERE
       OIB_NASTAVNIKA = 44102179316));
```

Opet imamo dva rješenja. Prvo rješenje zasnovano je na jednostrukoj SELECT naredbi koja simultano rabi tri relacije. Drugo rješenje je niz od tri SELECT naredbe, gdje je prva od njih ugniježđena drugoj, a druga u trećoj.

Upit 27: pronađi sve parove JMBAG-ova studenata koji su na istoj godini studija.

```
SELECT TEMP1.JMBAG, TEMP2.JMBAG
FROM STUDENT TEMP1, STUDENT TEMP2
WHERE TEMP1.JMBAG < TEMP2.JMBAG
AND TEMP1.GODINA_STUDIJA = TEMP2.GODINA_STUDIJA;
```

Kao što vidimo, rješenje je analogno onome kojim smo se koristili u relacijskoj algebri. Dakle naša SELECT naredba stvara Kartezijev produkt relacije STUDENT same sa sobom, a uvjet iza WHERE izdvaja bez ponavljanja kombinacije različitih studenata koji su na istoj godini. Da bi se ista relacija STUDENT bez dvoznačnosti mogla rabiti kao da je riječ o dvije relacije,

morali smo uvesti nadimke (aliase) TEMP1 odnosno TEMP2 za njezinu prvu odnosno drugu pojavu.

Upit 28: ispiši prezimena i plaće za sve nastavnike koji imaju veću plaću od nastavnika Cantora.

```
SELECT N2.PREZIME, N2.PLACA
FROM NASTAVNIK N1, NASTAVNIK N2
WHERE N1.PREZIME = 'Cantor'
AND N2.PLACA > N1.PLACA;
```

ili

```
SELECT PREZIME, PLACA
FROM NASTAVNIK
WHERE PLACA >
   (SELECT PLACA
    FROM NASTAVNIK
    WHERE PREZIME = 'Cantor');
```

Opet imamo dva rješenja. Prvo od njih simultano rabi dvije „kopije“ relacije NASTAVNIK, pa smo za te dvije kopije morali uvesti nadimke N1 odnosno N2. Drugo rješenje oslanja se na činjenicu da će ugniježđeni upit proizvesti samo jednu vrijednost atributa koju dalje možemo uspoređivati s operatorom >.

Upit 29: ispiši sve podatke o predmetima koje nije upisao student Kolar.

```
SELECT * FROM PREDMET
WHERE SIFRA_PREDMETA NOT IN
  (SELECT SIFRA_PREDMETA FROM UPISAO, STUDENT
   WHERE UPISAO.JMBAG = STUDENT.JMBAG
   AND PREZIME = 'Kolar');
```

Budući da su u relaciji UPISAO studenti zadani pomoću JMBAG-a a ne pomoću prezimena, u ugniježđenom upitu morali smo se također koristiti i podacima iz relacije STUDENT.

Upit 30: pronađi JMBAG-ove, prezimena i imena onih studenata koji su upisali sve predmete.

```
SELECT JMBAG, PREZIME, IME
FROM STUDENT
WHERE NOT EXISTS
   (SELECT *
    FROM  PREDMET
    WHERE NOT EXISTS
        (SELECT *
         FROM UPISAO
         WHERE UPISAO.JMBAG = STUDENT.JMBAG
          AND UPISAO.SIFRA_PREDMETA = PREDMET.SIFRA_PREDMETA));
```

Budući, da SQL nema univerzalni kvantifikator ali ima egzistencijalni, upit smo ovako morali preformulirati: pronađi one studente za koje ne postoji predmet kojeg oni nisu upisali. Rješenje slijedi takvu promijenjenu formulaciju, svodi se na tri ugniježđene SELECT naredbe, te se koristi egzistencijalnim kvantifikatorom EXISTS. Uvjet oblika NOT EXISTS (SELECT …) je istina ako i samo ako je rezultat uključene SELECT naredbe prazan.

Upit 31: ispiši prezimena i imena nastavnika koji ništa ne predaju.

```
SELECT PREZIME, IME
FROM NASTAVNIK
WHERE OIB NOT IN  (SELECT OIB_NASTAVNIKA FROM PREDMET);
```

Predloženo rješenje vrlo je čitljivo i zasniva se na ugniježđenom upitu. Upit je moguće izraziti i s jednostrukom SELECT naredbom, no tada morali rabiti neku vrstu vanjskog spoja.

(Odgovor na Upit 25)

JMBAG	PREZIME
1192130031	Horvat
1191205897	Janković
0246022858	Vuković

(Odgovor na Upit 26)

JMBAG	PREZIME
0036398757	Marković
1192130031	Horvat
1191205897	Janković
0165043021	Kolar
0246022858	Vuković

(Odgovor na Upit 27)

TEMP1.JMBAG	TEMP2.JMBAG
0036398757	1191205897
0036398757	0246022858
0246022858	1191205897
1191203289	1192130031
0036448430	0165043021

(Odgovor na Upit 28)

PREZIME	PLACA
Goedel	14000
Codd	16000
Turing	13000

(Odgovor na Upit 29)

SIFRA_ PREDMETA	NASLOV	IME_ZAVODA	OIB_ NASTAVNIKA	SEMESTAR	ECTS_ BODOVI
72005	Matematička analiza	Zavod za matematiku	25810043761	L	6
72009	Analitička geometrija	Zavod za matematiku	44102179316	L	5

(Odgovor na Upit 30)

JMBAG	PREZIME	IME
0246022858	Vuković	Janko

(Odgovor na Upit 31)

PREZIME	IME
Cantor	Georg
Pascal	Blaise

Slika 5.16: Odgovori na složenije upite u SQL-u.

5.3.3. Grupirajući upiti

Do sada su se odgovori na naše upite sastojali od vrijednosti koje se doslovno pojavljuju u pojedinim n-torkama pojedinih relacija baze. No često su nam potrebne i vrijednosti koje odgovaraju cijelim grupama (ustvari skupovima) n-torki. Takve grupne vrijednosti nastaju primjenom neke od grupnih funkcija. Na primjer, postoji funkcija koja daje broj n-torki u grupi, ili zbroj vrijednosti nekog izraza za sve n-torke u grupi, ili minimum vrijednosti izraza, maksimum, prosjek, i slično.

U nastavku slijedi nekoliko primjera grupirajućih SQL-upita, dakle upita gdje se n-torke iz baze razvrstavaju u grupe, pa se na te grupe primjenjuju grupne funkcije. Primjeri su opet vezani uz našu fakultetsku bazu podataka. Za stanje baze sa Slike 5.1 odgovori na naše grupirajuće upite prikazani su na Slici 5.17.

Upit 32: ispiši najmanju i najveću plaću nastavnika, te zbroj plaća za sve nastavnike.

```
SELECT MIN(PLACA), MAX(PLACA), SUM (PLACA)
FROM NASTAVNIK;
```

Ovo je najjednostavnija vrsta grupirajućeg upita. Grupa je samo jedna i sastoji se od svih n-torki relacije NASTAVNIK. To znači da će upit proizvesti samo jedan redak ispisa. Primjenom grupnih funkcija pronalazimo minimum, maksimum odnosno zbroj vrijednosti atributa PLACA na skupu svih nastavnika.

Upit 33: sastavi izvještaj koliko ima studenata na kojoj godini.

```
SELECT GODINA_STUDIJA, COUNT(*)
FROM STUDENT
GROUP BY GODINA_STUDIJA;
```

Ovdje je riječ o upitu gdje skup n-torki relacije STUDENT treba podijeliti u grupe s obzirom na vrijednosti atributa GODINA_STUDIJA. Upit će proizvesti onoliko redaka ispisa koliko ima različitih godina studija. Grupiranje po godini studija postiže se primjenom klauzule GROUP_BY. Funkcija COUNT(*) daje broj n-torki u grupi, s time da se broje sve n-torke, čak i one koje sadrže null-vrijednosti. Jedini izrazi koji se u grupirajućem upitu smiju pojaviti u listi iza klauzule SELECT su vrijednosti grupnih funkcija ili atribut po kojem je izvršeno grupiranje.

Upit 34: sastavi izvještaj koliko ima nastavnika u pojedinom zavodu.

```
SELECT IME_ZAVODA, COUNT(*) BROJ_NASTAVNIKA
FROM NASTAVNIK
GROUP BY IME_ZAVODA;
```

Upit je sličan prethodnom, jedino što sada dijelimo u grupe n-torke iz relacije NASTAVNIK, a kriterij grupiranja je vrijednost atributa IME_ZAVODA. U ispisu se pojavljuje mali kozmetički dodatak: izraz COUNT(*) preimenovan je u BROJ_NASTAVNIKA, pa će se takvo ime stupca pojaviti u zaglavlju ispisane tablice.

Upit 35: odredi prosječnu plaću nastavnika za svaki pojedini zavod.

```
SELECT IME_ZAVODA, AVG(PLACA) PROSJECNA_PLACA
FROM NASTAVNIK
GROUP BY IME_ZAVODA;
```

I ovaj upit je sličan prethodnima. Pojavljuje se funkcija AVG(…) koja daje srednju vrijednost (aritmetičku sredinu) zadanog izraza za n-torke u grupi.

Upit 36: ispiši popis JMBAG-ova, prezimena i imena svih studenata, te za svakog studenta broj koliko je predmeta on upisao. Popis treba biti leksikografski sortiran po prezimenima studenata.

```
SELECT STUDENT.JMBAG, PREZIME, IME, COUNT(*) UPISAO_PREDMETA
FROM STUDENT, UPISAO
WHERE STUDENT.JMBAG = UPISAO.JMBAG
GROUP BY STUDENT.JMBAG
ORDER BY PREZIME;
```

Ovdje vidimo složeniju SELECT naredbu, gdje se najprije kombiniraju n-torke iz dviju relacija STUDENT i UPISAO, zatim se klauzulom WHERE izdvajaju kombinacije s istim JMBAG-om, te se na kraju takve kombinirane i izdvojene n-torke grupiraju u skladu s JMBAG-om. Dakle uvjet iza klauzule WHERE primjenjuje se prije grupiranja. Klauzula ORDER BY primjenjuje se nakon grupiranja i njome se sortiraju redci ispisa koji odgovaraju grupama.

Upit 37: ispiši JMBAG-ove studenata koji su položili manje od dva predmeta, zajedno s brojem predmeta koje su položili.

```
SELECT JMBAG, COUNT(OCJENA) POLOZIO_PREDMETA
FROM STUDENT
GROUP BY JMBAG
HAVING COUNT(OCJENA) < 2;
```

Klauzula HAVING je analogna klauzuli WHERE, no odnosi se na grupe dobivene pomoću GROUP BY, a ne na polazne n-torke. Dakle, klauzula HAVING djeluje nakon grupiranja i njome je moguće izdvojiti samo neke grupe te izbaciti neželjene grupe. Funkcija COUNT(…) daje broj ne-null vrijednosti zadanog izraza u n-torkama iz grupe, s time da se broje sve pojave iste vrijednosti. Za prebrojavanje koliko ima različitih ne-null vrijednosti, morali bismo rabiti COUNT (DISTINCT …).

Upit 38: ispiši prezimena i imena studenata s druge ili treće godine koji su skupili barem 10 ECTS-bodova, zajedno s brojem skupljenih ECTS-bodova. Ispis treba biti silazno sortiran prema broju ECTS-bodova.

```
SELECT PREZIME, IME, SUM(ECTS_BODOVI) ZBROJ_ECTS
FROM UPISAO, STUDENT, PREDMET
WHERE UPISAO.JMBAG = STUDENT.JMBAG
AND UPISAO.SIFRA_PREDMETA = PREDMET.SIFRA_PREDMETA
AND GODINA_STUDIJA >= 2
GROUP BY UPISAO.JMBAG
HAVING SUM(ECTS_BODOVI) >= 10
ORDER BY 3  DESC;
```

Ovo je najsloženiji primjer u ovom odjeljku, gdje se pojavljuje više relacija, izdvajanje n-torki pomoću WHERE, grupiranje, izdvajanje grupa pomoću HAVING, te sortiranje ispisa. Naglasimo još jednom da WHERE djeluje prije grupiranja, a HAVING i ORDER BY nakon grupiranja.

(Odgovor na Upit 32)

MIN(PLACA)	MAX(PLACA)	SUM(PLACA)
11000	16000	78000

(Odgovor na Upit 33)

GODINA_ STUDIJA	COUNT(*)
1	3
2	2
3	2

(Odgovor na Upit 34)

IME_ZAVODA	BROJ_ NASTAVNIKA
Zavod za matematiku	4
Zavod za računarstvo	2

(Odgovor na Upit 35)

IME_ZAVODA	PROSJECNA_ PLACA
Zavod za matematiku	12250
Zavod za računarstvo	14500

(Odgovor na Upit 36)

STUDENT.JMBAG	PREZIME	IME	UPISAO_PREDMETA
0036448430	Grubišić	Katica	1
1192130031	Horvat	Dragica	3
1191205897	Janković	Marija	1
0165043021	Kolar	Ivan	3
0036398757	Marković	Marko	1
1191203289	Petrović	Petar	2
0246022858	Vuković	Janko	5

(Odgovor na Upit 37)

JMBAG	POLOZIO_PREDMETA
0036398757	0
1191205897	0
0036448430	1
0246022858	0

(Odgovor na Upit 38)

PREZIME	IME	ZBROJ_ECTS
Horvat	Dragica	16
Kolar	Ivan	16
Petrović	Petar	10

Slika 5.17: Odgovori na grupirajuće upite u SQL-u.

5.4. Zadaci za vježbu

Zadatak 5.1. Promatramo bazu podataka o gurmanima, jelima i restoranima. Baza bilježi koji gurman posjećuje koji restoran, koji restoran poslužuje koje jelo, te koji gurman voli koje jelo. Relacijska shema izgleda ovako:

POSJECUJE (IME_GURMANA, IME_RESTORANA)
POSLUZUJE (IME_RESTORANA, IME_JELA)
VOLI (IME_GURMANA, IME_JELA).

Za promatranu bazu sljedeće upite zapišite u relacijskoj algebri:

- pronađi restorane koji poslužuju bar jedno jelo koje voli gurman Pero.
- pronađi gurmane koji posjećuju bar jedan restoran koji poslužuje jelo koji oni vole.
- pronađi restorane koji poslužuju sva jela koje voli gurman Pero.

Zadatak 5.2. Za bazu podataka o gurmanima, upite iz Zadatka 5.1 zapišite u relacijskom računu orijentiranom na n-torke.

Zadatak 5.3. Za bazu podataka o gurmanima, upite iz Zadatka 5.1 zapišite u relacijskom računu orijentiranom na domene.

Zadatak 5.4. Za bazu podataka o gurmanima, upite iz Zadatka 5.1 zapišite u jeziku SQL.

Zadatak 5.5. Promatramo bazu podataka o knjižnici. Relacije govore o knigama, članovima i o posudbama knjiga članovima. Relacijska shema izgleda ovako:

KNJIGA (KATALOSKI_BROJ, NASLOV, AUTOR, IZDAVAC)
CLAN (BROJ_ISKAZNICE, PREZIME, IME, ADRESA)
POSUDBA (KATALOSKI_BROJ, BROJ_ISKAZNICE, DATUM_POSUDJIVANJA).

Atribut KATALOSKI_BROJ jednoznačno određuje izdanje knjige, a BROJ_ISKAZNICE jednoznačno određuje člana knjižnice. Za promatranu bazu sljedeće upite zapišite u relacijskoj algebri:

- pronađi naslove i autore svih knjiga koje je izdao izdavač Prentice-Hall.
- pronađi naslove svih knjiga koje su bile posuđene 15. veljače 2011. godine.
- pronađi prezime i ime člana kojem je knjiga s naslovom „Kome zvono zvoni“ bila posuđena 12. ožujka 2011. godine.
- ispiši naslov i autora bilo koje knjige izdane od Prentice-Hall koja je bila posuđena članu s prezimenom Perić prije 21. srpnja 2010. godine.
- ispiši prezimena, imena i adrese članova kojima su bile posuđene sve knjige napisane od autora Hemingwaya.
- ispiši naslove i autore svih knjiga koje nikad nisu bile posuđene.
- ispiši sve podatke o knjigama koje nikad nisu bile posuđene članu s brojem iskaznice 34216.
- pronađi imena članova koji su posudili sve one knjige koje je posudio član s brojem iskaznice 67542.

Zadatak 5.6. Za bazu podataka o knjižnici, upite iz Zadatka 5.5 zapišite u relacijskom računu orijentiranom na n-torke.

Zadatak 5.7. Za bazu podataka o knjižnici, upite iz Zadatka 5.5 zapišite u relacijskom računu orijentiranom na domene.

Zadatak 5.8. Za bazu podataka o knjižnici, upite iz Zadatka 5.5 zapišite u jeziku SQL.

Zadatak 5.9. Promatramo tri relacije: $R(A,B)$, $S(B,C)$, $T(C,D)$. Dokažite da vrijedi sljedeća jednakost algebarskih izraza:

$$(R \text{ join } S) \text{ join } T = R \text{ join } (S \text{ join } T) .$$

6. Fizičko oblikovanje i implementacija baze podataka

Ovo poglavlje posvećeno je trećoj fazi oblikovanja baze podataka, a to je fizičko oblikovanje. Glavni cilj te faze je stvoriti *fizičku shemu* baze, dakle opis njezine fizičke građe. Fizička shema zapravo je tekst sastavljen od naredbi u SQL-u ili nekom drugom jeziku kojeg razumije DBMS. Izvođenjem tih naredbi DBMS automatski stvara fizičku bazu. Dakle, od fizičkog oblikovanja baze do njezine stvarne implementacije vrlo je kratak put.

U ovom poglavlju također ćemo opisati načine na koji je baza podataka fizički realizirana. Pritom nas zanima i statički i dinamički aspekt te realizacije. Dakle opisat ćemo samu fizičku građu baze, ali isto tako i algoritme koji omogućuju rad s pohranjenim podacima, na primjer odgovaranje na složene upite. Ova znanja nisu izravno potrebna projektantu baze podataka, budući da se o detaljima fizičke realizacije zapravo brine DBMS. Ipak, dobro je da projektant razumije o čemu je tu riječ, jer će na taj način biti u stanju bolje podesiti one parametre unutar fizičkog oblikovanja kojima raspolaže.

Poglavlje je podijeljeno u tri potpoglavlja. U prvom potpoglavlju opisujemo fizičku građu baze. Drugo potpoglavlje izravno se bavi izradom fizičke sheme na osnovu relacijske sheme, te implementacijom baze. U trećem potpoglavlju govorimo o izvrednjavanju upita nad podacima koji su pohranjeni u implementiranoj bazi.

6.1. Fizička građa baze podataka

Fizička baza podataka gradi se od datoteka i indeksa pohranjenih na disku. U ovom potpoglavlju najprije proučavamo elemente fizičke građe, a to su blokovi, zapisi i pokazivači. Zatim objašnjavamo kako se ti elementi organiziraju u datoteke i indekse. To nam omogućuje da bolje razumijemo način na koji DBMS početnu verziju fizičke sheme, sastavljenu od SQL naredbi CREATE TABLE i CREATE INDEX, pretvara u fizičku bazu.

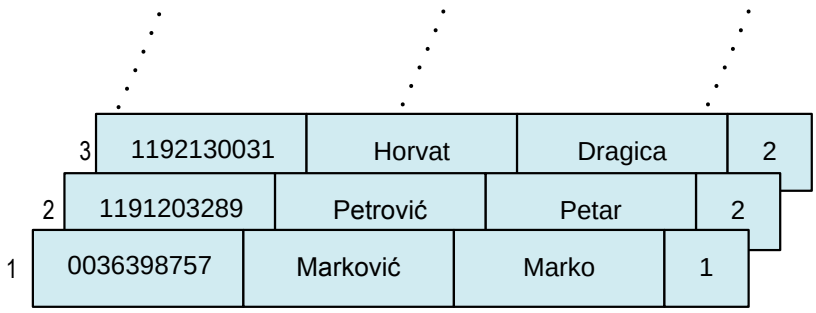
6.1.1. Elementi fizičke građe

Baza podataka fizički se pohranjuje u vanjskoj memoriji računala, najčešće na magnetskom disku. Važno je znati da operacijski sustav računala dijeli vanjsku memoriju u jednako velike *blokove* (sektore). Veličina bloka je konstanta operacijskog sustava i ona može iznositi na primjer 512 byte ili 4096 byte. Svaki blok jednoznačno je zadan svojom *adresom*.

Osnovna operacija s vanjskom memorijom je prijenos bloka sa zadanom adresom iz vanjske memorije u glavnu, ili obratno. Dio glavne memorije koji sudjeluje u prijenosu (i ima jednaku veličinu kao i sam blok) zove se *buffer*. Blok je najmanja količina podataka koja se može prenijeti; na primjer ako želimo pročitati samo jedan byte iz vanjske memorije, tada moramo prenijeti cijeli odgovarajući blok, pretražiti buffer u glavnoj memoriji i izdvojiti traženi byte. Vrijeme potrebno za prijenos bloka (mjereno u milisekundama) neusporedivo je veće od vremena potrebnog za bilo koju radnju u glavnoj memoriji (mjereno u mikro- ili nanosekundama). Zato je brzina nekog algoritma za rad s vanjskom memorijom određena brojem blokova koje algoritam mora prenijeti, a vrijeme potrebno za postupke u glavnoj memoriji je zanemarivo.

Osnovna struktura koja se pojavljuje u fizičkoj građi baze podataka naziva se *datoteka*. Riječ je o pojmu koji nam je poznat iz programskih jezika poput C-a ili COBOL-a. Datoteka je konačni niz *zapisa* (slogova) istog tipa pohranjenih u vanjskoj memoriji. *Tip zapisa* zadaje se kao uređena n-torka *osnovnih podataka* (komponenti), gdje je svaki osnovni podatak opisan svojim imenom i tipom (cijeli ili realni broj, znak, niz znakova, …). Sam zapis sastoji se od konkretnih vrijednosti osnovnih podataka. Smatramo da su zapisi fiksne duljine, dakle jedan zapis ima točno jednu vrijednost svakog od osnovnih podataka i ta vrijednost je prikazana fiksiranim brojem byte-ova. Tipične operacije koje se obavljaju nad datotekom su: ubacivanje novog zapisa, promjena postojećeg zapisa, izbacivanje zapisa, ili pronalaženje zapisa gdje zadani osnovni podaci imaju zadane vrijednosti.

Jedna datoteka obično služi za fizičko prikazivanje jedne relacije iz relacijske baze. Na primjer, promotrimo opet relaciju STUDENT sa Slike 3.1 koja sadrži podatke o studentima upisanim na fakultet. Da bismo fizički prikazali tu relaciju, svaku njezinu n-torku pretvaramo u zapis, te zapise poredamo u nekom redoslijedu, pa ih pohranimo na disk. Na taj način dobivamo niz zapisa pohranjenih na disk, dakle datoteku. Ideja je ilustrirana Slikom 6.1. Pogodan tip zapisa za podatke o studentu mogao bi se precizno definirati u programskom jeziku poput C-a ili COBOL-a, te bi se na primjer sastojao od: niza od 10 znamenki (JMBAG), dva niza od po 20 znakova (PREZIME, IME), te jedne dodatne znamenke (GODINA STUDIJA).



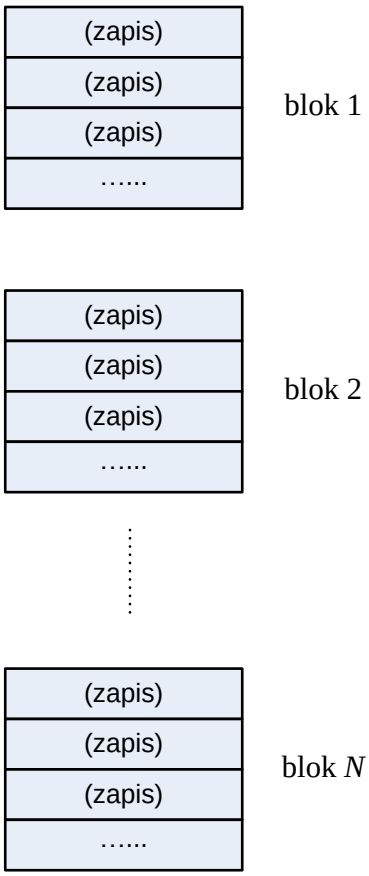
Slika 6.1: Datoteka s podacima o studentima na fakultetu.

Primijetimo da smo prilikom pretvorbe relacije STUDENT u datoteku nužno morali uvesti brojne fizičke detalje koji nisu postojali u relacijskom modelu, na primjer točnu duljinu pojedinog podatka u byte-ovima, redoslijed podataka unutar zapisa, međusobni redoslijed zapisa, i tako dalje.

Slično kao kod relacija, i za datoteke se može uvesti pojam ključa. *Kandidat za ključ* je osnovni podatak, ili kombinacija osnovnih podataka, čija vrijednost jednoznačno određuje zapis unutar datoteke. Ukoliko ima više kandidata za ključ, tada odabiremo jednog od njih da bude *primarni* ključ. Primijetimo da, za razliku od relacije, datoteka ne mora imati ključ, jer mogu postojati zapisi-duplikati. Ipak, ako je datoteka nastala kao fizički prikaz relacije, tada ona ima primarni ključ i on se poklapa s onim kojeg smo odabrali za relaciju.

U nastavku ovog odjeljka detaljnije opisujemo kako se zapisi koji čine datoteku pohranjuju u vanjskoj memoriji. Budući da se vanjska memorija sastoji od blokova, zapisi se moraju rasporediti po blokovima. S obzirom da je zapis obično znatno manji od bloka, više zapisa sprema se u jedan blok. Pritom uzimamo da je u jednom bloku smješten cijeli broj zapisa, što

znači da ni jedan zapis ne prelazi granicu između dva bloka, te da dio bloka možda ostaje neiskorišten. Ovakav način pohranjivanja omogućuje da jednoznačno odredimo položaj zapisa na disku. Naime, *adresa zapisa* gradi se kao uređeni par adrese bloka i pomaka u byte-ovima unutar bloka.



Slika 6.2: Datoteka sastavljena od blokova u kojima su zapisi.

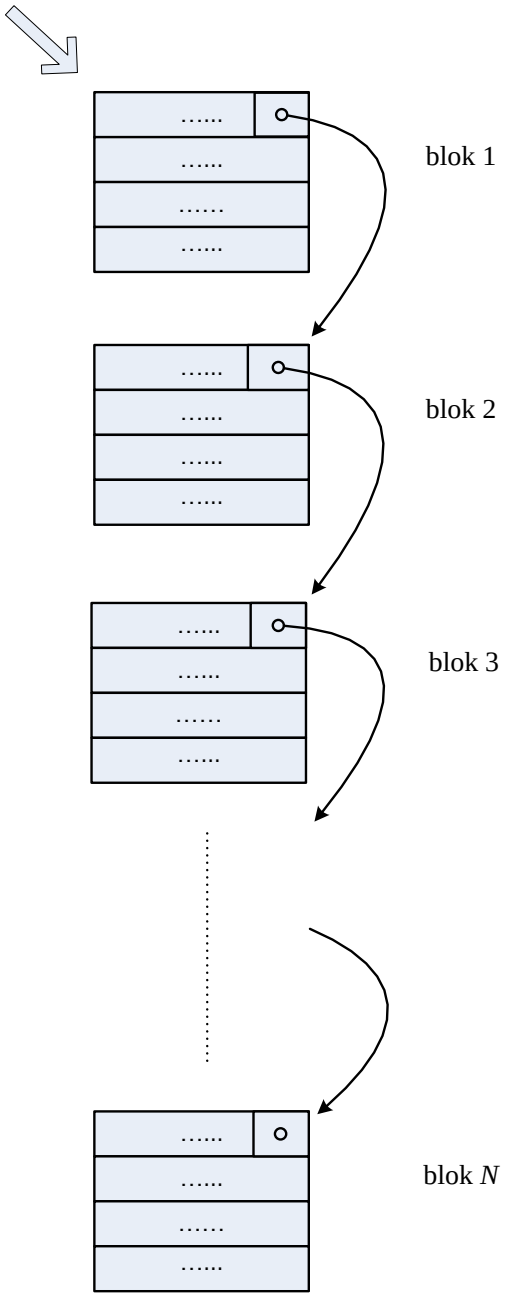
Budući da se datoteka obično sastoji od velikog broja zapisa, cijela datoteka obično zauzima više blokova, kao što je ilustrirano Slikom 6.2. Položaj i redoslijed blokova koji čine istu datoteku određen je posebnim pravilima koja čine takozvanu *organizaciju* datoteke. U svakom slučaju, ti blokovi se ne moraju nalaziti na uzastopnim adresama na disku. Neke od organizacija datoteka opisat ćemo u sljedećem odjeljku.

Na kraju ovog odjeljka spomenut ćemo još jedan važni element fizičke građe, a to je *pokazivač* (*pointer*). Riječ je o podatku unutar zapisa ili bloka jedne datoteke koji pokazuje na neki drugi zapis ili blok u istoj ili drugoj datoteci. Pokazivač se obično realizira tako da njegova vrijednost doslovno bude adresa zapisa ili bloka kojeg treba pokazati – to je takozvani *fizički* pokazivač. No mogući su i *logički* pokazivači koji pokazuju na implicitan način, na primjer navođenjem vrijednosti primarnog ključa zapisa kojeg treba pokazati. Pokazivači se obilato rabe u raznim organizacijama datoteka – oni omogućuju uspostavljanje veza između zapisa ili blokova, dakle povezivanje dijelova datoteke u cjelinu, te pristup iz jednog dijela te cjeline u drugi.

6.1.2. Organizacija datoteke

U prethodnom odjeljku vidjeli smo da se relacija iz relacijske baze fizički prikazuje kao datoteka u vanjskoj memoriji računala. Pritom su zapisi te datoteke raspoređeni u više blokova. Način međusobnog povezivanja blokova iz iste datoteke određen je posebnim pravilima koja čine organizaciju datoteke. U ovom odjeljku opisat ćemo nekoliko najvažnijih organizacija. Svaka od njih ima svoje prednosti i nedostatke u pogledu efikasnog obavljanja osnovnih operacija nad datotekom.

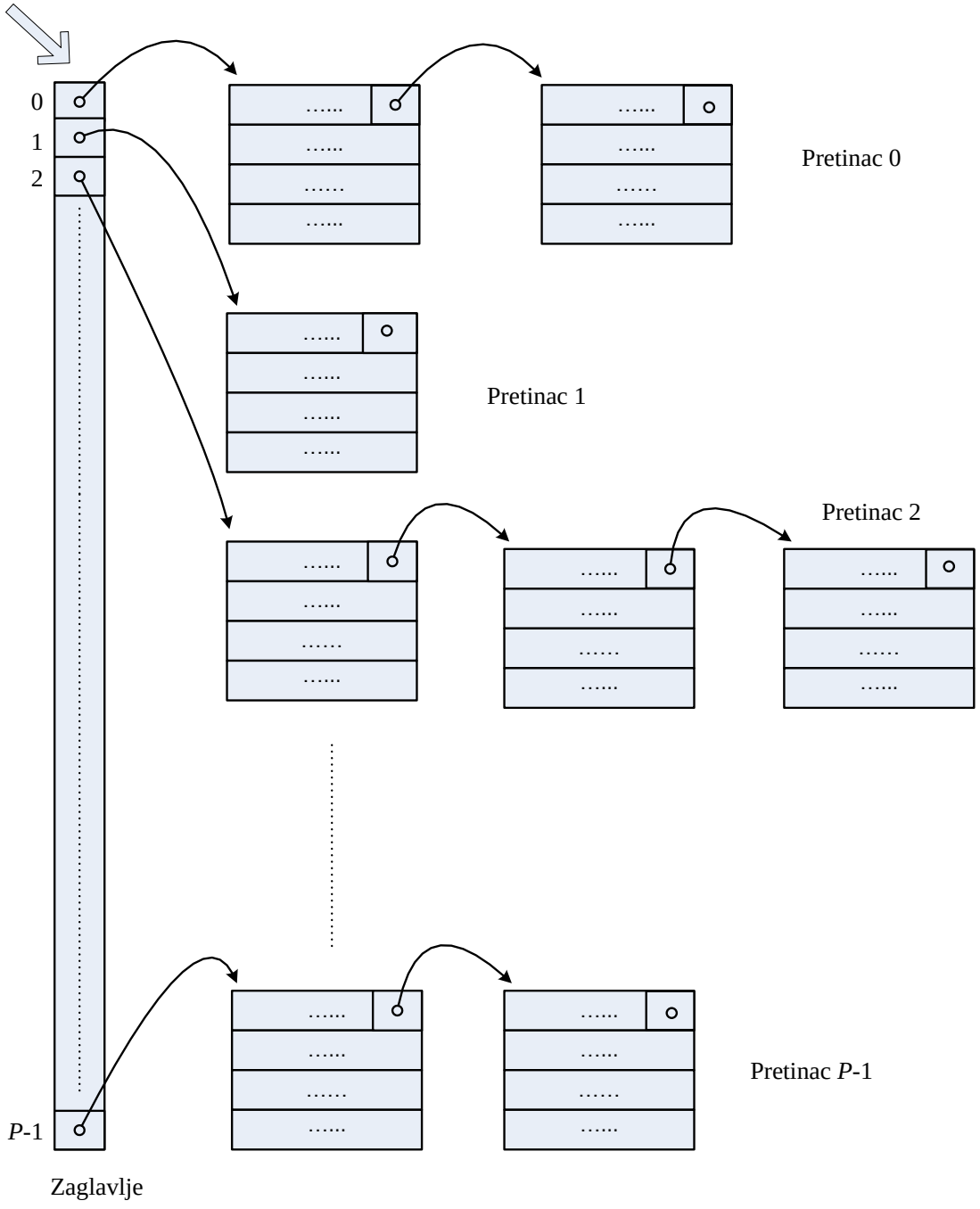
Jednostavna datoteka. Zapisi su poredani u onoliko blokova koliko je potrebno. Ti blokovi su međusobno povezani u vezanu listu, dakle svaki blok sadrži fizički pokazivač na idući blok. Kao adresu cijele datoteke pamtimo adresu prvog bloka. Građa je prikazana na Slici 6.3.



Slika 6.3: Jednostavna datoteka.

Prednost jednostavne organizacije je da se kod nje lagano ubacuju, izbacuju i mijenjaju zapisi. No nedostatak je da bilo kakvo traženje, na primjer traženje zapisa sa zadanom vrijednošću primarnog ključa, zahtijeva sekvencijalno čitanje blok po blok. To znači da će jedno traženje u prosjeku zahtijevati čitanje pola datoteke, pa vrijeme traženja linearno raste s veličinom datoteke.

Hash datoteka. Zapisi su raspoređeni u *P* cjelina, takozvanih *pretinaca*, (*buckets*) označenih rednim brojevima 0, 1, 2, … , *P*-1. Svaki pretinac građen je kao vezana lista blokova. Zadana je takozvana *hash funkcija h*( ) – ona daje redni broj *h*(*k*) pretinca u kojeg treba spremiti zapis s vrijednošću ključa *k*. Ista funkcija kasnije omogućuje i brzo pronalaženje zapisa sa zadanom vrijednošću ključa.



Slika 6:4: Hash datoteka.

Fizički pokazivači na početke pretinaca čine zaglavlje koje se smješta u prvi blok ili prvih nekoliko blokova datoteke. Adresu zaglavlja pamtimo kao adresu cijele datoteke. Zaglavlje je obično dovoljno malo, pa se za vrijeme rada s datotekom može držati u glavnoj memoriji. Cijela građa hash datoteke vidljiva je na Slici 6.4.

Skup mogućih vrijednosti ključa obično je znatno veći od broja pretinaca. Zato je važno da $h(\ )$ uniformno (jednoliko) distribuira vrijednosti ključa na pretince. Tada se naime neće dešavati da se pretinci neravnomjerno pune, pa će sve vezane liste biti podjednako kratke. Primjer dobre hash funkcije $h(\ )$ zasniva se na tome da se vrijednost ključa k shvati kao cijeli broj, te da $h(k)$ bude ostatak kod dijeljenja k s brojem pretinaca P.

Hash datoteka može se smatrati dijametralno suprotnom organizacijom od jednostavne. Naime, dok jednostavna organizacija dozvoljava jedino sekvencijalni pristup, prednost hash organizacije u tome što ona omogućuje gotovo izravni pristup na osnovu ključa. Da bismo pronašli zapis sa zadanom vrijednošću ključa k, najprije računamo $h(k)$, te zatim pretražimo samo $h(k)$-ti pretinac. Ako je $h()$ zaista uniformna, te ako je broj pretinaca P dobro odabran, tada ni jedan od pretinaca nije suviše velik. Pristup na osnovu ključa tada zahtijeva svega nekoliko čitanja blokova.

Nedostatak hash datoteke je da ona ne može sačuvati sortirani redoslijed po ključu za zapise. Naime, hash funkcija ima tendenciju „razbacivanja“ podataka na kvazi-slučajan način. Također, hash datoteka nije pogodna ako želimo pronaći zapise gdje je vrijednost ključa u nekom intervalu.

Daljnje dvije organizacije datoteke koje ćemo opisati zasnivaju se na tome da se uz osnovnu datoteku s podacima rabi i takozvani *indeks*. Preciznije: indeks je pomoćna datoteka koja olakšava traženje zapisa u osnovnoj datoteci.

Indeks koji omogućuje traženje po primarnom ključu naziva se *primarni* indeks. Zapisi u primarnom indeksu su parovi oblika (k, p), gdje je k vrijednost ključa, a p je fizički pokazivač na zapis u osnovnoj datoteci koji sadrži tu vrijednost ključa. Zbog svojstava primarnog ključa, za zadanu vrijednost k u indeksu može postojati najviše jedan par (k, p).

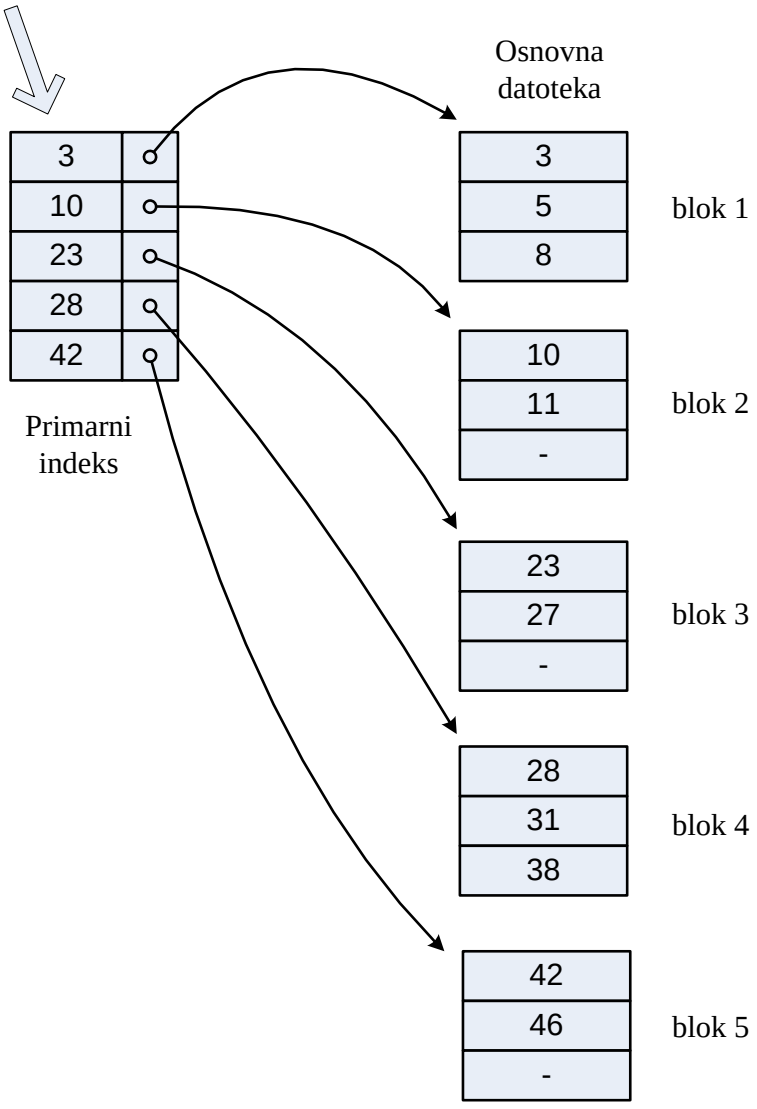
Indeks koji omogućuje traženje po podatku koji nije ključ naziva se *sekundarni* indeks. Zapisi u sekundarnom indeksu su parovi oblika (v, p), gdje je v vrijednost podatka, a p je fizički ili logički pokazivač na jedan od zapisa u osnovnoj datoteci koji sadrži tu vrijednost podatka. Budući da odabrani podatak nema svojstvo ključa, u indeksu može postojati više parova s istim v, dakle mogu postojati parovi (v, p_1), (v, p_2), (v, p_3), ….

Indeks-sekvencijalna datoteka je najpopularnija organizacija zasnovana na indeksu. Riječ je o osnovnoj datoteci koja je organizirana jednostavno i kojoj je zbog bržeg traženja po primarnom ključu dodan primarni indeks. Građa je prikazana Slikom 6.5. Brojevi na slici predstavljaju vrijednosti ključa. Ako je osnovna datoteka uzlazno sortirana po ključu, tada primarni indeks može biti *razrijeđen*, dakle on ne mora sadržavati pokazivače na sve zapise u osnovnoj datoteci, već je dovoljno da sadrži adrese blokova i najmanju vrijednost ključa za svaki blok. Kao adresu cijele datoteke pamtimo adresu indeksa.

Prednost indeks-sekvencijalne organizacije je da ona omogućuje prilično brz pristup na osnovu ključa, makar ipak malo sporiji nego hash datoteka. Zaista, da bismo pronašli zapis sa zadanom vrijednošću ključa k, čitamo indeks, saznajemo adresu odgovarajućeg bloka iz

osnovne datoteke, te izravno pristupamo tom bloku. Daljnja prednost indeks-sekvencijalne organizacije je da ona omogućuje čitanje svih zapisa osnovne datoteke sortirano po ključu – u tu svrhu slijedimo adrese blokova onim redom kako su one upisane u indeksu. Također, moguće je lako pronaći zapise gdje je vrijednost ključa u nekom intervalu. Dakle indeks-sekvencijalna datoteka predstavlja dobar kompromis između jednostavne i hash datoteke, pa je to razlog zašto je ona toliko popularna.

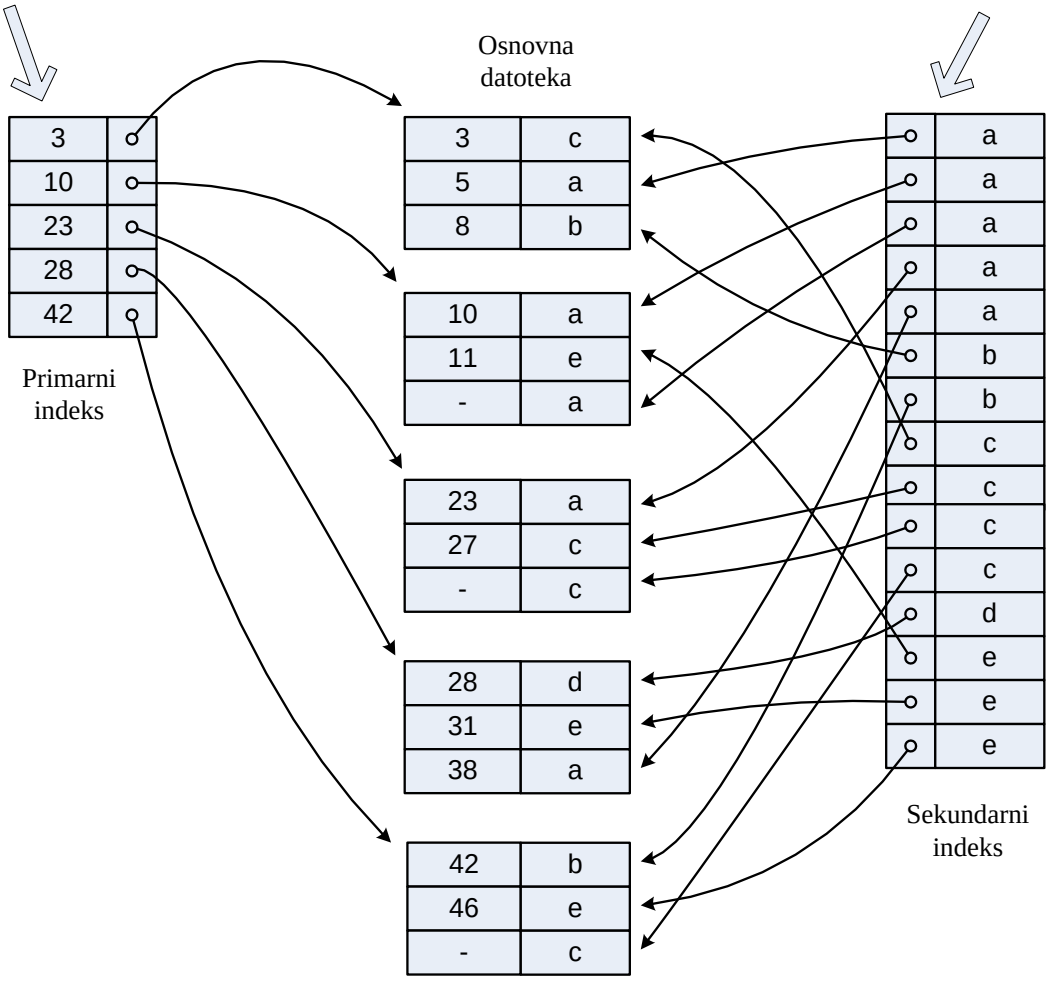
Nedostatak indeks-sekvencijalne organizacije je da se kod nje znatno kompliciraju operacije ubacivanja, mijenjanja ili izbacivanja podataka. Naime, svaka promjena u osnovnoj datoteci zahtijeva da se provede i odgovarajuća promjena u indeksu. Također, indeks je sam po sebi složena struktura koja troši dodatni prostor na disku.



Slika 6.5: Indeks-sekvencijalna datoteka.

Invertirana datoteka dobiva se daljnjom nadogradnjom indeks-sekvencijalne. I dalje imamo osnovnu datoteku i primarni indeks, no dodan je barem jedan sekundarni indeks koji omogućuje da se ista osnovna datoteka, osim po primarnom ključu, pretražuje i po nekom

drugom podatku. Sekundarni indeks je uvijek gust, dakle on sadrži pokazivač na svaki zapis iz osnovne datoteke. Ideja je ilustrirana Slikom 6.6. Brojevi na slici predstavljaju vrijednosti ključa, a slova vrijednosti drugog podatka po kojem pretražujemo. Kao adresu cijele datoteke pamtimo ili adresu primarnog ili adresu sekundarnog indeksa – dakle istim podacima može se pristupiti na dva različita načina.



Slika 6.6: Invertirana datoteka.

Prednost invertirane organizacije je brz pristup po više kriterija, te mogućnost sortiranog ispisa ili intervalnog pretraživanja po svim tim kriterijima. Nedostatak je da se ažuriranje podataka još više komplicira, te se troši još više dodatnog prostora na disku.

Hash datoteka s podijeljenom hash funkcijom. Riječ je poopćenju prije opisane hash datoteke. Cilj koji se želi postići je mogućnost traženja ne samo po primarnom ključu nego i po drugim podacima. Građa datoteke izgleda isto kao na Slici 6.4. No hash funkcija je definirana općenitije, tako da osim ključa uzima kao argumente i ne-ključne podatke.

Pretpostavimo da želimo pronaći zapise gdje istovremeno podatak A_1 ima vrijednost v_1, podatak A_2 ima vrijednost v_2, ..., podatak A_r ima vrijednost v_r. Pretpostavimo također da se redni broj pretinca u hash tablici može izraziti kao niz od B bitova – to znači da je broj pretinaca $P = 2^B$. Postupamo na sljedeći način.

- Podijelimo B bitova u skupine: b_1 bitova za podatak A_1, b_2 bitova za podatak A_2, ..., b_r bitova za podatak A_r.
- Zadamo pogodne „male“ hash funkcije $h_i(v_i)$, i=1,2, ..., r, gdje h_i preslikava vrijednost v_i podatka A_i u niz od b_i bitova.
- Redni broj pretinca za zapis u kojem je $A_1=v_1$, $A_2=v_2$, ..., $A_r=v_r$ zadaje se spajanjem malih nizova bitova. Dakle: $h(v_1,v_2, ..., v_r) = h_1(v_1) \mid h_2(v_2) \mid ... \mid h_r(v_r)$. Ovdje je | oznaka za „lijepljenje“ nizova bitova.

Doprinos jednog podatka u razdijeljenoj hash funkciji treba biti proporcionalan s veličinom domene tog podatka i s frekvencijom njegovog pojavljivanja u upitima. Dakle, veća domena ili češće pojavljivanje u upitima zahtijeva veći broj bitova.

Kao primjer, promatramo zapise o studentima fakulteta sa Slike 6.1. Želimo ih spremati u hash datoteku s 1024 ($=2^{10}$) pretinaca. Podijelimo 10 bitova u rednom broju pretinca na sljedeći način: 5 bitova za JMBAG, 3 bita za PREZIME, 2 bita za GODINU_STUDIJA. Zadajemo sljedeće hash funkcije, gdje su v_1, v_2, v_3 vrijednosti za JMBAG, PREZIME i GODINU_STUDIJA:

$h_1(v_1) = v_1$ % 32 (ostatak kod dijeljenja s 32) ,
$h_2(v_2)$ = (broj znakova u v_2 koji su različiti od bjeline) % 8,
$h_3(v_3) = v_3$ % 4 .

Tada je redni broj pretinca za zapis (1192130031, Horvat, Dragica, 2) zadan s $(01111|110|10)_2 = (506)_{10}$.

Da bismo pronašli zapis (ili više njih) u kojem je $A_1=v_1$, $A_2=v_2$, ..., $A_r=v_r$, računamo redni broj pretinca $h(v_1,v_2, ...,v_r)$ i sekvencijalno pretražimo taj jedan pretinac. Ako u upitu nije fiksirana vrijednost podatka A_i, tada b_i bitova u rednom broju pretinca ostaje nepoznato, a broj pretinaca koje moramo pretražiti povećava se 2^{bi} puta. Što manje vrijednosti za A_1, A_2, ... , A_r je poznato, to će biti veći broj pretinaca koje moramo pretražiti.

Na primjer, ako tražimo sve studente na drugoj godini, tada su nam u rednom broju pretinca poznata zadnja 2 bita: 10, no prvih 8 bitova je nepoznato. Treba pretražiti 2^8=256 pretinaca, dakle 1/4 datoteke. Ako tražimo sve studente s prezimenom Horvat, tada su nam u rednom broju pretinca poznata srednja tri bita: 110, no prvih 5 i zadnja 2 bita su nepoznata. Dakle sad treba pretražiti $2^{5+2}=2^7$=128 pretinaca, to jest 1/8 datoteke.

Prednost podijeljene hash funkcije u odnosu na invertiranu datoteku je da se ne troši dodatni prostor za indekse. Također, operacije ubacivanja, izbacivanja i promjene zapisa su znatno jednostavnije budući da nema indeksa koje treba održavati. No traženje zapisa u kojem su specificirane vrijednosti samo nekih od podataka traje dulje nego kod invertirane organizacije. Smatra se da je organizacija pomoću podijeljene hash funkcije dobra za datoteke koje nisu prevelike i čiji sadržaj se često mijenja.

6.1.3. Organizacija indeksa

U prethodnom odjeljku vidjeli smo da neke organizacije datoteke uključuju u sebi pomoćnu datoteku – indeks. Budući da je indeks sam za sebe također jedna datoteka, on isto mora biti organiziran na odgovarajući način. U ovom odjeljku opisat ćemo uobičajeni način fizičkog

prikazivanja indeksa pomoću takozvanog B-stabla. Riječ je o hijerarhijskoj strukturi podataka koja omogućuje da indeks zaista efikasno obavlja svoje osnovne zadaće, a to su brzo pronalaženje zadane vrijednosti podatka, te čuvanje sortiranog redoslijeda svih vrijednosti. Počinjemo s definicijom B-stabla.

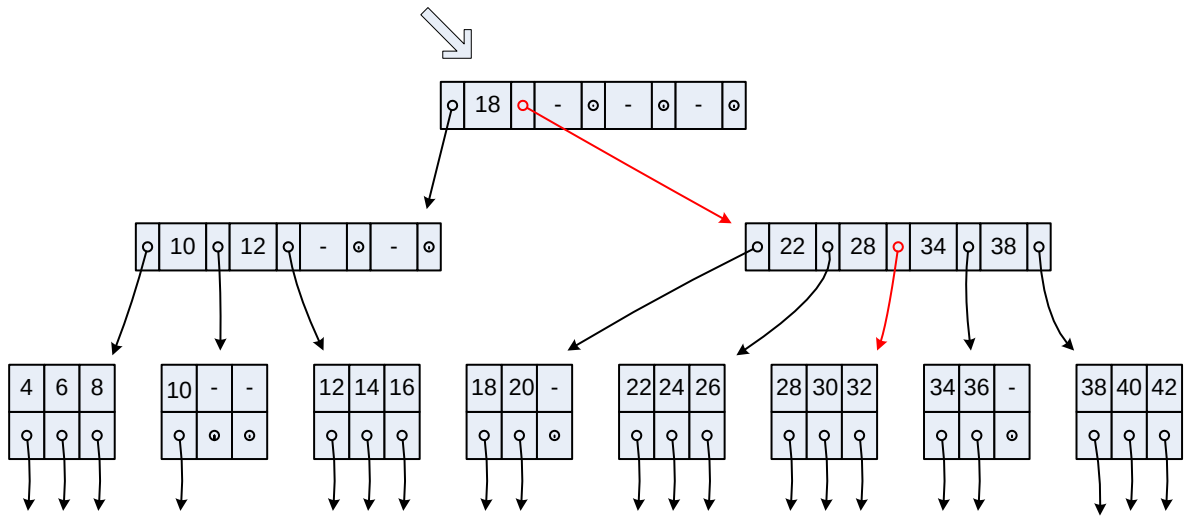
B-stablo reda m je m-narno stablo sa sljedećim svojstvima:

- korijen je ili list ili ima bar dvoje djece;
- svaki čvor, izuzev korijena i listova, ima između $\lceil m/2 \rceil$ i m djece;
- svi putovi od korijena do lista imaju istu duljinu.

Primijetimo da drugo svojstvo osigurava da je B-stablo „razgranato“ u širinu. Treće svojstvo osigurava „balansiranost“, dakle da su sve grane jednako visoke.

U nastavku detaljno opisujemo prikaz gustog primarnog indeksa pomoću B-stabla. Prikaz ostalih vrsta indeksa je vrlo sličan. Gusti primarni indeks prikazuje se kao B-stablo sagrađeno od blokova vanjske memorije, i to tako da jedan čvor bude jedan blok. Veza između roditelja i djeteta realizira se tako da u bloku-roditelju piše fizički pokazivač na blok-dijete. Također vrijedi sljedeće.

- Unutrašnji čvor ima sadržaj oblika (p_0, k_1, p_1, k_2, p_2, … , k_r, p_r), gdje je p_i pokazivač na i-to dijete dotičnog čvora ($0 \le i \le r$), k_i je vrijednost ključa ($1 \le i \le r$). Vrijednosti ključa unutar čvora su sortirane, dakle $k_1 \le k_2 \le \ldots \le k_r$. Sve vrijednosti ključa u pod-stablu koje pokazuje p_0 su manje od k_1. Za $1 \le i < r$, sve vrijednosti ključa u pod-stablu kojeg pokazuje p_i su u poluotvorenom intervalu $[k_i, k_{i+1})$. Sve vrijednosti ključa u pod-stablu kojeg pokazuje p_r su veće ili jednake k_r.
- List sadrži parove oblika (k, p), gdje je k vrijednost ključa, a p je fizički pokazivač na pripadni zapis u osnovnoj datoteci. Parovi unutar lista su uzlazno sortirani po k. List ne mora biti sasvim popunjen. Jednom zapisu osnovne datoteke odgovara točno jedan par (k, p) u listovima B-stabla.

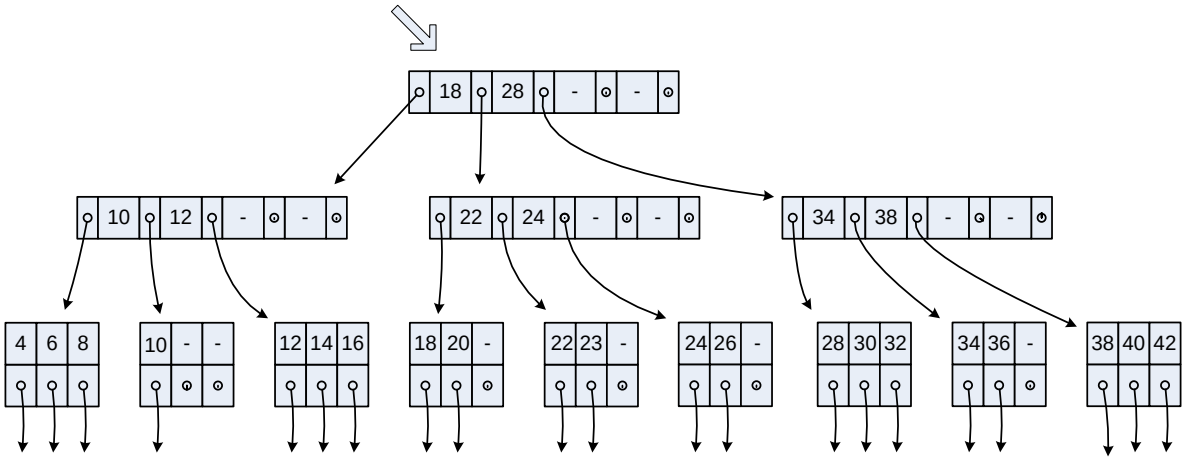


Slika 6.7: Gusti primarni indeks prikazan kao B-stablo reda 5.

Na Slici 6.7 vidimo gusti primarni indeks neke datoteke prikazan kao B-stablo reda 5. Primijetimo da se indeks u obliku B-stabla može shvatiti kao hijerarhija jednostavnijih indeksa.

U indeksu koji je prikazan kao B-stablo moguće je vrlo brzo za zadanu vrijednost ključa k pronaći pokazivač p na odgovarajući zapis u osnovnoj datoteci. U tu svrhu slijedimo put od korijena do lista koji bi morao sadržavati par (k, p). To se radi tako da redom čitamo unutrašnje čvorove oblika $(p_0, k_1, p_1, k_2, p_2, \ldots, k_r, p_r)$, te usporedimo k s $k_1, k_2, \ldots, k_r$. Ako je $k_i \leq k < k_{i+1}$, dalje čitamo čvor kojeg pokazuje p_i. Ako je $k < k_1$, dalje čitamo čvor s adresom p_0. Ako je $k \geq k_r$, koristimo se adresom p_r. Kad nas taj postupak konačno dovede u list, tražimo u njemu par sa zadanim k. Na primjer, ako u B-stablu sa Slike 6.7 tražimo vrijednost ključa 30, proći ćemo put koji je na slici označen crvenim strelicama.

Efikasnost prikaza indeksa pomoći B-stabla počiva na činjenici da u realnim situacijama B-stablo nikad nema preveliku visinu, to jest sastoji se od svega 3-4 razine. Naime, zbog odnosa veličine bloka, duljine ključa i duljine adrese, red B-stabla m može biti prilično velik, pa B-stablo postaje „široko i nisko“. Mali broj razina znači mali broj čitanja blokova s diska pri traženju.



Slika 6.8: Ubacivanje vrijednosti 23 u B-stablo s prethodne slike.

Za razliku od traženja koje se odvija brzo i efikasno, ubacivanje podataka u B-stablo je kompliciranа operacija koja često zahtijeva da se neki od čvorova rascijepi na dva, te da se nakon toga izvrši promjena i u nadređenom čvoru. Lančana reakcija promjena može doći sve do korijena, koji se također može rascijepiti čime se visina stabla povećava za 1. Slika 6.8 prikazuje B-stablo s prethodne Slike 6.7 nakon što je u njega ubačena nova vrijednost ključa 23. Izbacivanje podatka iz B-stabla odvija se analogno kao ubacivanje, samo u obrnutom smjeru. Prilikom izbacivanja može doći do sažimanja čvorova, te do smanjenja visine stabla.

6.2. Pretvorba relacijske sheme u fizičku shemu i njezina implementacija

Rekli smo da je fizička shema baze tekst sastavljen od naredbi u SQL-u ili nekom drugom jeziku. Izvođenjem tih naredbi DBMS stvara fizičku građu baze. U tom smislu, fizička shema može se shvatiti opisom fizičke građe. Ipak, taj opis je samo implicitan budući da iz njega ne možemo doslovno pročitati kako će datoteke biti organizirane. Projektant tu ima prilično ograničen utjecaj, a većinu detalja automatski određuje DBMS pomoću svojih ugrađenih pravila.

6.2.1. Stvaranje početne verzije fizičke sheme

Najvažnija SQL naredba koja se pojavljuje u fizičkoj shemi baze je naredba CREATE TABLE. Njome se definira jedna relacija iz baze, dakle ime relacije, te imena i tipovi atributa. Također je moguće zadati koji atribut ili kombinacija atributa čine primarni ključ te relacije, te smije li neki atribut imati neupisane vrijednosti ili ne smije.

Početnu verziju fizičke sheme dobivamo tako da svaku relaciju iz prethodno razvijene relacijske sheme opišemo jednom naredbom CREATE TABLE. Pritom tipove atributa odredimo najbolje što možemo u skladu s pripadnim rječnikom podataka. Kod određivanja tipova obično moramo napraviti neke kompromise budući da je popis tipova koje podržava DBMS ograničen.

```
CREATE TABLE STUDENT
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   GODINA_STUDIJA ENUM('1','2','3','4','5'),
   PRIMARY KEY(JMBAG));

CREATE TABLE PREDMET
  (SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   NASLOV CHAR(80),
   IME_ZAVODA CHAR(40),
   OIB_NASTAVNIKA NUMERIC(11) UNSIGNED,
   SEMESTAR ENUM('Z','L'),
   ECTS_BODOVI NUMERIC(2) UNSIGNED,
   PRIMARY KEY(SIFRA_PREDMETA));

CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB));

CREATE TABLE ZAVOD
  (IME_ZAVODA CHAR(40) NOT NULL,
   OIB_PROCELNIKA NUMERIC(11) UNSIGNED,
   OPIS_DJELATNOSTI CHAR(160),
   PRIMARY KEY(IME_ZAVODA));

CREATE TABLE UPISAO
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   DATUM_UPISA DATE,
   OCJENA ENUM('2','3','4','5'),
   PRIMARY KEY(JMBAG,SIFRA_PREDMETA));
```

Slika 6.9: Početna verzija fizičke sheme za bazu podataka o fakultetu.

Slika 6.9 prikazuje početnu fizičku shemu za bazu podataka o fakultetu, koja je dobivena na osnovu relacijske sheme sa Slike 3.5 i rječnika podataka sa Slike 3.6. Rabi se MySQL-ova inačica SQL-a. Pojavljuje se pet naredbi CREATE TABLE od kojih svaka odgovara jednoj relaciji sa Slike 3.5. Definirani su primarni ključevi. Tipovi atributa neznatno se razlikuju od onih sa Slike 3.6, na primjer tekstualni podaci imaju ograničenje duljine.

Izvođenjem naredbi sa Slike 6.9, MySQL će automatski stvoriti fizičku bazu gdje će svaka od pet relacija biti prikazana kao jedna datoteka. Rabit će se jedna vrsta indeks-sekvencijalne organizacije koja se u MySQL-ovoj terminologiji zove MyISAM: dakle zapisi datoteke bit će raspoređeni u blokove, te će u datoteku automatski biti ugrađen primarni indeks koji osigurava svojstvo primarnog ključa i ubrzava pretraživanje po ključu.

Naredbe CREATE TABLE sa Slike 6.9 mogle su se napisati i bez navođenja primarnog ključa. U tom slučaju MySQL bi se za prikaz relacije mogao koristiti i jednostavnom datotekom. Dakle ne bi bilo ugrađenog indeksa, ne bi se garantirala jedinstvenost vrijednosti ključa, a traženje po ključu bi zahtijevalo sekvencijalno čitanje cijele datoteke.

Neki DBMS-i, na primjer Oracle, omogućuju i prikaz relacije u obliku hash tablice. Tada u odgovarajućoj naredbi CREATE TABLE moramo navesti posebnu opciju. Odabirom hash tablice dodatno se ubrzava traženje po ključu, no usporavaju se sve operacije koje ovise o sortiranom redoslijedu po ključu. Jedinstvenost vrijednosti ključa u hash tablici DBMS može garantirati provjerom sadržaja pretinca prilikom svakog upisa podataka.

6.2.2. Daljnje dotjerivanje fizičke sheme

Zbog navođenja primarnih ključeva i uporabe primarnih indeksa, fizička shema sa Slike 6.9 osigurava da će se u svim datotekama traženje po primarnom ključu odvijati relativno brzo. No traženje po drugim podacima bit će sporo jer će zahtijevati sekvencijalno čitanje odgovarajućih datoteka. Pretraživanje po odabranim podacima koji nisu ključevi možemo ipak ubrzati ako DBMS-u naredimo da sagradi odgovarajuće sekundarne indekse. U tu svrhu rabe se SQL naredbe CREATE INDEX.

Slika 6.10 prikazuje dodatak shemi sa Slike 6.9 kojim se uvodi indeks za atribut PREZIME u relaciji STUDENT, te indeksi za atribute JMBAG odnosno SIFRA_PREDMETA u relaciji UPISAO. Stvaranjem tih sekundarnih indeksa, indeks-sekvencijalna datoteka za prikaz relacije STUDENT odnosno UPISAO nadograđuje se do invertirane organizacije. Time postaje moguće brzo pronalaženje studenta po prezimenu ili brzi ispis svih studenata sortirano po prezimenu. Također, efikasno se pronalaze svi predmeti koje je upisao zadani student, te svi studenti koji su upisali zadani predmet.

```
CREATE INDEX SP_IND ON STUDENT (PREZIME);
CREATE INDEX UJ_IND ON UPISAO (JMBAG);
CREATE INDEX US_IND ON UPISAO (SIFRA_PREDMETA);
```

Slika 6.10: Sekundarni indeksi za bazu podataka o fakultetu.

Uvođenjem sekundarnih indeksa poboljšavaju se performanse baze prilikom pretraživanja. No treba biti svjestan da svaki sekundarni indeks predstavlja dodatni teret za DBMS budući da on zauzima prostor na disku i mora se ažurirati. Zato projektant baze ne smije pretjerivati sa stvaranjem indeksa, već treba procijeniti koje su stvarne potrebe aplikacija. Indeks je zaista potreban samo za one podatke po kojima se vrlo često pretražuje, ili ako se zahtijeva izuzetno velika brzina odziva.

6.2.3. Stvaranje i inicijalno punjenje baze

Kao što smo već rekli, fizička baza podataka nastaje izvođenjem naredbi iz fizičke sheme. Na primjer, izvođenjem svih naredbi sa Slike 6.9 i 6.10 stvaraju se sve datoteke i indeksi koji čine fizičku bazu podataka o fakultetu. Detalji postupka donekle se razlikuju ovisno o DBMS-u, a u slučaju MySQL-a oni izgledaju ovako.

```
> CREATE DATABASE fakultet;

> USE fakultet;

> SOURCE CreateTables.txt;
```

Prethodni redci zapravo su naredbe koje se izvode na interaktivan način unutar komandne ljuske mysql. Pretpostavljeno je da je cjelokupna fizička shema, dakle ukupni sadržaj Slike 6.9 i 6.10, pohranjena kao jedna tekstualna datoteka s imenom CreateTables.txt. Prvi redak stvara praznu bazu fakultet, dakle bazu u kojom nema ni jedne relacije. Drugi redak otvara tu bazu. Treći redak pokreće CreateTables.txt kao komandnu datoteku (*script file*), čime će se izvesti sve SQL naredbe CREATE TABLE i CREATE INDEX sa Slike 6.9. i 6.10.

Nakon upravo opisanog postupka, fakultetska baza imat će sve potrebne relacije, no te relacije će biti prazne, to jest u njima neće biti n-torki. Da bi s takvom bazom mogli išta smisleno raditi, moramo je inicijalno napuniti s podacima.

Inicijalno punjenje u principu je moguće obaviti standardnim SQL naredbama za ažuriranje. Naime, u SQL-u postoje naredbe nalik na SELECT koje služe za ubacivanje n-torke u relaciju, odnosno promjenu ili brisanje jedne ili više n-torki. U nastavku navodimo nekoliko primjera takvih naredbi: prve dvije ubacuju novog studenta odnosno novog nastavnika, treća mijenja nastavnika za predmet Baze podataka, a četvrta briše sve upise predmeta za studenta s JMBAG-om 1191203289.

```
INSERT INTO STUDENT VALUES (0036398757, 'Markovic', 'Marko',   '1');

INSERT INTO NASTAVNIK
  VALUES (13257600947, 'Cantor', 'Georg',  'Zavod za matematiku',  102, 12000);

UPDATE PREDMET
  SET OIB_NASTAVNIKA = 67741205512  WHERE  NASLOV = 'Baze podataka';

DELETE FROM UPISAO WHERE  JMBAG = 1191203289;
```

Osim standardnih SQL naredbi, u većini DBMS-a stoje nam na raspolaganju i dodatni mehanizmi kojima se inicijalno punjenje baze može obaviti na efikasniji način. MySQL nam omogućuje da inicijalni sadržaj jedne relacije pripremimo u obliku tekstualne datoteke, te da onda jednom naredbom taj sadržaj pretočimo u bazu.

Na primjer, pretpostavimo da tekstualna datoteka StudentData.txt sadrži sljedeće podatke za 7 studenata. Svaki redak datoteke odgovara jednoj n-torki iz relacije STUDENT, vrijednosti atributa odvojene su znakovima tab i poredane su onako kako je određeno u odgovarajućoj naredbi CREATE TABLE.

```
0036398757  Markovic    Marko      1
1191203289  Petrovic    Petar      2
1192130031  Horvat      Dragica    2
1191205897  Jankovic    Marija     1
0165043021  Kolar       Ivan       3
0036448430  Grubisic    Katica     3
0246022858  Vukovic     Janko      1
```

Sljedeće naredbe izvode iz komandne ljuske mysql – njima se sadržaj iz StudentData.txt pretvara u 7 n-torki relacije STUDENT.

```
> USE fakultet;

> LOAD DATA INFILE “StudentData.txt“ INTO TABLE STUDENT;
```

Nakon inicijalnog punjenja, baza je spremna za rad. Na primjer, u slučaju naše fakultetske baze podataka bilo bi moguće izvesti upite iz Potpoglavlja 5.3. Također, pomoću raznih alata i programskih jezika mogli bismo dalje razvijati aplikacije koje bi služile za daljnje ažuriranje podataka, izvještavanje, računanje statistika i slično.

6.3. Izvrednjavanje i optimizacija upita

Kad smo u prethodnim potpoglavljima govorili o fizičkoj građi i implementaciji baze podataka, podrazumijevali smo da DBMS zna obavljati jednostavne radnje s relacijama (datotekama), kao što su unos nove n-torke (zapisa), promjenu ili brisanje n-torke, te pronalaženje n-torke sa zadanom vrijednošću ključa ili nekog drugog podatka. No od implementacije baze očekuje se i više: ona također treba efikasno odgovarati na složene upite gdje se povezuju ili grupiraju podaci iz raznih relacija. Dakle, uz algoritme za osnovne operacije s datotekama, DBMS također treba imati na raspolaganju i algoritme koji omogućuju interpretaciju i izvrednjavanje složenih upita.

U ovom potpoglavlju nastojat ćemo prikazati način na koji DBMS odgovara na upite. Prvi odjeljak prikazuje opći tijek tog postupka. Daljnji odjeljci opisuju neke detalje unutar postupka, kao što su algoritmi za izvrednjavanje algebarskih operacija, te pravila za optimizaciju upita.

6.3.1. Opći tijek izvrednjavanja i optimizacije

Upit se obično postavlja u neproceduralnom jeziku kao što je SQL. Pronalaženje odgovora na postavljeni upit odvija se četiri faze.

1. Prevođenje upita u relacijsku algebru.
2. Viša razina optimizacije: algebarska transformacija.
3. Niža razina optimizacije: odabir algoritma za izvrednjavanje pojedine algebarske operacije.
4. Izvrednjavanje pojedine algebarske operacije odabranim algoritmom.

Faza prevođenja upita iz SQL-a u relacijsku algebru potrebna je zato jer relacijska algebra, za razliku od SQL-a, prikazuje upit u proceduralnom obliku koji omogućuje izvrednjavanje. Nakon prevođenja, odgovor na upit moguće je dobiti izvršavanjem operacija koje se pojavljuju u algebarskom izrazu u redoslijedu koji je određen samim izrazom. Sam algoritam prevođenja iz SQL-a u relacijsku algebru nećemo opisivati jer je riječ o naprednijem gradivu koje izlazi iz okvira ovog teksta. Algoritam je utemeljen na prije spominjanoj ekvivalenciji SQL-a i relacijske algebre u smislu izražajnosti.

U fazi optimizacije na višoj razini, polazni algebarski izraz koji je bio dobiven prevođenjem iz SQL-a nastoji se transformirati u ekvivalentni izraz koji je pogodniji za izvrednjavanje. Ovdje pod ekvivalentnim izrazima smatramo one koji imaju jednaku vrijednost. Izraz je pogodniji za izvrednjavanje ako se on može izračunati u kraćem vremenu ili uz manji utrošak memorije. Transformacija se obavlja primjenom posebnih pravila o ekvivalenciji algebarskih izraza.

Zadnje dvije faze u pronalaženju odgovora na upit ponavljaju se više puta, dakle jednom za svaku operaciju u transformiranom algebarskom izrazu, poštujući redoslijed koji je zapisan u samom izrazu. Sjetimo se da se u algebarskim izrazima pojavljuju unarne ili binarne operacije koje od relacija rade relacije. S obzirom da su relacije fizički prikazane datotekama, algoritmi za izvrednjavanje algebarskih operacija zapravo su algoritmi koji čitaju jednu ili dvije ulazne datoteke te ispisuju izlaznu datoteku. Složenost takvih algoritama mjeri se količinom podataka koje treba pročitati ili ispisati.

Faza optimizacije na nižoj razini uvodi se zato jer za jednu algebarsku operaciju obično imamo na raspolaganju više algoritama koji je izvrednjavaju. Svaki od tih algoritama ima svoje prednosti i mane, te se u određenim okolnostima može pokazati bržim ili sporijim od ostalih. Optimizacija na nižoj razini trebala bi osigurati da se od raspoloživih algoritama bira onaj koji je u danoj situaciji najbrži. Odabir se zasniva na heurističkim pravilima koja procjenjuju brzinu algoritma na osnovu raznih parametara kao što su veličine ulaznih datoteka, način na koje su one sortirane, postojanje ili nepostojanje indeksa, i tako dalje.

Na osnovi svega rečenog u ovom odjeljku, vidimo da se postupak odgovaranja na upite u najvećoj mjeri svodi na izvrednjavanje operacija iz relacijske algebre. Sljedeća dva odjeljka zato detaljnije opisuju konkretne algoritme za izvrednjavanje algebarskih operacija. Pritom nam je u središtu pažnje izvrednjavanje prirodnog spoja, budući da se kod te operacije najbolje vide neke važne ideje koje su primjenjive i na druge operacije.

U ovom odjeljku također smo vidjeli da se postupak odgovaranja na upite nastoji ubrzati primjenom dvaju vidova optimizacije. Optimizacija upita je složeno područje koje također izlazi iz okvira ovog teksta. Ipak, u zadnjem odjeljku navodimo kao ilustraciju nekoliko primjera pravila za optimizaciju.

6.3.2. Izvrednjavanje prirodnog spoja

Promatramo relacije $R_1(A,B)$ i $R_2(B,C)$ sa zajedničkim atributom B. Označimo sa $S(A,B,C)$ prirodni spoj od R_1 i R_2. Svaka od ovih triju relacija fizički se prikazuje jednom (istoimenom) datotekom: n-torke se pretvaraju u zapise, a atributi u istoimene osnovne podatke. Razmotrit ćemo nekoliko načina kako se pomoću datoteka R_1 i R_2 može generirati datoteka S.

Algoritam ugniježđenih petlji. To je očigledan, makar ne nužno i najefikasniji način. Doslovno se primjenjuje definicija prirodnog spoja. Algoritam je opisan sljedećim pseudo-kodom.

```
inicijaliziraj praznu S;
učitaj prvi zapis iz R1;
dok ( nismo prešli kraj od R1 ) {
        učitaj prvi zapis iz R2;
        dok ( nismo prešli kraj od R2) {
                ako ( tekući zapisi iz R1 i R2 sadrže istu vrijednost za B )
                        stvori kombinirani zapis i ispiši ga u S;
                pokušaj učitati idući zapis iz R2;
        }
        pokušaj učitati idući zapis iz R1;
}
```

Prema ovom pseudo-kodu, datoteku R_1 čitamo samo jednom, no za svaki zapis iz R_1 iznova moramo čitati cijelu datoteku R_2. Algoritam se može poboljšati tako da u glavnu memoriju učitamo segment od što više blokova iz R_1. Preinačimo petlje tako da uspoređujemo svaki zapis učitan iz R_2 sa svakim zapisom iz R_1 koji je trenutno u glavnoj memoriji. Nakon što smo pročitali cijelu R_2 i obavili sva potrebna uspoređivanja, učitamo idući segment od R_1 u glavnu memoriju te ponavljamo postupak. Nakon ovog poboljšanja R_1 se očigledno čita samo jednom, a R_2 se čita onoliko puta koliko ima segmenata u R_1. Poboljšana verzija algoritma osobito je dobra onda kad je jedna od datoteka dovoljno mala da cijela stane u glavnu memoriju. Tada se i R_1 i R_2 čitaju samo jednom.

Algoritam zasnovan na sortiranju i sažimanju. Pretpostavimo da su datoteke R_1 i R_2 uzlazno sortirane po zajedničkom podatku B. Tada u R_1 i u R_2 možemo uočiti skupine uzastopnih zapisa s istom vrijednošću za B. Datoteka S koja sadrži prirodni spoj od R_1 i R_2 lagano se može generirati sljedećim algoritmom koji podsjeća na klasično sažimanje.

```
inicijaliziraj praznu S;
učitaj prvu skupinu zapisa iz R1;
učitaj prvu skupinu zapisa iz R2;
dok ( nismo prešli kraj ni od R1 ni od R2) {
        ako ( tekuća skupina zapisa iz R1 sadrži manju
        vrijednost za B nego tekuća skupina zapisa iz R2 )
                pokušaj učitati iduću skupinu zapisa iz R1;
        inače ako ( tekuća skupina zapisa iz R2 sadrži
        manju vrijednost za B nego tekuća skupina zapisa iz R1 )
                pokušaj učitati iduću skupinu zapisa iz R2;
```

```
        inače {
                svaki zapis iz tekuće skupine iz R1 kombiniraj sa svakim zapisom
                iz tekuće skupine iz R2 te sve generirane zapise ispiši u S;
                pokušaj učitati iduću skupinu zapisa iz R1;
                pokušaj učitati iduću skupinu zapisa iz R2;
        }
}
```

Pretpostavili smo da su skupine zapisa iz R_1 odnosno R_2 s istom vrijednošću za B dovoljno male tako da stanu u glavnu memoriju. Pod ovakvim pretpostavkama i R_1 i R_2 se čitaju samo jednom. Ukoliko skupine ne stanu u glavnu memoriju, algoritam treba preraditi tako da učitava segment po segment od svake skupine. Segmenti jedne datoteke tada će se morati više puta učitavati.

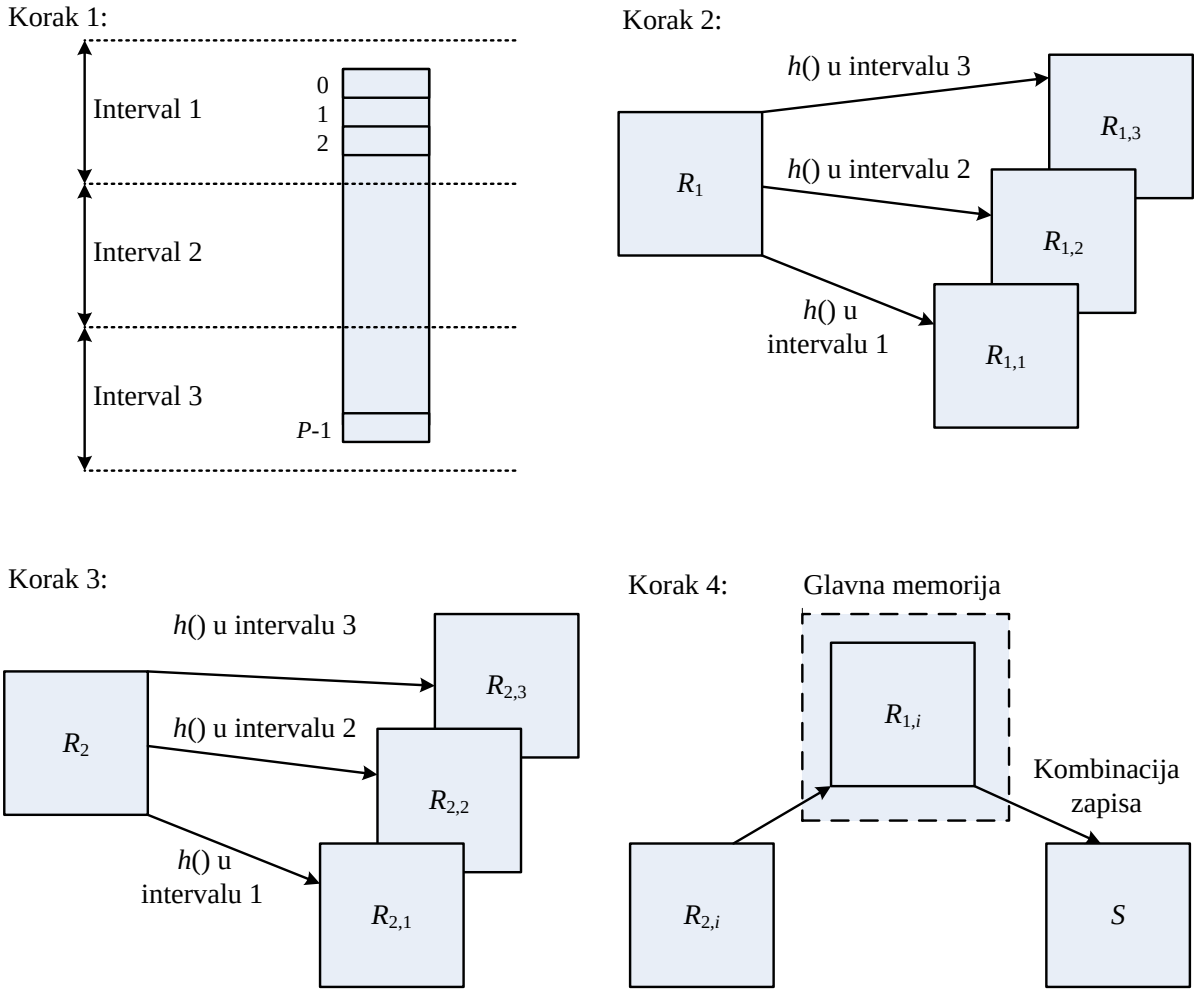
Ako R_1 i R_2 nisu sortirane kao što se tražilo u opisanom algoritmu, tada ih najprije treba sortirati, pa tek onda računati prirodni spoj. U literaturi postoje mnogi dobri algoritmi za sortiranje, no oni obično predviđaju da cijela datoteka stane u glavnu memoriju. Da bismo sortirali veću datoteku, dijelimo je na segmente koji stanu u glavnu memoriju, posebno sortiramo svaki segment i ispisujemo ga natrag u vanjsku memoriju. Dalje se sortirani segmenti postepeno sažimaju u sve veće i veće, sve dok na kraju ne dobijemo cijelu sortiranu datoteku. Riječ je o prilično dugotrajnom postupku koji zahtijeva višestruko prepisivanje cijele datoteke. Dakle sortiranje polaznih R_1 i R_2 trajat će znatno dulje nego generiranje S od već sortiranih R_1 i R_2. Ipak, ako su R_1 i R_2 jako velike, cijeli postupak se isplati u odnosu na algoritam ugniježđenih petlji.

Algoritam zasnovan na indeksu. Pretpostavimo da jedna od datoteka R_1 i R_2, na primjer R_2, ima sekundarni indeks za zajednički podatak B. Tada se datoteka S, koja sadrži prirodni spoj od R_1 i R_2, može generirati na sljedeći način.

```
inicijaliziraj praznu S;
učitaj prvi zapis iz R1;
dok ( nismo prešli kraj od R1 ) {
        pomoću indeksa pronađi i učitaj sve zapise iz R2
        koji imaju istu vrijednost za B kao tekući zapis iz R1;
        tekući zapis iz R1 kombiniraj sa svakim od učitanih
        zapisa iz R2 te generirane zapise ispiši u S;
        pokušaj učitati idući zapis iz R1;
}
```

Algoritam jednom pročita cijelu R_1. No iz R_2 se neposredno čitaju samo oni zapisi koji sudjeluju u prirodnom spoju. To može dovesti do značajne uštede u obimu posla. Ako i R_1 i R_2 imaju indeks za B, tada treba sekvencijalno čitati manju datoteku, a rabiti indeks veće datoteke.

Algoritam zasnovan na hash funkciji i razvrstavanju. Zadajemo hash funkciju h koja ovisi o zajedničkom podatku B. Kombinacija zadanog zapisa iz datoteke R_1 sa zadanim zapisom iz datoteke R_2 pojavljuje se u prirodnom spoju S ako i samo ako oba zadana zapisa imaju jednaku vrijednost za B. Zato hash funkcija za oba takva zapisa daje istu vrijednost. Razvrstavanjem zapisa iz R_1 i R_2 u skupine onih s bliskom vrijednošću h lakše ćemo odrediti koji parovi zapisa se mogu kombinirati.



Slika 6.11: Izvrednjavanje prirodnog spoja pomoću hash funkcije i razvrstavanja.

Neka je R_1 manja od R_2. Algoritam se tada sastoji od sljedećih pet koraka. Pritom su koraci 1, 2, 3 i 4 ilustrirani na Slici 6.11.

1. Inicijaliziraj praznu datoteku *S*. Odaberi hash funkciju *h*. Podijeli ukupni raspon hash vrijednosti na *k* podjednakih intervala. Pritom je *k* odabran tako da $1/k$ od datoteke R_1 stane u glavnu memoriju.
2. Čitaj sekvencijalno R_1 i razvrstaj njezine zapise u *k* skupina (pomoćnih datoteka) tako da jedna skupina sadrži sve zapise iz R_1 koje *h* preslikava u jedan od intervala. Ako je *h* zaista uniformna hash funkcija, tada su sve skupine podjednako velike - znači da jedna skupina stane u glavnu memoriju.
3. Čitaj sekvencijalno R_2 i razvrstaj njezine zapise u *k* skupina (pomoćnih datoteka), slično kao što smo to napravili s datotekom R_1.
4. Odaberi jedan od intervala za vrijednost *h*. Učitaj u glavnu memoriju odgovarajuću skupinu zapisa iz R_1. Sekvencijalno čitaj odgovarajuću skupinu zapisa iz R_2. Kombiniraj tekući zapis iz R_2 sa svim zapisima iz R_1 (u glavnoj memoriji) koji imaju jednaku vrijednost za *B*. Dobivene kombinacije ispiši u *S*.
5. Ponovi korak 4, s time da odabereš novi interval za vrijednost od *h*. Ako smo u koraku 4 već obradili svaki od *k* intervala, tada je algoritam završen.

Analizom algoritma lagano se vidi da se svaka od datoteka R_1 i R_2 čita točno dvaput.

6.3.3. Izvrednjavanje selekcije, projekcije i ostalih operacija

Osim prirodnog spoja, najvažnije relacijske operacije su selekcija i projekcija. Izložit ćemo osnovne ideje za izvrednjavanje tih dviju operacija, a zatim ćemo kratko napomenuti kako se implementiraju ostale operacije.

Izvrednjavanje selekcije. Zadana je relacija *R* i Booleovski uvjet $\mathcal{B}$. *R* je fizički prikazana istoimenom datotekom, na standardni način. Izvrednjavanje selekcije *R* where $\mathcal{B}$ ovisi o obliku uvjeta $\mathcal{B}$, no obično se svodi na traženje zapisa u datoteci *R* sa zadanom vrijednošću nekih podataka. Dakle, obično se radi o pristupu na osnovi primarnog ključa ili o pristupu na osnovi ostalih podataka. U Potpoglavlju 6.1 već smo opisali algoritme za pristup. Sad ćemo samo ukratko rezimirati ideje.

- Naivni algoritam za selekciju bio bi: sekvencijalno čitanje cijele *R* i provjera svakog zapisa zadovoljava li on $\mathcal{B}$. Ako je *R* velika, to traje predugo, pa treba izgraditi pomoćne strukture podataka koje omogućuju izravniji pristup do traženih zapisa.
- Invertiranje datoteka pomoću indeksa je vjerojatno najfleksibilnija metoda ubrzavanja selekcije. Većina današnjih relacijskih DBMS-a oslanja se na invertirane datoteke, dakle na jednostavno organizirane datoteke nadopunjene sekundarnim indeksima koji su prikazani B-stablima (primarni gusti indeks je samo specijalni slučaj sekundarnog). Drugi no znatno rjeđe rabljeni način ubrzavanja selekcije je primjena hash organizacije.
- Neki oblici uvjeta $\mathcal{B}$ zahtijevaju da u *R* tražimo zapise gdje vrijednost zadanog podatka nije fiksirana, već se kreće u zadanom intervalu. Indeks-B-stablo podržava ovakvo intervalno pretraživanje, budući da je u njemu sačuvan sortirani redoslijed zapisa po dotičnom podatku. Hash organizacija ne podržava intervalno pretraživanje.

Izvrednjavanje projekcije. Zadana je relacija *R* i njezin atribut *A*. *R* je fizički prikazana istoimenom datotekom na standardni način. Da bismo generirali datoteku koja odgovara projekciji *S* = *R*[*A*], očito treba pročitati cijelu datoteku *R* i izdvojiti sve vrijednosti podatka *A* koje se pojavljuju. No ista vrijednost za *A* može se pojaviti više puta. Osnovni problem implementiranja projekcije je: kako u *S* eliminirati zapise-duplikate?

- Najjednostavniji algoritam za računanje projekcije zasnovan je na ugniježđenim petljama. Vanjska petlja čita datoteku *R*, a unutrašnja petlja prolazi trenutno stvorenim dijelom datoteke *S*. Algoritam je opisan sljedećim pseudo-kodom.

```
inicijaliziraj praznu S;
učitaj prvi zapis iz R;
dok ( nismo prešli kraj od R ) {
    duplikat = 0;
    pokušaj učitati prvi zapis iz S;
    dok ( nismo prešli kraj od S i duplikat==0 ) {
            ako ( tekući zapisi iz R i S sadrže istu vrijednost za A ) duplikat = 1;
            pokušaj učitati idući zapis iz S;
    }
    ako ( duplikat == 0 )
            prepiši vrijednost za A iz tekućeg zapisa iz R na kraj od S kao novi zapis;
    pokušaj učitati idući zapis iz R;
}
```

- Ako je *R* velika, algoritam s ugniježđenim petljama zahtijeva previše vremena. Tada je bolje postupiti na sljedeći način: izdvojiti sve vrijednosti za *A* koje se pojavljuju u *R* te zatim sortirati niz izdvojenih vrijednosti. U sortiranom nizu duplikati će se pojavljivati jedan iza drugoga, pa ih je lako eliminirati jednim sekvencijalnim čitanjem.
- Još jedna ideja je: razvrstati izdvojene vrijednosti za *A* u skupine pomoću hash funkcije. Duplikati će se tada naći u istoj skupini, pa ih je opet lako eliminirati.

Izvrednjavanje ostalih operacija. Kod implementacije ostalih operacija pojavljuju se slične ideje kao kod prirodnog spoja ili projekcije. Na primjer:

- Kartezijev produkt dviju relacija R_1 i R_2 računa se ugniježđenom petljom. Bolji algoritam nije moguć jer se ionako svaki zapis iz datoteke R_1 mora kombinirati sa svakim zapisom iz datoteke R_2.
- Unija dviju relacija R_1 i R_2 dobiva se na očigledni način, dakle spajanjem datoteka. Pritom, slično kao kod projekcije, treba eliminirati zapise-duplikate. Opet pomaže sortiranje datoteke R_1 i R_2, ili uporaba hash funkcije.
- Presjek dviju relacija može se shvatiti kao specijalni slučaj prirodnog spoja gdje su svi atributi zajednički. Zato algoritmi za računanje presjeka liče na one za računanje prirodnog spoja. Računanje skupovne razlike također je slično.
- Daljnji relacijski operatori mogu se izraziti pomoću već razmatranih, pa se oni obično ne implementiraju zasebno.

6.3.4. Optimizacija upita

U prethodnim odjeljcima vidjeli smo da se optimizacija upita provodi na dvije razine. Viša razina bavi se transformacijom algebarskog izraza u oblik koji je pogodniji za izvrednjavanje. Niža razina svodi se na izbor pogodnog algoritma za izvrednjavanje svake pojedine operacije unutar algebarskog izraza. Obje razine optimizacije provode se tako da DBMS primjenjuje odgovarajuća pravila, dakle pravila o ekvivalenciji algebarskih izraza, odnosno pravila o tome koji od raspoloživih algoritama je pogodan u kojoj situaciji.

Da bismo zornije prikazali postupak optimizacije upita, u nastavku ovog odjeljka navodimo sedam primjera pravila za optimizaciju. Pritom se prvih šest primjera odnose na algebarsku transformaciju, a zadnji primjer na izbor algoritma za izvrednjavanje. Naš popis nije ni u kojem slučaju cjelovit ni konačan, već služi samo za ilustraciju. Pravila ovog tipa obično predstavljaju poslovnu tajnu proizvođača DBMS-a i sredstvo kojim oni pokušavaju postići komparativnu prednost jedan pred drugim.

Kombiniranje selekcija. Očito vrijedi ovakva ekvivalencija:

$$(R \text{ where } \mathcal{B}_1) \text{ where } \mathcal{B}_2 = R \text{ where } (\mathcal{B}_1 \text{ and } \mathcal{B}_2).$$

Izraz s lijeve strane treba pretvoriti u oblik s desne strane. Time smanjujemo potrebno vrijeme ukoliko se obje selekcije izvrednjavaju podjednako „sporo“ (dakle pregledom cijele relacije). Ako se jedna relacija odvija brzo (na primjer zahvaljujući postojanju pomoćnih fizičkih struktura podataka) a druga sporo, tada se kombiniranje ne isplati, jer će rezultirajuća selekcija također biti spora. Znači, odluka što je bolje ovisi o fizičkoj građi baze podataka.

Izvlačenje selekcije ispred prirodnog spoja ili Kartezijevog produkta. Ako uvjet $\mathcal{B}$ sadrži samo atribute od *R*, a ne one od *S*, tada vrijedi:

(R join S) where $\mathcal{E}$ = (R where $\mathcal{E}$) join S,
(R times S) where $\mathcal{E}$ = (R where $\mathcal{E}$) times S,

Ovakva transformacija se preporuča jer ona može znatno smanjiti broj n-torki koje ulaze u prirodni spoj odnosno Kartezijev produkt. Općenitije, ako $\mathcal{E}$ rastavimo na $\mathcal{E}$ = $\mathcal{E}_R$ and $\mathcal{E}_S$ and $\mathcal{E}_C$ and $\mathcal{E}'$, gdje $\mathcal{E}_R$ sadrži samo atribute od R, $\mathcal{E}_S$ sadrži samo atribute od S, $\mathcal{E}_C$ sadrži zajedničke atribute od R i S, $\mathcal{E}'$ predstavlja ostatak od $\mathcal{E}$, tada vrijedi:

(R join S) where $\mathcal{E}$ = ((R where ($\mathcal{E}_R$ and $\mathcal{E}_C$)) join (S where ($\mathcal{E}_S$ and $\mathcal{E}_C$))) where $\mathcal{E}'$.

Slična ekvivalencija može se napisati i za operaciju times.

Na primjer, želimo naći JMBAG-ove svih studenata na drugoj godini studija koji su upisali neki predmet kod nastavnika Kleina i dobili ocjenu veću od 2. Upit se može izraziti kao:

```
RESULT := ( (STUDENT join UPISAO join PREDMET) where
          ((GODINA_STUDIJA=2) and (OCJENA>2) and (OIB_NASTAVNIKA=44102179316)) )
          [JMBAG].
```

Primjenom transformacije dobivamo optimizirani izraz:

```
RESULT := ( (STUDENT where GODINA_STUDIJA=2) join
             (UPISAO where OCJENA>2) join
             (PREDMET where OIB_NASTAVNIKA=44102179316) ) [JMBAG].
```

Izvlačenje selekcije ispred projekcije. Ako uvjet $\mathcal{E}$ sadrži samo atribute X po kojima se obavlja projiciranje, tada je:

$R[X]$ where B = (R where B) $[X]$.

Projiciranje može dugo trajati zbog eliminacije „duplikata“. Zato je dobro najprije smanjiti broj n-torki selektiranjem. To je posebno preporučljivo onda kad postoje fizička sredstva za brzu selekciju.

Kombiniranje projekcija. Ako su X, Y i Z atributi od relacije R, tada vrijedi:

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

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

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

(R join S)$[X]$ = $R[X]$ join $S[X]$.

No pravilo ne vrijedi za proizvoljan skup atributa X. Neka su A_R atributi od R, A_S atributi od S, $A_C = A_R \cap A_S$ zajednički atributi od R i S. Tada vrijedi:

$$(R \text{ join } S)[X] = (R[(X \cap A_R) \cup A_C] \text{ join } S[(X \cap A_S) \cup A_C])[X] .$$

Opet se smanjuje broj n-torki koje ulaze u prirodni spoj. Zahvat ne mora uvijek biti koristan, jer projekcija može spriječiti efikasnu implementaciju prirodnog spoja. Na primjer, projekcija može stvoriti privremenu relaciju na koju nisu primjenjive postojeće pomoćne fizičke strukture. Znači, odluka što je bolje opet ovisi o fizičkoj građi.

Optimizacija skupovnih operacija. Koji put su od koristi sljedeća pravila koja se odnose na skupovnu uniju ili razliku:

$$(R \text{ union } S) \text{ where } \mathcal{E} = (R \text{ where } \mathcal{E}) \text{ union } (S \text{ where } \mathcal{E}) ,$$
$$(R \text{ minus } S) \text{ where } \mathcal{E} = (R \text{ where } \mathcal{E}) \text{ minus } (S \text{ where } \mathcal{E}) ,$$
$$(R \text{ union } S)[X] = R[X] \text{ union } S[X] ,$$
$$(R \text{ minus } S)[X] = R[X] \text{ minus } S[X] ,$$
$$(R \text{ where } \mathcal{E}_1)[X] \text{ union } (R \text{ where } \mathcal{E}_2)[X] = (R \text{ where } (\mathcal{E}_1 \text{ or } \mathcal{E}_2))[X] .$$

Predzadnja jednakost vrijedi pod pretpostavkom da X sadrži primarne atribute od R (a time i od S). Transformacijama se nastoji smanjiti broj n-torki koje sudjeluju u operacijama union i minus. Operacija intersect je specijalni slučaj od join, pa za nju vrijede ista pravila kao za join.

Optimalni izbor algoritma za izvrednjavanje prirodnog spoja. Treba izračunati prirodni spoj za relacije koje su pohranjene su u datotekama R_1 i R_2. Stoje nam na raspolaganju četiri algoritma za računanje prirodnog spoja koji su bili opisani u Odjeljku 6.2.2. Tada odluku o izboru algoritma donosimo na sljedeći način.

- Ako su datoteke R_1 i R_2 već sortirane po zajedničkom podatku B, tada je najefikasniji algoritam zasnovan na sortiranju i sažimanju.
- Ukoliko je jedna datoteka dovoljno mala da stane u glavnu memoriju, treba odabrati algoritam ugniježđenih petlji.
- Ako je jedna datoteka znatno veća od druge i ima odgovarajući indeks, tada je najbolji algoritam zasnovan na indeksu.
- Za velike datoteke R_1 i R_2 bez indeksa, najbolji je algoritam zasnovan na hash funkciji i razvrstavanju.

Ovo pravilo zasnovano je na analizi složenosti pojedinih algoritama koju smo proveli u Odjeljku 6.3.2.

6.4. Zadaci za vježbu

Zadatak 6.1. Datoteka s najboljim rezultatima neke računalne igre sastoji se od zapisa čiji oblik je prikazan na slici 6.12. Duljina zapisa je 40 byte. Datoteka nikad ne sadrži više od 100 zapisa. Najčešća operacija je ispis rang liste najboljih rezultata (prvih n mjesta). Predložite pogodnu organizaciju datoteke. Nacrtajte odgovarajući dijagram. Pretpostavljamo da se blok vanjske memorije sastoji od 512 byte, a adresa bloka zauzima 4 byte.

BODOVI	PREZIME I IME	DATUM

Slika 6.12: Zapis s najboljim rezultatima računalne igre.

Zadatak 6.2. Datoteka automobila u nekom gradu sastoji se od zapisa čiji oblik je prikazan na Slici 6.13. Duljina zapisa je 150 byte. Očekujemo da će datoteka sadržavati nekoliko tisuća zapisa. Najčešća operacija je: traženje podataka o automobilu sa zadanim registarskim brojem. Predložite pogodnu organizaciju datoteke. Nacrtajte odgovarajući dijagram. Blok vanjske memorije sastoji se od 512 byte, a adresa bloka zauzima 4 byte.

REGISTARSKA OZNAKA	TIP AUTOMOBILA	GODINA PROIZVODNJE AUTOMOBILA	PREZIME I IME VLASNIKA	….

Slika 6.13: Zapis o automobilu.

Zadatak 6.3. Datoteka sadrži rječnik stranih riječi i sastoji se od zapisa čiji oblik je prikazan na Slici 6.14. Strana riječ zauzima 15 znakova, a prijevod 35 znakova; znači duljina zapisa je 50 byte. Rječnik sadrži oko 10000 riječi. Najčešća operacija je ispis prijevoda za zadanu stranu riječ. Pritom se dozvoljava zadavanje početka strane riječi (prvih nekoliko slova) - tada treba ispisati sve riječi koje počinju na zadani način i njihove prijevode. Predložite pogodnu organizaciju datoteke. Nacrtajte odgovarajući dijagram. Pretpostavljamo da se blok vanjske memorije sastoji od 512 byte, a adresa bloka zauzima 4 byte.

STRANA RIJEČ	PRIJEVOD

Slika 6.14: Zapis o stranoj riječi.

Zadatak 6.4. Datoteka sadrži tehničke karakteristike vijaka koji se rabe u nekoj tvornici. Oblik zapisa vidljiv je na Slici 6.15. Duljina zapisa je oko 120 byte. Očekujemo da će datoteka sadržavati više od tisuću zapisa. Osim traženja po identifikacijskom broju, javlja se potreba za odgovaranjem na sljedeće upite.

- Koji vijci imaju promjer u zadanom intervalu?
- Koji su vijci s duljinom navoja u zadanom intervalu?
- Koji vijci imaju promjer glave u zadanom intervalu?
- Razne Booleovske kombinacije prethodnih upita.

Predložite pogodnu organizaciju datoteke. Nacrtajte odgovarajući dijagram. Pretpostavljamo da se blok vanjske memorije sastoji od 512 byte, a adresa bloka zauzima 4 byte.

IDENTIFIKACIJSKI BROJ	PROMJER VIJKA (mm)	DULJINA NAVOJA (mm)	PROMJER GLAVE	….

Slika 6.15: Zapis o vijku.

Zadatak 6.5. Datoteka sadrži podatke o zaposlenima u nekom poduzeću. Oblik zapisa prikazan je na Slici 6.16. Zbog fluktuacije kadra česta su ubacivanja i izbacivanja zapisa. Duljina zapisa je 200 byte. Ukupan broj zaposlenih je nekoliko stotina. Osim traženja na osnovi matičnog broja, traže se i zaposleni sa zadanim zanimanjem ili iz zadanog odjela. Često se traže i zaposleni koji istovremeno zadovoljavaju dva ovakva uvjeta, dakle oni sa zadanim zanimanjem i iz zadanog odjela. Predložite pogodnu organizaciju datoteke. Nacrtajte odgovarajući dijagram. Pretpostavljamo da se blok vanjske memorije sastoji od 512 byte, a adresa bloka zauzima 4 byte.

MATIČNI BROJ	PREZIME I IME	ZANIMANJE	ODJEL	….

Slika 6.16: Zapis o zaposleniku.

Zadatak 6.6. Zapisi osnovne datoteke sadrže sljedeće vrijednosti primarnog ključa: 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256. Prikažite gusti indeks za tu datoteku pomoću B-stabla reda 3. Nacrtajte odgovarajući dijagram.

Zadatak 6.7. U osnovnu datoteku iz prethodnog zadatka ubačen je novi zapis s vrijednošću primarnog ključa 32. Uskladite indeks B-stablo s tom promjenom, dakle prepravite dijagram iz prethodnog zadatka.

Zadatak 6.8. Rješavanjem Zadatka 3.1 ili 4.1 dobili ste nadopunjenu relacijsku shemu baze podataka o fakultetu. Pretvorite tu relacijsku shemu u fizičku shemu za MySQL.

Zadatak 6.9. Rješavanjem Zadatka 3.2 ili 4.2 dobili ste relacijsku shemu baze podataka o knjižnici. Pretvorite tu relacijsku shemu u fizičku shemu za MySQL.

Zadatak 6.10. Rješavanjem Zadatka 3.3 ili 4.8 dobili ste relacijsku shemu za bazu podataka iz vašeg područja interesa. Pretvorite tu relacijsku shemu u fizičku shemu za MySQL.

Zadatak 6.11. U Prilogu 1 pronađite relacijsku shemu baze podataka o bolnici. Na osnovu te sheme i bez čitanja ostatka priloga sami oblikujte odgovarajuću fizičku shemu za MySQL. Usporedite vaše rješenje s onim u prilogu.

Zadatak 6.12. U Prilogu 2 pronađite normaliziranu relacijsku shemu baze podataka o znanstvenoj konferenciji. Na osnovu te sheme i bez čitanja ostatka priloga sami oblikujte odgovarajuću fizičku shemu za MySQL. Usporedite vaše rješenje s onim u prilogu.

7. Integritet i sigurnost baze podataka

Ovo poglavlje posvećeno je integritetu i sigurnosti baze podataka. Riječ je o problematici koja je izuzetno važna za ispravno funkcioniranje baze nakon što je ona bila implementirana te je ušla je u redovnu uporabu. Integritet i sigurnost u prvom redu su briga administratora baze, no dio brige može preuzeti i projektant.

U ovom poglavlju upoznat ćemo se s nizom mehanizama za čuvanje integriteta i sigurnosti baze. Neke od tih mehanizama automatski pokreće DBMS prilikom svog redovnog rada, drugi se implementiraju onda ako ih projektant ugradi u svoju fizičku shemu, treći opet stoje na raspolaganju administratoru koji ih pokreće po potrebi.

Poglavlje je podijeljeno u tri potpoglavlja. Prvo potpoglavlje govori o integritetu baze, te o načinima kako projektant može dograditi svoju fizičku shemu tako da osigura čuvanje integriteta. U drugom poglavlju proučavamo dva aspekta sigurnosti: to su mogućnost oporavka baze u slučaju tehničkog oštećenja, te zaštita od neovlaštenih radnji korisnika. Treće poglavlje bavi se mehanizmima kojima DBMS omogućuje istovremeni rad korisnika, održavajući pritom integritet baze čak i onda kad korisnici istovremeno izvode konfliktne radnje.

7.1. Čuvanje integriteta

Čuvati *integritet* baze znači čuvati korektnost i konzistentnost podataka. *Korektnost* znači da svaki pojedini podatak ima ispravnu vrijednost. *Konzistentnost* znači da su različiti podaci međusobno usklađeni, dakle ne protuslove jedan drugome. Integritet baze lako bi se mogao narušiti na primjer pogrešnim radom aplikacija.

Voljeli bismo kad bi se baza podataka sama mogla braniti od narušavanja integriteta. U tu svrhu, suvremeni DBMS-i dozvoljavaju projektantu baze da definira takozvana *ograničenja* (*constraints*). Riječ je o uvjetima (pravilima) koje korektni i konzistentni podaci moraju zadovoljavati.

Projektant uvodi ograničenja tako da ih upiše u fizičku shemu baze. Uvedena ograničenja DBMS će uključiti u konačnu realizaciju baze. To znači da će u kasnijem radu kod svake promjene podataka DBMS automatski provjeravati jesu li sva ograničenja zadovoljena. Ako neko ograničenje nije zadovoljeno, tada DBMS neće izvršiti traženu promjenu, već će dotičnoj aplikaciji poslati poruku o greški. Sljedeća tri odjeljka opisuju tri vrste ograničenja, te konkretne načine njihovog uvođenja u fizičku shemu.

7.1.1. Uvođenje ograničenja kojima se uspostavlja integritet domene

Ograničenja za integritet domene izražavaju činjenicu da vrijednost atributa mora biti iz odgovarajuće domene. Zahtjev da vrijednost primarnog atributa ne smije biti prazna također spada u ovu kategoriju.

Ograničenje na integritet domene uvodi se u prvom redu tako da se u naredbi CREATE TABLE atributu pridruži odgovarajući tip, uz eventualnu klauzulu NOT NULL. No popis podržanih tipova obično je presiromašan, tako da samim pridruživanjem tipa često ne uspijevamo u potpunosti izraziti potrebno ograničenje.

Kao primjer, pogledajmo opet početnu fizičku shemu za bazu podataka o fakultetu na Slici 6.9. Vidimo da smo za neke atribute uspjeli vrlo precizno odrediti tipove, čime smo osigurali integritet domene. Zaista, za GODINU_STUDIJA zadali smo da je tip ENUM, što znači da GODINA_STUDIJA može poprimati samo vrijednosti iz navedene liste '1', '2', '3', '4', '5'. U skladu s time, DBMS će spriječiti upis studenta u 6. godinu studija. Da smo GODINU_STUDIJA proglasili malim cijelim brojem, tada ne bismo mogli spriječiti pogrešan upis.

Zamislimo sada da se naš fakultet nalazi u zgradi koja ima tri etaže, označene s 1, 2 i 3. Pretpostavimo da su troznamenkasti brojevi soba tako oblikovani da prva znamenka odgovara etaži. Tada tip NUMERIC(3) UNSIGNED kojeg smo pridružili atributu BROJ_SOBE ne čuva integritet domene. Naime, ispravni brojevi soba su samo oni između 101 i 399, a DBMS će zbog tipa NUMERIC(3) UNSIGNED dozvoliti unos bilo kojeg pozitivnog troznamenkastog broja, dakle i 099 i 501.

Budući da pridruživanje tipa atributu ne osigurava da će se u potpunosti čuvati integritet domene, mnogi DBMS-i dozvoljavaju da se u shemu ugradi i precizniji uvjet kojeg vrijednosti atributa moraju zadovoljavati. Takav uvjet može se uklopiti u naredbu CREATE TABLE ili se on može pojaviti kao zasebna naredba.

```
CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3)
     CHECK (BROJ_SOBE BETWEEN 101 AND 399),
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB));
```

Slika 7.1: Osiguravanje integriteta domene za BROJ_SOBE.

Na Slici 7.1 vidi se jedan način kako se uvođenjem dodatnog uvjeta može osigurati integritet domene za naš atribut BROJ_SOBE u zgradi s tri etaže. Dodatni uvjet zadan je unutar naredbe CREATE TABLE NASTAVNIK –dakle prikazana je nova verzija te naredbe koja bi trebala zamijeniti verziju sa Slike 6.9. Budući da MySQL ovdje ne daje pravu podršku, Slika 7.1 iznimno rabi sintaksu MS SQL Server-a. Slično rješenje postoji i u Oracle-u.

7.1.2. Uvođenje ograničenja za čuvanje integriteta unutar relacije

Ograničenja za čuvanje integriteta unutar relacije čuvaju korektnost veza između atributa te relacije, na primjer funkcionalne ovisnosti. Najvažniji primjer takvog ograničenja je ono koje traži da dvije n-torke unutar iste relacije ne smiju imati jednaku vrijednost primarnog ključa. Slično ograničenje može se postaviti i za atribut koji nije odabran za primarni ključ ali je kandidat za ključ.

Ograničenje kojim se izražava svojstvo ključa uvodi se u prvom redu tako da se u naredbu CREATE TABLE stavi klauzula PRIMARY KEY odnosno UNIQUE, zajedno s imenom ili imenima atributa koji čine primarni ključ odnosno kandidat za ključ. Drugi način uvođenja istih ograničenja je eksplicitno stvaranje primarnih indeksa naredbom CREATE UNIQUE INDEX.

Ako pogledamo fizičku shemu za bazu podataka o fakultetu na Slici 6.9, vidimo da u svim naredbama CREATE TABLE postoje odgovarajuće klauzule PRIMARY KEY. Znači, već u polaznoj verziji sheme osigurali smo da se čuva svojstvo ključa. Umjesto klauzule PRIMARY KEY mogli smo uporabiti naredbu CREATE UNIQUE INDEX, no način sa Slike 6.9 smatra se čitljivijim.

Kao primjer, zamislimo sada da na našem fakultetu vrijedi pravilo da svaki nastavnik mora imati zasebnu sobu. Tada atribut BROJ_SOBE postaje kandidat za ključ unutar relacije NASTAVNIK, naime ne smije se desiti da dva nastavnika sjede u istoj sobi, pa BROJ_SOBE jednoznačno određuje n-torku o nastavniku. Shema sa Slike 6.9 tada ne čuva integritet unutar relacije jer se njome ne osigurava jedinstvenost vrijednosti za BROJ_SOBE.

```
CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB),
   UNIQUE(BROJ_SOBE));
```

```
CREATE UNIQUE INDEX NB_IND ON NASTAVNIK (BROJ_SOBE);
```

Slika 7.2: Dva načina osiguravanja da BROJ_SOBE ima svojstvo ključa.

Na Slici 7.2 vide se dva načina kako se u MySQL-u uvođenjem dodatnog ograničenja može osigurati svojstvo ključa za BROJ_SOBE. Prvi način svodi se na ubacivanje klauzule UNIQUE u naredbu CREATE TABLE NASTAVNIK – ovakva verzija naredbe trebala bi zamijeniti verziju sa Slike 6.9. Drugi način svodi se na to da se shemi sa Slike 6.9 doda nova naredba CREATE UNIQUE INDEX, s time da naredba CREATE TABLE ostaje nepromijenjena.

7.1.3. Uvođenje ograničenja kojima se čuva referencijalni integritet

Ograničenja za čuvanje referencijalnog integriteta služe zato da se očuva konzistentnost veza između relacija. Najčešće je riječ o ograničenjima koja se odnose na strani ključ, dakle na atribut u jednoj relaciji koji je ujedno primarni ključ u drugoj relaciji. Svaka vrijednost takvog atributa u prvoj relaciji mora biti prisutna i u drugoj relaciji.

Ako pogledamo našu bazu podataka o fakultetu sa Slike 3.5 i 3.6, tada uočavamo da u njoj postoji mnogo stranih ključeva. Relacija PREDMET sadrži strane ključeve IME ZAVODA i OIB NASTAVNIKA koji odgovaraju primarnom ključu u relaciji ZAVOD odnosno NASTAVNIK. Relacija NASTAVNIK ima kao strani ključ IME ZAVODA, dakle ključ iz relacije ZAVOD. U relaciji ZAVOD nalazi se strani ključ OIB PROČELNIKA koji odgovara ključu OIB iz relacije NASTAVNIK. Svaki od primarnih atributa JMBAG i ŠIFRA PREDMETA iz relacije UPISAO predstavlja sam za sebe strani ključ vezan uz primarni ključ iz relacije STUDENT odnosno PREDMET. Fizička shema sa Slike 6.9 ne čuva integritet ni jednog od ovih stranih ključeva, naime u njoj nisu označeni nikakvi odnosi između atributa u različitim relacijama.

U današnjim DBMS-ima ograničenje kojim se čuva svojstvo stranog ključa uvodi se tako da se u odgovarajuću naredbu CREATE TABLE stavi klauzula FOREIGN KEY … REFERENCES … . Dakle eksplicitno navodimo ime atributa koji predstavlja strani ključ u dotičnoj relaciji, te ime relacije u kojoj taj isti atribut predstavlja strani ključ.

Slika 7.3 sadrži novu verziju fizičke sheme za bazu podataka o fakultetu, koju možemo rabiti kao zamjenu za shemu sa Slike 6.9. Naredbe su opet zapisane u sintaksi MySQL-a. Zahvaljujući klauzulama FOREIGN KEY, ova nova verzija ima uvedena ograničenja za čuvanje integriteta gotovo svih prije navedenih stranih ključeva. Jedini strani ključ kojeg smo ispustili je OIB PROČELNIKA u relaciji ZAVOD.

Nakon realizacije sheme sa Slike 7.3, DBMS na primjer neće dozvoliti da se u relaciju UPISAO ubaci n-torka s vrijednošću JMBAG koje nema u relaciji STUDENT. Također, iz relacije PREDMET neće se moći brisati n-torka sa ŠIFROM PREDMETA koja se pojavljuje u relaciji UPISAO.

Primijetimo da je redoslijed naredbi CREATE TABLE na Slici 7.3 drukčiji nego na Slici 6.9. To je zato što se bilo koja relacija sa stranim ključem može stvoriti tek nakon što već postoji ona relacija koju taj strani ključ referencira. Na primjer, relacija NASTAVNIK se zbog stranog ključa IME ZAVODA mora stvarati nakon relacije ZAVOD. Pritom nije moguće u relaciji ZAVOD istovremeno zadati i strani ključ OIB PROČELNIKA jer bi taj strani ključ zahtijevao baš suprotni redoslijed stvaranja dviju relacija.

Na Slici 7.3 vidimo također i neke specifičnosti MySQL-a. Kao prvo, morali smo zahtijevati da se za sve datoteke umjesto standardne indeks-sekvencijalne organizacije rabi specifična organizacija InnoDB. Naime, jedino takva građa datoteka unutar MySQL-a omogućuje provjeru svojstava stranog ključa. Kao drugo, uz svaki strani ključ morali smo eksplicitno deklarirati sekundarni indeks koji će služiti za traženje po tom ključu. Kod drugih DBMS-a, odgovarajuće naredbe CREATE TABLE obično izgledaju jednostavnije, to jest sadrže samo klauzule FOREIGN KEY, a ostali parametri se biraju automatski. Zahvaljujući navođenju sekundarnih indeksa, Slika 7.3 usput zamjenjuje i Sliku 6.10, naime nema više potrebe za zasebnim naredbama CREATE INDEX.

```
CREATE TABLE STUDENT
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   GODINA_STUDIJA ENUM('1','2','3','4','5'),
   PRIMARY KEY(JMBAG))
   ENGINE=INNODB;

CREATE TABLE ZAVOD
  (IME_ZAVODA CHAR(40) NOT NULL,
   OIB_PROCELNIKA NUMERIC(11) UNSIGNED,
   OPIS_DJELATNOSTI CHAR(160),
   PRIMARY KEY(IME_ZAVODA))
   ENGINE=INNODB;

CREATE TABLE NASTAVNIK
  (OIB NUMERIC(11) UNSIGNED NOT NULL,
   PREZIME CHAR(20),
   IME CHAR(20),
   IME_ZAVODA CHAR(40),
   BROJ_SOBE NUMERIC(3) UNSIGNED,
   PLACA NUMERIC(5) UNSIGNED,
   PRIMARY KEY(OIB),
   INDEX NI_IND (IME_ZAVODA),
   FOREIGN KEY (IME_ZAVODA) REFERENCES ZAVOD(IME_ZAVODA))
   ENGINE=INNODB;

CREATE TABLE PREDMET
  (SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   NASLOV CHAR(80),
   IME_ZAVODA CHAR(40),
   OIB_NASTAVNIKA NUMERIC(11) UNSIGNED,
   SEMESTAR ENUM('Z','L'),
   ECTS_BODOVI NUMERIC(2) UNSIGNED,
   PRIMARY KEY(SIFRA_PREDMETA),
   INDEX PI_IND (IME_ZAVODA),
   INDEX PO_IND (OIB_NASTAVNIKA),
   FOREIGN KEY (IME_ZAVODA) REFERENCES ZAVOD(IME_ZAVODA),
   FOREIGN KEY (OIB_NASTAVNIKA) REFERENCES NASTAVNIK(OIB))
   ENGINE=INNODB;

CREATE TABLE UPISAO
  (JMBAG NUMERIC(10) UNSIGNED NOT NULL,
   SIFRA_PREDMETA NUMERIC(5) UNSIGNED NOT NULL,
   DATUM_UPISA DATE,
   OCJENA ENUM('2','3','4','5'),
   PRIMARY KEY(JMBAG, SIFRA_PREDMETA),
   INDEX UJ_IND (JMBAG),
   INDEX US_IND (SIFRA_PREDMETA),
   FOREIGN KEY (JMBAG) REFERENCES STUDENT(JMBAG),
   FOREIGN KEY (SIFRA_PREDMETA) REFERENCES
     PREDMET(SIFRA_PREDMETA))
   ENGINE=INNODB;
```

Slika 7.3: Čuvanje integriteta za strane ključeve.

Na kraju, treba napomenuti da provjera svakog ograničenja, pogotovo onog za referencijalni integritet, predstavlja teret za DBMS i doprinosi usporavanju rada. Projektant zato treba odmjeriti koja ograničenja su stvarno potrebna, a koja se mogu zanemariti ili zamijeniti odgovarajućim provjerama u aplikacijama. Shema sa Slike 7.3 predstavlja svojevrsno pretjerivanje jer smo za 5 relacija uveli 5 referencijalnih integriteta. Vjerojatno bi bilo dovoljno provjeravati samo sekundarne ključeve u relaciji UPISAO, budući da se jedino tu radi o nešto većoj količini podataka.

7.2. Sigurnost baze

Baza podataka predstavlja dragocjen resurs. Njezin nastanak i održavanje iziskuju goleme količine ljudskog rada. Zato se od DBMS-a očekuje da on u što većoj mjeri garantira sigurnost podataka. To znači da se ne smije desiti da podaci budu uništeni ili oštećeni zbog tehničkog kvara, pogrešnih transakcija, nepažnje korisnika ili zlonamjernih radnji.

Današnji DBMS-i raspolažu djelotvornim mehanizmima za sigurnost. U ovom potpoglavlju opisujemo kako projektant odnosno administrator baze mogu postići da se ti mehanizmi aktiviraju tijekom rada baze. Prvi odjeljak bavi se tehničkim aspektom sigurnosti, dakle oporavkom baze u slučaju njezinog većeg ili manjeg oštećenja. Ostali odjeljci bave se suptilnijim aspektom koji se tiče zaštite baze od neovlaštenih radnji korisnika.

7.2.1. Stvaranje pretpostavki za oporavak baze

Da bismo bolje razumjeli načine kako sve može doći do oštećenja baze, sjetimo se da se rad s bazom u pravilu svodi na pokretanje takozvanih *transakcija*. Makar jedna transakcija s korisničkog stanovišta predstavlja jednu nedjeljivu cjelinu, ona se obično realizira kao niz od nekoliko elementarnih zahvata u samoj bazi.

Tipičan primjer transakcije je bankovna transakcija, gdje se zadani novčani iznos prebacuje s jednog bankovnog računa na drugi. Takvo prebacivanje predstavlja jednu logičku cjelinu, no ono se realizira kroz dvije zasebne promjene u bazi: smanjivanje salda na jednom računu, te povećanje salda na drugom.

Osnovno svojstvo transakcije je da ona prevodi bazu iz jednog konzistentnog stanja u drugo. No među-stanja koja nastaju nakon pojedinih operacija unutar transakcije mogu biti nekonzistentna. Da bi se očuvao integritet baze, transakcija mora u cijelosti biti izvršena ili uopće ne smije biti izvršena. Transakcija koja iz bilo kojeg razloga nije do kraja bila obavljena morala bi biti *neutralizirana* – dakle svi podaci koje je ona do trenutka prekida promijenila morali bi natrag dobiti svoje polazne vrijednosti.

Na primjer, prebacivanje novaca s jednog bankovnog računa na drugi čuva konzistenciju u smislu da ukupna količina novca na svim računima ostaje ista. No u tijeku realizacije tog prebacivanja doći će do privremene nekonzistencije, jer će novci biti skinuti s jednog računa a neće još biti stavljeni na drugi, ili obratno, bit će stavljeni na drugi račun prije nego što su bili skinuti s prvog. Ako se transakcija prekine tijekom realizacije, novci će netragom nestati ili će se stvoriti niotkud.

Rekli smo već da se baza podataka u toku svog rada može naći u neispravnom stanju. Budući da se ispravni rad s bazom postiže cjelovitim izvođenjem transakcija, najčešći razlog koji dovodi do oštećenja baze je baš neispravno izvedena transakcija, dakle transakcija koja se počela izvršavati, nije bila obavljena do kraja, no nije bila ni neutralizirana. Daljnji, no mnogo rjeđi razlozi koji također mogu dovesti do oštećenja baze su: pogrešno sastavljena transakcija, softverske greške u DBMS-u ili operacijskom sustavu, te hardverske greške, na primjer kvar diska.

Od suvremenog DBMS-a očekuje se da u svim slučajevima oštećenja baze omogući njezin *oporavak*, dakle povratak u stanje koje je što ažurnije i pritom još uvijek konzistentno. No da bi oporavak zaista bio moguć, mora biti ispunjena bar neka od sljedećih pretpostavki.

- **Uključen je DBMS-ov mehanizam za upravljanje transakcijama.** Tada DBMS očekuje od aplikacije da ona eksplicitno najavi početak transakcije, te da eksplicitno objavi njezin završetak. Pritom završetak može biti potvrda (*commit*) da je transakcija ispravno obavljena, ili njezin opoziv (*rollback*). Za vrijeme izvršavanja transakcije DBMS samo privremeno pamti promjene u bazi. U slučaju potvrde, te promjene se trajno upisuju u bazu, a u slučaju opoziva one se neutraliziraju odnosno zaboravljaju.
- **Povremeno se stvara rezervna kopija baze.** Ona se dobiva snimanjem cijele baze na drugi medij (drugi disk ili magnetsku traku), i to u trenutku kad smatramo da je baza u konzistentnom stanju. Stvaranje kopije je dugotrajna operacija koja može ometati redovni rad korisnika. Zato se kopiranje ne obavlja suviše često, već periodički u unaprijed predviđenim terminima – na primjer jednom tjedno.
- **Održava se žurnal-datoteka.** Riječ je o datoteci gdje je ubilježena „povijest“ svake transakcije koja je mijenjala bazu nakon zadnjeg stvaranja rezervne kopije. Za jednu transakciju žurnal evidentira adresu svakog zapisa kojeg je transakcija promijenila, zajedno s prethodnom vrijednošću tog zapisa i novom vrijednošću.

Navedene pretpostavke za oporavak baze zaista omogućuju razne oblike oporavka. Na primjer.

- DBMS-ovom kontrolom transakcija uz mogućnost opoziva znatno se smanjuje broj transakcija koje će oštetiti bazu svojim djelomičnim izvođenjem. Naime, kad god aplikacija ustanovi da se transakcija ne može izvršiti do kraja, ona će ju opozvati, a DBMS će neutralizirati promjene podataka. Sam postupak neutralizacije može se promatrati kao svojevrsni mikroskopski oblik oporavka.
- Žurnal datoteka omogućuje neutralizaciju transakcije koja je došla do kraja i trajno je promijenila bazu, ali se naknadno utvrdilo da je bila pogrešna ili nepotrebna. Rabi se postupak odmotavanja unatrag (*roll-back*). Dakle čita se žurnal od kraja prema početku, pronalaze se stare vrijednosti zapisa koje je transakcija mijenjala, pa se te stare vrijednosti ponovo upisuju na odgovarajuća mjesta u bazu.
- Rezervna kopija baze omogućuje ponovno uspostavljanje baze u slučaju njezinog znatnijeg oštećenja. Postupak se svodi na presnimavanje svih podataka iz rezervne kopije natrag u bazu. Time se uspostavlja stanje zabilježeno zadnjom rezervnom kopijom (možda od prošlog tjedna).
- Rezervna kopija i žurnal datoteka zajedno omogućuju još bolji oporavak baze koja je pretrpjela znatnije oštećenje. Najprije se uspostavlja stanje iz zadnje rezervne kopije. Zatim se rabi postupak odmotavanja unaprijed (*roll-forward*). Dakle čita se žurnal datoteka od početka prema kraju, te se ponovo unose u bazu sve zabilježene promjene podataka redom za svaku potvrđenu transakciju. Time se uspostavlja prilično ažurno stanje koje je prethodilo oštećenju.

Opisani mehanizmi i sredstva za oporavak baze mogu se uključivati i dodavati po potrebi, tako da njima obično rukuju administrator baze i programeri koji razvijaju aplikacije. Ipak, odluka koja će se sredstva rabiti zapravo spada u nadležnost projektanta. Naime, uključivanje pojedinog mehanizma donosi veću sigurnost, ali opterećuje svakodnevni rad baze i smanjuje performanse. Projektant treba procijeniti kakve vrste transakcija će se izvršavati, koliki je stvarni rizik od oštećenja, te u kojoj mjeri se isplati žrtvovati performanse zbog veće sigurnosti.

Na osnovu svoje procjene, projektant treba dati odgovarajuće preporuke budućem administratoru i programerima. Štoviše, u mnogim DBMS-ima uporaba određenih mehanizama zaštite moguća je samo pod uvjetom da su datoteke organizirane na odgovarajući način. Takva ograničenja projektant mora uzeti u obzir kod oblikovanja fizičke građe, to jest projektantova fizička shema mora biti kompatibilna s njegovim vlastitim planovima za kasniju zaštitu.

U nastavku ćemo opisati kako se opisani mehanizmi i sredstva za oporavak baze realiziraju u MySQL-u.

- MySQL po defaultu ne upravlja transakcijama, dakle svaku SQL naredbu on smatra zasebnom cjelinom i odmah ju izvršava u bazi. To se zove *autocommit mode*. No taj default može se promijeniti tako da aplikacija ili korisnik pošalju naredbu

  SET AUTOCOMMIT = 0;

  Nakon toga, ali pod uvjetom da su sve datoteke tipa InnoDB, moguće je upravljati transakcijama. Početak transakcije mora se eksplicitno označiti naredbom

  BEGIN;

  Potvrda da je transakcija uredno završila postiže se naredbom

  COMMIT;

  a opoziv neuspjele transakcije naredbom

  ROLLBACK;

- Stvaranje rezervne kopije MySQL baze obavlja se pozivom uslužnog programa mysqldump. Program se poziva iz komandne ljuske operacijskog sustava i prima opcije koje određuju koja baza će se presnimiti kamo.
- MySQL po defaultu održava žurnal datoteku. Naime, sam DBMS realiziran je kao program mysqld . Taj program prilikom pokretanja čita komandnu datoteku koja u svojem standardnom obliku sadrži opciju oblika

  --log-update=*ime_datoteke* .

  Time je određeno ime datoteke u koju će se upisivati sve promjene baze. Ako želimo isključiti žurnal datoteku, tada treba zaustaviti mysqld i ponovo ga pokrenuti bez navedene opcije.

Kao primjer upravljanja transakcijama u MySQL-u, u nastavku je prikazan niz naredbi kojima se plaća nastavnika Turinga smanjuje za 1000, a plaća nastavnika Pascala povećava za 1000.

```
SET AUTOCOMMIT = 0;

BEGIN;

UPDATE NASTAVNIK
  SET PLACA = PLACA – 1000
  WHERE PREZIME = 'Turing';

UPDATE NASTAVNIK
  SET PLACA = PLACA + 1000
  WHERE PREZIME = 'Pascal';

COMMIT; (ili ROLLBACK;)

SET AUTOCOMMIT = 1;
```

Cijeli ovaj niz naredbi proglašen je nedjeljivom cjelinom koja se mora izvršiti u cijelosti ili se uopće ne smije izvršiti. Na taj način čuva se konzistencija u smislu da ukupan zbroj plaća ostaje isti. Promjene se obavljaju u relaciji NASTAVNIK koja je opisana fizičkom shemom na Slici 7.3. Primijetimo da fizička shema sa Slike 7.3 traži da sve datoteke budu tipa InnoDB. Odabir InnoDB je nužnost jer MySQL znade upravljati jedino transakcijama koje se odvijaju nad takvim datotekama.

7.2.2. Davanje ovlaštenja korisnicima

Zaštita podataka od neovlaštene uporabe uglavnom se postiže tako da se korisnicima pomoću SQL naredbi GRANT i REVOKE dodjeljuju ili uskraćuju ovlaštenja. Uvođenje korisnika i upravljanje njihovim ovlaštenjima obično je posao administratora baze. Ipak, dobro je da projektant u sklopu fizičkog oblikovanja baze već predvidi nekoliko tipičnih korisnika, te predloži njihova ovlaštenja. Projektantovi naputci mogu kasnije služiti administratoru kao obrazac za uvođenje daljnjih korisnika.

U MySQL, pojedino ovlaštenje veže se uz kombinaciju MySQL-ovog imena korisnika i imena stroja s kojeg se korisnik prijavljuje. Dakle, ovlaštenje je pridruženo „e-mail adresi“ ime_korisnika@ime_stroja. To znači da ista osoba može imati drukčija ovlaštenja ukoliko se prijavljuje s drugog stroja. Uobičajena ovlaštenja su: SELECT, INSERT, DELETE, UPDATE, ALTER, CREATE, DROP, ALL, …, a njima se korisnika ovlašćuje da pokreće istoimene SQL naredbe, odnosno u slučaju ALL daju mu se sva ovlaštenja.

Doseg ovlaštenja može biti: globalno za sve baze koje kontrolira dotična instalacija MySQL (*.*), za jednu takvu bazu (ime_baze.*), za jednu relaciju jedne baze (ime_baze.ime_relacije), ili čak za pojedine atribute unutar jedne relacije.

Kao prvi primjer davanja ovlaštenja u MySQL, najprije navodimo naredbu GRANT kojom administrator neregistriranom (anonymous) korisniku dozvoljava pretraživanje fakultetske baze fakultet.

```
> GRANT SELECT on fakultet.* TO ''@localhost;
```

Nakon što se izvela ova naredba GRANT, moguća je ovakva sesija neregistriranog korisnika koji uspješno pregledava sadržaj fakultetske baze no ne uspijeva izvršiti promjene u njoj. Redci koji počinju sa $ izvode se u ljusci operacijskog sustava, a redci koji počinju s > izvode se unutar komandne ljuske mysql.

```
$ mysql –u nepoznato_ime
> SELECT USER;

  (ispisuje se: nepoznato_ime@local host)

> USE fakultet;
> SELECT * FROM PREDMET;
  ...
  (ispis)
  ...
> INSERT INTO PREDMET VALUES
    (33333, 'Novi naslov' , 'Zavod za racunarstvo', 33571209458, 'L', 5));
  ...
  (poruka o greški)
  ...
```

Kao drugi primjer davanja ovlaštenja u MySQL-u, navodimo naredbe kojima administrator stvara korisnika someuser s lozinkom loz000, koji se prijavljuje s lokalnog računala, može raditi sve što hoće u svojoj bazi s imenom someuser, te smije pretraživati fakultetsku bazu fakultet.

```
> CREATE DATABASE someuser;

> GRANT ALL ON someuser.* TO
    someuser@localhost IDENTIFIED BY 'loz000';

> GRANT SELECT ON fakultet.* TO someuser@localhost;
```

Dalje prikazujemo jednu moguću sesiju korisnika someuser. Nakon što se prijavio navođenjem lozinke, someuser najprije neuspješno pokušava gledati bazu tudja_baza za koju nema nikakvih ovlaštenja. Zatim uspješno stvara i ažurira relaciju unutar svoje baze someuser gdje ima sva ovlaštenja. Na kraju radi s bazom fakultet gdje uspješno pregledava sadržaj jedne relacije, no ne uspijeva izvršiti promjene u njoj.

```
$ mysql –u someuser –p
  Enter password: ****** (loz000)

> SELECT USER( );

  (ispisuje se: someuser@localhost)

> USE tudja_baza;
```

```
> SHOW TABLES;
   ...
   (poruka o greški)
   ...
> USE someuser;

> CREATE TABLE PROBA
     (ID NUMERIC(2) UNSIGNED NOT NULL,
     NAZIV CHAR(20),
     PRIMARY KEY(ID));

> INSERT INTO PROBA VALUES (44, 'Novi naziv');

> USE fakultet;

> SELECT * FROM PREDMET;
   ...
   (ispis)
   ...
> INSERT INTO PREDMET VALUES
      (33333, 'Novi naslov', 'Zavod za racunarstvo', 33571209458, 'L', 5));
   ...
   (poruka o greški)
   ...
```

U nastavku slijedi primjer oduzimanja ovlaštenja u MySQL. Najprije vidimo naredbe kojima administrator korisniku someuser oduzima pravo čitanja relacija PREDMET, ZAVOD i UPISAO u fakultetskoj bazi fakultet.

```
> REVOKE SELECT ON fakultet.* FROM ''@localhost;

> REVOKE SELECT on fakultet.* FROM someuser@localhost;

> GRANT SELECT ON fakultet.STUDENT TO someuser@localhost;

> GRANT SELECT ON fakultet.NASTAVNIK TO someuser@localhost;
```

Zatim je moguća sljedeća sesija korisnika someuser, gdje on nakon prijave neuspješno pokušava čitati relaciju PREDMET iz fakultetske baze, no uspijeva pročitati relaciju NASTAVNIK iz iste baze.

```
$ mysql –u someuser -p
   Enter password: ****** (loz000)

> USE fakultet;

> SELECT * FROM PREDMET;
   ...
   (poruka o greški)
   ...
```

```
> SELECT * FROM NASTAVNIK;
  ...
  (ispis)
  ...
```

Prethodni primjeri naredbi GRANT i REVOKE sadržavali su u sebi specifičnosti MySQL-a. Svaki DBMS ima neke svoje specifičnosti, tako da se mogućnosti davanja i oduzimanja ovlaštenja dosta razlikuju. Ono što MySQL nema, a mnogi drugi DBMS-i imaju, mogućnost je uvođenja korisničkih skupina. Ako postoji podrška za korisničke skupine, tada se ovlaštenja ne trebaju dodjeljivati individualnim korisnicima već skupinama. Svakog individualnog korisnika smještavamo u neku skupinu, čime on nasljeđuje sva ovlaštenja za tu skupinu. Takav način rada je spretan ako imamo velik broj korisnika sa sličnim ovlaštenjima.

Na kraju ovog odjeljka treba još naglasiti da se sustav ovlaštenja uvijek oslanja na zaštitu fizičkih direktorija i datoteka na razini operacijskog sustava računala. Tako na primjer u slučaju MySQL-a zaštita u operacijskom sustavu mora biti postavljena tako da jedini proces koji smije pristupati fizičkim direktorijima i datotekama bude sam MySQL, točnije program mysqld.

7.2.3. Uporaba pogleda kao mehanizma zaštite

U Poglavlju 1 spomenuli smo poglede (pod-sheme) kao sredstvo za postizavanje logičke nezavisnosti podataka. No pogledi mogu služiti i za zaštitu podataka. Naime, projektant ili administrator mogu određenom korisniku pridružiti njegov pogled na bazu. Korisnik tada „vidi“ samo dio baze, pa su mu time bitno ograničene njegove mogućnosti zlouporabe podataka.

U relacijskom modelu, i globalna shema i pogled (pod-shema) zadaju se kao skup relacija. Pritom se virtualne relacije koje čine pogled izvode iz stvarnih relacija koje čine globalnu shemu. U SQL-u se relacija-pogled zadaje naredbom CREATE VIEW, a izvođenje iz globalnih relacija opisuje se naredbom SELECT koja je ugniježđena u CREATE VIEW.

Da bi zaštita preko pogleda zaista funkcionirala, potrebno je još dotičnom korisniku regulirati ovlaštenja. Naime, naredbe GRANT i REVOKE primjenjive su ne samo na stvarne relacije nego i na poglede. Pomoću GRANT i REVOKE mora se osigurati da korisnik nema pristupa do stvarnih relacija, ali da može pristupiti pogledima. Na taj način, korisnik je prisiljen raditi samo s onim podacima koje smo obuhvatili pogledima, a ostali podaci su mu nedostupni.

Definiranje pogleda za pojedine vrste korisnika moglo bi se prepustiti administratoru baze. No bolje je da se time bavi projektant, budući da je to aktivnost koja zadire u logičku razinu oblikovanja. Ako to nije učinio prije, projektant bi najvažnije poglede trebao definirati u sklopu svoje fizičke sheme navođenjem odgovarajućih naredbi CREATE VIEW.

Gornji dio Slike 7.4 sadrži prvi primjer uporabe pogleda kao mehanizma zaštite u fakultetskoj bazi sa Slike 6.13. Cilj koji se u tom primjeru želi postići je skrivanje povjerljivog podatka o plaći nastavnika. Navedena naredba CREATE VIEW stvara virtualnu relaciju NAST_VIEW1 koja izgleda skoro isto kao i stvarna relacija NASTAVNIK, jedino što u njoj nema atributa PLACA. Naredba je pisana u sintaksi MySQL. Da bismo zaista zaštitili podatak

o plaći, većinu korisnika baze moramo ovlaštenjima prisiliti da rabe NAST_VIEW1 kao zamjenu za NASTAVNIK. Korisnik tada vidi „vertikalni“ segment originalne relacije (samo neke stupce).

```
CREATE VIEW NAST_VIEW1
  AS SELECT OIB, PREZIME, IME, IME_ZAVODA, BROJ_SOBE
  FROM NASTAVNIK;

CREATE VIEW NAST_VIEW2
  AS SELECT * FROM NASTAVNIK
  WHERE IME_ZAVODA = 'Zavod za računarstvo';

CREATE VIEW UPISAO_VIEW
  AS SELECT  PREDMET.NASLOV, UPISAO.DATUM_UPISA, UPISAO.OCJENA
  FROM UPISAO, PREDMET
  WHERE UPISAO.JMBAG = 0036398757
  AND UPISAO.SIFRA_PREDMETA = PREDMET.SIFRA_PREDMETA;
```

Slika 7.4: Uporaba pogleda u fakultetskoj bazi podataka.

U srednjem dijelu Slike 7.4 vidimo još jedan primjer zaštite fakultetske baze podataka pomoću pogleda. Prikazana naredba CREATE VIEW stvara pogled NAST_VIEW2 namijenjen tajnici Zavoda za računarstvo unutar fakulteta. Opet se služimo sintaksom MySQL-a. Tajnica smije pristupiti svim podacima o nastavnicima, čak i njihovim plaćama, ali sve to samo za nastavnike iz njezinog zavoda. Dakle tajnica vidi „horizontalni“ segment originalne relacije (samo neke retke).

Donji dio Slike 7.4 sadrži treći primjer uporabe pogleda u fakultetskoj bazi podataka. Naredba CREATE VIEW pisana u MySQL-u definira pogled UPISAO_VIEW namijenjen studentu s JMBAG-om 0036398757. Dotični student vidi podatke o predmetima koje je on upisao, no skriveni su mu podaci o upisima drugih studenata. Šifre upisanih predmeta zamijenjene su naslovima predmeta. Dakle student vidi relaciju koja u tom obliku ne postoji u bazi već se dobiva spajanjem podataka iz dviju postojećih relacija.

7.3. Istovremeni pristup

Većina baza podataka po svojoj prirodi je *višekorisnička*. Znači jedan te isti podatak, pohranjen na jednom mjestu, potreban je raznim osobama i raznim aplikacijama, možda čak u isto vrijeme. Na primjer, broj prodanih autobusnih karata za istu vožnju autobusom treba simultano biti dostupan raznim potencijalnim kupcima koji bi istovremeno htjeli kupiti kartu za tu vožnju. Od suvremenog DBMS-a traži se da korisnicima omogući *istovremeni pristup* do podataka. Svaki korisnik pritom treba imati dojam da sam radi s bazom. Obično je riječ o prividnoj istovremenosti (dijeljenje vremena istog računala). Ipak, DBMS i u tom slučaju mora pažljivo koordinirati radnje korisnika, naime on ne smije dopustiti da te istovremene radnje, koje bi mogle biti u međusobnom konfliktu, naruše integritet baze.

U ovom potpoglavlju bavimo se mehanizmima i metodama za čuvanje integriteta baze u uvjetima istovremenog rada korisnika. Riječ je o mehanizmima koji su ugrađeni u DBMS i

aktiviraju se automatski, ili o metodama koje moraju rabiti programeri prilikom pisanja svojih aplikacija. Zahtjev da se integritet baze ne smije narušiti zbog istovremenog pristupa preciznije se izražava kao svojstvo serijalizabilnosti.

7.3.1. Serijalizabilnost paralelnih transakcija

Kao što smo rekli u prošlom potpoglavlju, rad korisnika s bazom podataka svodi se na pokretanje unaprijed definiranih *transakcija*. Makar jedna transakcija sa korisničkog stanovišta predstavlja jednu nedjeljivu cjelinu, ona se obično realizira kao niz od nekoliko elementarnih zahvata u samoj bazi.

U višekorisničkoj bazi dešavat će se da se nekoliko transakcija izvodi paralelno. Osnovne operacije koje pripadaju raznim transakcijama tada će se vremenski ispreplesti. Da bismo održali integritet baze, moramo zahtijevati da učinak tih paralelnih transakcija bude isti kao da su se one izvršavale sekvencijalno, dakle jedna iza druge u nekom (bilo kojem) redoslijedu.

Traženo svojstvo, da učinak istovremenog izvršavanja transakcija mora biti ekvivalentan nekom sekvencijalnom izvršavanju, naziva se *serijalizabilnost*. Dakle, ako to svojstvo zaista vrijedi, kažemo da je dotično paralelno izvršavanje skupa transakcija bilo *serijalizabilno*.

Nažalost, svojstvo serijalizabilnosti ne može se a priori garantirati. Nekontrolirano paralelno izvršavanje transakcija može vrlo lako dovesti do neželjenih efekata. Da bismo to pokazali, zamislimo da studenti iz naše fakultetske baze podataka organiziraju ekskurziju autobusom. Broj studenata koji će ići na ekskurziju ograničen je brojem sjedala u autobusu. Zbog evidencije tko sve ide na ekskurziju, naša fakultetska baza podataka proširena je novim relacijama, koje za svako sjedalo u autobusu bilježe koji student ga je zauzeo, te koliko još ima slobodnih sjedala. Studenti koji žele ići na ekskurziju moraju na *on-line* način rezervirati svoje mjesto u autobusu. Tada je prilikom uporabe baze moguć sljedeći scenarij.

> Dva studenta, svaki sa svog računala od kuće, istovremeno pokušavaju dobiti kartu za ekskurziju. U tom trenutku postoji samo jedno slobodno sjedalo u autobusu. Pokreću se dvije transakcije, T_1 i T_2, koje DBMS „istovremeno“ obavlja. Dolazi do slijedećeg vremenskog redoslijeda izvršavanja elementarnih operacija.
> 1. T_1 učitava iz baze broj slobodnih sjedala.
> 2. T_1 smanjuje pročitanu vrijednost s 1 na 0. Promjena je za sada učinjena samo u radnoj memoriji računala.
> 3. T_2 učitava iz baze broj slobodnih sjedala. Budući da je stanje baze još uvijek nepromijenjeno, učitani broj je neažuran, dakle 1.
> 4. T_2 smanjuje pročitanu vrijednost s 1 na 0. Promjena je opet učinjena samo u radnoj memoriji.
> 5. T_1 unosi u bazu promijenjenu vrijednost za broj slobodnih sjedala. Dakle u bazi sada piše da nema slobodnih sjedala.
> 6. Slično i T_2 unosi u bazu svoju promijenjenu vrijednost za broj slobodnih sjedala. Dakle u bazu se ponovo upisuje da nema slobodnih sjedala.
> 7. Budući da je na početku svog rada naišla na broj slobodnih sjedala veći od 0, T_1 izdaje studentu kartu.
> 8. Iz istih razloga i T_2 izdaje drugom studentu kartu.
>
> Vidimo da se ovaj način stvorila jedna karta previše. Ili, drugim riječima, dva studenta morala bi sjediti na istom sjedalu u autobusu.

Ozbiljni DBMS ne smije dopustiti slijed događaja iz prethodnog primjera. Dakle, DBMS u svakom trenutku mora garantirati serijalizabilnost. U prethodnoj situaciji jedan (bilo koji) student trebao je dobiti kartu, dok je drugi trebao dobiti obavijest da više nema slobodnih mjesta. Očito je potrebna neka vrsta kontrole (koordinacije) istovremenog izvršavanja transakcija. Razni DBMS-i to rade na razne načine. Uobičajena tehnika koju među ostalima rabi i MySQL zasniva se na lokotima.

7.3.2. Lokoti i zaključavanja

Lokoti su pomoćni podaci koji služe za koordinaciju konfliktnih radnji. Baza je podijeljena na više dijelova, tako da jednom dijelu odgovara točno jedan lokot. Transakcija koja želi pristupiti nekom podatku najprije mora „uzeti“ odgovarajući lokot i time *zaključati* dotični dio baze. Čim je obavila svoju operaciju, transakcija treba „vratiti“ lokot i time *otključati* podatke. Kad transakcija naiđe na podatke koji su već zaključani, ona mora čekati dok ih prethodna transakcija ne otključa. Time se zapravo izbjegava (sasvim) istovremeni pristup istom podatku.

Ovaj mehanizam dovoljan je da otkloni probleme iz prethodnog primjera. Zaista, zamislimo da sada obje transakcije T_1 i T_2 zaključavaju podatak u bazi kojem misle pristupiti. Tada će T_1 prva zaključati broj slobodnih sjedala, a otključat će ga tek nakon što ga ažurira. T_2 će (nakon kratkog čekanja) učitati već ažuriranu vrijednost (0 slobodnih sjedala), pa drugi student neće dobiti kartu. Znači „istovremeno“ izvršavanje T_1 i T_2 sada je serijalizabilno.

Veličina dijela baze kojem je pridružen jedan lokot određuje takozvanu *zrnatost* zaključavanja (*granularity*). Što je zrno krupnije, to je kontrola zaključavanja jednostavnija za DBMS, no stupanj paralelnosti rada je manji. Zrnatost suvremenih DBMS-a je obično reda veličine n-torke ili bar fizičkog bloka.

Uporaba lokota krije u sebi i određene opasnosti. Najveća od njih je mogućnost međusobne *blokade* dviju ili više transakcija (takozvani *deadlock)*. Da bismo točnije objasnili o čemu se radi, opet ćemo se poslužiti jednim zamišljenim scenarijem vezanim uz našu studentsku ekskurziju autobusom.

> Uzmimo da za svako sjedalo u autobusu baza podataka pohranjuje ime putnika (studenta) koji sjedi na tom sjedalu. Pretpostavimo da postoji transakcija kojom dva putnika mogu zamijeniti sjedala (na primjer pušač i nepušač). Zamislimo sada da su istovremeno pokrenute dvije transakcije ovakve vrste, nazovimo ih T_1 i T_2. Neka T_1 mijenja imena putnika na sjedalima 1 i 2, a T_2 obavlja istu zamjenu ali u suprotnom redoslijedu. Tada je moguć slijedeći način obavljanja elementarnih operacija.
> 1. T_1 zaključa podatak o sjedalu 1 jer mu misli pristupiti.
> 2. Iz istih razloga T_2 zaključa podatak o sjedalu 2.
> 3. T_1 traži lokot za sjedalo 2, ali mora čekati jer je T_2 već zaključala dotični podatak.
> 4. Slično, T_2 traži lokot za sjedalo 1, ali mora čekati jer je T_1 već zaključala taj podatak.
>
> Očigledno ni T_1 ni T_2 više ne mogu nastaviti rad - one će vječno čekati jedna drugu.

DBMS koji rabi lokote mora računati s upravo opisanom mogućnošću blokade transakcija, te mora osigurati da se ta blokada spriječi ili prekine. Rješenje koje je danas najčešće u uporabi izgleda ovako.

- Privremeno se dopušta blokada, no povremeno se kontrolira ima li blokiranih transakcija (traži se ciklus u usmjerenom grafu koji prikazuje koja transakcija čeka koju). Ako takve blokirane transakcije postoje, jedna od njih se prekida, neutralizira se njezin dotadašnji učinak, te se ona se ponovo starta u nekom kasnijem trenutku.

U nastavku ovog odjeljka opisat ćemo detaljnije način uporabe lokota u MySQL-u. Ti detalji ovise o tipu datoteke koja služi za prikaz relacije.

- Kod jednostavnijih tipova datoteka, na primjer MyISAM, rabe se lokoti na razini cijele relacije. MySQL na početku izvršavanja transakcije automatski uzima lokote za sve relacije koje sudjeluju u toj transakciji. Pritom se kod svih transakcija lokoti uzimaju u istom redoslijedu, čime se izbjegava blokada.
- Kod složenijih tipova datoteki, na primjer InnoDB, rabe se lokoti na razini n-torke. Lokoti se opet uzimaju automatski, dakle bez eksplicitne naredbe u transakciji. Blokada je moguća, no ona se otklanja prekidanjem i neutralizacijom jedne transakcije.
- U slučaju InnoDB postoji i mogućnost zaključavanja ili otključavanja na razini cijele relacije. No ta mogućnost se ostvaruje jedino ako transakcija sadrži eksplicitne naredbe LOCK TABLES … odnosno UNLOCK TABLES …

7.3.3. Dvofazni protokol zaključavanja

Na osnovu do sada pokazanih primjera, moglo bi se povjerovati da uporaba lokota (uz izbjegavanje blokade) daje garanciju za serijalizabilnost. To nažalost nije točno, a pogrešni utisak stekao se zato što su primjeri bili suviše jednostavni. Naime, postoji mogućnost za nekorektno izvršavanje istovremenih transakcija čak i onda kad se podaci zaključavaju. Da bismo to pokazali, još jednom ćemo se poslužiti zamišljenim „scenarijem“ vezanim uz našu studentsku ekskurziju autobusom.

Opet promatramo transakciju kojom u autobusu dva putnika (studenta) mijenjaju sjedala. Uzmimo da su istovremeno pokrenute dvije identične transakcije, T_1 i T_2, koje obje mijenjaju imena putnika na istim sjedalima 1 i 2. Neka na početku na tim sjedalima sjede studenti Ivan i Marko. Tada je moguć sljedeći redoslijed obavljanja elementarnih operacija.

1. T_1 zaključa sjedalo 1, čita ime Ivan i pamti ga kao prvo ime, te otključa sjedalo 1.
2. T_1 zaključa sjedalo 2, čita ime Marko i pamti ga kao drugo ime, te otključa sjedalo 2.
3. T_1 ponovo zaključa sjedalo 1, upisuje mu zapamćeno drugo ime (dakle Marko), te otključa sjedalo 1.
4. T_2 zaključa sjedalo 1, čita ime Marko i pamti ga kao svoje prvo ime, te otključa sjedalo 1.
5. T_2 zaključa sjedalo 2, čita ime Marko i pamti ga kao svoje drugo ime, te otključa sjedalo 2.
6. T_1 ponovo zaključa sjedalo 2, upisuje mu zapamćeno prvo ime (dakle Ivan), te otključa sjedalo 2.
7. T_2 ponovo zaključa sjedalo 1, upisuje mu svoje zapamćeno drugo ime (dakle Marko), te otključa sjedalo 1.
8. T_2 ponovo zaključa sjedalo 2, upisuje mu svoje zapamćeno prvo ime (dakle opet Marko), te otključa sjedalo 2.

Vidimo da sada na oba sjedala sjedi Marko, a Ivana nema nigdje. Znači, opisani način izvršavanja transakcija T_1 i T_2 nije serijalizabilan. Naime, bilo koji sekvencijalni redoslijed

izvršavanja proizveo bi dvostruku zamjenu, to jest Ivan bi opet bio na sjedalu 1, a Marko na sjedalu 2.

Upravo navedeni primjer uvjerio nas je da zaključavanje podataka samo po sebi nije garancija za serijalizabilnost. No srećom, stvar se lagano može spasiti. Dovoljno je od transakcija zahtijevati da se pokoravaju jednom strožem „pravilu ponašanja“. Preciznije, može se dokazati da vrijedi sljedeća tvrdnja.

- Ako u svakoj od transakcija sva zaključavanja slijede prije prvog otključavanja, tada proizvoljno istovremeno izvršavanje tih transakcija mora biti serijalizabilno.

Navedeno pravilo zove se *dvofazni protokol zaključavanja*. Naime, zaključavanja i otključavanja se zbivaju u dvije razdvojene faze tokom izvršavanja transakcije.

Dvofazni protokol zaključavanja bit će bolje razumljiv ukoliko se vratimo prethodnom primjeru na Slici 7.11 s dvostrukom zamjenom sjedala. Razlog zašto transakcije T_1 i T_2 nisu korektno radile je baš taj što se one nisu pokoravale protokolu. Zaista, obje su otključavale podatke, pa ih zatim opet zaključavale. Da bi bila u skladu sa protokolom, transakcija mora samo jednom zaključati podatak o sjedalu (prije čitanja), te ga samo jednom otključati (nakon ažuriranja). U razdoblju između čitanja i ažuriranja podatak mora ostati zaključan. Lako se uvjeriti da, uz ovu promjenu „ponašanja“, prije opisani redoslijed događaja više nije moguć. U najmanju ruku, koraci 5 i 6 morat će zamijeniti redoslijed, jer T_2 neće moći pristupiti sjedalu 2 dok ga T_1 nije ažurirala i otključala. Obje transakcije tada će korektno obaviti svoj posao.

7.3.4. Vremenski žigovi

Dvofazni protokol zaključavanja (uz izbjegavanje blokade) predstavlja primjer tehnike za koordinaciju paralelnog izvršavanja transakcija zasnovane na lokotima. No postoje i metode koje ne rabe lokote. Kao primjer, ukratko spominjemo tehniku zasnovanu na *vremenskim žigovima* (*time stamps*).

Svakoj transakciji pridružuje se identifikacijski broj, takozvani vremenski žig. Čitanja i promjene istog podatka dozvoljavaju se samo ako se one odvijaju u redoslijedu vremenskih žigova pripadnih transakcija. U slučaju narušavanja tog redoslijeda, jedna od transakcija mora se prekinuti, neutralizirati i ponovo startati s većim vremenskim žigom. Na primjer, ako transakcija T_1 ima žig t_1, T_2 ima žig $t_2 > t_1$, T_1 želi pročitati podatak x, a T_2 je već mijenjala taj isti x, tada se T_1 mora neutralizirati.

Opisani način izvršavanja transakcija u skladu s vremenskim žigovima garantira serijalizabilnost. Naime, ukupni učinak svih transakcija je isti kao da se svaka od njih izvršavala trenutačno, u svom posebnom trenutku koji je određen vremenskim žigom.

Ako usporedimo dvije razmatrane metode za koordinaciju istovremenih transakcija, dakle dvofazni protokol zaključavanja i vremenske žigove, tada je teško reći koja metoda je bolja. Obje garantiraju serijalizabilnost, no obje stvaraju dodatno opterećenje za DBMS. Za očekivati je da će metoda s vremenskim žigovima biti efikasnija ukoliko je riječ o bazi gdje transakcije najčešće pristupaju različitim podacima i ne smetaju jedna drugoj. S druge strane, uporaba lokota se više isplati u bazi gdje postoje zajednički podaci kojima sve transakcije moraju pristupiti.

7.4. Zadaci za vježbu

Zadatak 7.1. Rješavanjem Zadatka 6.1 dobili ste fizičku shemu nadopunjene baze podataka o fakultetu. Nadogradite tu shemu tako da osigurate čuvanje integriteta za neke od stranih ključeva.

Zadatak 7.2. Rješavanjem Zadatka 6.2 ili 4.2 dobili ste fizičku shemu baze podataka o knjižnici. Nadogradite tu shemu tako da osigurate čuvanje integriteta za neke od stranih ključeva.

Zadatak 7.3. Rješavanjem Zadatka 6.3 dobili ste fizičku shemu za bazu podataka iz vašeg područja interesa. Nadogradite tu shemu tako da osigurate čuvanje integriteta za neke od stranih ključeva.

Zadatak 7.4. Razmislite koje sve vrste (skupine) korisnika bi mogle rabiti bazu podataka o fakultetu. Precizno definirajte ovlaštenja za tipičnog korisnika iz svake skupine. Napišite odgovarajuće naredbe GRANT i REVOKE, te eventualne naredbe CREATE VIEW. Služite se sintaksom MySQL.

Zadatak 7.5. Razmislite koje sve vrste (skupine) korisnika bi mogle rabiti bazu podataka iz vašeg područja interesa koju ste dobili rješavanjem Zadatka 6.3. Precizno definirajte ovlaštenja za tipičnog korisnika iz svake skupine. Napišite odgovarajuće naredbe GRANT i REVOKE, te eventualne naredbe CREATE VIEW. Služite se sintaksom MySQL.

Prilozi

U prethodnim poglavljima opisali smo postupak oblikovanja baze podataka, te smo ga ilustrirali na studijskom primjeru baze podataka o fakultetu. U ovim prilozima isti postupak provodimo kroz još dva studijska primjera, a to su baza podataka o bolnici, odnosno baza podataka o znanstvenoj konferenciji.

P.1. Oblikovanje baze podataka o bolnici

U ovom studijskom primjeru bavimo se pojednostavnjenom i donekle neobičnom bolnicom. Oblikovanje kreće od specifikacije koja je dobivena utvrđivanjem i analizom zahtjeva, a zatim se odvija u tri uobičajene faze: Konceptualno, logičko, odnosno fizičko oblikovanje.

P.1.1. Specifikacija za bolnicu

Utvrđivanjem i analizom zahtjeva dobili smo sljedeću specifikaciju. Ona govori o bolnici, njezinim prostorijama, liječnicima, osoblju i pacijentima, te o odgovarajućim odnosima i pravilima.

> **Pacijenti koji zauzimaju sobe.** Pacijent se obično smješta u bolničku sobu prilikom dolaska u bolnicu. Svaka soba može primiti mnogo pacijenata. Konzultanti (stariji kirurzi) bolnice smiju imati i svoje privatne pacijente, koji su smješteni u jednokrevetnim privatnim sobama. Informacije koje treba pamtiti o pacijentu uključuju osobni identifikacijski broj (OIB), prezime, ime, adresu i tako dalje.
>
> **Medicinske sestre zadužene za sobe.** Sestra može ili ne mora biti zadužena za sobu. Pritom jedna sestra može biti zadužena najviše za jednu sobu, no za istu sobu može biti zaduženo više sestara. Sestra je jednoznačno određena svojim OIB-om.
>
> **Kirurške operacije koje se obavljaju nad pacijentima.** Nad istim pacijentom može se obaviti više kirurških operacija. Informacije o jednoj operaciji su: tip operacije, pacijent, kirurg, datum, vrijeme i mjesto.
>
> **Kirurzi koji obavljaju operacije.** Jednu operaciju obavlja samo jedan kirurg, a za ostale prisutne kirurge se smatra da oni asistiraju pri operaciji. Kirurge nadgledaju stariji kirurzi, takozvani konzultanti, koji također mogu obavljati operacije ili asistirati. Informacije o jednom kirurgu su: OIB, prezime i ime, adresa, broj telefona, i tako dalje. Svaki konzultant ima svoju specijalnost.
>
> **Operacijske sale u kojima se odvijaju operacije.** Jedna operacija se odvija samo u jednoj sali, no ista sala može biti poprište mnogih operacija. Svaka sala ima svoju identifikacijsku oznaku. Neke sale su specijalno opremljene za neke vrste operacija.
>
> **Medicinske sestre zadužene za sale.** Sestra može ili ne mora biti zadužena za salu, no ne može biti zadužena za više od jedne sale. Za jedno salu može biti zaduženo više sestara.

P.1.2 . Konceptualna shema za bolnicu

Početni dio oblikovanja baze podataka o bolnici je konceptualno oblikovanje. Čitanjem prethodne specifikacije otkrivamo elemente od kojih se sastoji konceptualna shema naše baze, dakle entitete, veze i atribute. Konceptualnu shemu oblikujemo crtanjem reduciranog Chen-ovog dijagrama entiteta i veza, te sastavljanjem dodatnog teksta koji prati dijagram. Dobiveni dijagram prikazan je na Slici P.1, a pripadni popratni tekst nalazi se na Slici P.2.

Iz Slike P.1 vidljivo je od kojih se sve tipova entiteta i veza sastoji naša shema, a zbog upisanih kardinalnosti zadane su i funkcionalnosti veza, te obaveznost članstva entiteta u vezama. Slike P.1 određuje popis atributa za pojedini tip entiteta odnosno vezu.

P.1.3. Relacijska shema i rječnik podataka za bolnicu

Nastavak oblikovanja baze podataka o bolnici je logičko oblikovanje. Služeći se uobičajenim pravilima, konceptualnu shemu sa Slike P.1 i Slike P.2 pretvaramo u relacijsku shemu, dakle skup relacija od kojih svaka ima zadano ime, atribute i primarni ključ. Također usput sastavljamo rječnik podataka, dakle popis svih atributa s objašnjenjem njihovog tipa i značenja. Dobivena relacijska shema prikazana je na Slici P.3, a rječnik podataka na Slici P.4.

Vidimo da je svaki tip entiteta iz konceptualne sheme prikazan jednom relacijom u logičkoj shemi. S druge strane, prikaz veze zavisi o njezinoj funkcionalnosti, te o obaveznosti članstva njezinih entiteta.

- Veza ZAUZIMA prikazana je stranim ključem ID SOBE u relaciji PACIJENT, budući da u njoj PACIJENT ima skoro obavezno članstvo.
- Slično, veza LIJEČI prikazana je stranim ključem OIB KONZULTANTA u relaciji PRIVATNI PACIJENT.
- Također, strani ključevi OIB KIRURGA, OIB PACIJENTA i ID SALE u relaciji OPERACIJA prikazuju veze OBAVLJA, PODVRGAVA SE odnosno ODVIJA SE, za koje OPERACIJA ima obavezno članstvo.
- Veza ASISTIRA je zbog svoje funkcionalnosti M:M morala biti prikazana posebnom relacijom koja sadrži ključne atribute od KIRURG i OPERACIJA zajedno s dodatnim atributom ULOGA.
- 1:M veze ZADUŽENA ZA SOBU odnosno ZADUŽENA ZA SALU su zbog neobaveznosti članstva prikazane posebnim relacijama. Te relacije sadrže ključne atribute odgovarajućih entiteta i dodatni atribut DATUM ZADUŽIVANJA. Ako bi većina sestara bile zadužene za sobe, tada bi možda bilo bolje vezu ZADUŽENA ZA SOBU prikazati stranim ključem ID SOBE unutar relacije SESTRA, no tada bi u istu relaciju morali ugurati i dodatni atribut DATUM ZADUŽIVANJA.
- 1:M veza NADGLEDA je zbog neobaveznosti članstva prikazana posebnom relacijom. Mogla je biti prikazana i ubacivanjem OIB KONZULTANTA u relaciju KIRURG, no tada bi ubačeni atribut bio prazan za sve kirurge koje nitko ne nadgleda.

Primijetimo da je dobivena relacijska shema već u četvrtoj normalnoj formi, tako da nije potrebno provoditi dodatni postupak normalizacije. To je zato što je polazna konceptualna shema bila zdravo oblikovana.

P.1.4. Fizička shema za bolnicu

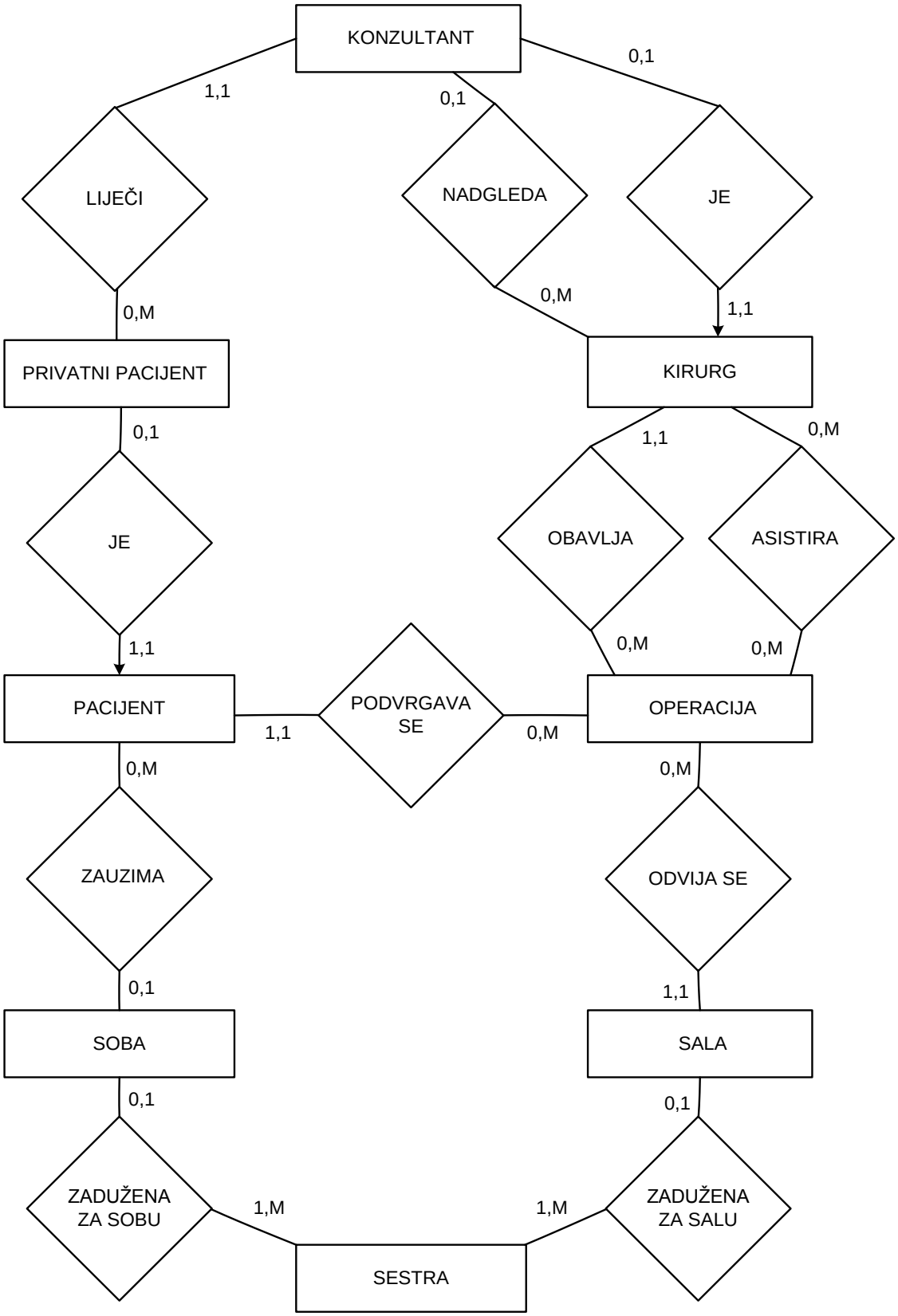
Zadnji dio oblikovanja baze podataka o bolnici predstavlja fizičko oblikovanje. Relacijsku shemu naše baze pretvaramo u fizičku shemu tako da građu svake relacije sa Slike P.3 opišemo odgovarajućom SQL naredbom CREATE TABLE. Pritom tipove atributa određujemo u skladu s rječnikom podataka sa Slike P.4. Ukoliko se služimo sintaksom iz MySQL, tada dobivena fizička shema izgleda kao na Slikama P.5 i P.6.

Tekst sa Slike P.5 i P.6 treba smatrati početnom verzijom fizičke sheme koju je moguće dalje dotjerivati. Ta početna verzija osigurava integritet domena za atribute u onoj mjeri koliko to dopuštaju mogućnosti zadavanja tipova u MySQL-u. Također, osigurana je jedinstvenost vrijednosti primarnog ključa unutar svake relacije.

Primijetimo da shema sa Slika P.5 i P.6 za sada ne osigurava referencijalni integritet, dakle konzistentnu uporabu vrijednosti za strane ključeve. Naime, u našoj bazi postoji vrlo velik broj stranih ključeva:

- ID SOBE u relaciji PACIJENT,
- OIB KONZULTANTA u relaciji PRIVATNI PACIJENT,
- OIB KIRURGA, ID SALE, i OIB PACIJENTA u relaciji OPERACIJA,
- ID OPERACIJE, i OIB KIRURGA u relaciji ASISTIRA,
- OIB NADGLEDANOG, i OIB KONZULTANTA u relaciji NADGLEDA,
- ID SOBE u relaciji ZADUŽENA ZA SOBU,
- ID SALE u relaciji ZADUŽENA ZA SALU.

Automatska provjera svih ovih ključeva ne dolazi u obzir jer bi to previše zakompliciralo fizičku građu baze, te degradiralo njezine performanse. Ipak, neke važnije provjere mogle bi se implementirati uvođenjem sekundarnih indeksa i klauzulama FOREIGN KEY.



Slika P.1: Dijagram s entitetima i vezama za bazu podataka o bolnici.

.

Tip entiteta KIRURG ima atribute:
OIB, PREZIME, IME, ADRESA, BROJ TELEFONA.

Tip entiteta KONZULTANT je pod-tip od KIRURG, ima dodatni atribut:
SPECIJALNOST (grana kirurgije u kojoj se specijalizirao).

Tip entiteta PACIJENT ima atribute:
OIB, PREZIME, IME, ADRESA, DATUM ROĐENJA, SPOL.

Tip entiteta PRIVATNI PACIJENT JE POD-tip od PACIJENT, ima dodatni atribut:
ID PRIVATNE SOBE.

Tip entiteta SESTRA ima atribute:
OIB, PREZIME, IME, STRUČNI STUPANJ.

Tip entiteta SOBA (misli se na ne-privatnu sobu) ima atribute:
ID SOBE, TIP SOBE, BROJ KREVETA.

Tip entiteta SALA ima atribute:
ID SALE, TIP SALE.

Tip entiteta OPERACIJA ima atribute:
ID OPERACIJE, TIP OPERACIJE, DATUM, VRIJEME.

Veza ASISTIRA ima atribut:
ULOGA (kirurga u operaciji).

Veza ZADUŽENA ZA SOBU ima atribut:
DATUM ZADUŽIVANJA.

Veza ZADUŽENA ZA SALU ima atribut:
DATUM ZADUŽIVANJA.

Ostale veze nemaju atribute.

Slika P.2: Popratni tekst uz dijagram sa Slike P.1.

KIRURG (OIB, PREZIME, IME, ADRESA, BROJ TELEFONA)

KONZULTANT (OIB, SPECIJALNOST).

PACIJENT (OIB, PREZIME, IME, ID SOBE, ADRESA, DATUM ROĐENJA, SPOL)

PRIVATNI PACIJENT (OIB, OIB KONZULTANTA, ID PRIVATNE SOBE)

SESTRA (OIB, PREZIME, IME, STRUČNI STUPANJ)

SOBA (ID SOBE, TIP SOBE, BROJ KREVETA)

SALA (ID SALE, TIP SALE)

OPERACIJA (ID OPERACIJE, OIB KIRURGA, ID SALE, OIB PACIJENTA,
TIP OPERACIJE, DATUM, VRIJEME)

ASISTIRA (ID OPERACIJE, OIB KIRURGA, ULOGA)

NADGLEDA (OIB NADGLEDANOG, OIB KONZULTANTA)

ZADUŽENA ZA SOBU (OIB SESTRE, ID SOBE, DATUM ZADUŽIVANJA)

ZADUŽENA ZA SALU (OIB SESTRE, ID SALE, DATUM ZADUŽIVANJA)

Slika P.3: Relacijska shema za bazu podataka o bolnici.

IME ATRIBUTA	TIP	OPIS
OIB	Niz od točno 11 znamenki	Šifra koja jednoznačno određuje osobu
PREZIME	Niz znakova	Prezime osobe
IME	Niz znakova	Ime osobe
ADRESA	Niz znakova	Ulica, kućni broj, poštanski broj, grad, država
BROJ TELEFONA	Niz znamenki	Pozivni broj zemlje, grada ili mreže, broj unutar mreže
SPECIJALNOST	Niz znakova	Naziv grane u kojoj se kirurg specijalizirao
ID (PRIVATNE) SOBE	Kratki niz znakova	Šifra koja jednoznačno određuje bolničku sobu
DATUM	Datum	Dan, mjesec i godina kad se nešto dogodilo
VRIJEME	Vrijeme	Sat i minuta kad se nešto dogodilo
SPOL	„muško“ ili „žensko“	Oznaka spola osobe
STRUČNI STUPANJ	Kratki niz znakova	Oznaka stručnog stupnja sestre
TIP SOBE	Kratki niz znakova	Oznaka tipa bolničke sobe
BROJ KREVETA	Cijeli broj	Broj koliko kreveta ima u bolničkoj sobi
ID SALE	Kratki niz znakova	Šifra koja jednoznačno određuje operacijsku salu
TIP SALE	Kratki niz znakova	Oznaka tipa operacijske sale
ID OPERACIJE	Niz znamenki	Šifra koja jednoznačno određuje operaciju
TIP OPERACIJE	Kratki niz znakova	Oznaka tipa operacije
ULOGA	Niz znakova	Opis uloge kirurga u operaciji

Slika P.4: Rječnik podataka za bazu podataka o bolnici.

```
CREATE TABLE KIRURG
    (OIB NUMERIC(11) UNSIGNED NOT NULL,
     PREZIME CHAR(20),
     IME CHAR(20),
     ADRESA CHAR(80),
     BROJ_TELEFONA NUMERIC(15) UNSIGNED,
     PRIMARY KEY (OIB))
     ENGINE = INNODB;

CREATE TABLE KONZULTANT
    (OIB NUMERIC(11) UNSIGNED NOT NULL,
     SPECIJALNOST CHAR(40),
     PRIMARY KEY (OIB))
     ENGINE = INNODB;

CREATE TABLE PACIJENT
    (OIB NUMERIC(11) UNSIGNED NOT NULL,
     PREZIME CHAR(20),
     IME CHAR(20),
     ID_SOBE CHAR(5),
     ADRESA CHAR(80),
     DATUM_RODJENJA DATE,
     SPOL ENUM('M', 'Z'),
     PRIMARY KEY (OIB))
     ENGINE = INNODB;

CREATE TABLE PRIVATNI_PACIJENT
    (OIB NUMERIC(11) UNSIGNED NOT NULL,
     OIB_KONZULTANTA NUMERIC(11) UNSIGNED NOT NULL,
     ID_PRIVATNE_SOBE CHAR(5),
     PRIMARY KEY (OIB))
     ENGINE = INNODB;

CREATE TABLE SESTRA
    (OIB NUMERIC(11) UNSIGNED NOT NULL,
     PREZIME CHAR(20),
     IME CHAR(20),
     STRUCNI_STUPANJ ENUM('SSS', 'VSHS', 'VSS'),
     PRIMARY KEY (OIB))
     ENGINE = INNODB;

CREATE TABLE SOBA
    (ID_SOBE CHAR(5) NOT NULL,
     TIP_SOBE CHAR(20),
     BROJ_KREVETA NUMERIC(2) UNSIGNED,
     PRIMARY KEY (ID_SOBE))
     ENGINE = INNODB;
```

Slika P.5: Fizička shema za bazu podataka o bolnici (prvi dio).

```
CREATE TABLE SALA
    (ID_SALE CHAR(5) NOT NULL,
     TIP_SALE CHAR(20),
     PRIMARY KEY (ID_SALE))
     ENGINE = INNODB;

CREATE TABLE OPERACIJA
    (ID_OPERACIJE NUMERIC(7) UNSIGNED NOT NULL,
     OIB_KIRURGA NUMERIC(11) UNSIGNED NOT NULL,
     ID_SALE CHAR(5) NOT NULL,
     OIB_PACIJENTA NUMERIC(11) UNSIGNED NOT NULL,
     TIP_OPERACIJE CHAR(20),
     DATUM DATE,
     VRIJEME TIME,
     PRIMARY KEY (ID_OPERACIJE))
     ENGINE = INNODB;

CREATE TABLE ASISTIRA
    (ID_OPERACIJE NUMERIC(7) UNSIGNED NOT NULL,
     OIB_KIRURGA NUMERIC(11) UNSIGNED NOT NULL,
     ULOGA CHAR(40),
     PRIMARY KEY (ID_OPERACIJE, OIB_KIRURGA))
     ENGINE = INNODB;

CREATE TABLE NADGLEDA
    (OIB_NADGLEDANOG NUMERIC(11) UNSIGNED NOT NULL,
     OIB_KONZULTANTA NUMERIC(11) UNSIGNED NOT NULL,
     PRIMARY KEY (OIB_NADGLEDANOG))
     ENGINE = INNODB;

CREATE TABLE ZADUZENA_ZA_SOBU
    (OIB_SESTRE NUMERIC(11) UNSIGNED NOT NULL,
     ID_SOBE CHAR(5) NOT NULL,
     DATUM_ZADUZIVANJA DATE,
     PRIMARY KEY (OIB_SESTRE))
     ENGINE = INNODB;

CREATE TABLE ZADUZENA_ZA_SALU
    (OIB_SESTRE NUMERIC(11) UNSIGNED NOT NULL,
     ID_SALE CHAR(5) NOT NULL,
     DATUM_ZADUZIVANJA DATE,
     PRIMARY KEY (OIB_SESTRE))
     ENGINE = INNODB;
```

Slika P.6: Fizička shema za bazu podataka o bolnici (drugi dio).

P2. Oblikovanje baze podataka o znanstvenoj konferenciji

U ovom studijskom primjeru želimo oblikovati bazu podataka koja će služiti kao podrška organizatorima neke znanstvene konferencije. Kao i prethodnim primjerima, postupak kreće od specifikacije, te se nastavlja kroz faze konceptualnog, logičkog, odnosno fizičkog oblikovanja.

P.2.1. Specifikacija za znanstvenu konferenciju

Utvrđivanjem i analizom zahtjeva dobili smo sljedeću specifikaciju. Ona govori o znanstvenoj konferenciji, njezinim organizatorima i sudionicima, te znanstvenim radovima koji će se izlagati na sjednicama. Opisani su postupci vezani uz pripremu, organizaciju i odvijanje konferencije.

> **Općenito o konferenciji.** Computing Conference 2011 (kratica CC 2011) omogućuje prezentaciju novih rezultata u računarskim znanostima. Organizatori upućuju svoj poziv za sudjelovanje raznim fakultetima, institutima i kompanijama. Kao odgovor stiže nekoliko stotina radova (članaka). Recenzenti pregledavaju pristigle radove. Zbog ograničenja trajanja, samo 120 radova bit će zaista prihvaćeno za prezentaciju na CC 2011. Svaki rad se svrstava u jednu od tema konferencije.
>
> **Raspored tema i sjednica.** CC 2011 traje 4 dana, a svaki dan radi se 8 sati. Obrađuje se 8 tema (na primjer Umjetna inteligencija, Baze podataka, Računalna grafika, Softversko inženjerstvo, i tako dalje). Prezentacije se održavaju u dva paralelna toka (dvije dvorane istovremeno). Svaka sjednica traje 2 sata i posvećena je točno jednoj temi. Svaka tema ima dakle 4 sjednice.
>
> **Postupak recenziranja.** Recenzente imenuje organizacijski odbor konferencije. Svaki recenzent je poznati stručnjak za određenu temu konferencije i sam je odgovoran za recenziranje svih pristiglih radova koji su svrstani u njegovu temu. Organizatori žele da sve teme budu podjednako zastupljene, zato će biti prihvaćeno najviše 15 radova po temi.
>
> **Evidencija sudionika.** Očekuje se da će na CC 2011 sudjelovati nekoliko tisuća ljudi. O svakom sudioniku treba pamtiti nekoliko osobnih podataka (na primjer, titula, prezime i ime, radno mjesto, poštanska adresa, e-mail adresa, …). Također, sudionik se treba odlučiti na kojim sve sjednicama misli prisustvovati. Zadnja informacija služi za računanje kotizacije.
>
> **Računanje kotizacije.** Standardna kotizacija za jednu sjednicu je 50 EUR. Popusti su sljedeći: 20% za sudionika koji je i autor rada prihvaćenog za prezentaciju; 30% za autora koji će također i prezentirati rad, 40% za sudionika koji je ujedno i predsjedavajući neke od sjednica.
>
> **Sudjelovanje na sjednicama.** Svaki sudionik može prijaviti prisustvovanje na po volji mnogo sjednica, pod uvjetom da se te sjednice vremenski ne preklapaju. Sudionici mogu mijenjati svoje polazne prijave, dodavanjem novih sjednica ili

odustajanjem od njih. No sudionikove prijave dva tjedna prije početka CC 2011 smatraju se konačnima.

Novčane doznake. Da bi namirio svoju kotizaciju, sudionik šalje organizatorima jednu ili više novčanih doznaka. Ukoliko sudionik preplati kotizaciju, organizatori mu vraćaju preplaćeni iznos u obliku jedne doznake.

Baza podataka treba čuvati sve relevantne podatke o potencijalnim sudionicima, radovima, sjednicama, temama, recenzentima i doznakama.

P.2.2 . Konceptualna shema za znanstvenu konferenciju

U skladu s pravilima konceptualnog oblikovanja, čitamo prethodnu specifikaciju, te otkrivamo entitete, veze i atribute od kojih se sastoji konceptualna shema naše baze. Tu konceptualnu shemu opet dokumentiramo u obliku reduciranog Chen-ovog dijagrama s popratnim tekstom. Dijagram je prikazan je na Slici P.7, a pripadni popratni tekst je na Slici P.8.

Iz Slike P.7 vidljivi su tipovi entiteta, veze, te kardinalnosti veza. Posredno se vide i funkcionalnosti veza, te obaveznost članstva entiteta u vezama. Slike P.8 daje popis atributa za pojedini tip entiteta odnosno vezu.

P.2.3. Relacijska shema i rječnik podataka za znanstvenu konferenciju

U prvom dijelu logičkog oblikovanja, na osnovu konceptualne sheme sa Slika P.7 i P.8 izravno dolazimo do početne verzije relacijske sheme i do rječnika podataka. Dobivena relacijska shema prikazana je na Slici P.9, a sastoji se od skupa relacija gdje svaka od njih ima zadano ime, atribute i primarni ključ. Rječnik podataka vidljiv je na Slici P.10 i on detaljnije objašnjava tip i značenje za svaki atribut.

Uočavamo da je svaki tip entiteta iz konceptualne sheme prikazan jednom relacijom u dobivenoj logičkoj shemi. S druge strane, način prikaza veze zavisi o njezinoj funkcionalnosti, te o obaveznosti članstva njezinih entiteta.

- Veze JE AUTOR i PRISUTAN NA su veze s funkcionalnošću M:M, pa su zato prikazane posebnim relacijama AUTORSTVO odnosno PRISUSTVO.
- Veza PRIHVAĆEN ZA prikazana je posebnom relacijom PRIHVAĆANJE. Alternativno rješenje, zasnovano na umetanju stranog ključa OZNAKA SJEDNICE u relaciju RAD, nije pogodno zato jer većina radova neće biti prihvaćena za prezentaciju na konferenciji.
- Veza ODGOVORAN ZA prikazana je pomoću stranog ključa OZNAKA TEME u relaciji RECENZENT. Čini se da je to bolje rješenje nego da smo u relaciju TEMA stavili strani ključ E-MAIL ADRESA RECENZENTA. Naime, prvo se zadaju teme, a onda se za svaku od njih traži recenzent.
- Veza POSLAO već je implicitno prikazana time što se u relaciji DOZNAKA pojavljuje E-MAIL ADRESA SUDIONIKA (pošiljatelja doznake).
- Ostale veze imaju funkcionalnost 1:M, s time da odgovarajući tip entiteta ima obavezno članstvo. Zato se te veze prikazuju ubacivanjem stranog ključa u pripadnu relaciju.

U nastavku logičkog oblikovanja bavimo se normalizacijom. Za svaku relaciju iz sheme sa Slike P.9 provjeravamo je li ona u dovoljno visokoj normalnoj formi, te treba li je prevesti u višu normalnu formu. Zaključujemo sljedeće.

- Relacije RAD, TEMA, DOZNAKA, AUTORSTVO, PRIHVAĆANJE i PRISUSTVO su očito u 4NF pa ih ne treba mijenjati.
- U relaciji SJEDNICA kombinacije atributa (DATUM, VRIJEME OD, OZNAKA DVORANE) odnosno (DATUM, VRIJEME DO, OZNAKA DVORANE) čine kandidate za ključ. No mi smo ipak uveli OZNAKU SJEDNICE kao spretniju kraticu.
- Primijetimo da u relaciji SJEDNICA postoji funkcionalna ovisnost VRIJEME OD → VRIJEME DO ili VRIJEME DO → VRIJEME OD. Zato, strogo govoreći, SJEDNICA nije u BCNF, čak ni u 3NF. No razbijanje te relacije na manje ne bi imalo smisla. Naime, atribute DATUM, VRIJEME OD, VRIJEME DO, OZNAKA DVORANE upisujemo (zbog udobnosti) uvijek zajedno s OZNAKOM SJEDNICE. Ne može doći do anomalija koje su inače prisutne kod relacija koje nisu u 3NF. Zato relaciju SJEDNICA ostavljamo u sadašnjem obliku.
- Smatramo da NAZIV USTANOVE u relaciji SUDIONIK odnosno RECENZENT ne određuje POŠTANSKU ADRESU. Naime, ista ustanova može biti raspoređena na više adresa. Nas ovdje zanima adresa na kojoj je dotična osoba, a ne matična adresa cijele ustanove. Zbog toga ovdje nije riječ o tranzitivnoj ovisnosti, pa je relacija RECENZENT u 4NF.
- U relaciji SUDIONIK ipak postoji jedna druga tranzitivna ovisnost: E-MAIL ADRESA → STATUS → IZNOS KOTIZACIJE. Zbog toga SUDIONIK nije u 3NF, pa tu relaciju moramo normalizirati. Postupak normalizacije svodi se na razbijanje relacije SUDIONIK na dvije, čime nastaje nova relacija koju možemo zvati TARIFA.

Nova verzija relacijske sheme koja nastaje normalizacijom prikazana je na Slici P.11. U odnosu na prethodnu verziju sa Slike P.9, u novoj verziji postoje samo dvije razlike:

- relacija SUDIONIK ima jednostavniju građu,
- pojavila se nova relacija TARIFA.

Shema sa Slike P.11 je u dovoljnoj mjeri normalizirana, naime sve njezine relacije su u 4NF.

Primijetimo da je odstupanje od 4NF u shemi sa Slike P.9 nastupilo zbog propusta u oblikovanju entiteta i veza. Naime, trebalo je uočiti da postoji tip entiteta TARIFA koji govori da bilo koji sudionik s određenim statusom plaća istu kotizaciju. Također, trebalo je uočiti da postoji M:1 veza između SUDIONIKA i TARIFE koja određuje po kojoj tarifi dotični sudionik plaća kotizaciju. Da smo to sve uvažili otpočetka, postupak oblikovanja relacijske sheme odmah bi nam dao shemu u 4NF i nikakav daljnji postupak normalizacije ne bi bio potreban.

P.2.4. Fizička shema za znanstvenu konferenciju

U sklopu fizičkog oblikovanja, normaliziranu relacijsku shemu naše baze za znanstvenu konferenciju pretvaramo u fizičku shemu. To radimo tako da građu svake relacije sa Slike P.11 opišemo odgovarajućom SQL naredbom CREATE TABLE. Pritom tipove atributa nastojimo što bolje uskladiti s rječnikom podataka sa Slike P.10. Dobivena fizička shema prikazana je na Slikama P.12 i P.13. Koristili smo se sintaksom MySQL-a.

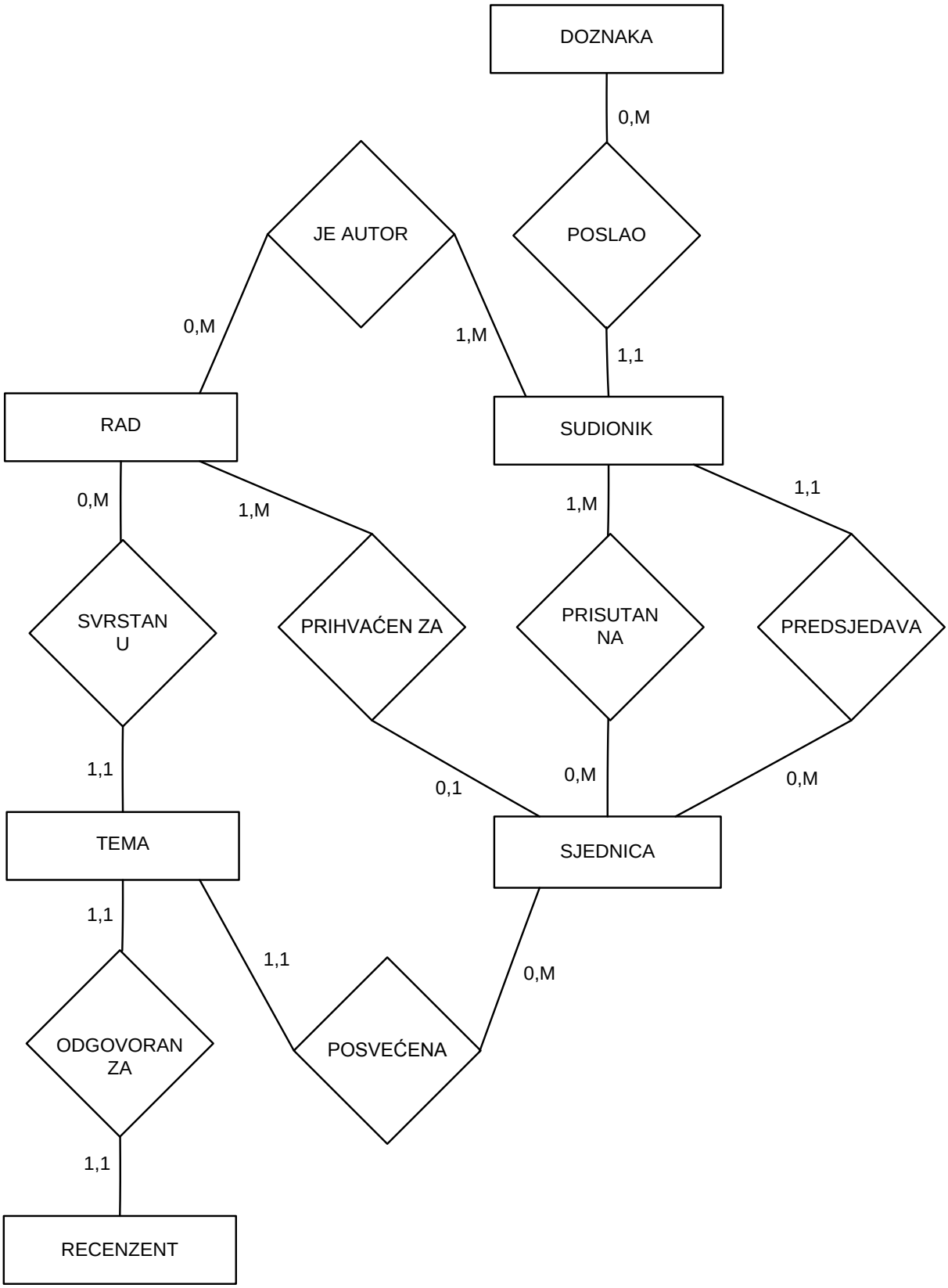
Kao i u prošlim primjerima, tekst sa Slika P.12 i P.13 tek je početna verzija fizičke sheme koju je moguće dalje dotjerivati. Naime, ta početna verzija uglavnom osigurava integritet

domena za atribute i jedinstvenost vrijednosti primarnih ključeva, no ne osigurava referencijalni integritet za strane ključeve.

Primijetimo da u našoj bazi za znanstvenu konferenciju postoji velik broj stranih ključeva:

- STATUS u relaciji SUDIONIK,
- OZNAKA TEME u relaciji RECENZENT,
- E-MAIL ADRESA SUDIONIKA u relaciji DOZNAKA.
- OZNAKA TEME i E-MAIL ADRESA PREDSJEDAVAJUĆEG u relaciji SJEDNICA,
- BROJ RADA i E-MAIL ADRESA AUTORA u relaciji AUTORSTVO,
- BROJ RADA i OZNAKA SJEDNICE u relaciji PRIHVAĆANJE,
- E-MAIL ADRESA SUDIONIKA i OZNAKA SJEDNICE u relaciji PRISUSTVO.

Automatska provjera referencijalnog integriteta za neke od tih ključeva mogla bi se po potrebi implementirati uvođenjem sekundarnih indeksa i klauzulama FOREIGN KEY. To bi zahtijevalo nadopunu nekih od naredbi CREATE TABLE sa slika P.12 odnosno P.13.



Slika P.7: Dijagram entiteta i veza za bazu podataka o znanstvenoj konferenciji.

Tip entiteta RAD ima atribute:
BROJ RADA, NASLOV RADA, BROJ STRANICA.

Tip entiteta SUDIONIK ima atribute:
E-MAIL ADRESA, TITULA, PREZIME, IME, NAZIV USTANOVE,
POŠTANSKA ADRESA, STATUS (obični, autor, izlagač, predsjednik),
IZNOS KOTIZACIJE (po sjednici).

Tip entiteta RECENZENT ima atribute:
E-MAIL ADRESA, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA.

Tip entiteta TEMA ima atribute:
OZNAKA TEME, NAZIV TEME, OPIS TEME.

Tip entiteta DOZNAKA ima atribute:
E-MAIL ADRESA SUDIONIKA, DATUM, PLAĆENI IZNOS
(jedan sudionik u jednom danu može imati samo jednu doznaku).

Tip entiteta SJEDNICA ima atribute:
OZNAKA SJEDNICE (pon1, pon2, … čet8), DATUM, VRIJEME OD,
VRIJEME DO, OZNAKA DVORANE (gdje se održava).

Ni jedna veza nema atribute veze.

Slika P.8: Popratni tekst uz dijagram sa Slike P.7.

RAD (BROJ RADA, NASLOV RADA, BROJ STRANICA)

SUDIONIK (E-MAIL ADRESA, TITULA, PREZIME, IME, NAZIV USTANOVE,
POŠTANSKA ADRESA, STATUS, IZNOS KOTIZACIJE).

RECENZENT (E-MAIL ADRESA, TITULA, PREZIME, IME, NAZIV USTANOVE,
POŠTANSKA ADRESA, OZNAKA TEME)

TEMA (OZNAKA TEME, NAZIV TEME, OPIS TEME)

DOZNAKA (E-MAIL ADRESA SUDIONIKA, DATUM, PLAĆENI IZNOS)

SJEDNICA (OZNAKA SJEDNICE, DATUM, VRIJEME OD,
VRIJEME DO, OZNAKA DVORANE, OZNAKA TEME,
E-MAIL ADRESA PREDSJEDAVAJUĆEG)

AUTORSTVO (BROJ RADA, E-MAIL ADRESA AUTORA)

PRIHVAĆANJE (BROJ RADA, OZNAKA SJEDNICE)

PRISUSTVO (E-MAIL ADRESA SUDIONIKA, OZNAKA SJEDNICE)

Slika P.9: Početna verzija relacijske sheme za bazu podataka
o znanstvenoj konferenciji.

IME ATRIBUTA	TIP	OPIS
BROJ RADA	Cijeli broj	Šifra koja jednoznačno određuje rad
NASLOV RADA	Niz znakova	Naslov koji piše na radu
BROJ STRANICA	Mali cijeli broj	Broj koliko rad ima stranica
E-MAIL ADRESA	Niz znakova	Rabi se kao šifra koja jednoznačno određuje osobu
TITULA	„Prof.dr.sc“, „Doc.dr.sc“ i sl	Jedna od uobičajenih kratica za akademski ili stručni naziv
PREZIME	Niz znakova	Prezime osobe
IME	Niz znakova	Ime osobe
NAZIV USTANOVE	Niz znakova	Jednoznačno određuje ustanovu gdje radi osoba
POŠTANSKA ADRESA	Niz znakova	Adresa osobe u ustanovi gdje radi: ulica, kućni broj, poštanski broj, grad, država
STATUS	„obični“, „autor“, „izlagač“ ili „predsjedavajući“	Status sudionika koji određuje kolika će mu biti kotizacija
IZNOS KOTIZACIJE	Mali cijeli broj	Iznos kotizacije u eurima koji sudionik mora platiti za svaku sjednicu gdje je prisutan
OZNAKA TEME	Kratki niz znakova	Kratka šifra koja jednoznačno određuje temu
NAZIV TEME	Niz znakova	Puni naziv teme
OPIS TEME	Niz znakova	Opširnije obrazloženje što spada a što ne spada u određenu temu
DATUM	Datum	Dan, mjesec i godina kad se nešto događa
PLAĆENI IZNOS	Mali cijeli broj	Novčani iznos u eurima koji je uplaćen preko doznake
OZNAKA SJEDNICE	„pon1“, „pon2“, … „čet8“	Šifra koja jednoznačno određuje dan i termin održavanja sjednice
VRIJEME (OD ili DO)	Vrijeme	Sat i minuta početka odnosno kraja sjednice
OZNAKA DVORANE	Kratki niz znakova	Šifra koja jednoznačno određuje dvoranu

Slika P.10: Rječnik podataka za bazu podataka o znanstvenoj konferenciji.

RAD (BROJ RADA, NASLOV RADA, BROJ STRANICA)

SUDIONIK (E-MAIL ADRESA, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA,
STATUS).

TARIFA (STATUS, IZNOS KOTIZACIJE)

RECENZENT (E-MAIL ADRESA, TITULA, PREZIME, IME, NAZIV USTANOVE,
POŠTANSKA ADRESA, OZNAKA TEME)

TEMA (OZNAKA TEME, NAZIV TEME, OPIS TEME)

DOZNAKA (E-MAIL ADRESA SUDIONIKA, DATUM, PLAĆENI IZNOS)

SJEDNICA (OZNAKA SJEDNICE, DATUM, VRIJEME OD,
VRIJEME DO, OZNAKA DVORANE, OZNAKA TEME,
E-MAIL ADRESA PREDSJEDAVAJUĆEG)

AUTORSTVO (BROJ RADA, E-MAIL ADRESA AUTORA)

PRIHVAĆANJE (BROJ RADA, OZNAKA SJEDNICE)

PRISUSTVO (E-MAIL ADRESA SUDIONIKA, OZNAKA SJEDNICE)

Slika P.11: Normalizirana verzija relacijske sheme za bazu podataka o znanstvenoj konferenciji.

```
CREATE TABLE RAD
     (BROJ_RADA NUMERIC(4) UNSIGNED NOT NULL,
      NASLOV_RADA CHAR(160),
      BROJ_STRANICA NUMERIC(2) UNSIGNED,
      PRIMARY KEY (BROJ_RADA))
      ENGINE = INNODB;

CREATE TABLE SUDIONIK
     (E_MAIL_ADRESA CHAR(40) NOT NULL,
      TITULA ENUM('Prof.dr.sc', 'Doc.dr.sc', 'Dr.sc','Mr.sc'),
      PREZIME CHAR(20),
      IME  CHAR(20),
      NAZIV_USTANOVE CHAR(40),
      POSTANSKA_ADRESA CHAR(80),
      STATUS ENUM('O', 'A', 'I', 'P'),
      PRIMARY KEY (E_MAIL_ADRESA))
      ENGINE = INNODB;

CREATE TABLE TARIFA
     (STATUS ENUM('O', 'A', 'I', 'P') NOT NULL,
      IZNOS_KOTIZACIJE NUMERIC(3) UNSIGNED,
      PRIMARY KEY (STATUS))
      ENGINE = INNODB;

CREATE TABLE RECENZENT
     (E_MAIL_ADRESA CHAR(40) NOT NULL,
      TITULA  ENUM('Prof.dr.sc', 'Doc.dr.sc', 'Dr.sc','Mr.sc'),
      PREZIME CHAR(20),
      IME CHAR(20),
      NAZIV_USTANOVE CHAR(40),
      POSTANSKA_ADRESA CHAR(80),
      OZNAKA_TEME CHAR(3) NOT NULL,
      PRIMARY KEY (E_MAIL_ADRESA))
      ENGINE = INNODB;

CREATE TABLE TEMA
     (OZNAKA_TEME CHAR(3) NOT NULL,
      NAZIV_TEME CHAR(40),
      OPIS_TEME CHAR(160),
      PRIMARY KEY (OZNAKA_TEME))
      ENGINE = INNODB;
```

Slika P.12: Fizička shema za bazu o znanstvenoj konferenciji (prvi dio).

```
CREATE TABLE DOZNAKA
    (E_MAIL_ADRESA_SUDIONIKA CHAR(40) NOT NULL,
     DATUM DATE NOT NULL,
     PLACENI_IZNOS NUMERIC(4) UNSIGNED,
     PRIMARY KEY (E_MAIL_ADRESA_SUDIONIKA, DATUM))
     ENGINE = INNODB;

CREATE TABLE SJEDNICA
    (OZNAKA_SJEDNICE ENUM ('pon1', 'pon2', 'pon3', 'pon4', 'pon5',
          'pon6', 'pon7', 'pon8', 'uto1','uto2', 'uto3', 'uto4', 'uto5', 'uto6',
          'uto7', 'uto8', 'sri1', 'sri2', 'sri3', 'sri4', 'sri5', 'sri6',  'sri7', 'sri8',
          'cet1', 'cet2', 'cet3', 'cet4', 'cet5', 'cet6', 'cet7', 'cet8')  NOT NULL,
     DATUM DATE,
     VRIJEME_OD TIME,
     VRIJEME_DO TIME,
     OZNAKA_DVORANE CHAR(4),
     OZNAKA_TEME CHAR(3) NOT NULL,
     E_MAIL_ADRESA_PREDSJEDAVAJUCEG  CHAR(40)  NOT NULL,
     PRIMARY KEY (OZNAKA_SJEDNICE))
     ENGINE = INNODB;

CREATE TABLE AUTORSTVO
    (BROJ_RADA NUMERIC(4) UNSIGNED NOT NULL,
     E_MAIL_ADRESA_AUTORA CHAR(40) NOT NULL,
     PRIMARY KEY (BROJ_RADA, E_MAIL_ADRESA_AUTORA))
     ENGINE = INNODB;

CREATE TABLE PRIHVACANJE
    (BROJ_RADA NUMERIC(4) UNSIGNED NOT NULL,
     OZNAKA_SJEDNICE ENUM ('pon1', 'pon2', 'pon3', 'pon4', 'pon5',
          'pon6', 'pon7', 'pon8', 'uto1','uto2', 'uto3', 'uto4', 'uto5', 'uto6',
          'uto7', 'uto8', 'sri1', 'sri2', 'sri3', 'sri4', 'sri5', 'sri6',  'sri7', 'sri8',
          'cet1', 'cet2', 'cet3', 'cet4', 'cet5', 'cet6', 'cet7', 'cet8')  NOT NULL,
     PRIMARY KEY (BROJ_RADA))
     ENGINE = INNODB;

CREATE TABLE PRISUSTVO
    (E_MAIL_ADRESA_SUDIONIKA CHAR(40) NOT NULL,
     OZNAKA_SJEDNICE ENUM ('pon1', 'pon2', 'pon3', 'pon4', 'pon5',
          'pon6', 'pon7', 'pon8', 'uto1','uto2', 'uto3', 'uto4', 'uto5', 'uto6',
          'uto7', 'uto8', 'sri1', 'sri2', 'sri3', 'sri4', 'sri5', 'sri6',  'sri7', 'sri8',
          'cet1', 'cet2', 'cet3', 'cet4', 'cet5', 'cet6', 'cet7', 'cet8')   NOT NULL,
     PRIMARY KEY (E_MAIL_ADRESA_SUDIONIKA, OZNAKA_SJEDNICE))
     ENGINE = INNODB;
```

Slika P.13: Fizička shema za bazu o znanstvenoj konferenciji (drugi dio).

Literatura

Općeniti udžbenici o bazama podataka:
- C.J. Date: An Introduction to Database Systems, 8th Edition. Addison-Wesley, Reading MA, 2003.
- R. Elmasri, S. Navathe: Fundamentals of Database Systems, 6th Edition. Addison-Wesley, Reading MA, 2010.
- R. Ramakrishnan, J. Gehrke: Database Management Systems, 3rd Edition, McGraw- Hill, New York, 2002.
- A. Silberschatz, H.F. Korth, S. Sudarshan: Database System Concepts, 6th Edition. McGraw-Hill, New York, 2010.
- M. Varga: Baze podataka – konceptualno, logičko i fizičko modeliranje podataka, DRIP, Zagreb, 1994.

Udžbenici o oblikovanju baza podataka:
- C. Churcher: Beginning Database Design - From Novice to Professional. Apress, Berkley CA, 2007.
- M.J. Hernandez: Database Design for Mere Mortals - A Hands-On Guide to Relational Database Design. 2nd Edition. Addison-Wesley, Reading MA, 2003.
- R. Stephens: Beginning Database Design Solutions. Wrox, Hoboken NJ, 2008.

Priručnici za jezik SQL:
- A. Beaulieu: Learning SQL. O'Reilly Media Inc, Sebastopol CA, 2009.
- A. Molinaro: SQL Cookbook. O'Reilly Media Inc, Sebastopol CA, 2005.
- R.F. Van der Lans: Introduction to SQL. 4th Edition, Addison-Wesley, Upper Saddle River NJ, 2006.

Dokumentacija za MySQL:
- P. DuBois: MySQL. 4th Edition. Addison-Wesley, Upper Saddle River NJ, 2008.
- M. Widenius, D. Axmark: MySQL Reference Manual. O'Reilly & Associates, Sebastopol CA, 2002.
- On-line dokumentacija : http://www.mysql.com/documentation/