Osnove projektiranja baza podataka

D310

OBRAZOVNI PROGRAMI srca

srce
Sveučilište u Zagrebu
Sveučilišni računski centar

Ovu inačicu priručnika izradio je autorski tim Srca u sastavu:

Autor: Robert Manger (dorade: Nikolina Čorkalo)

Urednica: Sabina Rako (dorade: Gorana Urukalo Čorkalo)

Lektorica: Jasna Novak Milić

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

edu@srce.hr

ISBN 978-953-382-030-9

Verzija priručnika D310-20250303

Sadržaj

Uvod

Trajanje uvoda: 15 min.

Baza podataka može biti dio aplikacije ili samostalni resurs koji podržava različite aplikacije. 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.

Projektiranje baze podataka predstavlja ključni dio njezina razvoja. Cilj projektiranja je oblikovati bazu podataka koja zadovoljava utvrđene potrebe korisnika i podataka. Ta građa u pravilu ne bi smjela biti optimizirana za jednu određenu aplikaciju, već bi trebala odražavati smisao i unutrašnju povezanost samih podataka. Time bi baza dugoročno trebala biti pogodna za promjene i evoluciju u skladu sa zahtjevima budućih aplikacija.

Uobičajeni postupak projektiranja baze podataka sastoji se od tri faze: projektiranje na konceptualnoj, logičkoj i na fizičkoj razini. 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 logičku 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 kao rezultat daje naredbe u jeziku SQL kojima se realizira potrebna fizička građa baze s pomoćnim strukturama za pretraživanje i čuvanje integriteta odnosno sigurnosti podataka.

Cilj tečaja D310 je upoznavanje polaznika s postupkom projektiranja baza podataka. Tijekom tečaja prate se sve tri faze projektiranja na jednom odabranom studijskom primjeru. Također, svaki polaznik ima priliku projektirati svoju vlastitu bazu iz područja koje sam odabere. Za dokumentaciju i realizaciju baze koriste se standardni uredski paketi, alati za crtanje dijagrama i sustav za upravljanje bazom podataka – interpreter SQL-a. Predznanje koje se očekuje od polaznika je osnovno znanje o bazama podataka i osnove jezika SQL, koje su obuhvaćene tečajem D301.

Tečaj D310 traje pet dana, s time da se svakog dana održavaju četiri školska sata. Ovaj priručnik slijedi tijek tečaja pa je podijeljen u pet poglavlja, koja otprilike odgovaraju danima, odnosno 19 potpoglavlja, koja otprilike odgovaraju školskim satima. Prvo poglavlje sadrži uvod u baze podataka, dakle ponavljanje znanja koja bi polaznici već trebali imati s prethodnih tečajeva. Drugo poglavlje obrađuje projektiranje na konceptualnoj razini, dakle izradu sheme entiteta, atributa i veza za zamišljenu bazu podataka. Treće poglavlje pokriva prvi dio projektiranja na logičkoj razini i rezultira relacijskom shemom baze. U četvrtom poglavlju bavimo se nastavkom projektiranja na logičkoj razini, dakle normalizacijom relacijske sheme dobivene metodama trećeg poglavlja. Posljednje peto poglavlje odnosi se na projektiranje na fizičkoj razini, dakle na izradu koda u SQL-u kojim se realizira fizička građa baze.

Ovaj priručnik sadrži velik broj primjera i zadataka koji prate i ilustriraju obrađeno gradivo. Kroz većinu poglavlja provlači se spomenuti odabrani studijski primjer. Također, u prilogu se nalazi cjelovita projektna dokumentacija za dodatna dva studijska primjera. Na kraju svakog poglavlja nalaze se zadaci za vježbu. Neki od tih zadataka su samostalni,

neki se odnose na studijske primjere, a neki na bazu podataka iz polaznikova područja zanimanja.

1. dan
3:15
Predavanja:
3 x 45 min.
Vježbe:
1 x 45 min.
Pauza:
1x15 min.

1. Uvod u projektiranje baza podataka

Po završetku ovog poglavlja moći ćete:

- *definirati osnovne pojmove vezane uz baze podataka i sustav za upravljanje bazama podataka*
- *analizirati faze razvojnog ciklusa baze podataka*
- *analizirati važnost projektne dokumentacije za baze podataka*
- *objasniti predloške za izradu dokumentacije baze podataka.*

U ovom tečaju bavimo se problematikom trajnog pohranjivanja većih količina podataka u vanjskoj memoriji računala. To je izuzetno važna problematika: 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 i podržana je u svim programskim jezicima. Na primjer, program razvijen u jeziku COBOL ili C može stvoriti datoteku na disku i u nju upisati podatke, ili može otvoriti postojeću datoteku i iz nje pročitati podatke. Sličan pristup prisutan je i u drugim jezicima poput Pythona, Jave ili Go-a. Unatoč tome, rad s datotekama i osnovne funkcionalnosti za pohranu podataka danas su često nedostatni za zahtjeve aplikacija koje moraju upravljati velikim količinama strukturiranih i nestrukturiranih podataka.

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

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* (engleski *Data Base Management System* – DBMS). 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 (DBMS) je poslužitelj (server) baze podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom. Također, on u ime klijenata obavlja sve operacije s podacima. Dalje, on je u stanju podržati razne baze, od kojih svaka može imati svoju logičku strukturu, ali u skladu s istim *modelom*. Isto tako, brine se za sigurnost podataka i automatizira administrativne poslove s bazom.

Slično kao i operacijski sustav, DBMS spada u temeljni softver koji većina korisnika i organizacija ne razvija samostalno već ga nabavlja kao gotov proizvod. DBMS je ključna komponenta informacijskih sustava koja omogućuje učinkovito upravljanje, pohranu i pristup podacima. Danas postoji svega nekoliko važnih i široko zastupljenih DBMS-a:

- **DB2.** Proizvod tvrtke IBM, namijenjen prvenstveno velikim *mainframe* računalima. Posebno je prilagođen potrebama velikih korporacija koje zahtijevaju skalabilnost, pouzdanost i integraciju s ostalim IBM tehnologijama.
- **Oracle Database.** Proizvod istoimene tvrtke, pokriva gotovo sve računalne platforme, uključujući UNIX, Linux i MS Windows. Poznat je po svojoj fleksibilnosti, naprednim sigurnosnim značajkama i širokoj podršci za različite poslovne aplikacije.
- **MS SQL Server.** Microsoftov proizvod, namijenjen poslužiteljskim računalima s operacijskim sustavima MS Windows. Često se koristi u poslovnim okruženjima zbog jednostavne integracije s drugim Microsoft alatima kao što su Excel, Power BI i Azure.
- **MySQL.** Popularan je na mnogim platformama, posebno u kontekstu web-aplikacija i sustava otvorenog koda. Radi na širokom rasponu operacijskih sustava, uključujući Linux, Windows i macOS, te podržava različite procesorske arhitekture (npr. x86, x64, ARM). Osim toga, MySQL se često koristi u kombinaciji s alatima i tehnologijama kao što su PHP, Python, i Java, što ga čini izvrsnim izborom za LAMP (Linux, Apache, MySQL, PHP) aplikacije. Također, MySQL je popularan na cloud platformama kao što su Amazon RDS, Google Cloud SQL i Azure Database for MySQL, gdje se koristi za razvoj i hosting aplikacija. Svi ovi proizvodi, osim osnovnih DBMS funkcionalnosti, dolaze i s dodatnim alatima za razvoj aplikacija, administraciju baza podataka, sigurnosno upravljanje, analitiku i optimizaciju performansi. Također, uključuju alate za izradu i povrat sigurnosnih kopija, koji omogućuju korisnicima da redovito čuvaju podatke te brzo i jednostavno vrate baze podataka u slučaju tehničkog kvara, gubitka podataka ili cyber napada. Takvi alati omogućuju korisnicima i administratorima da lako razvijaju i održavaju sustave temeljene na bazama podataka, pružajući visoku učinkovitost i pouzdanost u radu.

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 predstavlja osnovu za projektiranje

i implementaciju baze. Točnije, podaci u bazi moraju biti logički organizirani u skladu s modelom koji podržava odabrani DBMS.

Dosadašnji DBMS-i podržavaju razne modele podataka, a najvažnijih su:

- **Relacijski model.** Zasnovan je na matematičkom pojmu *relacije*. I podaci i veze među podacima prikazuju se tablicama koje se sastoje od redaka (zapisa) i stupaca (atributa).
- **Mrežni model.** Baza je prikazana kao mreža koja se sastoji od *čvorova* i usmjerenih *lukova*. Čvorovi predstavljaju tipove zapisa (slogova podataka), a lukovi definiraju veze među njima
- **Hijerarhijski model.** Poseban slučaj mrežnog modela. Baza je predstavljena kao jedno *stablo* (hijerarhija) ili skup stabala. Svako stablo sastoji se od čvorova i veza „nadređeni-podređeni“ između čvorova. Čvorovi su tipovi zapisa, a hijerarhijske veze prikazuju odnose među njima.
- **Objektni model.** Inspiriran je objektno-orijentiranim programskim jezicima. Baza je predstavljena kao skup trajno pohranjenih *objekata* koji se sastoje od svojih internih „atributa“ (podataka) i „metoda“ (operacija) za rukovanje tim podacima. Svaki objekt pripada klasi, a između klasa se uspostavljaju veze nasljeđivanja, agregacije i druge vrste veza.
- **NoSQL modeli.** Namijenjeni za rad s nestrukturiranim (poput slika i videa) i polustrukturiranim podacima (poput JSON ili XML formata), omogućujući veću fleksibilnost od relacijskog modela. Nestrukturirani podaci uključuju slike, videozapise, e-mailove ili tekstualne dokumente koji nemaju jasno definiranu shemu, dok su polustrukturirani podaci organizirani u fleksibilne formate kao što su JSON ili XML, gdje zapisi mogu imati različite atribute. NoSQL baze podržavaju i strukturirane podatke, često bez stroge definicije sheme, što omogućuje veću prilagodljivost.

Hijerarhijski i mrežni model bili su u upotrebi 60-ih i 70-ih godina 20. stoljeća. Od 80-ih godina pa do danas prevladava relacijski model. Iako se očekivao prijelaz na objektni model, većina današnjih baza podataka i dalje se temelji na relacijskom modelu.

Relacijski model ostaje srž suvremenih informacijskih sustava, pružajući standardiziran, učinkovit i pouzdan način organizacije i upravljanja podacima. Međutim, rast popularnosti NoSQL i graf baza podataka pokazuje kako različiti pristupi i modeli imaju svoje mjesto u današnjem raznolikom ekosustavu baza podataka.

1.1.3. Ciljevi koji se nastoje postići uporabom baza podataka

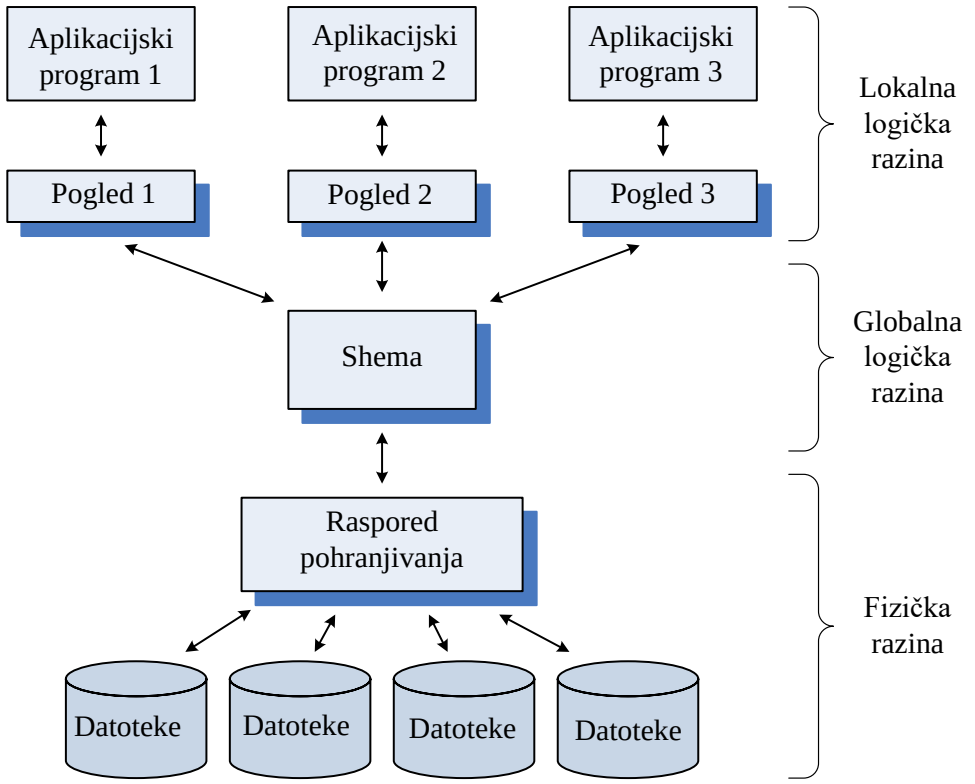
Spomenuli smo da baze podataka predstavljaju višu razinu rada s podacima u odnosu na klasične programske jezike. Ta viša razina očituje se u nastojanju tehnologije baza podataka da ispuni sljedeće ciljeve:

- **Fizička nezavisnost podataka**. Logička definicija baze odvojena je od njezine fizičke građe. Dakle, 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.** Globalna logička definicija baze odvojena je od lokalne definicije za pojedine aplikacije. Dakle, 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 samo onim redoslijedom koji je bio predviđen u vrijeme projektiranja i implementiranja baze. Danas se podrazumijeva da korisnik može slobodno prebirati po podacima i po svojem nahođenju uspostavljati veze među podacima. Tom zahtjevu zaista udovoljavaju samo relacijske baze.
- **Istovremeni pristup do podataka.** Baza mora omogućiti da se veći broj korisnika istovremeno koristi istim podacima. Pritom ti korisnici ne smiju ometati jedan drugoga, a svaki od njih treba imati dojam da sam radi s bazom.
- **Čuvanje integriteta.** Nastoji se automatski sačuvati korektnost i konzistencija podataka, čak i u slučaju grešaka u aplikacijama ili konflikata kod istovremenih aktivnosti korisnika.
- **Mogućnost oporavka nakon kvara.** Mora postojati pouzdana zaštita baze u slučaju kvara hardvera ili grešaka u radu sistemskog softvera, uključujući vraćanje podataka iz sigurnosnih kopija.
- **Zaštita od neovlaštenog korištenja.** Mora postojati mogućnost da se korisnicima ograniče prava pristupa bazi, tj. svakom korisniku moraju se regulirati ovlaštenja, što 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 i izborom pogodnih algoritama za pretraživanje.
- **Mogućnost podešavanja i kontrole.** Velike baze zahtijevaju stalnu brigu: praćenje performansi, mijenjanje parametara u fizičkoj građi, rutinsko pohranjivanje sigurnosnih kopija, reguliranje ovlaštenja korisnika. Također, budući da se svrha baze s vremenom mijenja, potrebno je povremeno podesiti i logičku strukturu. Ovakvi poslovi moraju se obavljati centralizirano. Odgovorna osoba zove se administrator baze podataka, kojemu na raspolaganju stoje odgovarajući 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.



Slika 1.1: Arhitektura baze podataka

- **Fizička razina** odnosi se na fizički prikaz i raspored podataka na jedinicama vanjske memorije. To je aspekt koji vide samo sistemski programeri (oni koji su razvili DBMS). Sama fizička razina može se dalje podijeliti na više podrazina 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 koji vidi projektant baze odnosno njezin administrator. Opis globalne logičke definicije naziva se *shema* (eng. *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 korištenog modela. Također, shema može uvesti i ograničenja kojima se čuva integritet podataka.
- **Lokalna logička razina** odnosi se na logičku predodžbu o dijelu baze kojim se koristi pojedina aplikacija. To je aspekt koji vidi korisnik ili aplikacijski programer. Opis jedne lokalne logičke definicije zove se *pogled* (eng. *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 korištenog modela. Također, pogled u svojoj konačnoj realizaciji zadaje i način na koji se iz globalnih podataka i veza izvode lokalni podaci.

Primijetimo da se fizička neovisnost podataka spomenuta u Odjeljku 1.1.3 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. Tako 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 potreban raspored pohranjivanja i fizičku bazu. Projektant odnosno administrator može samo donekle utjecati na fizičku građu baze, podešavanjem njemu dostupnih parametara.

Programi i korisnici fizičkoj bazi ne pristupaju izravno, 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 ove kategorije.

- **Jezik za opis podataka** (eng. *Data Description Language* – DDL). Služi projektantu baze ili administratoru za zapisivanje sheme ili pogleda. Tim se jezikom definiraju podaci i veze među podacima 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** (eng. *Data Manipulation Language* – DML). Služi programeru za uspostavljanje veze između aplikacijskog programa i baze. Naredbe DML omogućuju „manevriranje“ po bazi i jednostavne operacije kao što su upis, promjena, brisanje ili čitanje zapisa. U nekim softverskim paketima DML je zapravo biblioteka potprograma: „naredba“ u DML-u svodi se na poziv potprograma. U drugim paketima zaista se radi o posebnom jeziku: programer tada piše program u kojem su izmiješane naredbe dvaju jezik, pa takav program treba prevoditi pomoću dvaju prevoditelja (DML-*precompiler*, obični *compiler*).
- **Jezik za postavljanje upita** (eng. *Query Language* – QL). Služi neposrednom korisniku za interaktivno pretraživanje baze. To je jezik koji podsjeća na govorni (engleski) jezik. Naredbe su neproceduralne, dakle takve da samo specificiraju rezultat koji želimo dobiti, a ne i postupak za dobivanje rezultata.

Takva 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 koristiti interaktivno (preko *on-line interpretera*) ili se može pojavljivati uklopljen u aplikacijske programe. Svi DBMS-i spomenuti u odjeljku 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. Ti jezici su nužni da bi stvorili bazu i povezali se s njom, no oni 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, itd.) s ugniježđenim DML-naredbama. Tijekom 80-ih godina 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 bio je u njihovoj nestandardnosti: svaki od njih u pravilu je bio dio nekog određenog softverskog paketa za baze podataka te se nije mogao koristiti izvan tog paketa (baze).

U današnje vrijeme aplikacije se najčešće razvijaju u standardnim *objektno-orijentiranim* programskim jezicima (Java, C++, C#, itd.) uz korištenje ORM alata kao što su Hibernate, Entity Framework ili SQLAlchemy, što omogućuje jednostavnije mapiranje modela objekata na tablice u bazi. Sve češće se koriste NoSQL baze (MongoDB, Firebase) koje imaju vlastite jezike za upite prilagođene specifičnostima tih sustava. Moderni razvoj sve više uključuje i alate za brzo razvijanje aplikacija (low-code/ no-code platforme) koje omogućuju jednostavno upravljanje podacima bez predznanja programiranja.

1.2. Razvojni ciklus baze podataka

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, 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 proizvoda. No u slučaju baza podataka taj ciklus ima neke svoje specifičnosti. Od navedenih pet aktivnosti, u ovom nas tečaju zanima samo jedna, a to je projektiranje. Ipak, zbog cjelovitosti izlaganja, u ovom potpoglavlju ukratko ćemo opisati sve aktivnosti.

1.2.1. Utvrđivanje i analiza zahtjeva

Kako 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 skupine korisnika, pojavit će se razna tumačenja značenja i svrhe pojedinih podataka i razni načini njihove uporabe. Analiza zahtjeva treba pomiriti te razlike, tako da se eliminiraju 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, jer 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, jer ujedno definira i najvažnije transakcije s podacima, a često i cijele aplikacije.

1.2.2. Projektiranje (na konceptualnoj, logičkoj i fizičkoj razini)

Cilj projektiranja 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, projektiranje predlaže način kako da se podaci na pogodan način grupiraju, strukturiraju i međusobno povežu. Glavni rezultat projektiranja trebala bi biti shema cijele baze, oblikovana u skladu s pravilima korištenog modela podataka i zapisana tako da ju korišteni DBMS može razumjeti i realizirati. Kod većih baza rezultat projektiranja mogu također biti i pogledi (pod-sheme) za potrebe pojedinih važnijih aplikacija.

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

- **Projektiranje na konceptualnoj razini.** Glavni rezultat prve faze projektiranja 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 lako razumljiv ljudima, no još je nedovoljno razrađen da bi omogućio izravnu implementaciju.
  - Na primjer, u sustavu za upravljanje podacima knjižnice, konceptualna shema može uključivati sljedeće entitete i njihove atribute:
    1. Entitet: Knjiga

       Ovaj entitet predstavlja knjige koje knjižnica posjeduje.

       Atributi:

       - Naslov: Naslov knjige.
       - Autor: Ime i prezime autora knjige.
       - Godina izdanja: Godina kada je knjiga izdana.
       - ISBN: Jedinstveni identifikator za knjigu.

    2. Entitet: Član

       Ovaj entitet predstavlja osobe koje su članovi knjižnice.

       Atributi:

       - Ime člana: Ime osobe.
       - Prezime člana: Prezime osobe.
       - Članski broj: Jedinstveni identifikator za člana.
       - Adresa: Adresa stanovanja člana.

    3. Entitet: Posudba (veza između Knjiga i Član)

       Ovaj entitet/veza prati posudbe knjiga članova knjižnice.

       Atributi:

- Datum posudbe: Datum kada je knjiga posuđena.
- Datum vraćanja: Datum kada je knjiga vraćena.
- Status: Trenutačni status posudbe (npr. posuđeno, vraćeno).

Konceptualna shema često se izrađuje pomoću dijagrama entitet-veza (ERD), koristeći alate poput draw.io, MySQL Workbench, ili slične alate za modeliranje podataka. Ovi dijagrami koriste jednostavne oblike, poput pravokutnika za entitete, elipsa za atribute i rombova za veze, kako bi prikazali odnose između podataka.

Takav intuitivan prikaz omogućuje sudionicima projekta da lako razumiju strukturu baze podataka, no on još uvijek nije dovoljno detaljan za izravnu implementaciju. U kasnijim fazama projektiranja, konceptualna shema pretvara se u logičku i fizičku shemu, koje su prilagođene specifičnostima DBMS-a koji će se koristiti.

- **Projektiranje na logičkoj razini.** Kao glavni rezultat druge faze projektiranja nastaje logička shema, koja je u slučaju relacijskog modela sastavljena od relacija (tablica).Tablice su međusobno povezane pomoću primarnih i stranih ključeva. Na primjer, informacije o knjigama ne smiju biti duplicirane u tablici posudbi, već se povezuju preko stranog ključa ISBN. Sastavni dio projektiranja na logičkoj razini 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. Logička shema detaljno razrađuje podatke s konceptualne razine. U slučaju knjižnice, logička shema može uključivati relacije poput:
    - Knjiga: (ISBN, Naslov, Autor, GodinaIzdavanja)
    - Član: (ČlanID, Ime, Prezime, Adresa)
    - Posudba: (PosudbaID, ČlanID, ISBN, DatumPosudbe, DatumVraćanja)

  Proces normalizacije koristi se za uklanjanje suvišnih podataka i smanjenje mogućnosti anomalija.
- **Projektiranje na fizičkoj razini.** Glavni rezultat treće faze projektiranja je *fizička shema* cijele baze, dakle opis njezine fizičke građe. U slučaju korištenja DBMS-a zasnovanog na jeziku SQL, pojam „fizička razina“ 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. Fizička razina definira način na koji se podaci pohranjuju i upravljaju na fizičkom mediju. U SQL bazama podataka, fizička shema realizira logičke tablice kao SQL naredbe za kreiranje tablica, indeksa i pogleda. Na primjer, naredba CREATE TABLE koristi se za definiranje strukture tablica u bazi podataka, uključujući nazive stupaca, njihove tipove podataka te ograničenja poput primarnih i stranih ključeva.
- S druge strane, naredba CREATE VIEW omogućuje stvaranje virtualnih tablica koje nisu fizički pohranjene u bazi, već služe kao unaprijed definirani upiti za specifične aplikacije. VIEW je koristan za

pojednostavljenje složenih upita ili za ograničavanje pristupa određenim podacima na način prilagođen korisniku.

Navedene tri faze projektiranja detaljno ćemo obraditi u 2. poglavlju, zatim u 3. i 4. poglavlju, te konačno u 5. poglavlju. Svi rezultati projektiranja opisuju se u odgovarajućim dokumentima, koji zajedno čine *projektnu dokumentaciju* baze podataka. O toj dokumentaciji opširnije ćemo govoriti u Potpoglavlju 1.3.

1.2.3. Implementacija

Implementacija se svodi na fizičku realizaciju projektirane 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, primjerice kao obične datoteke ili tekstualni dokumenti. Danas većina sustava za upravljanje bazama podataka (DBMS) nudi ETL alate (Extract, Transform, Load) koji omogućuju brzi prijenos podataka u bazu. Iako DBMS sustavi često dolaze s alatima koji olakšavaju prijenos podataka iz različitih izvora, ovaj proces je obično zahtjevan i teško ga je u potpunosti automatizirati. Razlog tome je potreba za čišćenjem, ispravljanjem i usklađivanjem podataka kako bi se osigurala njihova točnost i konzistentnost unutar baze.

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 udovoljava 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.

Cilj testiranja je identificirati i ispraviti pogreške nastale u prethodnim fazama razvoja baze: u analizi zahtjeva, projektiranju ili u implementaciji. Pogreške u ranijim aktivnostima imaju teže posljedice jer se provlače i kroz kasnije aktivnosti pa zahtijevaju više truda da se poprave. Na primjer, pogreška 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 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 pogrešaka u projektiranju, na primjer posljedica neuočavanja važnih veza između određenih vrsta podataka. U takvom slučaju opet slijedi popravak pogrešaka, dakle nova revizija shema, dokumenata i baze.

1.2.5. Održavanje

Održavanje baze podataka odvija se 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. Ovaj proces često se naziva i evolucijom baze podataka, jer se baze s vremenom razvijaju kako bi udovoljile novim zahtjevima.

Kao i u softverskom inženjerstvu općenito, 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 pogrešaka 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* odnosi se na prilagodbu baze novoj infrastrukturi, poput migracije na cloud platforme, prelaska na NoSQL sustave ili novom DBMS-u koji se nije koristio u vrijeme projektiranja i početne implementacije.

U današnje vrijeme održavanje uključuje i kontinuirani nadzor performansi baze, kako bi se pravovremeno otkrili i riješili problemi poput usporavanja upita, preopterećenja resursa ili sigurnosnih prijetnji. Suvremeni DBMS sustavi često nude alate za automatizaciju održavanja, uključujući automatsko kreiranja indeksa, optimizaciju upita ili prediktivno upozoravanje na potencijalne probleme.

Naglasimo da je održavanje baze nužnost na koju se treba pripremiti već tijekom njezina razvoja i uzimati je u obzir tijekom njezina cijelog 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 poteškoća, a promjene će se svoditi na povremeno ubacivanje novih podataka u postojeće relacije ili dodavanje sasvim novih relacija. Pritom te promjene neće utjecati na ispravan rad postojećih aplikacija, jer su one zaštićene svojstvima fizičke i logičke nezavisnosti opisanim u Odjeljku 1.1.3.

1.3. Dokumentacija

Svaki softverski proizvod, 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 ovom priručniku ograničit ćemo se isključivo na *projektnu* dokumentaciju za baze podataka, dakle na dokumente koji nastaju tijekom aktivnosti projektiranja baze. Istaknut ćemo važnost takve dokumentacije, prikazati nekoliko predložaka za njezinu izradu, te spomenuti alate koji se pritom obično koriste.

1.3.1. Važnost izrade dokumentacije

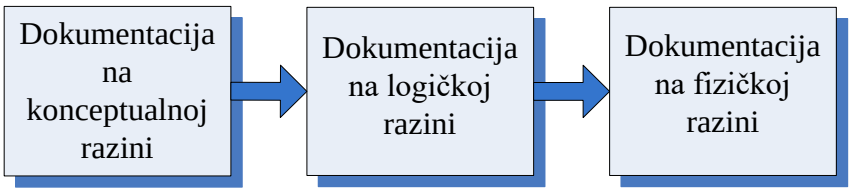
Projektna dokumentacija za bazu podataka važna je zato što ona omogućuje ispravan tijek samog projektiranja i implementacije u skladu s pravilima struke te konzistentan prelazak iz pojedine razvojne faze ili aktivnosti u drugu. Također, dokumentacija je neophodna tijekom testiranja i kasnijeg održavanja baze, jer 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, i njihovu kolektivnu memoriju.

Projektna dokumentacija za bazu podataka prati sve tri faze projektiranja, 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, tj. konceptualna i logička shema. Sva tri dijela moraju se čuvati zato što 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 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. Osoba koja namjerava izvršiti promjenu u bazi prisiljena je koristiti se dokumentacijom da bi utvrdila gdje i što treba promijeniti. Kad god dođe do promjene u građi baze, ta se promjena mora unijeti i u dokumentaciju. Štoviše, sva tri dijela dokumentacije moraju se mijenjati istovremeno tako da ostanu međusobno konzistentni i konzistentni s realiziranom bazom. Jedino pod tim uvjetom dokumentacija će ostati relevantna za daljnje održavanje.

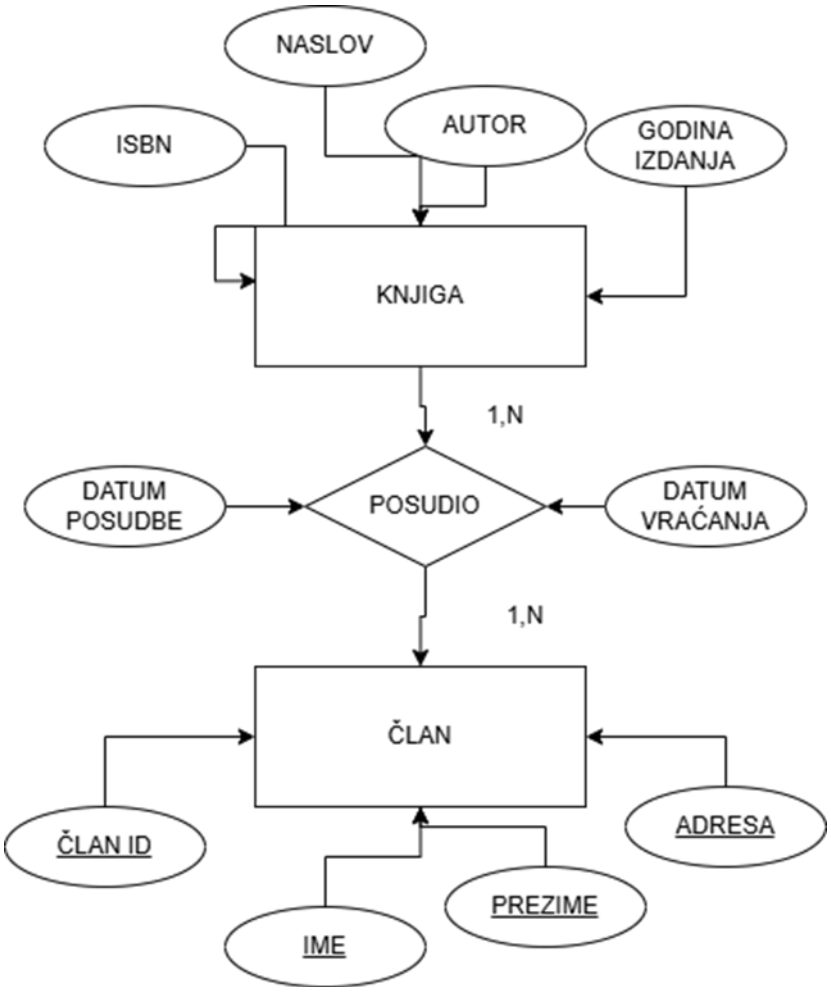
Kod manjih baza dovoljno je da projektna dokumentacija u svakom trenutku odražava ažurno stanje. No kod većih i složenijih baza korisno je da se osim ažurnog stanja dokumentira i povijest promjena. To se može realizirati tako da se sami dokumenti čuvaju u više inačica, ili tako da se u jednom dokumentu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

1.3.2. Predlošci za izradu dokumentacije

Projektna dokumentacija sastoji se od tekstovnih 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 tekstovne 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 i veze. Postoji nekoliko predložaka za prikaz konceptualne sheme, a svi se sastoje od neke vrste dijagrama, uz koji stoji više ili manje tekstovnih dopuna. Opisat ćemo tri najvažnija predloška (preostali se mogu smatrati njihovim varijacijama).

- **Izvorni Chenov 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 tekstovni elementi ubačena imena entiteta, veza i atributa te oznake takozvanih kardinalnosti veza. Prednost takvog načina prikazivanja je da je sva informacija prikazana na dijagramu. Nedostatak je u tome što dijagram može postati nepregledan i prenatrpan mjehurićima ako on sadrži mnogo atributa.
- **Reducirani Chenov dijagram.** Riječ je o pojednostavnjenoj vrsti izvornog Chenova 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 nadomješta se tekstom uz 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 uporabiti za prikaz konceptualne sheme baze tako 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 pa nema potrebe za tekstovnim nadopunama.



Slika 1.3: Izvorni Chenov dijagram

Prethodna i sljedeće tri slike prikazuju konceptualnu shemu jedne baze prikazanu na tri načina uporabom opisanih triju obrazaca. Riječ je o vrlo jednostavnoj bazi koja sadrži podatke o članovima i knjigama u knjižnici te pamti koju je knjigu posudio određeni član. Slika 1.3 prikazuje shemu u obliku izvornog Chenova dijagrama. Na Slikama 1.4 i 1.5 vidimo ekvivalentni reducirani Chenov dijagram s popratnim tekstom. Slika 1.6 daje istu informaciju u obliku UML-ova *class*-dijagrama.

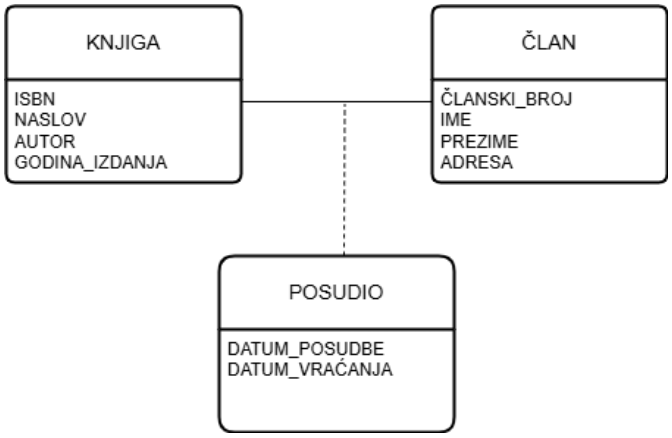
Slika 1.4: Reducirani Chenov dijagram

Tip entiteta KNJIGA ima atribute:
ISBN, NASLOV, AUTOR, GODINA_IZDANJA.

Tip entiteta ČLAN ima atribute:
ČLANSKI_BROJ, IME, PREZIME, ADRESA.

Veza POSUDBA ima atribute:
DATUM_POSUDBE, DATUM_VRAĆANJA.

Slika 1.5: Popratni tekst uz reducirani Chen-ov dijagram



Slika 1.6: UML-ov *class*-dijagram

Od tri spomenuta predloška za prikaz konceptualne sheme u nastavku ovog priručnika rabit ćemo isključivo reducirani Chenov dijagram. Naime, taj se predložak u praksi pokazuje najspretnijim jer predstavlja dobar kompromis između izražajnosti i jednostavnosti.

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 naziva i relacijska shema. Opisat ćemo dva bitno različita predloška za prikaz relacijske sheme koji su danas najčešće u uporabi.

- **Tekstovni 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 takvih 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 njihova značenja i tipa vrijednosti koji mogu poprimiti. Relacije se obično opisuju jednim redom teksta, npr. Knjiga(ISBN, Naslov, Autor, Godina izdanja). Primarni ključevi mogu se vizualno označiti, npr. podcrtavanjem Knjiga(ISBN, Naslov, Autor, Godina izdanja)**,** gdje je **ISBN** podcrtani atribut koji označava primarni ključ.

- **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-ovi kao što je 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, one dakle spajaju atribut u jednoj relaciji koji je ujedno ključ u drugoj relaciji. Uz takav dijagram opet je poželjno priložiti rječnik podataka.

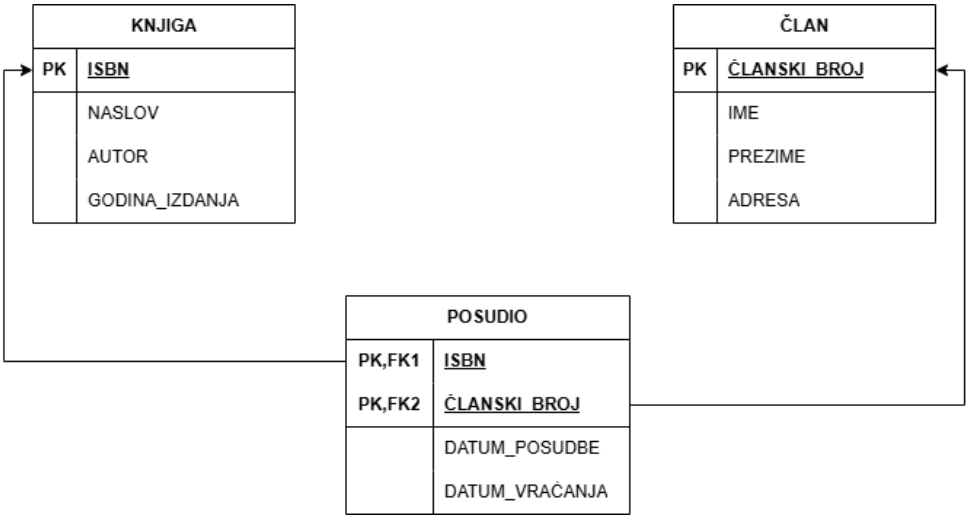
Sljedeće dvije slike prikazuju relacijsku shemu jedne baze prikazanu na dva načina, uporabom dvaju opisanih obrazaca. Riječ je ponovo o bazi o članovima, knjigama i posudbi koju smo upoznali na prethodnim slikama. Slika 1.7 sadrži tekstualni prikaz relacijske sheme, a Slika 1.8 grafički prikaz. U oba se slučaja kao nadopuna može dodati rječnik podataka koji se nalazi na Slici 1.9.

KNJIGA(ISBN, NASLOV, AUTOR, GODINA_IZDANJA)

ČLAN (ČLANSKI_BROJ, PREZIME, IME, ADRESA)

POSUDBA (ISBN, ČLANSKI BROJ, DATUM_POSUDBE, DATUM_VRAĆANJA)

Slika 1.7: Tekstualni prikaz relacijske sheme



Slika 1.8: Grafički prikaz relacijske sheme

IME PODATKA	TIP	OPIS
ISBN	Niz od točno 13 znamenki	Jedinstveni identifikator knjige
NASLOV	Niz znakova	Naslov knjige
AUTOR	Niz znakova	Ime i prezime autora knjige
GODINA IZDANJA	Cijeli broj	Godina kada je knjiga izdana
ČLANSKI BROJ	Niz od točno 10 znamenki	Jedinstveni identifikator člana knjižnice.
IME	Niz znakova	Ime člana
PREZIME	Niz znakova	Prezime člana
ADRESA	Niz znakova	Adresa stanovanja člana
DATUM POSUDBE	Datum	Datum kada je knjiga posuđena
DATUM VRAĆANJA	Datum	Predviđeni datum vraćanja posuđene knjige

Slika 1.9: Rječnik podataka kao prilog relacijskoj shemi

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

Kao što smo već prije spomenuli, 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 koji 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 i raspolaže drukčijim komandnim jezikom.

Slika 1.10 prikazuje fizičku shemu naše baze o knjigama, članovima i posudbama. U primjeru se koristi inačica sintakse SQL-a iz DBMS-a SQLServera. U slučaju korištenja drugog DBMS-a, naredbe bi se mogle donekle razlikovati.

```
CREATE TABLE KNJIGA (
  ISBN CHAR(13) NOT NULL,
  Naslov VARCHAR(100),
  Autor VARCHAR(100),
  Godina_izdanja INT,
  PRIMARY KEY (ISBN)
);
CREATE TABLE CLAN (
  Clanski_broj CHAR(10) NOT NULL,
  Ime VARCHAR(50),
  Prezime VARCHAR(50),
  Adresa VARCHAR(100),
  PRIMARY KEY (Clanski_broj)
);
CREATE TABLE POSUDBA (
  ISBN CHAR(13) NOT NULL,
  Clanski_broj CHAR(10) NOT NULL,
  Datum_posudbe DATE,
  Datum_vracanja DATE,
  PRIMARY KEY (ISBN, Clanski_broj),
  FOREIGN KEY (ISBN) REFERENCES KNJIGA(ISBN),
  FOREIGN KEY (Clanski_broj) REFERENCES CLAN(Clanski_broj)
);
```

Slika 1.10: Fizička shema zapisana u jeziku SQL

U nastavku ovog priručnika, gotovo sve fizičke sheme i naredbe u SQL-u bit će napisane u skladu sa sintaksom SQL Servera. 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 tekstovni dokument s umetnutim dijagramima, osnovni alat za izradu te dokumentacije je tekst-procesor kao što je MS Word. Ipak, za izradu dijagrama koje ćemo umetnuti u tekst potreban nam je dodatni alat, a 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 projektiranja i prilagođena je 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 projektiranje baze podataka. Kao primjer kvalitetnog CASE-alata koji je pogodan za projektiranje baza podataka navodimo Visual Paradigm. Osim svih standardnih UML-dijagrama, taj paket podržava i crtanje jedne inačice Chenovih 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 draw.io. U njemu već postoje predlošci za većinu UML-dijagrama, predložak za grafički prikaz logičke sheme baze podataka, a također, općeniti elementi poput pravokutnika, mjehurića i rombova koji mogu poslužiti za sastavljanje Chenovih dijagrama.

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

- Prednost CASE-alata je da on „razumije“ sintaksu i semantiku dijagrama. Zato on može korisniku ispravljati pogreške 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.
- Mana 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 običnim operacijama „kopiraj i zalijepi“.
- Mana općenitog alata za crtanje je da je on samo program za uređivanje slika koji ne razumije smisao dijagrama za baze podataka. Zato nam on ne može baš u velikoj mjeri kontrolirati ili automatizirati posao projektiranja.

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. U skladu s time, mi ćemo se u ovom tečaju uz MS Word koristiti i alatom draw.io.

1.4. Vježbe

- **Zadatak 1.1.** Postoji li u ustanovi gdje radite neka veća baza podataka? Kakve vrste podataka ona sadrži? Koje aplikacije ona podržava? Koji DBMS služi za njezinu realizaciju?
- **Zadatak 1.2.** Kako to da od 90-ih godina 20. stoljeća do danas objektne baze podataka nisu uspjele istisnuti iz upotrebe 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 1.3.** Objasnite zašto se mogućnost istovremenog rada više korisnika s istim podacima ističe kao poseban cilj koji bi tehnologija baza podataka trebala ostvariti. Opišite što bi se sve loše moglo dogoditi kad bi više korisnika nekontrolirano pristupalo istim podacima u isto vrijeme.
- **Zadatak 1.4.** Objasnite zašto se mogućnost oporavka baze nakon kvara ističe kao važan cilj koji bi tehnologija baza podataka trebala ostvariti. Što mislite, na koji način DBMS obavlja oporavak baze? Kojim se pomoćnim strukturama podataka on pritom koristi?
- **Zadatak 1.5.** Precrtajte Sliku 1.6 (UML-ov *class*-dijagram), služeći se najprije alatom MS Visio, a zatim draw.io. Koji vam se alat čini spretniji? Jeste li uočili neke prednosti ili mane jednog alata u odnosu na drugi?

Dodatni zadaci:

- **Zadatak 1.6.** Napišite specifikaciju za neku jednostavniju bazu podataka iz vašeg područja interesa.
- **Zadatak 1.7.** Pod pretpostavkom da znate programirati, napišite program u Pythonu, Javi ili u bilo kojem programskom jeziku kojim se služite. Program treba omogućiti izravno pohranjivanje podataka o studentima (JMBAG, PREZIME, IME, GODINA_STUDIJA) u datoteku, te kasnije čitanje tih podataka iz iste datoteke.
  Razmislite: 0sigurava li ovakav način pohranjivanja podataka fizičku i logičku nezavisnost podataka? Obrazložite svoj odgovor.
- **Zadatak 1.8.** Kreirajte CHENOV dijagram za jednostavniji sustav baze podataka po vašem izboru. Prikazujte entitete, njihove atribute i veze među entitetima koristeći alat draw.io. Usporedite svoj dijagram s notacijama naučenim na nastavi i navedite eventualne razlike.
- **Zadatak 1.9** Kreirajte UML dijagram za sustav upravljanja hotelom prema sljedećem opisu:
  Opis sustava:
  - Gosti hotela imaju mogućnost pretraživanja ponude hotela putem interneta i rezervacije soba. Tijekom rezervacije mogu odabrati dodatne usluge poput parkirnog mjesta ili paketa prehrane (doručak, polupansion, puni pansion). Nakon dolaska u hotel, recepcionar je odgovoran za provjeru rezervacija, izdavanje soba te programiranje ključeva/kartica za pristup sobama. Pri završetku boravka, recepcionar naplaćuje usluge i izdaje račun.
  - Administrator sustava ima ovlasti za upravljanje podacima o sobama, dodavanje ili uklanjanje soba, ažuriranje informacija

(broj kreveta, dodatni sadržaji itd.), upravljanje parkirnim mjestima te uređivanje cjenika hotela.

- **Zadatak 1.10** Kreirajte CHEN-ov dijagram za sustav narudžbe i plaćanja u web trgovini prema sljedećem opisu:
    - Opis sustava: Web trgovina omogućuje kupovinu registriranim i neregistriranim korisnicima. Korisnici biraju artikle koje žele kupiti, određuju količinu i dodaju ih u košaricu. Svaka stavka u košarici predstavlja artikl s nazivom, opisom i jediničnom cijenom. Korisnici mogu mijenjati količinu artikala u košarici, uklanjati artikle ili pregledavati ukupnu cijenu.
    - Nakon završetka odabira artikala, korisnik kreira narudžbu koja sadrži: ukupni iznos za plaćanje, podatke o korisniku (adresa isporuke, kontakt telefon i adresa e-pošte), datum i vrijeme narudžbe te odabrani način plaćanja (kartica ili gotovina pri dostavi). Kod plaćanja karticom korisnik unosi potrebne podatke, a plaćanje se obrađuje putem vanjskog sustava. Narudžba ima status koji može biti:
        - "u tijeku" – dok se narudžba ne potvrdi,
        - "poslana" – kada je narudžba potvrđena i čeka isporuku,
        - "isporučena" – kada je narudžba dostavljena.
- Za registrirane korisnike sustav automatski dohvaća prethodno spremljene podatke (kontakt telefon, e-pošta, adresa isporuke) koji se mogu prilagoditi prije potvrde narudžbe. Registrirani korisnici mogu pregledavati povijest narudžbi, ponavljati ili izbrisati prije potvrde, pri čemu artikli ostaju u košarici. **Zadatak 1.11.** Napišite specifikaciju za bazu podataka koja će se koristiti u sustavu upravljanja autopraonicom. Specifikacija treba sadržavati:
    - Opis sustava: Sustav treba omogućiti praćenje usluga autopraonice, evidenciju klijenata i vođenje evidencije svih obavljenih pranja vozila.
    - Entiteti i atributi:
        - Entitet Klijent: ID_klijenta, Ime, Prezime, Kontakt, Broj_registracije.
        - Entitet Usluga: ID_usluge, Naziv_usluge, Opis, Cijena.
        - Entitet Pranje: ID_pranja, Datum, Vrijeme, ID_klijenta, ID_usluge.

Odnosi: Svaki klijent može koristiti više usluga autopraonice, dok se svako pranje povezuje s jednim klijentom i jednom uslugom.

2. Projektiranje na konceptualnoj razini

2. dan

3:15
Predavanja: 3 x 45 min.
Vježbe: 1 x 45 min.
Pauza: 1x15 min.

Po završetku ovog poglavlja moći ćete:

- *razlikovati entitete, atribute i veze unutar tekstualne specifikacije*
- *opisati funkcionalnost, obaveznost članstva i kardinalnost veza između entiteta*
- *prikazati složenije veze između entiteta koristeći odgovarajuće dijagram*
- *primijeniti pravila za crtanje složenijih veza na dijagramima.*

U ovom poglavlju opisujemo prvu fazu projektiranja baze podataka, a to je projektiranje na konceptualnoj razini. Glavni je cilj te faze 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 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 i 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, jer postoje jasna pravila koja kažu kako se ona dalje pretvara u relacijsku shemu i koje joj daljnje informacije 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 pohraniti podatke, nešto što može postojati ili ne postojati i što se može jednoznačno identificirati. Entitet može predstavljati stvar ili biće, na primjer: KNJIGA, FAKULTET, STUDENT, PREDMET (na fakultetu), NASTAVNIK, GRAD itd,, ili događaj i pojavu, poput: FILMSKA PROJEKCIJA, PUTOVANJE, POLAGANJE ISPITA itd.

Entitet se opisuje skupom *atributa.* Na primjer: atributi entiteta KNJIGA su: ISBN, NASLOV, AUTOR, GODINA IZDANJA, i sl., atributi entiteta STUDENT su JMBAG (jedinstveni matični broj akademskog građanina), PREZIME, IME, GODINA STUDIJA itd., a atributi PREDMETA koji se predaje na fakultetu su ŠIFRA PREDMETA, NASLOV, SEMESTAR u kojem se predaje, ECTS-BODOVI koji označavaju težinu predmeta itd.

Ako atribut zahtjeva svoje atribute, treba ga smatrati posebnim entitetom. Na primjer, za entitet KNJIGA mogli bismo definirati atribut AUTOR kao jednostavni podatak. Međutim, ako za opis autora trebamo dodatne atribute, poput IME, PREZIME, GODINA ROĐENJA, NACIONALNOST, tada AUTOR postaje zaseban entitet, a odnos između KNJIGE i njenog AUTORA interpretira se kao veza između dva entiteta. Isto pravilo vrijedi i ako atribut može imati više vrijednosti istovremeno. Na primjer, za entitet FILMSKA PROJEKCIJA, atribut FILM mogao bi biti definiran kao jednostavan podatak. No, ako za opis filma trebamo dodatne atribute, poput ŽANR, REDATELJ, GODINA IZDANJA, tada FILM postaje zaseban entitet, a odnos između FILMSKE PROJEKCIJE i njenog FILMA interpretira se kao veza između dva entiteta.

Ime entiteta, s pripadajućim popisom atributa. određuje *tip* entiteta. Za svaki zadani tip entiteta može postojati cijeli skup *primjeraka* (pojava) tog tipa, od kojih svaki ima različite vrijednosti atributa. Na primjer, STUDENT je tip entiteta čiji su primjerci konkretni studenti poput Petar Petrović, Marko Marković, Dragica Horvat itd. Svaki od tih studenata ima različite kombinacije vrijednosti atributa, poput JMBAG, PREZIME, IME, GODINA 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 identificiraju 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 KNJIGA, kandidat za ključ je atribut ISBN, jer je svaka knjiga jedinstveno identificirana svojim ISBN-om. Za tip entiteta FILMSKA PROJEKCIJA kandidat za ključ je kombinacija atributa FILM i VRIJEME (kada se projekcija održava).

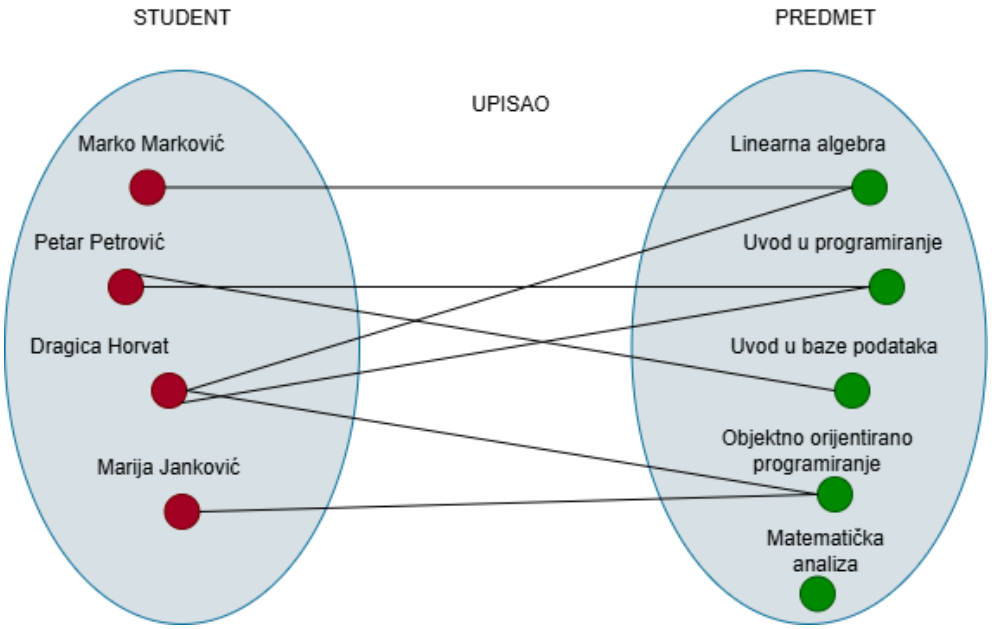
Ako jedan tip entiteta ima više kandidata za ključ, tada biramo jedan od njih koji nam se čini najpogodnijim za identifikaciju i proglašavamo ga *primarnim* ključem. Na primjer, za tip entiteta STUDENT, dobri kandidati za ključ su JMBAG, OIB i JMBG. U svakodnevnom životu služimo se i kombinacijom atributa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

Unutar jedne konceptualne sheme tipovi entiteta moraju imati različita imena. Također, svi atributi koji opisuju isti entitet moraju imati jedinstvena imena. Dopušta se da dva entiteta imaju atribute s istim imenom, ali 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 što 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 trenutačno povezani.

Kao primjer, promotrimo binarnu vezu UPISAO između tipova entiteta STUDENT i PREDMET. Njome se izražava činjenica da studenti upisuju izborne predmete na fakultetu. U svakom se 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 četiri primjerka entiteta STUDENT, koje smo zbog jednostavnosti pretpostavili da ih možemo razlikovati prema PREZIMENU i IMENU. Također, postoji pet primjeraka entiteta PREDMET, koje možemo razlikovati prema NASLOVU. 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, Uvod u programiranje i Objektno orijentirano programiranje; dok predmet Matematička analiza nije upisao nijedan student. Stanje veze s vremenom se može mijenjati: novi parovi mogu se pojaviti, a postojeći nestati. Na primjer, može se dogoditi da Marko Marković dodatno upiše Matematičku analizu, a da Petar Petrović odustane od predmeta Uvod u baze podataka.

Osim što povezuje tipove entiteta, veza može imati i svoje atribute, dakle atribute koji se ne mogu pripisati nijednom od tih tipova. Na primjer, veza UPISAO može imati atribut DATUM UPISA. Zaista, DATUM UPISA ne

možemo smatrati atributom entiteta STUDENT, jer isti student može upisati razne predmete na različite datume. Slično, DATUM UPISA ne može biti atribut entiteta PREDMET jer isti predmet mogu upisati razni studenti na različite datume. Konkretna vrijednost DATUMA UPISA zapravo se pridružuje konkretnom paru primjeraka entiteta STUDENT i PREDMET koji su povezani. Na primjer, možemo zabilježiti da je Marko Marković upisao Linearnu algebru 3. rujna 2024., a Marija Janković je upisala Objektno orijentirano programiranje 5. rujna 2024.

Slično kao i 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 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 projektiranja ispravno prikazala u relacijskoj shemi. Navedena ćemo 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 na to da se ista veza može promatrati u dva smjera, od E_1 do E_2 i obratno, postoje četiri vrste funkcionalnosti koje su opisane u Tablici 2.1.

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 vidi na Slici 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.

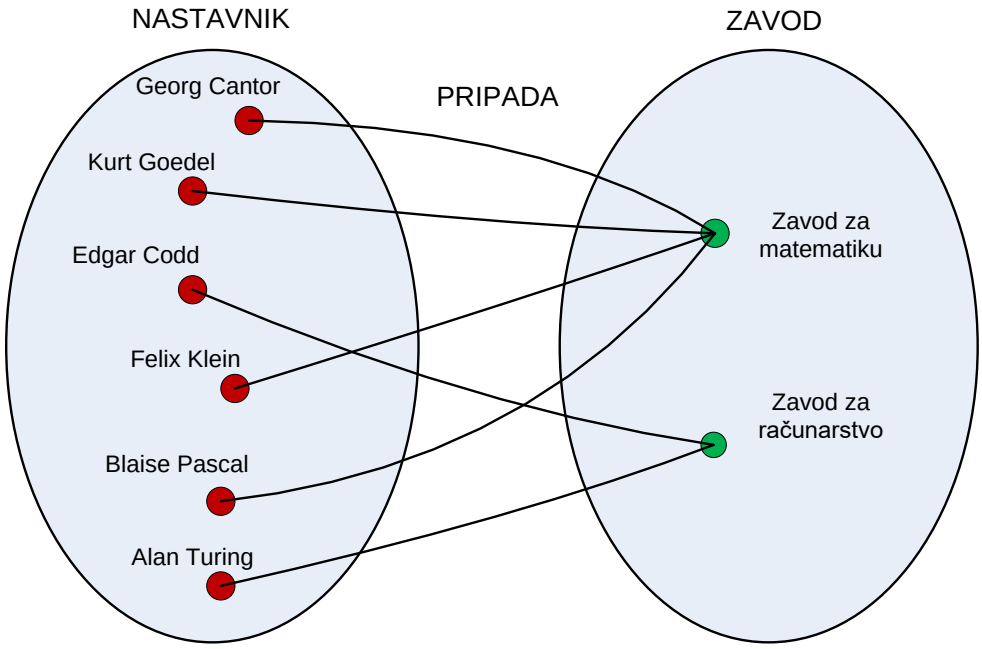
Tablica 2.1: Vrste funkcionalnosti za vezu između tipova entiteta E_1 i E_2

Kao primjer za funkcionalnost M:1 navodimo vezu PRIPADA između tipova entiteta NASTAVNIK i ZAVOD (organizacijska jedinica unutar fakulteta). 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 osnovi imena i prezimena. Ista veza ako se gleda u suprotnom smjeru može služiti kao primjer za funkcionalnost 1:M.

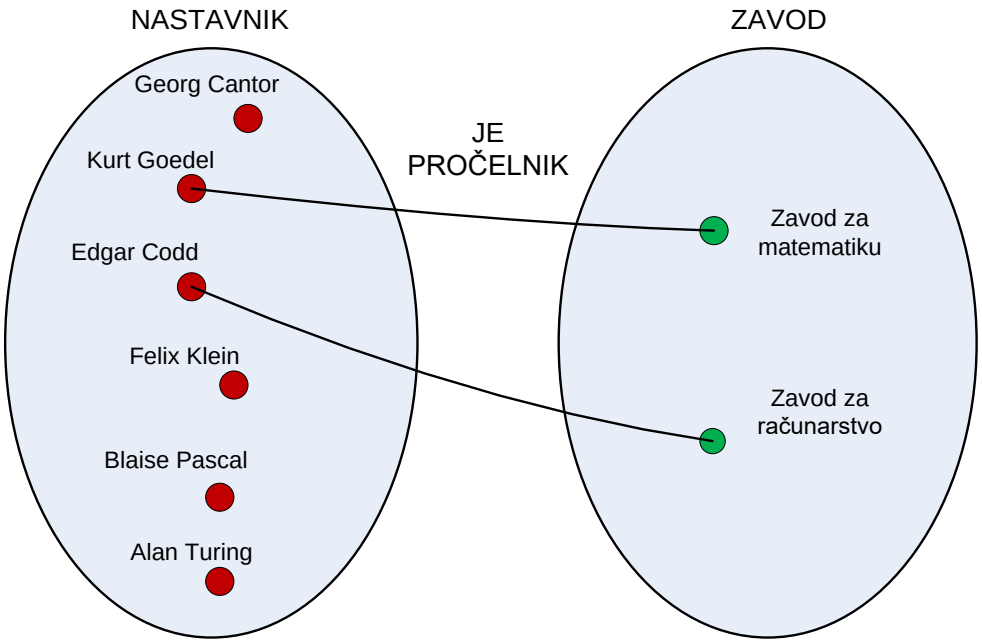
Kao primjer za funkcionalnost 1:1 navodimo vezu JE PROČELNIK između tipova entiteta NASTAVNIK i ZAVOD. Jedan nastavnik može biti pročelnik najviše jednog zavoda (onog kojem inače pripada), a jedan zavod može imati samo jednog pročelnika. Slika 2.3 prikazuje jedno stanje te veze.

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 2.2: Veza s funkcionalnošću M:1



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 se granice označavaju oznakama 0, 1 ili M i odvajaju se zarezom. Uobičajene kardinalnosti navedene su i opisane u Tablici 2.2.

OZNAKA	**OPIS**
0,1	Jedan primjerak od E_1 može biti povezan ni s jednim 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 ni s jednim, 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.

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

Navedimo kao primjer 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 od svakog studenta tražimo 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 je ako je donja granica 1 – tada bilo koji primjerak od E_1 mora biti povezan s barem 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 i 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.

2.1.4. 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 osnovi specifikacije dolazi do projektne dokumentacije na konceptualnoj razini. Opis koraka treba shvatiti uvjetno. Naime treba biti svjestan da se postupak projektiranja ne može do kraja opisati ni definirati. U projektiranju uvijek ostaje mjesta za dosjetljivost, improvizaciju i kreativnost.

2.1.5. 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 ova pitanja:

- Ima li pojam označen imenicom neka 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 zapravo pripadaju vezi između entiteta, a ne pojedinim entitetima. Također, za svaku vezu treba odrediti njezinu funkcionalnost, obaveznosti članstva i 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 ako je specifikacija nejasna ili nepotpuna. U takvim slučajevima projektant treba dodatno razgovarati s korisnicima kako bi otklonio nejasnoće. Također, projektant može po vlastitom nahođenju dodavati umjetne atribute (npr. šifre, oznake i sl.) koji služe za identifikaciju ili klasifikaciju, kao i atribute za koje je očito da nedostaju.

Zamislimo, na primjer, da trebamo oblikovati konceptualnu shemu baze podataka knjižnice. U specifikaciji su navedene sljedeće rečenice:

- Knjižnica posjeduje veliki broj knjiga. Svaka knjiga ima svoj naslov, autora, godinu izdanja i ISBN.
- Knjige su raspoređene po kategorijama (npr. roman, znanstvena literatura, dječja literatura). Svaka kategorija ima svoj naziv i opis.
- Članovi knjižnice mogu posuđivati knjige. Prilikom posudbe bilježe se datum posudbe i datum povrata. Svaki član ima svoje ime, prezime, adresu i članski broj.
- Knjižničari su odgovorni za unos knjiga i evidenciju posudbi. Svaki knjižničar ima ime, prezime i šifru zaposlenika.
- Na kraju godine knjižnica evidentira članove koji su posudili najveći broj knjiga.

Analizom tih rečenica otkrivamo ove entitete, veze i atribute.

- Tipovi entiteta su: KNJIGA, ČLAN, KATEGORIJA, KNJIŽNIČAR
- Veze su: POSUDIO između ČLAN i KNJIGA, PRIPADA između KATEGORIJA i KNJIGA, EVIDENTIRAO između KNJIŽNIČAR i POSUDIO
- Atributi su: NASLOV (knjige), AUTOR (knjige), GODINA IZDANJA (knjige), ISBN (jedinstveni identifikator knjige), KATEGORIJA (kojom knjiga pripada), DATUM POSUDBE (kada je član posudio knjigu), DATUM POVRATA (kada je korisnik vratio knjigu).

Za svaki tip entiteta ili vezu utvrđuje se popis atributa.

- KNJIGA ima jasno definirane atribute NASLOV, AUTOR, GODINA IZDANJA i KATEGORIJA. Kako bi svaka knjiga imala jedinstvenu identifikaciju, dodaje se ISBN kao primarni ključ.
- ČLAN na temelju specifikacije ima atribute IME, PREZIME i ADRESA. Za pouzdanu identifikaciju svakog člana, dodaje se ČLANSKI BROJ kao primarni ključ.
- KATEGORIJA ima atribute NAZIV i OPIS. Pretpostavlja se da je NAZIV jedinstven i stoga se koristi kao primarni ključ.
- KNJIŽNIČAR ima atribute IME, PREZIME i ŠIFRA ZAPOSLENIKA. ŠIFRA ZAPOSLENIKA služit će kao primarni ključ.
- Veza POSUDIO povezuje entitete ČLAN i KNJIGA te ima atribute DATUM POSUDBE i DATUM POVRATA, čime se bilježi povijest posudbi.

- veze (PRIPADA i EVIDENTIRAO) nemaju dodatnih atributa jer samo povezuju entitete.

Dalje za svaku vezu određujemo njezinu kardinalnost u oba smjeru. Budući da su sve veze binarne, određuje se njihova funkcionalnost te obaveznost članstva entiteta u njima.

KARDINALNOST VEZA:

- POSUDIO - u smjeru od ČLAN prema KNJIGA ima kardinalnost 1,M, a u obratnom smjeru 0,M. To znači da svaki korisnik može posuditi više knjiga, dok pojedina knjiga može biti posuđena više puta, ako bilježimo povijest posudbi.
- PRIPADA u smjeru od KNJIGA prema KATEGORIJA ima kardinalnost 1,1, a u obratnom smjeru 0,M. To znači da svaka knjiga pripada točno jednoj kategoriji, dok jedna kategorija može sadržavati više knjiga.
- EVIDENTIRAO u smjeru od KNJIŽNIČAR prema POSUDIO ima kardinalnost 0,M, a u obratnom smjeru 1,1. To znači da knjižničar može evidentirati više posudbi, dok svaka posudba mora je evidentirati jedan knjižničar.

Primijetimo da ova jednostavna shema opisuje samo trenutačno stanje u knjižnici. Ne pamti se koje su knjige korisnik posudio u prošlosti (osim ako za svaku posudbu čuvamo povijesne podatke u vezi POSUDIO).Kategorije knjiga ne bilježe povijest izmjena – svaka knjiga ima samo trenutačnu kategoriju. Evidencija knjižničara ne bilježi promjene vezane uz odgovornost ili status u knjižnici.

Za složeniju shemu koja bi pratila povijest događaja, trebalo bi dodati nove entitete (npr. POVIJESNA POSUDBA) ili proširiti atribute postojećih entiteta, primjerice dodavanjem vremenskih oznaka ili atributa koji prate izmjene u kategorijama i odgovornostima.

2.1.6. Crtanje dijagrama

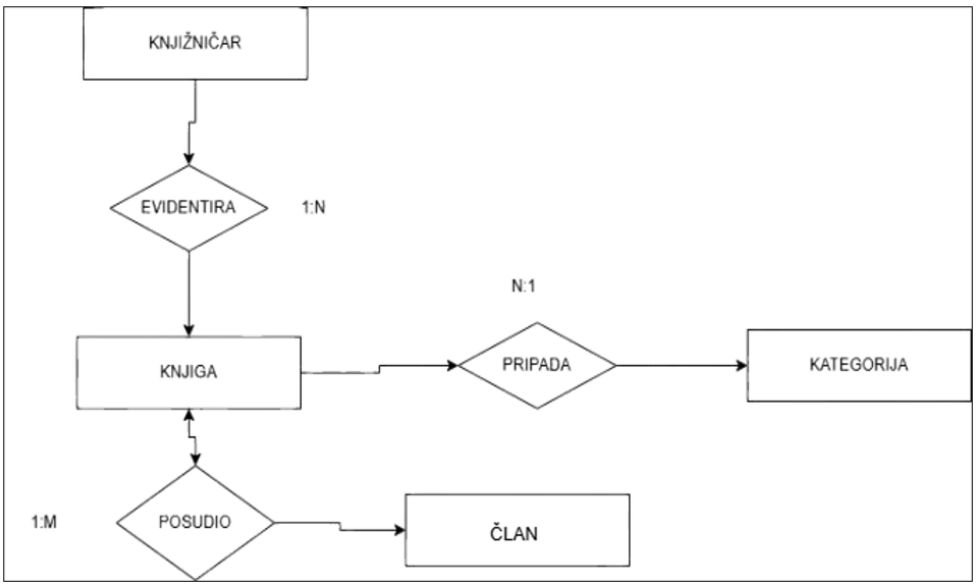
Nakon što smo identificirali sve entitete, veze i atribute, idući korak u oblikovanju konceptualne shema jest povezivanje tih elemenata i prikazivanje u obliku dijagrama. Koristi se reducirani Chenovim dijagram, u kojem su tipovi entiteta prikazani kao pravokutnici, a veze kao rombovi. Imena tipova entiteta i veza upisana su u odgovarajuće pravokutnike odnosno rombove. Kako bi bilo jasno između kojih je tipova entiteta uspostavljena određena veza, odgovarajući romb povezan je linijama s odgovarajućim pravokutnicima. Uz linije su upisane oznake kardinalnosti veza, pri čemu se kardinalnost u smjeru od jednog do drugog tipa entiteta piše bliže drugom tipu.

Primjer dijagrama nacrtanog prema navedenim pravilima prikazan je na Slici 2.4. Riječ je o prikazu konceptualne sheme baze podataka knjižnice, čiji su elementi identificirani u prethodnom odjeljku.

Nakon izrade dijagrama, projektant ga svakako treba prezentirati korisnicima te s njima treba provjeriti njegovu ispravnost. Ovim procesom mogu se otkriti eventualni propusti u konceptualnoj shemi, poput nedostataka entiteta ili veza, pogrešno uspostavljenih veza, netočnih kardinalnost i slično. Ako se uoče propusti, projektant se vraća na korak identificiranja veza i atributa, ponovo crta dijagram i ponovno ga predstavlja

korisnicima. Ciklus oblikovanja konceptualne sheme može se ponavljati više puta, sve dok korisnici ne odobre konačni dijagram.

Važno je napomenuti da reducirani Chenov dijagram kakvim se koristimo opisuje samo entitete i veze, a ne sadrži informacije o atributima. Informacije o atributima izostavljaju se radi jednostavnosti i preglednosti, budući da velik broj atributa može otežati čitljivost. Međutim, to znači da konceptualna shema nije u potpunosti opisana dijagramom, te da se informacije o atributima moraju dostaviti u tekstu koji prati dijagram.



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

2.1.7. Sastavljanje teksta koji prati dijagram

Posljednji korak u oblikovanju konceptualne sheme je sastavljanje teksta koji prati dijagram. Taj tekst treba pružiti informacije o konceptualnoj shemi koje nisu vidljive na samom dijagramu. Prvenstveno, tekst treba uključivati popis atributa za svaki tip entiteta i svaku vezu.

Za naš primjer baze podataka o knjižnici, tekst koji prati dijagram sa Slike 2.4 mogao bi sadržavati informacije prikazane na Slici 2.5. To znači da za svaki entitet i vezu trebamo navesti sve atribute, a primarne ključeve označiti podvlačenjem.

Osim tih osnovnih informacija, tekst koji prati dijagram može po potrebi uključivati dodatne sadržaje, poput objašnjenja projektanta za uvođenje atributa koji nisu bili predviđeni specifikacijom, odabira primarnog ključa ili pretpostavki o kardinalnosti veza.

Tip entiteta KNJIGA ima atribute:
ISBN, NASLOV, AUTOR, GODINA IZDANJA, KATEGORIJA.

Tip entiteta ČLAN ima atribute:
ČLANSKI BROJ, IME, PREZIME, ADRESA.

Tip entiteta KATEGORIJA ima atribute:
NAZIV KATEGORIJE, OPIS.

Tip entiteta KNJIŽNIČAR ima atribute:
ŠIFRA ZAPOSLENIKA, IME, PREZIME.

Veza POSUDIO ima atribute:
DATUM POSUDBE, DATUM POVRATA.

Ostale veze nemaju atribute.

Slika 2.5: Popratni tekst uz dijagram sa Slike 2.4

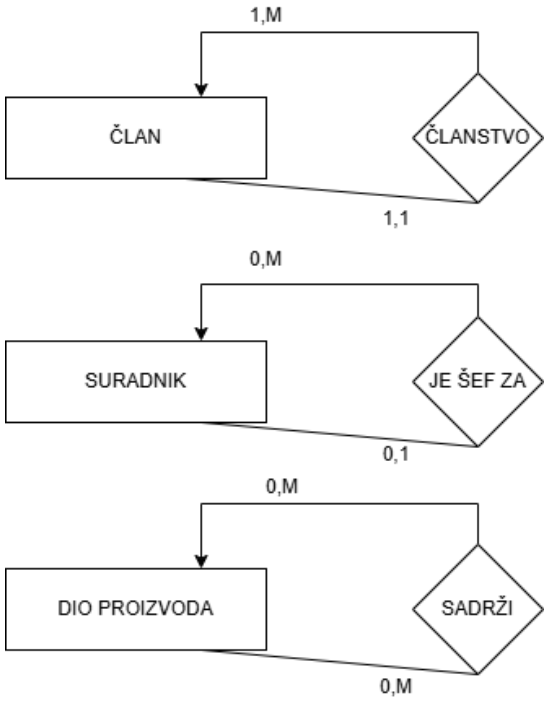
Ako se smatra korisnim, u tekst s može dodati i rječnik podataka, gdje će za svaki atribut biti objašnjeno njegovo značenje i tip vrijednosti koji može poprimiti. Ipak, takav rječnik nije obavezan na konceptualnoj razini projektiranja, već je namijenjen logičkoj razini. Na konceptualnoj razini dovoljno je da atributi imaju smislene nazive iz kojih se naslućuje njihovo značenje i tip. Na primjer, ako atribut nosi naziv NASLOV, jasno je da označava naslov knjige i da će poprimiti vrijednosti tekstualne podatke. Ako se zove ISBN, to je jedinstveni identifikator knjige koji će vjerojatno biti niz brojeva ili kombinacija znakova prema standardiziranom obrascu. Ako je naziv atributa DATUM POSUDBE, tada je jasno da označava datum kada je knjiga posuđena, a vrijednosti će biti u formatu datuma itd.

2.2. Složenije veze i njihov prikaz na dijagramima

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

2.2.1. Prikaz involuirane veze

Involuirana veza povezuje jedan tip entiteta s istim tipom. Njezino se stanje opisuje 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 ČLANSTVO, koja povezuje tip entiteta ČLAN. Ova veza dokumentira status članstva osobe u organizaciji, kao što je knjižnica. Funkcionalnost veze je 1:M, jer jedna osoba može imati više različitih članstava (npr., u različitim sekcijama knjižnice ili u različitim periodima), dok svako članstvo odgovara samo jednoj osobi. Dijagram prikazuje strelicu koja ukazuje na smjer tumačenja veze: jedna osoba je povezana s više članstava, a ne obratno. Članstvo u vezi je obavezno, jer se pretpostavlja da svaka osoba unutar baze ima barem jedno članstvo. Ovi podaci omogućuju organizaciji da efikasno upravlja svojim članovima, bilježeći početak i završetak svakog članstva, kao i ostale relevantne informacije poput tipa članstva.

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.

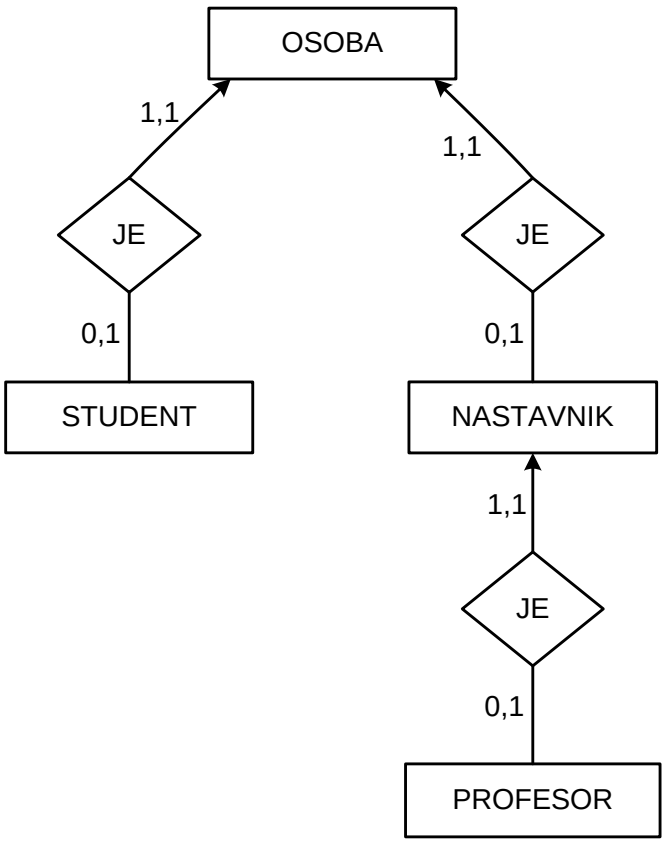
Treći dijagram na Slici 2.6 prikazuje vezu SADRŽI koja povezuje entitet DIO PROIZVODA sa samim sobom. Primjerci entiteta opisuju dijelove proizvoda koji se proizvode u nekoj tvornici. Veza prikazuje odnos između složenijih i jednostavnijih dijelova, dakle da jedan složeniji dio sadrži više jednostavnijih te da se isti jednostavniji dio pojavljuje 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.

2.2.2. Prikaz podtipova i nadtipova entiteta

Tip entiteta E_1 je *podtip* tipa entiteta E_2 ako je svaki primjerak od E_1 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 podtip 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 koristiti u druge svrhe osim za povezivanje podtipova i nadtipova. Primijetimo također da se sama veza JE može pojaviti više puta na istom dijagramu ako imamo više parova entiteta koji su u odnosu podtip-nadtip. Time je napravljena iznimka od općenitog pravila da svaka veza u shemi mora imati jedinstveno ime; no taj izuzetak ne smeta jer sve pojave veze JE imaju u biti isto značenje.



Slika 2.7: Primjeri za podtipove i nadtipove

Slika 2.7 sadrži dijagram sheme s podtipovima i nadtipovima. 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 itd. Tipovi STUDENT i NASTAVNIK su podtipovi 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 podtip od NASTAVNIK, a time posredno i podtip 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.

2.2.3. Prikaz ternarne veze

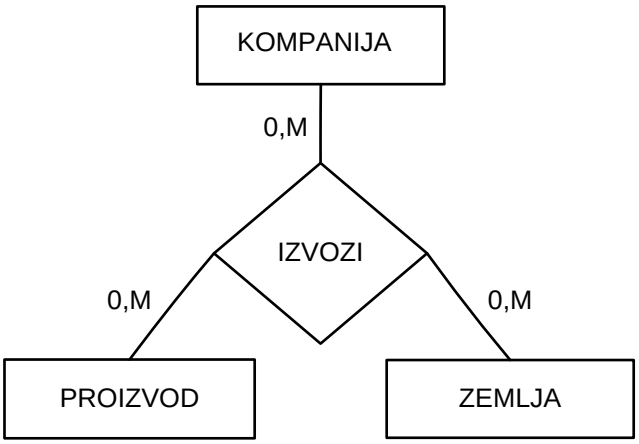
Ternarna veza uspostavlja se između tri tipa entiteta. Stanje ternarne veze opisuje se kao skup uređenih trojki primjeraka entiteta koji su trenutačno povezani. Svojstva ternarne veze teže je opisati nego kod binarne veze, jer postoji više vrsta funkcionalnosti i više kombinacija za obaveznost članstva. 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 kao romb s upisanim imenom veze. Razlika je u tome što iz romba izlaze tri spojnice prema odgovarajućim pravokutnicima, a ne dvije. Uz spojnice su i ovdje upisane oznake kardinalnosti veze i to tako da se kardinalnost u smjeru od odabrana dva tipa prema trećem tipu piše bliže trećem tipu.

Primjer ternarne veze sa Slike 2.4 odnosi se na podatke o knjigama, članovima i knjižničarima u procesu posuđivanja knjiga. Stanje te veze bilježi se kao skup uređenih trojki sastavljenih od primjeraka triju tipova. Pritom jedna takva trojka znači da je određeni član posudio određenu knjigu, a tu je posudbu evidentirao određeni knjižničar.

Sva tri entiteta u vezi na Slici 2.4 imaju obavezno članstvo, jer je u pravilu svaka knjiga evidentirana od strane barem jednog knjižničara, svaki knjižničar mora biti zadužen za evidenciju barem jedne knjige ili posudbe, a svaki član mora imati barem jednu moguću interakciju s knjižničarom i knjigom (odnosno, barem jednu posudbu). Funkcionalnost veze može biti mnogo-naprama-mnogo-naprama-jedan (M:M:1) ili mnogo-naprama-mnogo-naprama-mnogo (M:M:M), ovisno o poslovnim pravilima knjižnice. Na primjer, ako pretpostavimo da točno jedan knjižničar može evidentirati jednu posudbu (u jednom trenutku), a više različitih članova može posuditi istu knjigu u različito vrijeme, onda bi veza bila M:M:1.

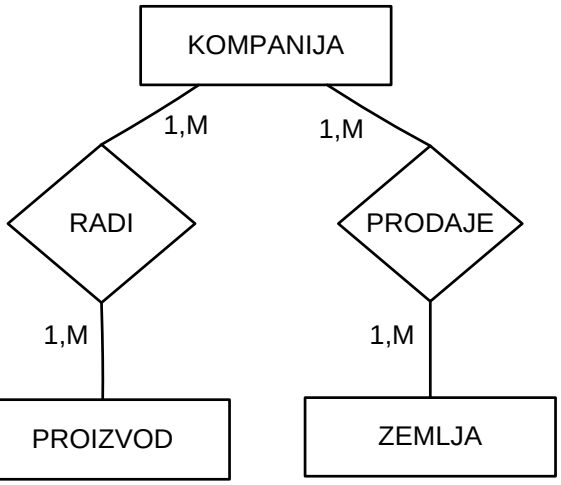
Moguće je također da za zadani par (član, knjiga) postoji više različitih knjižničara koji su u različito vrijeme evidentirali posudbu te knjige tom članu, što upućuje na višu razinu slobode (M:M:M). Međutim, može se dogoditi i da za neki par (član, knjiga) ta knjiga nije nikada bila posuđena, pa je kardinalnost u tom smjeru oblika 0,M.



Slika 2.8: Primjer ternarne veze

Ternarnu vezu uvodimo samo 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 to 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 kompanija na tržištu neke zemlje.

Iz dviju binarnih veza RADI i PRODAJE može se reproducirati polazna ternarna veza IZVOZI. Naime, da bi 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 ta 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.

2.3. Vježbe

- **Zadatak 2.1.** Očito je da već u specifikaciji iz Odjeljka 2.2.1, kao i u konceptualnoj shemi sa Slika 2.4 i 2.5, nedostaju mnogi važni podaci o knjižnici. 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 teretani. Predvidite da ta baza mora pohranjivati podatke o programima i aktivnostima u teretani, članovima teretane i zaposlenicima teretane. Također, moraju se evidentirati prijave članova na pojedine programe.

Dodatni zadaci:

- **Zadatak 2.3.** Na temelju specifikacije koju ste dobili rješavanjem Zadatka 1.6 oblikujte konceptualnu shemu za bazu podataka iz svojeg 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 osnovi te specifikacije i bez čitanja ostatka priloga sami oblikujte odgovarajuću konceptualnu shemu. Usporedite svoje rješenje s onim u prilogu.
- **Zadatak 2.6.** Oblikujte konceptualnu shemu za sustav upravljanja događajima u kulturnom centru:
    - Događaji: ID_događaja, Naziv, Datum, Vrijeme, Opis, Lokacija.
    - Organizatori: ID_organizatora, Ime, Kontakt, Email.
    - Sudionici: ID_sudionika, Ime, Prezime, Kontakt.
    - Karte: ID_karte, Cijena, Mjesto_sjedala, ID_događaja.
- Definirajte veze između entiteta:
    - Jedan događaj može imati više organizatora i sudionika.
    - Karte su vezane uz događaje, a jedan događaj može imati više karata.
- **Zadatak 2.7.** Oblikujte konceptualnu shemu za bazu podataka transportne tvrtke koja prati:
    - Vozači: ID_vozača, Ime, Prezime, Kontakt, Broj_licence
    - Vozila: ID_vozila, Marka, Model, Registracija, Kapacitet_tereta
    - Putovanja: ID_putovanja, Datum_polaska, Datum_dolaska, ID_vozača, ID_vozila).
    - Tereti: ID_tereta, naziv, težina, ID_putovanja
- Definirajte veze među entitetima:
    - Svako putovanje ima jednog vozača i jedno vozilo.
    - Jedan teret se dodjeljuje jednom putovanju, ali jedno putovanje može imati više tereta.

Zadatak 2.8 Ljekarna koristi bazu podataka za praćenje lijekova, pacijenata, recepata i izdavanja lijekova. Pacijenti mogu podizati lijekove na temelju recepata koje im izdaju liječnici. Ljekarna prati stanje zaliha lijekova i evidentira povijest izdanih lijekova. Zaposlenici ljekarne odgovorni su za unos podataka o lijekovima, obradu recepata i izdavanje lijekova pacijentima.

1. Izradite CHENOV dijagram za sustav prema sljedećoj specifikaciji:

   Entiteti i atributi:

   - Lijek: ID_lijeka (primarni ključ), Naziv, Opis, Cijena, Zaliha
   - Pacijent: ID_pacijenta (primarni ključ), Ime, Prezime, Kontakt, Adresa
   - Recept: ID_recepta (primarni ključ), Datum_izdavanja, Datum_iskoristivosti, ID_pacijenta (vanjski ključ)
   - Zaposlenik: ID_zaposlenika (primarni ključ), Ime, Prezime, Pozicija, Kontakt
   - Izdavanje: ID_izdavanja (primarni ključ), Datum, Količina, ID_lijeka (vanjski ključ), ID_recepta (vanjski ključ), ID_zaposlenika (vanjski ključ)

2. Veze među entitetima:

   - Pacijent → Recept: Jedan pacijent može imati više recepata.
   - Recept → Izdavanje: Jedan recept može biti iskorišten za više lijekova.
   - Lijek → Izdavanje: Jedan lijek može biti izdavan više puta.
   - Zaposlenik → Izdavanje: Jedan zaposlenik može obraditi više izdanja lijekova.

Identificirajte entitete, veze i atribute te povežite entitete s odgovarajućim relacijama koristeći rombove za veze. Navedite kardinalnosti za svaku vezu.

Na temelju CHENOVOG dijagrama, pretvorite sustav u relacijsku shemu.

3. Projektiranje na logičkoj razini

3. dan

3:15

Predavanja: 2 x 45 min.
Vježbe: 2 x 45 min.
Pauza: 1x15 min.

Po završetku ovog poglavlja moći ćete:

- *analizirati pravila pretvorbe konceptualne u relacijsku shemu*
- *kreirati relacijsku shemu prema pravilima relacijskog modela*
- *primijeniti pravila pretvorbe konceptualne sheme u relacijsku shemu*
- *pretvoriti složenije veze u relacije.*

U ovom poglavlju počinjemo govoriti o drugoj fazi projektiranja baze podataka, a to je projektiranje na logičkoj razini. 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, jer 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 i način kako se zapisuje relacijska shema. Zatim ćemo izložiti pravila kojima se konceptualna shema baze dobivena u prethodnom poglavlju pretvara u relacijsku shemu. Tako dobivena relacijska shema obično još nije u svojem konačnom obliku – ona se dalje podvrgava postupku dotjerivanja (normalizacije), koji će biti opisan u sljedećem poglavlju.

3.1. Relacijski model za bazu podataka

Relacijski model bio je teoretski zasnovan još krajem 60-ih godina 20. stoljeća, u radovima Edgara Codda. Model se dugo pojavljivao samo u akademskim raspravama i knjigama. Prve realizacije na računalu bile su prespore i neučinkovite. Zahvaljujući intenzivnom istraživanju i napretku računala, učinkovitost relacijskih baza postepeno se poboljšavala. Sredinom 80-ih godina 20. stoljeća relacijski je model prevladao. I danas se velika 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 – *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 zapravo atributi s istim značenjem. Vrijednosti jednog atributa su podaci iste vrste. Dakle, definiran je *tip* ili skup dopuštenih vrijednosti za atribut, koji se 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.

Na Slici 3.1 prikazana je relacija KNJIGA s atributima ISBN, NASLOV, AUTOR, GODINA IZDANJA, KATEGORIJA. Relacija sadrži podatke o knjigama u knjižnici.

KNJIGA

ISBN	NASLOV	AUTOR	GODINA IZDANJA	KATEGORIJA
9781234567897	Uvod u programiranje	Marko Novak	2022	Informatika
9789876543210	Osnove matematike	Ana Horvat	2023	Znanstvena literatura
9781122334455	Povijest umjetnosti	Petra Kolar	2012	Umjetnost
9781122334455	Znanstvena fantastika	Ivan Janković	2021	Beletristika
9789988776655	Priče za djecu	Luka Petrović	2016	Dječja literatura

Slika 3.1: Relacija s podacima o knjigama

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* s ovim svojstvima:

1. Vrijednosti atributa iz *K* jednoznačno određuju n-torku u *R*. Dakle u *R* ne mogu postojati dvije n-torke s istim vrijednostima atributa iz *K*.
2. Ako iz *K* izbacimo bilo koji atribut, tada se narušava svojstvo 1.

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

Na primjer, u relaciji o posudbama sa Slike 3.1 atribut ISBN predstavlja primarni ključ. Kombinacija naslova knjige i autora vjerojatno nije prikladna kao primarni ključ, budući da se mogu pojaviti više knjiga s istim naslovom i autorom, osobito kod različitih izdanja ili prijevoda.

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.

Događa se da relacija ima više kandidata za ključ. Tada jedan od njih proglašavamo *primarnim ključem*. Atributi koji sastavljaju primarni ključ

zovu se *primarni atributi*. Vrijednost primarnog atributa ni u jednoj n-torki ne smije 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 od popisa imena atributa odvojenih zarezima i zatvorenih u zagrade. Primarni atributi su podvučeni. Na primjer, za relaciju o knjigama sa Slike 3.1, shema izgleda ovako:

KNJIGA (ISBN, NASLOV, AUTOR, GODINA_IZDANJA, KATEGORIJA).

Relacijska shema cijele baze zapisuje se tako da se nanižu sheme za sve relacije od kojih se ta baza sastoji. Dakle, shema baze ima toliko redaka koliko u njoj ima relacija. Na primjer, za bazu podataka o knjižnici 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 na Slici 3.6.

3.2. Pretvaranje konceptualne u relacijsku shemu

U nastavku objašnjavamo kako se pojedini elementi konceptualne sheme pretvaraju u relacije. Tako ćemo pokazat 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 KNJIGA iz baze podataka knjižnice sa Slika 2.4 i 2.5 prikazuje se relacijom

KNJIGA (ISBN, NASLOV, IME_AUTORA, GODINA_IZDANJA).

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.

KNJIGA (ISBN, NASLOV, AUTOR, GODINA IZDANJA)

ČLAN (ČLANSKI BROJ, PREZIME, IME, ADRESA)

KNJIŽNIČAR (ŠIFRA_ZAPOSLENIKA, PREZIME, IME)

KATEGORIJA (NAZIV KATEGORIJE, OPIS)

Slika 3.2: Pretvorba entiteta iz baze podataka o knjižnici u relacije

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.

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 svaka knjiga mora biti povezana s jednom kategorijom (odnosno, da svaka kategorija može imati više knjiga, ali je za knjigu obavezno navesti kategoriju), tada se atribut koji predstavlja primarni ključ iz entiteta KATEGORIJA ubacuje u relaciju KNJIGA kao strani ključ:

KNJIGA (ISBN, NASLOV, AUTOR, GODINA_IZDANJA, NAZIV).

KATEGORIJA (NAZIV_KATEGORIJE, OPIS)

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

Ovakvo uvođenje stranog ključa omogućuje da se polazna veza reproducira u oba smjera:

- Za zadanu knjigu dovoljno je pogledati vrijednost atributa *NAZIV* u relaciji **KNJIGA** kako bismo saznali kojoj kategoriji knjiga pripada.
- Obrnuto, da bismo za zadanu kategoriju pronašli sve knjige koje joj pripadaju, pretražimo relaciju **KNJIGA** i izdvojimo sve n-torke sa zadanom vrijednošću *NAZIV* – svaka od njih opisuje jednu od knjiga iz te kategorije.

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 rješenje za prikaz te veze i pripadnih entiteta izgleda ovako:

ČLAN (ČLANSKI BROJ, PREZIME, IME, ADRESA, …)

KNJIGA (ISBN, ČLANSKI BROJ, NASLOV, …).

Relacije ČLAN i KNJIGA odgovaraju samim tipovima entiteta. Kao primarne ključeve uveli smo atribute ČLANSKI BROJ (člana) odnosno ISBN (knjige). Da bismo prikazali vezu posuđivanja, u relaciju KNJIGA kao strani ključ dodali smo ČLANSKI BROJ osobe koja je posudila knjigu.

Slično kao i prije, taj strani ključ omogućuje da se veza POSUDBA reproducira u oba smjera. Primijetimo da će vrijednost atributa ČLANSKI BROJ ostati prazna u mnogim n-torkama relacije KNJIGA, to jest za sve knjige koje trenutačno nisu posuđene.

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

ČLAN (ČLANSKI BROJ, PREZIME, IME, ADRESA, …)

KNJIGA (ISBN, NASLOV, …)

POSUDBA (ISBN, ČLANSKI BROJ).

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

Slično kao i strani ključ, posebna relacija za prikaz veze opet omogućuje da se ta veza reproducira u oba smjera. Na primjer, da bismo utvrdili status određene knjige, u relaciji POSUDBA tražimo n-torku s odgovarajućom vrijednošću ISBN-a. Ako takvu n-torku ne nađemo, tada knjiga nije posuđena; u protivnom je posuđena i to članu s upisanom vrijednošću ČLANSKOG BROJA. Obratno, da bismo pronašli koje je sve knjige posudio određeni član, pretražujemo relaciju POSUDBA i izdvajamo sve n-torke s odgovarajućom vrijednošću za ČLANSKI BROJ – svaka od tih n-torki sadrži ISBN jedne od posuđenih knjiga.

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

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 knjižnici, 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.

KNJIGA (ISBN, NASLOV, AUTOR, GODINA IZDANJA, KATEGORIJA)

ČLAN (ČLANSKI BROJ, IME, PREZIME, ADRESA)

KATEGORIJA (NAZIV_KATEGORIJE, OPIS)

KNJIŽNIČAR (ŠIFRA_ZAPOSLENIKA, IME, PREZIME)

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

Na Slici 3.4 sve novosti u odnosu na Sliku 3.2 označene su crvenom bojom. Vidimo da su u relacije koje odgovaraju entitetima (KNJIGA, ČLAN, KATEGORIJA, KNJIŽNIČAR) dodani strani ključevi koji omogućuju reproduciranje svih M:1, 1:M i 1:1 veza. Atribut KATEGORIJA u relaciji KNJIGA određuje kojoj kategoriji pripada određena knjiga. Slično, atribut NAZIV u relaciji KATEGORIJA može (ovisno o modelu) bilježiti i tko je za tu kategoriju odgovoran (npr. ako bismo imali ŠIFRA_ZAPOSLENIKA kao strani ključ u KATEGORIJA koji upućuje na KNJIŽNIČAR). Atribut AUTOR u relaciji KNJIGA određuje tko je napisao knjigu (ako je u ovom jednostavnom modelu zabilježen samo jedan autor). Atribut ČLANSKI_BROJ u relaciji ČLAN služi za jednoznačnu identifikaciju člana knjižnice. Ipak, dobivena shema još uvijek ne bilježi M:N vezu **POSUDIO** (koja bi spajala više članova s više knjiga). Za takvu vezu potrebno je uvesti zasebnu relaciju (tablicu) s dva strana ključa – jedan prema

ČLAN(ČLANSKI_BROJ), a drugi prema KNJIGA(ISBN) – i dodatnim atributima poput DATUM_POSUDBE, DATUM_POVRATA i sl.

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 s eventualnim

atributima veze. Na primjer, veza POSUDIO iz baze knjižnice sa Slike 2.4 prikazuje se relacijom:

POSUDIO (ČLANSKI_BROJ, ISBN, DATUM POSUDBE, DATUM POVRATA).

Činjenica da je jedan član posudio jednu knjigu prikazuje se jednom n-torkom u relaciji POSUDIO. Ključ za POSUDIO je očito složen, to jest sastoji se od kombinacije atributa ČLANSKI_BROJ i ISBN. Naime, budući da isti član može posuditi više knjiga, a ista knjiga može biti posuđena od strane više članova (u različitim vremenima), 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 utvrdili koje je sve knjige posudio određeni član, pretražujemo relaciju POSUDIO i izdvajamo sve n-torke s odgovarajućom vrijednošću za ČLANSKI BROJ – svaka od izdvojenih n-torki sadrži ISBN posuđene knjige. Obrnuto, za utvrđivanje svih članova koji su posudili određenu knjigu, pretražuje se relacija POSUDIO prema ISBN – svaki pronađeni zapis otkriva ČLANSKI BROJ člana koji je posudio tu knjigu.

.

KNJIGA (ISBN, NASLOV, AUTOR, GODINA IZDANJA, KATEGORIJA)

ČLAN (ČLANSKI BROJ, IME, PREZIME, ADRESA)

KATEGORIJA (NAZIV_KATEGORIJE, OPIS)

KNJIŽNIČAR (ŠIFRA_ZAPOSLENIKA, IME, PREZIME)

POSUDIO (ČLANSKI BROJ, ISBN, DATUM POSUDBE, DATUM POVRATA)

Slika 3.5: Relacijska shema za bazu podataka o knjižnici

Ubacivanjem relacije POSUDIO u shemu sa Slike 3.4, dobivamo shemu prikazanu na Slici 3.5. To je napokon cjelovita relacijska shema za našu bazu podataka o knjižnici, koja je ekvivalentna konceptualnoj shemi jer 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. Iz te se sheme vidi od kojih se relacija sastoji cijela baza i od kojih se atributa sastoji svaka pojedina relacija. Ipak, uočili smo da se iz sheme ne vide tipovi tributa. Također se katkad može dogoditi da se iz imena pojedinih atributa ne može lako odrediti njihovo značenje.

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

Rječnik podataka stvaramo tako da prođemo svim relacijama iz sheme i upišemo u tablicu sve atribute na koje naiđemo, s time da isti atribut upišemo samo jednom. Zatim atributima u tablici na što razumljiviji način opišemo njihovo značenje. Na kraju svakom atributu odredimo tip.

Kod određivanja tipova treba uzeti u obzir da današnji DBMS-ovi očekuju da će atributi biti cijeli ili realni brojevi, znakovi ili nizovi znakova, datumi ili novčani iznosi. Dakle, 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 knjižnici dobivamo rječnik podataka koji je prikazan na Slici 3.6. Taj rječnik služi kao nadopuna relacijske sheme sa Slike 3.5.

IME ATRIBUTA	TIP	OPIS
ISBN	Niz od točno 13 znamenki	Šifra koja jednoznačno određuje knjigu.
NASLOV	Niz znakova	Naslov knjige.
AUTOR	Niz znakova	Ime i prezime autora
GODINA IZDANJA	Cijeli broj	Godina kada je knjiga izdana.
KATEGORIJA	Niz od točno 10 znamenki	Naziv ili oznaka kategorije u koju se knjiga svrstava.
ČLANSKI BROJ	Niz znakova	Šifra koja jednoznačno određuje člana knjižnice.
IME	Niz znakova	Ime člana
PREZIME	Niz znakova	Prezime člana

IME ATRIBUTA	TIP	OPIS
ADRESA	Niz znakova	Adresa stanovanja člana.
ŠIFRA_ZAPOSLENIKA	Niz od točno 5 znamenki	Jedinstvena šifra koja identificira knjižničara.
IME	Niz znakova	Ime člana
PREZIME	Niz znakova	Prezime člana
DATUM POSUDBE	Datum	Datum kada je knjiga posuđena
DATUM POVRATA	Datum	Datum kada je knjiga vraćena

Slika 3.6: Rječnik podataka za bazu podataka o fakultetu

Budući da se još uvijek bavimo logičkom razinom projektiranja, ograničenja vezana uz tip biramo na najprirodniji način, bez obzira na to 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 moći obavljati tako detaljnu kontrolu znak-po-znak. Slično tomu, tip atributa PREZIME definiramo kao niz znakova proizvoljne duljine, makar će DBMS sigurno nametnuti neku gornju granicu duljine.

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.

Fokusiramo se na entitet ČLAN i vezu ČLANSTVO koja zbog svoje fleksibilnosti u članstvu modeliramo kao 1:M vezu. Relacija ČLAN odgovara entitetu osobe, gdje svaka n-torka opisuje jednu osobu s atributima poput OIB-a koji služi kao primarni ključ, prezimena, imena i adrese. Relacija ČLANSTVO bilježi članske veze, gdje svaka n-torka zapisuje članstvo povezano s osobom identificiranom preko OIB-a. Članski broj u ovoj relaciji koristi se kao primarni ključ, a relacija također sadrži informacije o datumu

početka i završetka članstva. Model je dizajniran tako da svaki član može imati više članstava, dok svako članstvo jasno identificira jednog člana, reflektirajući 1:M vezu gdje jedan član može imati više različitih statusa članstva kroz vrijeme.

Ovaj pristup osigurava jasnoću i funkcionalnost u upravljanju članstvima unutar organizacije, poput knjižnice, i omogućava efikasno praćenje i administraciju članstva.

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 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. To 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 utvrdili tko mu je šef, iz odgovarajuće n-torke relacije SURADNIK čitamo MATIČNI BROJ ŠEFA. Obratno, da bismo za zadanog šefa ustanovili koji su mu sve suradnici 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 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 u sebi sadrži 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 se jednostavni dio 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

Podtip nekog tipa entiteta prikazuje se posebnom relacijom koja sadrži primarne atribute nadređenog tipa i atribute specifične za taj podtip. 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 …).

Svaka od tih relacija odgovara jednom od tipova entiteta. Kao primarni ključ za identificiranje osobe uveli smo OIB. Veza JE između podtipova i nadtipova uspostavlja se na osnovi 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 podtipova i nadtipova. 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 podtipova i nadtipova po potrebi reproducira unutar *aplikacija*. Pritom sama baza, odnosno njezina logička struktura, nije svjesna hijerarhije tipova.

Odnosi podtipova i nadtipova 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 se veza gotovo uvijek prikazuje posebnom relacijom, koja sadrži primarne atribute svih triju tipova entiteta 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 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, niti jedna kombinacija dvaju od ta tri atributa ne određuje n-torku, jer na primjer za zadanu kompaniju i zadani proizvod može biti više zemalja u koje ta kompanija izvozi taj proizvod itd. Kod ternarnih veza čija funkcionalnost nije M:M:M broj primarnih atributa može biti manji.

Slično kao i 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 utvrdili 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. Vježbe

- **Zadatak 3.1.** Rješavanjem Zadatka 2.1 dobili ste nadopunjenu konceptualnu shemu baze podataka o knjižnici. Sad tu nadopunjenu konceptualnu shemu pretvorite u (nadopunjenu) relacijsku shemu. Također sastavite odgovarajući rječnik podataka. Koje su se nove relacije ili novi atributi 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 teretani. Sad tu konceptualnu shemu pretvorite u relacijsku shemu. Također sastavite odgovarajući rječnik podataka.

Dodatni zadaci:

- **Zadatak 3.3.** Na temelju konceptualne sheme koju ste dobili rješavanjem Zadatka 2.3. oblikujte relacijsku shemu i rječnik podataka za bazu podataka iz svojeg područja interesa.
- **Zadatak 3.4.** U Prilogu 1 pronađite konceptualnu shemu baze podataka o bolnici. Na osnovi te sheme i bez čitanja ostatka priloga sami oblikujte odgovarajuću relacijsku shemu i rječnik podataka. Usporedite svoje rješenje s onim u prilogu.
- **Zadatak 3.5.** U Prilogu 2 pronađite konceptualnu shemu baze podataka o znanstvenoj konferenciji. Na temelju te sheme i bez čitanja ostatka priloga sami oblikujte odgovarajuću relacijsku shemu i rječnik podataka. Usporedite svoje rješenje s onim u prilogu.
- **Zadatak 3.6.** Na temelju konceptualne sheme sustava za upravljanje događajima u kulturnom centru iz Zadatka 2.6, pretvorite konceptualnu shemu u relacijsku shemu. Shema treba uključivati:
    - Tablice za događaje, organizatore, sudionike i karte.
    - Primarne ključeve za sve tablice i odgovarajuće vanjske ključeve koji povezuju tablice.
    - Kardinalnosti između entiteta koje osiguravaju ispravnu povezanost podataka.
    - Rječnik podataka za svaku tablicu, uključujući nazive atributa, tipove podataka (INT, DATE) i eventualna ograničenja (npr. NOT NULL, UNIQUE).
- **Zadatak 3.7.** Na temelju konceptualne sheme sustava za upravljanje transportom iz Zadatka 2.7, pretvorite konceptualnu shemu u relacijsku shemu. Uključite sljedeće:
    - Tablice za vozače, vozila, putovanja i terete.
    - Definirajte primarne i vanjske ključeve.
    - Dodajte rječnik podataka za svaku tablicu s opisom atributa (npr. naziv, tip podataka, ograničenja).

Razmislite kako osigurati referencijalnu cjelovitost između putovanja i vozača te između putovanja i tereta.

- **Zadatak 3.8.** Na temelju konceptualne sheme sustava za upravljanje ljekarnom iz Zadatka 2.8, oblikujte relacijsku shemu koja uključuje:

- Tablice za lijekove, pacijente, recepte, zaposlenike i izdavanje lijekova.
- Definirajte primarne i vanjske ključeve za svaku tablicu.
- Navedite rječnik podataka za tablice s opisima atributa i tipovima podataka.

Razmislite kako biste implementirali vezu između recepata i izdavanja lijekova kako bi se osigurala točnost podataka.

- **Zadatak 3.9.** Ako biste dodali novu funkcionalnost sustavu (npr. popusti na lijekove, grupne karte za događaje, dodatni kapacitet za teret), kako biste proširili postojeću shemu?

4. dan

3:15
Predavanja: 3 x 45 min.
Vježbe: 1 x 45 min.
Pauza: 1x15 min.

4. Nastavak projektiranja na logičkoj razini – normalizacija

Po završetku ovog poglavlja moći ćete:

- *primijeniti normalizaciju baza podataka i prepoznati nepravilnosti u relacijama*
- *analizirati funkcionalne ovisnosti i pravilno oblikovati relacije*
- *objasniti potrebu za normalizacijom*
- *prepoznati teškoće povezane s nenormaliziranim podacima.*

U ovom poglavlju nastavljamo govoriti o drugoj fazi projektiranja baze podataka, dakle opet govorimo o projektiranju na logičkoj razini. 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 projektiranje na fizičkoj razini. Sam postupak dotjerivanja zove se *normalizacija*.

U daljnjem tekstu najprije opisujemo jednostavniji i u praksi više korišteni dio normalizacije, a to je prevođenje u prvu, drugu i treću normalnu formu. Zatim se bavimo složenijim dijelom normalizacije koji je vezan uz Boyce-Coddovu i četvrtu normalnu formu. Na kraju raspravljamo o tome zašto je normalizacija uopće potrebna te možemo li od nje ipak 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“ koji 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-ih godina definirao daljnja mjerila kvalitete koja su izražena drugom i trećom normalnom formom.

4.1.1. Podzapisi, ponavljajuće skupine, prevođenje podataka u prvu normalnu formu

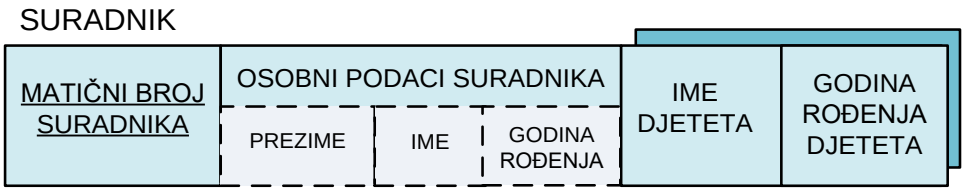
Kad smo govorili o relacijskom modelu za bazu podataka, naglasili smo da vrijednost atributa u relaciji (sadržaj jedne kućice u tablici) mora biti *jednostruka* i *jednostavna*. Dakle u jednu se kućicu ne može upisati više vrijednosti nego najviše jedna. Također, ta upisana vrijednost ne smije biti složena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

To svojstvo jednostrukosti i jednostavnosti zove se jednim imenom svojstvo *prve normalne forme* (oznaka: 1NF). Dakle, kažemo da je relacija u 1NF, jer su vrijednosti njezinih atributa jednostruke i nedjeljive. Primijetimo da 1NF zapravo ne predstavlja nikakav poseban zahtjev na relaciju, jer je to svojstvo već ugrađeno u relacijski model i definiciju relacije. Dakle, u relacijskoj bazi podataka 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.

Ako bazu podataka projektiramo kako je opisano u drugom poglavlju, 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 trećeg poglavlja dobit ćemo korektne relacije, dakle one u 1NF.

Znanje o 1NF potrebno nam je prvenstveno u situaciji kad su neki podaci već pohranjeni u nekom ne-relacijskom obliku, a želimo ih izravno prebaciti u relacijsku bazu. Zapisi (slogovi) u ne-relacijskim kolekcijama podataka mogu biti nenormalizirani, naime oni mogu sadržavati takozvane *podzapise* ili *ponavljajuće skupine*. Na primjer, u nekoj tvrtki mogli bismo imati datoteku s podacima o suradnicima. Zapis o jednom suradniku mogao bi imati oblik prikazan na Slici 4.1. Vidimo da se tu pojavio podzapis s osobnim podacima 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 podzapise. Postupak prevođenja izgleda otprilike ovako:

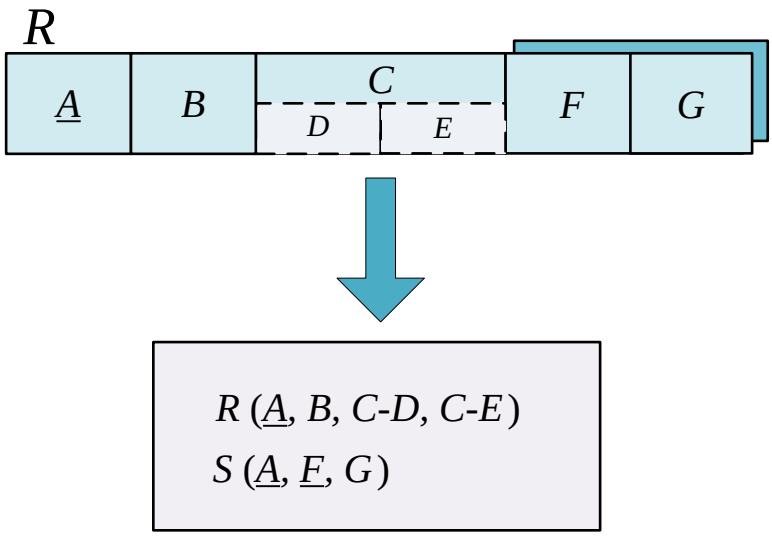
- Uvodi se jedna osnovna relacija i onoliko pomoćnih relacija koliko ima ponavljajućih skupina.
- Fiksni dio zapisa prikazuje se kao jedna n-torka u osnovnoj relaciji, s time da je svaki podzapis rastavljen na nekoliko zasebnih atributa.
- Svaka pojava ponavljajuće skupine u zapisu prikazuje se kao zasebna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

Ako taj postupak prevođenja u 1NF primijenimo na naše zapise o suradnicima sa Slike 4.1, dobit ćemo ove 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 osnovi iste vrijednosti MATIČNOG BROJA SURADNIKA.



Slika 4.2: Shematski prikaz prevođenja u 1NF

Postupak prevođenja u 1NF shematski je prikazan na Slici 4.2. Na toj slici vidi se pretvorba jednog podzapisa i jedne ponavljajuće skupine. Ako imamo više podzapisa ili više ponavljajućih skupina, tada se zahvati sa Slike 4.2 moraju ponavljati.

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:

POSUDIO (ČLANSKI BROJ, ISBN, NASLOV KNJIGE, AUTOR KNJIGE, GODINA IZDANJA, DATUM POSUDBE).

Ova relacija evidentira koje knjige su posuđene u knjižnici, s detaljima o knjigama i članovima koji su ih posudili. Pretpostavimo da ČLANSKI BROJ jednoznačno određuje člana, a ISBN jednoznačno određuje knjigu. Također, možemo pretpostaviti da svaka knjiga ima jednog autora i jednu godinu izdanja. Vidimo da u relaciji postoje brojne funkcionalne ovisnosti. Evo nekoliko primjera:

(ČLANSKI BROJ, ISBN) → DATUM POSUDBE,

ISBN→ NASLOV KNJIGE,

ISBN → AUTOR KNJIGE,

ISBN → GODINA IZDANJA.

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. Svaki je atribut u relaciji funkcionalno ovisi o ključu, ali ta ovisnost ne mora biti potpuna. Na primjer, u prethodnoj relaciji POSUDIO, atribut DATUM POSUDBE je potpuno funkcionalno ovisan o primarnom ključu (ČLANSKI BROJ, ISBN) jer oba atributa zajedno jednoznačno određuju datum posudbe. Međutim, atributi NASLOV KNJIGE, AUTOR KNJIGE, i GODINA IZDANJA nisu potpuno funkcionalno ovisni o primarnom ključu. Ovi atributi ovise samo o dijelu ključa, konkretno o ISBN, a ne o kombinaciji ČLANSKI BROJ. Takva djelomična ovisnost ukazuje na potencijalnu redundantnost podataka i može biti znak za normalizaciju relacije.

4.1.3. Parcijalne ovisnosti, prevođenje relacije u drugu normalnu formu

Vidjeli smo da u prethodnoj relaciji POSUDIO postoje atributi NASLOV KNJIGE, AUTOR KNJIGE, i GODINA IZDANJA koji nisu potpuno funkcionalno ovisni o primarnom ključu (ČLANSKI BROJ, ISBN). 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. Kako 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 POSUDIO nije u 2NF jer, kao što smo već rekli, u njoj postoje ove parcijalne ovisnosti:

ISBN → NASLOV KNJIGE

ISBN → AUTOR KNJIGE

ISBN → GODINA IZDANJA..

U skladu s prethodno rečenim, relacija koja nije u 2NF loše je oblikovana te se preporučuje 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 se relaciju *prepisuje* i odgovarajući dio primarnog ključa pa on postaje ključ u toj novoj relaciji.

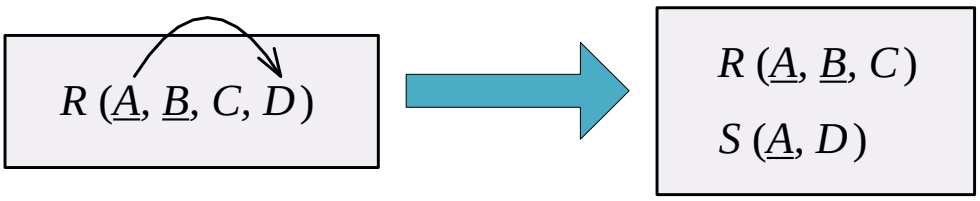
U našem primjeru, postupak prevođenja relacije POSUDIO u drugu normalnu formu (2NF) razbit će polaznu relaciju na sljedeće dvije relacije:

POSUDIO (ČLANSKI BROJ, ISBN, DATUM POSUDBE)

KNJIGA (ISBN, NASLOV KNJIGE, AUTOR KNJIGE, GODINA IZDANJA)

U polaznoj relaciji POSUDIO sve parcijalne ovisnosti proizlaze iz istog dijela ključa, a to je ISBN. Kao rezultat, postupak prevođenja stvara novu relaciju KNJIGA, u koju su premješteni svi atributi koji su parcijalno ovisni o ISBN (tj. NASLOV KNJIGE, AUTOR KNJIGE, i GODINA IZDANJA). ISBN postaje primarni ključ nove relacije KNJIGA, jer jednoznačno identificira knjige u knjižnici.

Nakon opisanog prevođenja obje su relacije u 2NF. Naime, u preostaloj verziji POSUDIO više nema parcijalnih ovisnosti, pa ona zadovoljava definiciju 2NF. Također, relacija KNJIGA ima jednostavan ključ (ISBN) 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 ponoviti.

4.1.4. Tranzitivne ovisnosti, prevođenje relacije u treću normalnu formu

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

ISBN → OIB AUTORA → IME AUTORA.

Takav niz funkcionalnih ovisnosti, pod uvjetom da srednji atribut (OIB AUTORA) nije kandidat za ključ, nazivamo *tranzitivna* ovisnost. U ovom slučaju, IME AUTORA je tranzitivno ovisan o ISBN, posredstvom OIB AUTORA. Pritom OIB AUTORA nije kandidat za ključ u relaciji KNJIGA, jer isti autor može napisati više knjiga, a ISBN je jedinstven za svaku knjigu.

Slično kao i parcijalna ovisnost, i tranzitivna se ovisnost 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 KNJIGA nije u 3NF jer imamo funkcionalnu ovisnost

OIB AUTORA → IME AUTORA,

a pritom OIB AUTORA nije ključ, a IME AUTORA nije primarni atribut.

Opet u skladu s prethodno rečenim, relacija koja nije u 3NF loše je oblikovana pa se preporučuje da se ona prevede u 3NF. Postupak prevođenja u 3NF sličan je postupku za 2NF. Dakle, polazna relacija se razbija u barem dvije nove relacije, tako da se prekinu nepoželjne tranzitivne ovisnosti, a da pritom ne dođe do gubitka informacija. 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 se tranzitivnim ovisnostima pojavljuje isti srednji atribut.
- Uz prebačene atribute, u novu se relaciju *prepisuje* i odgovarajući srednji atribut pa on postaje ključ u toj novoj relaciji.

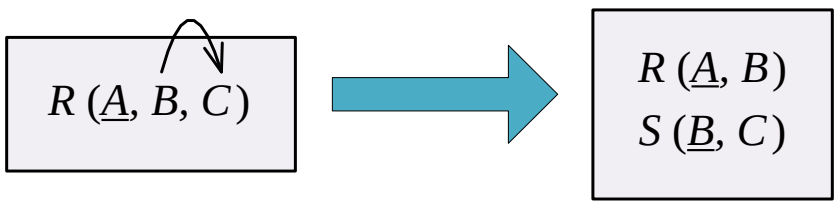
U našem konkretnom primjeru, postupak prevođenja u 3NF razbija polaznu relaciju KNJIGA u ove dvije relacije:

KNJIGA (ISBN, NASLOV KNJIGE, OIB AUTORA, GODINA IZDANJA)

AUTOR (OIB AUTORA, IME AUTORA).

Naime, u polaznoj inačici od KNJIGA postojala je samo jedna tranzitivna ovisnost sa srednjim atributom OIB AUTORA. Zato postupak prevođenja stvara samo jednu novu relaciju i u nju je prebačen tranzitivno-ovisni atribut IME AUTORA zajedno s OIB AUTORA. Vidimo da OIB AUTORA postaje ključ u novoj relaciji i da nova relacija zapravo opisuje autora te u skladu s time dobiva ime AUTOR.

Nakon opisanog prevođenja, obje relacije su u 3NF. Naime, u preostaloj verziji KNJIGA više nema tranzitivnih ovisnosti pa ona zadovoljava definiciju 3NF. Također, AUTOR 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 ponoviti.

4.2. Boyce-Coddova 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-ih godina 20. stoljeća, Edgar Codd je definirao pojačanu varijantu 2NF i 3NF koja se zove Boyce-Coddova 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 učinkovita zamjena za 2NF i 3NF.

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

Definicija Boyce-Coddove 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 u nekoj relaciji o kojoj je neki drugi atribut u istoj relaciji 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 koju smo promatrali kad smo govorili o 2NF i 3NF, dakle:

POSUDIO (ČLANSKI BROJ, ISBN, DATUM POSUDBE, NASLOV KNJIGE, AUTOR KNJIGE, OIB AUTORA).

Ponovo uočavamo ove funkcionalne ovisnosti:

ISBN → NASLOV KNJIGE

ISBN → AUTOR KNJIGE

NASLOV KNJIGE → GODINA IZDANJA.

Vidimo da postoji jedna determinanta: ISBN, koji determinira NASLOV KNJIGE i AUTOR KNJIGE, ali imamo tranzitivnu ovisnost NASLOV KNJIGE → GODINA IZDANJA, gdje NASLOV KNJIGE nije kandidat za ključ. Dakle, relacija nije u BCNF.

Slično kao prije, relacija koja nije u BCNF smatra se loše oblikovanom i stoga mogućim izvorom poteškoća. Preporučuje 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 takvih različitih determinanti.
- Iz polazne relacije *izbacuju se i prebacuju* u nove svi oni atributi koji su ovisni o nekoj od tih determinanti.

- Pritom u jednu novu relaciju idu oni atributi koji su ovisni o istoj determinanti.
- Uz prebačene atribute, u novu se relaciju *prepisuje* i sama determinanta, pa ona postaje ključ u toj novoj relaciji.

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

POSUDIO (ČLANSKI BROJ, ISBN, DATUM POSUDBE)

KNJIGA (ISBN, NASLOV KNJIGE, AUTOR KNJIGE)

AUTOR (OIB AUTORA, GODINA IZDANJA).

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 učinkovitu i kompaktnu zamjenu za 2NF i 3NF.

4.2.2. Odnos Boyce-Coddove 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 uopćava razne oblike funkcionalnih ovisnosti koje smo prije razmatrali. I parcijalne i tranzitivne ovisnosti zapravo određuju neku vrstu determinante koja nije kandidat za ključ. Zato BCNF u sebi uključuje 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 je u BCNF, mora nužno biti i u 2NF i u 3NF.

Na temelju 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 dogoditi da relacija koja je u 2NF i 3NF ipak nije u BCNF. Drugim riječima, BCNF se može smatrati „trećom-i-pol normalnom formom“.

Primjeri relacija koje su u 2NF i 3NF, a 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 knjižnicu u kojoj jednu knjigu može napisati više autora, a svaki autor može napisati više knjiga. Svaki član knjižnice može posuditi više knjiga. Godina izdanja knjige ovisi o autoru i specifična je za svaku knjigu. Situacija je slična onoj u kojoj više autora piše knjige iste tematike, ali svaka knjiga ima specifičnu godinu izdanja. Sve navedeno može se opisati relacijom, uz sljedeće pretpostavke: da ČLANSKI BROJ jednoznačno identificira člana, ISBN knjigu, a AUTOR KNJIGE autora:

POSUDIO (ČLANSKI BROJ, ISBN, AUTOR KNJIGE, GODINA IZDANJA).

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

AUTOR KNJIGE → GODINA IZDANJA.

No primarni ključ se može se odabrati i na drugačiji način. U tom slučaju, relacija izgleda ovako:

POSUDIO (ČLANSKI BROJ, ISBN, AUTOR KNJIGE, GODINA IZDANJA).

Ova relacija je u 2NF i 3NF, ali ne i u BCNF jer i dalje postoji ovisnost:

AUTOR KNJIGE → GODINA IZDANJA.

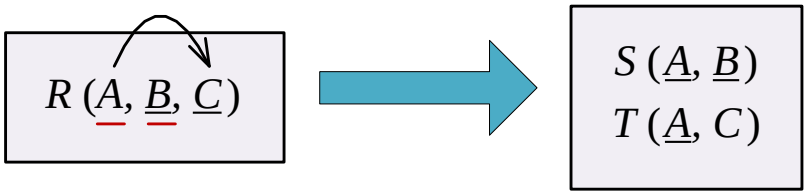
Pritom, determinanta AUTOR KNJIGE nije kandidat za ključ, jer može postojati više knjiga koje je napisao isti autor . Dakle, imamo primjer relacije koja je u 2NF i 3NF, ali nije u BCNF.

Ako se na posljednju inačicu relacije POSUDIO primijeni postupak prevođenja u BCNF, relacija se raspada na dvije nove relacije:

POSUDBA (ČLANSKI BROJ, ISBN)

KNJIGA (ISBN, AUTOR KNJIGE, GODINA IZDANJA).

Obje novonastale relacije sada su u BCNF. Pritom smo im dali imena koja najbolje opisuju njihovo značenje. Iz prve relacije vidi se koji član knjižnice je posudio koju knjigu, Iz druge relacije vidi se koji autor je napisao određenu knjigu i u kojoj godini je izdana.



Slika 4.5: Shematski prikaz prevođenja u BCNF

Postupak prevođenja u BCNF za slučaj relacije koja je 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. Lako se može 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 temelju podataka na Slici 4.6 stječe se dojam da vrijedi pravilo: *čim kompanija izvozi u neku zemlju, u tu zemlju izvozi sve svoje proizvode*. Ako prihvatimo da vrijedi takvo pravilo, tada postaje očito da relacija IZVOZI mora sadržavati veliku dozu redundancije.

Uočena redundancija može se eliminirati tako da se polazna relacija IZVOZI zamijeni 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.

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

Dosadašnja pravila normalizacije ne pomažu za uklanjanje redundancije 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.

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 ove višeznačne ovisnosti:

IME KOMPANIJE →→ IME PROIZVODA,

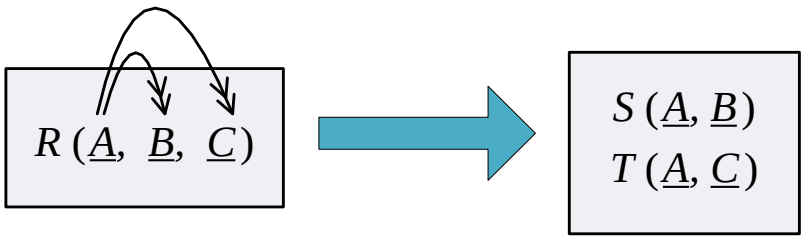
IME KOMPANIJE →→ IME ZEMLJE.

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

Iz upravo objašnjenih razloga, višeznačna ovisnost smatra se nepoželjnom pojavom. Da bi se takva ovisnost mogla ukloniti, 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* →→ *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. Dakle, IZVOZI nije u 4NF i zato je treba rastaviti na RADI i PRODAJE. Te dvije jednostavnije relacije sigurno su u 4NF, jer svaka od njih ima po dva atributa.



Slika 4.8: Shematski prikaz prevođenja u 4NF

Primjer s relacijom IZVOZI otkriva nam i općeniti 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.

To je shematski prikazano Slikom 4.8.

4.3. Potreba za normalizacijom

Normalizacija je u prvom redu potrebna zato što se njome izbjegavaju teškoće koje bi nastupile kad bismo radili s nenormaliziranim podacima.

Normalizacija je korisna i zato što se njome naknadno otkrivaju i ispravljaju pogreške 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 tezama.

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 inačicu relacije POSUDIO iz Odjeljka 4.1.3. koja nije u 2NF. Primijetimo da ta relacija, osim što bilježi tko je posudio koju knjigu, također pruža informacije i o knjigama i autorima:

POSUDIO (ČLANSKI BROJ, ISBN, NASLOV KNJIGE, AUTOR KNJIGE, GODINA IZDANJA).

Do teškoća u radu s relacijom POSUDIO dolazi kad preko nje pokušavamo upravljati podacima o knjigama ili članovima. Na primjer:

- Ako želimo unijeti podatke o novoj knjizi u bazu, to ne možemo učiniti sve dok barem jedan član ne posudi tu knjigu (naime, ne smijemo imati praznu vrijednost za primarni atribut ČLANSKI BROJ). Slično tome, ako želimo unijeti podatke o novom autoru, to ne možemo učiniti dok taj autor ne napiše barem jednu knjigu koju mora posuditi barem jedan član.
- Ako želimo promijeniti naslov postojeće knjige, moramo pronaći sve n-torke koje sadrže odgovarajući ISBN i promijeniti vrijednost za NASLOV KNJIGE u svim tim n-torkama. Broj promjena bit će jednak broju članova koji su posudili tu knjigu. Ako zaboravimo izvršiti neku promjenu, doći će do kontradiktornih podataka.
- Pretpostavimo da svi članovi koji su posudili neku knjigu vrate tu knjigu i više je ne posuđuju. Ako potom izbrišemo odgovarajuće n-torke, iz baze će nestati svi podaci o toj knjizi.

Promatrajmo dalje relaciju KNJIGA iz Odjeljka 4.1.4. koja nije u 3NF. Primijetimo da ta relacija, osim o knjigama, govori i o autorima:

KNJIGA (ISBN, NASLOV KNJIGE, AUTOR KNJIGE, GODINA IZDANJA).

Do teškoća u radu s relacijom KNJIGA dolazi upravo kada preko nje pokušavamo mijenjati podatke o autorima Na primjer:

- Ne možemo unijeti podatke o novom autoru sve dok on nije napisao barem jednu knjigu.
- Ako želimo promijeniti godinu izdanja određene knjige koju je napisao autor, moramo izvršiti promjenu u svakoj n-torki koja odgovara knjigama tog autora. Ako zaboravimo promijeniti podatke za neku od knjiga, imat ćemo kontradiktorne podatke.
- Ako autor privremeno ne napiše nijednu knjigu, tada iz baze nestaju svi podaci o tom autoru.

Promatrajmo zatim relaciju POSUDIO iz Odjeljka 4.2.2, koja nije u BCNF. Primijetimo da ta relacija, osim što bilježi tko je posudio koju knjigu, također sadrži informacije o knjigama i autorima:

POSUDIO (ČLANSKI BROJ, ISBN, AUTOR KNJIGE).

Do teškoća u radu s ovom inačicom relacije POSUDIO ponovno dolazi kada pokušavamo raditi s podacima o knjigama i autorima. Na primjer:

- Ne možemo evidentirati činjenicu da zadani autor piše zadanu knjigu sve dok bar jedan član knjižnice ne posudi tu knjigu koju je napisao taj autor.
- Veza između autora i knjige zapisana je s velikom redundancijom, onoliko puta koliko ima članova koji su posudili tu knjigu, što otežava ažuriranje podataka.
- Ako svi članovi koji su posudili knjigu određenog autora vrate tu knjigu, briše se evidencija da je taj autor napisao tu knjigu.

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 (NAZIV 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 pogrešaka

U prethodnom odjeljku uočili smo da svi nenormalizirani podaci imaju jednu zajedničku osobinu: pokušavaju govoriti o više stvari u isto vrijeme. Neki od njih pokušavaju istovremeno opisati više tipova entiteta, drugi bilježe vezu između entiteta no istovremeno navode i svojstva samih entiteta, treći istovremeno zapisuju više veza. To nije u skladu s postupkom projektiranja opisanom u drugom i trećem poglavlju. Naime, taj postupak, ako 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 pogreške u postupku projektiranja. Izvor takve pogreške 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 i odrediti pogreške 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. Kad bismo poštovali to pravilo, podzapis s osobnim podacima suradnika odmah bismo razbili u više jednostavnih atributa, a ponavljajuću skupinu o djetetu proglasili bismo zasebnim entitetom te 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 POSUDIO iz Odjeljka 4.1.3., koja nije u 2NF, nastala je jer je događaj posuđivanja knjige od člana knjižnice pogrešno interpretiran kao zaseban tip entiteta s vlastitim atributima. Umjesto toga, trebalo je prepoznati da su tipovi entiteta zapravo članovi knjižnice i knjige, te da je posuđivanje samo veza između tih tipova. Također, trebalo je uočiti postojanje tipa entiteta za autore, pri čemu posuđivanje knjige predstavlja vezu između autora i knjiga.

Postupak prevođenja iz Odjeljaka 4.1.3. i 4.1.4., najprije u 2NF, a zatim u 3NF, postupno ispravlja ove pogreške. Naime, tim postupkom relacija POSUDIO reducira se u oblik koji služi isključivo za bilježenje veze između članova knjižnice i knjiga. Također nastaju nove relacije KNJIGA i AUTOR, koje odgovaraju istoimenim entitetima. Veza između knjiga i autora ispravno se realizira preko stranog ključa u relaciji KNJIGA. Iako ne dobivamo posebnu relaciju za članove knjižnice, to je zato što u početnim podacima nismo imali nikakve atribute za članove osim ČLANSKI BROJ.

Relacija POSUDIO iz Odjeljka 4.2.2., koja je u 2NF i 3NF, ali nije u BCNF, nastala je jer smo odnose između članova knjižnice, knjiga i autora pogrešno tumačili kao ternarnu vezu. Zapravo se radilo o dvije nezavisne binarne veze. Da smo odmah prepoznali te binarne veze, podaci bi umjesto relacijom POSUDIO bili prikazani relacijama POSUDBA i KNJIGA, koje smo 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 dobili kao rezultat prevođenja u 4NF.

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

To naravno ne znači da je postupak normalizacije nepotreban, baš naprotiv. Naime, pogreške su uvijek moguće i nikad ne znamo je li do njih došlo ili nije. Normalizacija predstavlja mehanizam naknadnog prepoznavanja i ispravljanja pogrešaka. Svođenjem u normalnu formu pogrešno oblikovane relacije ponovo se vraćaju u oblik koje bi imale da pogrešaka nije bilo. Zahvaljujući normalizaciji, postupak projektiranja postaje proces koji se sam ispravlja, gdje se pogreške 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**. Događa se da nekoliko atributa u relaciji čini cjelinu koja se u aplikacijama nikad ne rastavlja na dijelove. Na primjer, promatrajmo 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 znamo da POŠTANSKI BROJ, IME GRADA i 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. Ne preporučuje se razbijati tu relaciju na dvije.

- **Učinkovita 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 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 koristiti. Drugim riječima, treba utvrditi mogu li se ili ne mogu teškoće u radu s nenormaliziranim podacima, koje smo opisali u Odjeljku 4.3.1, zaista dogoditi u tom konkretnom slučaju. Također, treba predvidjeti hoće li se zahtijevati izuzetno velika brzina kod pronalaženja podataka.

4.4. Vježbe

Zadatak 4.1. Rješavanjem Zadatka 3.1 dobili ste nadopunjenu relacijsku shemu baze podataka o knjižnici. 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 teretani. 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.

ISPORUKA

BROJ ISPORUKE	DATUM SLANJA	BROJ KUPCA	IME KUPCA	ADRESA KUPCA	BROJ PROIZ-VODA	IME PROIZ-VODA	KOMADA	CIJENA PO KOMADU

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 ovom relacijom:

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

  Ako je potrebno, prevedite tu relaciju u 3NF.

- **Zadatak 4.5.** Suradnici neke ustanove rade na različitim projektima. Pritom jedan suradnik radi na točno jednom projektu. Situacija je opisana ovom relacijom:

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

  Ako je potrebno, prevedite tu relaciju u 3NF.

- **Zadatak 4.6.** Na fakultetu se nastava iz jednog predmeta održava uvijek u istoj predavaonici, ali u nekoliko termina tjedno. Situacija je opisana ovom relacijom:

  RASPORED (BROJ PREDAVAONICE, TERMIN,
  ŠIFRA PREDMETA).

  Ako je potrebno, prevedite tu relaciju u BCNF.

- **Zadatak 4.7**. Studenti upisuju izborne predmete iz matematike i izborne predmete iz računarstva. Ne postavljaju se nikakvi uvjeti na izbor jednih u odnosu na druge. Situacija je opisana ovom relacijom:

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

  Prevedite tu relaciju u 4NF ako je potrebno.

Dodatni zadatak:

- **Zadatak 4.8.** Rješavanjem Zadatka 3.3 dobili ste relacijsku shemu za bazu podataka iz svojeg područja interesa. Normalizirajte tu shemu tako da sve relacije budu barem u 3NF.
- **Zadatak 4.9**. Relacija za upravljanje posudbama u knjižnici se definira kao: Posudba (Id_Posudba, Datum_Posudbe, Datum_Povrata, ID_Člana, Ime_Člana, Prezime_Člana, ID_Knjige, Naslov_Knjige, Autor, Žanr).
  Analizirajte relaciju kako biste identificirali funkcionalne zavisnosti. Provedite normalizaciju:
  - Pretvorite relaciju u 1NF uklanjanjem složenih atributa
  - Razdvojite relaciju u 2NF eliminirajući parcijalne zavisnosti
  - Provedite normalizaciju do 3NF uklanjanjem tranzitivnih zavisnosti

Zadatak 4.10 Proučite sljedeću relaciju o upravljanju narudžbama: Narudžba (ID_Narudžbe, Datum, ID_Kupca, Naziv_Kupca, ID_Artikla, Naziv_Artikla, Količina, Ukupna_Cijena)

- Identificirajte funkcionalne zavisnosti među atributima.
- Pretpostavite da Ukupna_Cijena nije spremljena, već da se računa. Kako biste uklonili redundanciju u relaciji?
- Predložite novu strukturu za relacije koja ne sadrži redundanciju, omogućava jednostavnije proširenje (npr. popusti, porezi) i podržava normalizaciju do 3NF.

5. Projektiranje na fizičkoj razini

5. dan

3:15

Predavanja: 3 x 45 min.
Vježbe: 1 x 45 min.
Pauza: 1x15 min.

Po završetku ovog poglavlja moći ćete:

- *objasniti osnovne elemente fizičke baze podataka*
- *analizirati različite organizacije datoteka u fizičkom dizajnu baze podataka*
- *opisati postupak uvođenja ograničenja za očuvanje integriteta baze podataka*
- *definirati mehanizme oporavka baze podataka u slučaju oštećenja.*

U ovom poglavlju govorimo o trećoj fazi projektiranja baze podataka, a to je projektiranje na fizičkoj razini. Glavni je cilj te faze stvoriti *fizičku shemu* baze, dakle opis njezine fizičke građe. Fizička shema zapravo je tekst sastavljen od naredbi u SQL-u ili nekom drugom jeziku koji razumije DBMS. Izvođenjem tih naredbi DBMS stvara fizičku građu baze.

U prvom potpoglavlju ovog poglavlja najprije opisujemo fizičku građu baze i način kako da se dođe do početne inačice fizičke sheme. Početna inačica već implementira sve relacije iz logičke sheme i osigurava potrebnu brzinu pristupa do podataka. U daljnja dva potpoglavlja opisujemo načine kako da se fizička shema dalje dogradi u svrhu čuvanja integriteta baze, odnosno u svrhu postizavanja sigurnosti podataka.

5.1. Fizička građa baze

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 kako DBMS početnu inačicu fizičke sheme, sastavljenu od SQL-naredbi CREATE TABLE i CREATE INDEX, pretvara u fizičku bazu.

5.1.1. Elementi fizičke građe

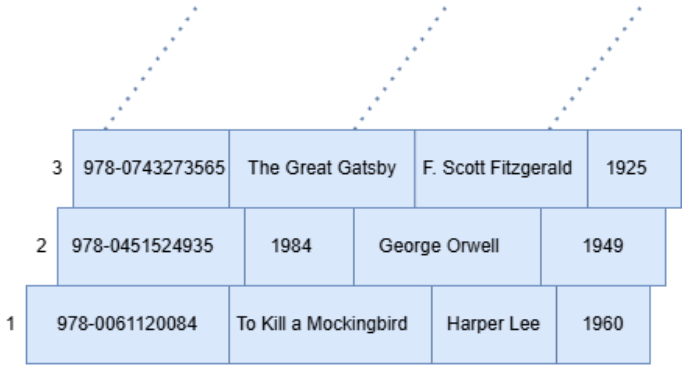
Baza podataka fizički se pohranjuje na vanjskoj memoriji računala, obično na SSD ili HDD diskovima. Važno je znati da operacijski sustav računala dijeli vanjsku memoriju u jednako velike *blokove* (sektore). Veličina bloka je konstanta operacijskog sustava i ona može iznositi na primjer 512 bajta ili 4096 bajta. 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 bajt iz vanjske memorije, tada moramo prenijeti cijeli odgovarajući blok, pretražiti *buffer* u glavnoj memoriji i izdvojiti traženi bajt. 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 još starijih programskih jezika poput C 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 itd.). 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 bajtova. 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 KNJIGA sa Slike 3.1 koja sadrži podatke o knjigama u knjižnici. Da bi fizički prikazali tu relaciju, svaku njezinu n-torku pretvaramo u zapis, te zapise poredamo u nekom redoslijedu pa ih pohranimo na disk. Tako dobivamo niz zapisa pohranjenih na disk, dakle datoteku. Ideja je ilustrirana Slikom 5.1. Pogodan tip zapisa za podatke o knjigama mogao bi se precizno definirati u programskom jeziku C#, Java ili Python, te bi se na primjer sastojao od: niza od 50 znakova (NASLOV), niza od 30 znakova (AUTOR), niza od 13 znamenki (ISBN), jednog cijelog broja (GODINA IZDANJA).



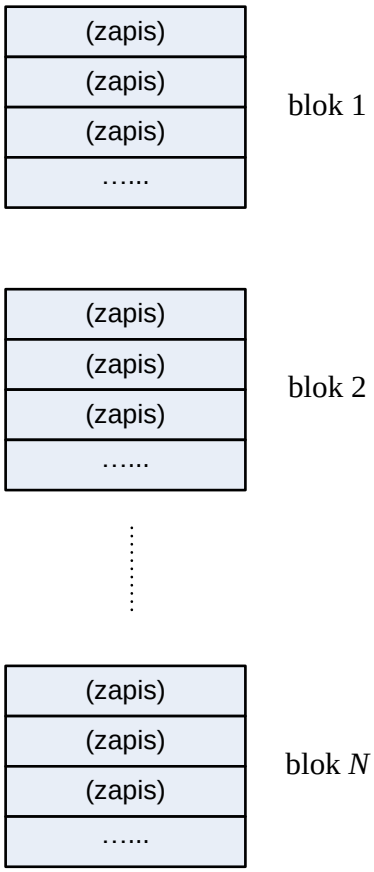
Slika 5.1: Datoteka s podacima o knjigama u knjižnici.

Primijetimo da smo prilikom pretvorbe relacije KNJIGA u datoteku nužno morali uvesti brojne fizičke detalje koji nisu postojali u relacijskom modelu, na primjer točnu duljinu pojedinog podatka u bajtovima, redoslijed podataka u zapisu, 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 u datoteci. Ako ima više kandidata za ključ, tada odabiremo jedan 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 koji 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 na

to 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. Takav način pohranjivanja omogućuje da jednoznačno odredimo položaj zapisa na disku. Naime, *adresa zapisa* gradi se kao uređeni par adrese bloka i pomaka u bajtovima unutar bloka.



Slika 5.2: 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 5.2. Položaj i redoslijed blokova koji čine istu datoteku određen je posebnim pravilima koja čine takozvanu *organizaciju* datoteke. U svakom slučaju, ti se blokovi 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 važan element fizičke građe, a to je *pokazivač* (*pointer*). Riječ je o podatku u zapisu ili bloku 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 na implicitan način pokazuju na zapis koji treba pokazati, na primjer navođenjem odgovarajuće vrijednosti primarnog ključa. Pokazivači se obilato koriste 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.

5.1.2. 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 mane u pogledu učinkovitog obavljanja osnovnih operacija nad datotekom.

Jednostavna datoteka. Zapisi su poredani u onoliko blokova koliko je potrebno. Ti su blokovi međusobno povezani u vezanu listu, dakle svaki blok sadrži fizički pokazivač na idući blok. Kao adresu cijele datoteke pamtimo adresu prvog bloka. Ova organizacija koristi se u specifičnim slučajevima, poput arhivskih datoteka ili linearne obrade podataka. Struktura je prikazana na Slici 5.3.

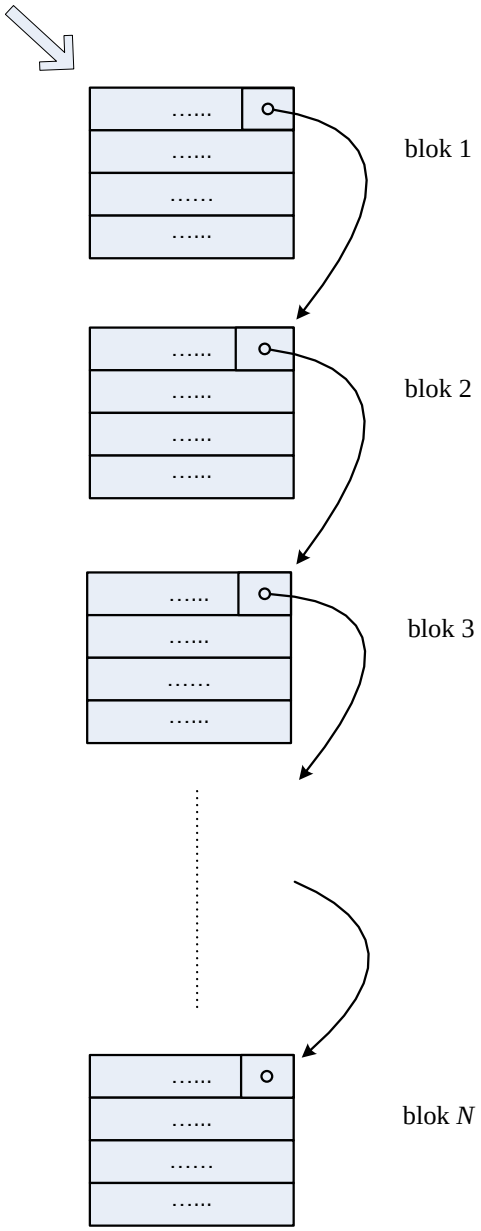
Prednost jednostavne organizacije je da se kod nje lagano ubacuju, izbacuju i mijenjaju zapisi. No mana je da bilo kakvo traženje, na primjer traženje zapisa sa zadanom vrijednošću primarnog ključa, zahtijeva sekvencijalno čitanje blok po blok. To znači da će jedno traženje u prosjeku zahtijevati čitanje 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 koji treba spremiti zapis s vrijednošću ključa *k*. Ista funkcija kasnije omogućuje i brzo pronalaženje zapisa sa zadanom vrijednošću ključa. Ova se struktura koristi u distribuiranim bazama podataka poput Apache Cassandra i Amazon DynamoDB, gdje se hash funkcije primjenjuju za raspodjelu podataka među čvorovima.

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 5.4.

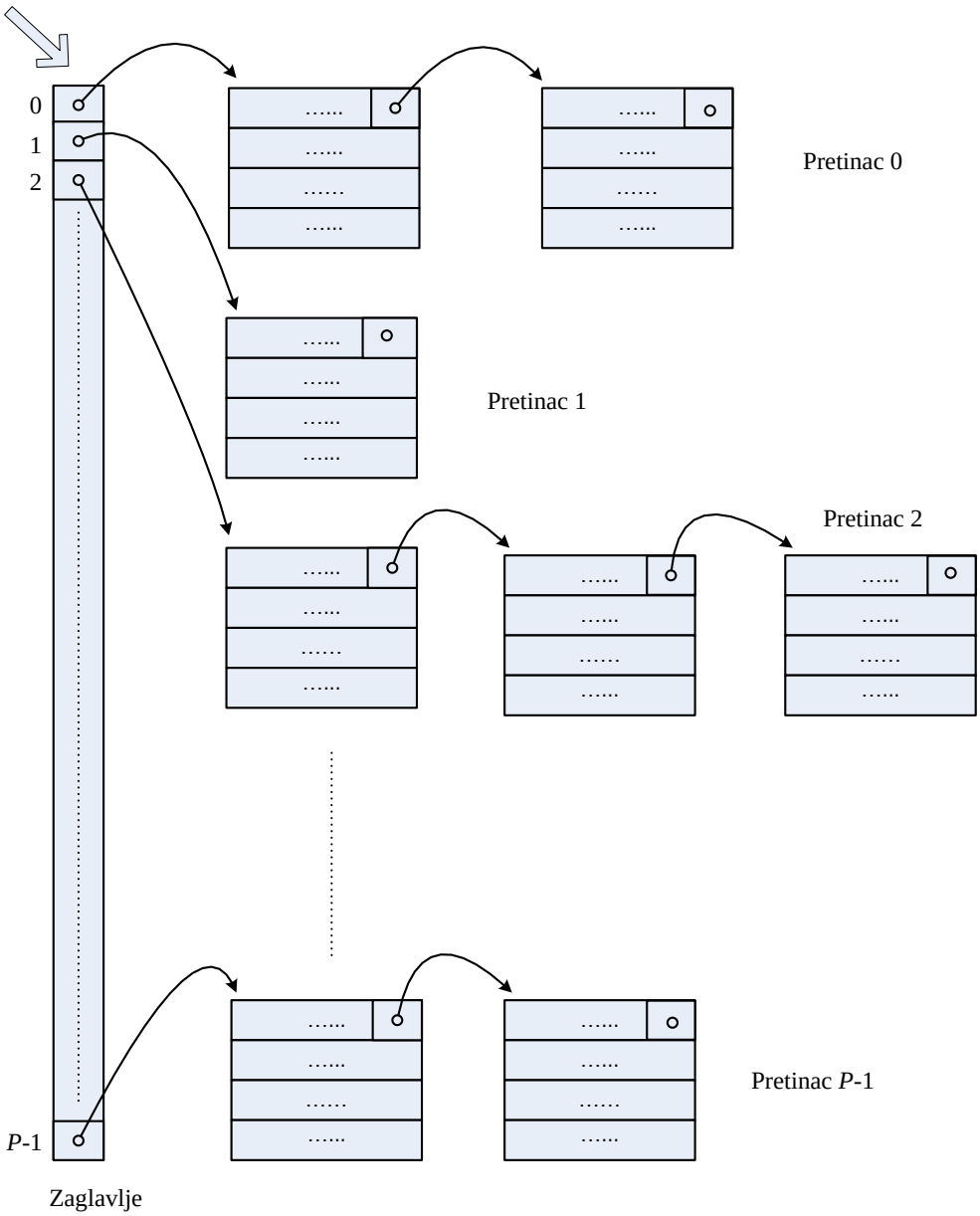
Skup mogućih vrijednosti ključa obično je znatno veći od broja pretinaca. Zato je važno da *h*() uniformno (jednoliko) distribuira vrijednosti ključa na pretince. Tada se naime neće događati da se pretinci neravnomjerno pune pa će sve vezane liste biti podjednako kratke. Primjer dobre *hash*-funkcije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



Slika 5.3: Jednostavna datoteka

Hash-datoteka može se smatrati dijametralno suprotnom organizacijom od jednostavne. Naime, dok jednostavna organizacija dopušta jedino sekvencijalni pristup, prednost je *hash*-organizacije u tome što ona omogućuje gotovo izravni pristup na osnovi ključa. Da bismo pronašli zapis sa zadanom vrijednošću ključa *k*, najprije računamo *h*(*k*), a zatim pretražimo samo *h*(*k*)-ti pretinac. Ako je *h*() zaista uniformna i 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.



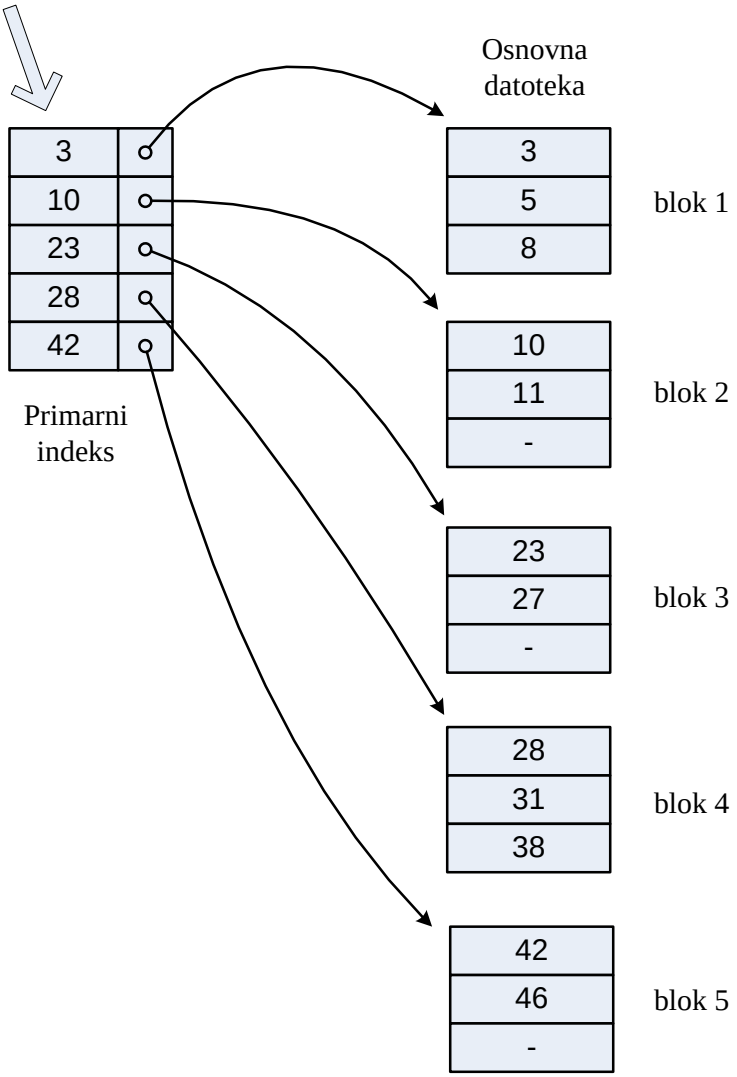
Slika 5:4: *Hash*-datoteka

Daljnje organizacije datoteke koje ćemo opisati zasnivaju se na tome da se uz osnovnu datoteku s podacima koristi 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) itd.

Indeks-sekvencijalna datoteka je najpopularnija organizacija zasnovana na indeksu. Riječ je o osnovnoj datoteci koja je organizirana jednostavno i kojoj je radi bržeg traženja po primarnom ključu dodan primarni indeks. Građa je prikazana Slikom 5.5. 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.

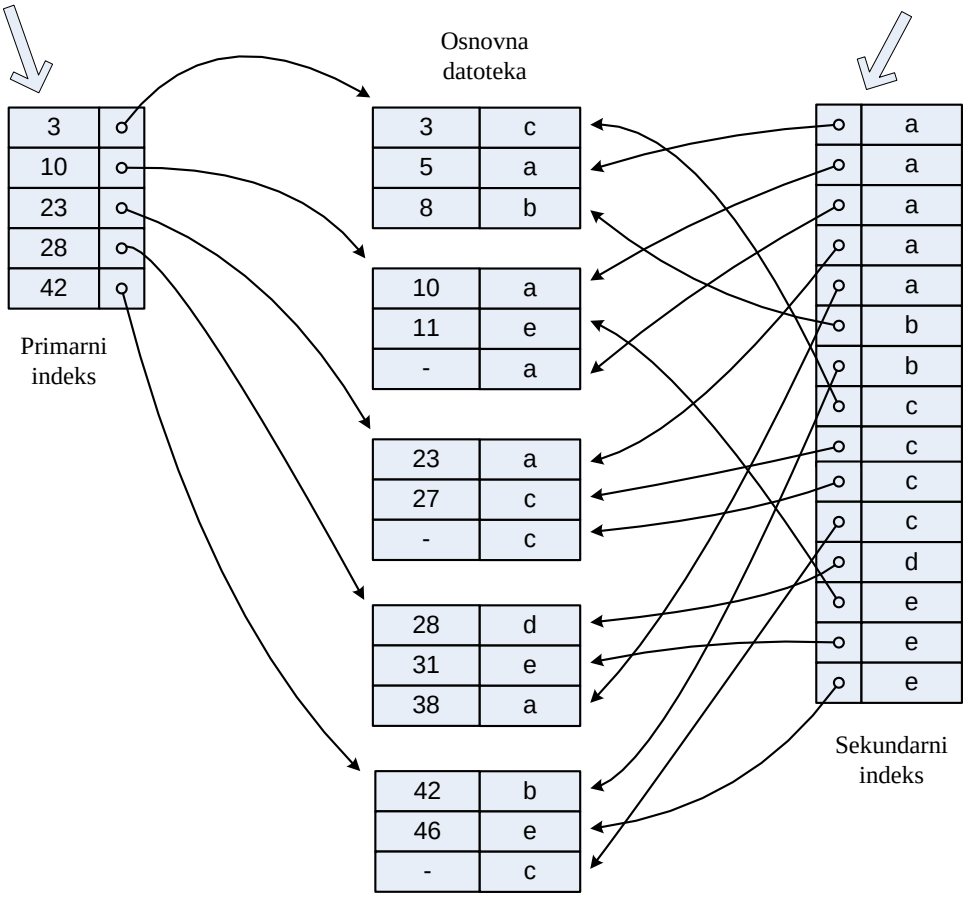


Slika 5.5: Indeks-sekvencijalna datoteka

Prednost indeks-sekvencijalne organizacije je da ona omogućuje prilično brz pristup na osnovi 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 i 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 redoslijedom kako su upisane u indeksu. Također, moguće je jednostavno pronaći zapise čija se vrijednost ključa nalazi u određenom intervalu, što je posebno korisno u analitičkim bazama podataka.

Dakle indeks-sekvencijalna datoteka predstavlja dobar kompromis između jednostavne i *hash*-datoteke pa je to razlog zašto je ona toliko popularna.

Nedostatak je indeks-sekvencijalne organizacije 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 5.6: Invertirana 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*, jer sadrži pokazivač na svaki zapis iz osnovne datoteke. Ideja je ilustrirana Slikom 5.6. 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 se podacima dakle može pristupiti na dva različita načina.

Prednost invertirane organizacije je brz pristup po više kriterija te mogućnost sortiranog ispisa ili intervalnog pretraživanja po svim tim kriterijima. Ovaj se tip organizacije široko koristi u sustavima pretraživanja poput Elasticsearcha, gdje se sekundarni indeksi primjenjuju za brzu pretragu tekstualnih podataka. Nedostatak je da se ažuriranje podataka još više komplicira i troši se još više dodatnog prostora na disku.

5.1.3. 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 također mora biti organiziran na odgovarajući način. U ovom odjeljku opisat ćemo uobičajeni način fizičkog prikazivanja indeksa pomoću takozvanog B-stabla. Ova se struktura danas često koristi u sustavima upravljanja bazama podataka, kao što su MySQL-a i PostgreSQL-a, za implementaciju primarnih i sekundarnih indeksa. Riječ je o hijerarhijskoj strukturi podataka koja omogućuje da indeks zaista učinkovito obavlja svoje osnovne zadaće, a to su brzo pronalaženje zadane vrijednosti podatka te čuvanje sortiranog redoslijeda svih vrijednosti. Počinjemo s definicijom B-stabla.

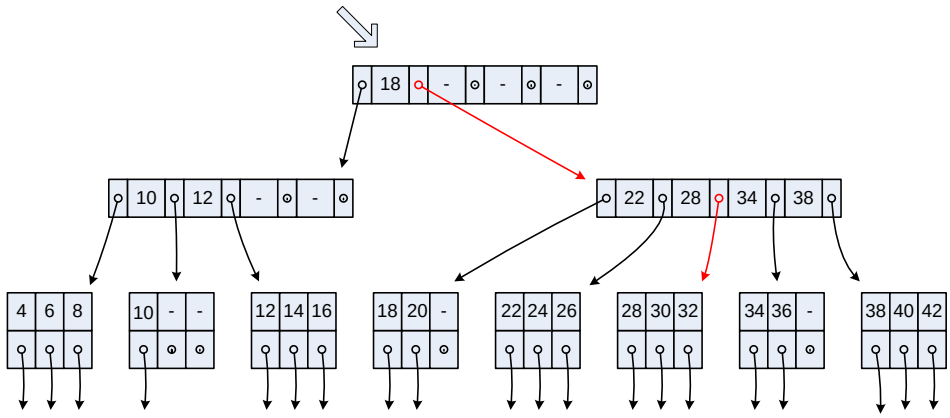
B-stablo reda m je m-narno stablo s ovim svojstvima:

- 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 je ostalih vrsta indeksa vrlo sličan. B+ stablo, koje je proširenje B-stabla, danas je najčešće korišteno zbog optimizacija u traženju i radu s diskovima. 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

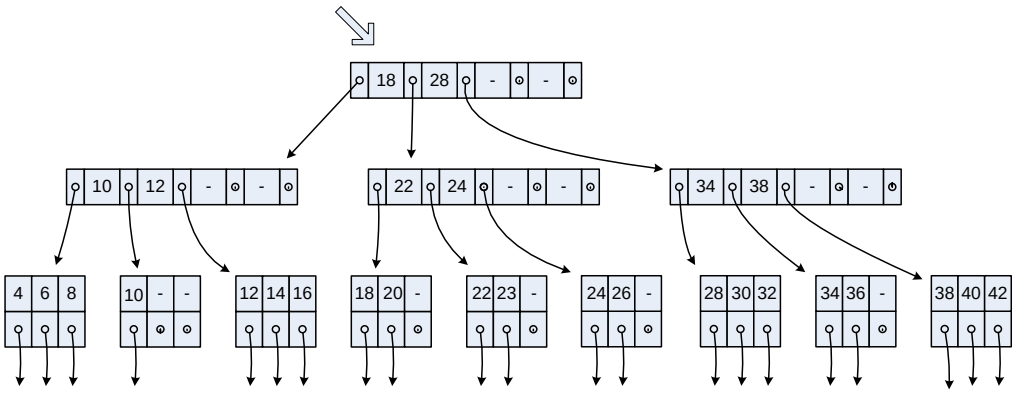
- Unutrašnji čvor ima sadržaj oblika (p_0, k_1, p_1, k_2, p_2, … , k_r, p_r), gdje je p_i pokazivač na i-to dijete dotičnog čvora ($0 \le i \le r$), k_i je vrijednost ključa ($1 \le i \le r$). Vrijednosti ključa u čvoru su sortirane, dakle $k_1 \le k_2 \le \ldots \le k_r$. Sve vrijednosti ključa u podstablu koje pokazuje p_0 su manje od k_1. Za $1 \le i < r$, sve vrijednosti ključa u pod-stablu koje pokazuje p_i su u poluotvorenom intervalu $[k_i, k_{i+1})$. Sve vrijednosti ključa u podstablu 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 u listu sortirani su uzlazno prema k. Listovi B-stabla često su međusobno povezani radi optimiziranog sekvencijalnog čitanja, što je ključno za obradu velikih količina podataka. Jednom zapisu osnovne datoteke odgovara točno jedan par (k, p) u listovima B-stabla.



Slika 5.7: Gusti primarni indeks prikazan kao B-stablo reda 5

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, … , k_r, p_r) i usporedimo k s k_1, k_2, … , 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 $k \geq k_r$, rabimo adresu 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 5.7 tražimo vrijednost ključa 30, proći ćemo put koji je na slici označen crvenim strelicama.

Učinkovitost prikaza indeksa pomoći B-stabla počiva na činjenici da u stvarnim situacijama B-stablo nikad nema preveliku visinu, to jest sastoji se od 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 tijekom pretraživanja. B+ stablo dodatno optimizira ovaj proces pružajući sve pokazivače na listovima, što omogućuje učinkovitije sekvencijalno pretraživanje.



Slika 5.8: Ubacivanje vrijednosti 23 u B-stablo s prethodne slike

Za razliku od traženja koje se odvija brzo i učinkovito, ubacivanje podataka u B-stablo komplicirana je 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. Ova

operacija je posebno optimizirana u modernim DBMS-ovima, gdje se koristi *write-ahead logging* kako bi se osigurala transakcijska sigurnost i integritet podataka.

Slika 5.8 prikazuje B-stablo s prethodne Slike 5.7 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. Suvremeni sustavi poput PostgreSQL-a i MySQL-a osiguravaju učinkovito upravljanje takvim promjenama kroz algoritme prilagođene radu s velikim količinama podataka.

5.1.4. Početno oblikovanje fizičke građe

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 je opis samo implicitan, jer 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.

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 čini primarni ključ te relacije i smije li ili ne smije neki atribut imati neupisane vrijednosti.

Početnu inačicu 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, jer je popis tipova koje podržava DBMS ograničen.

Slika 5.9 prikazuje početnu fizičku shemu za bazu podataka knjižnice, koja je dobivena na temelju relacijske sheme sa Slike 3.5 i rječnika podataka sa Slike 3.6. Koristi se inačica SQL-a za SQL Server. Pojavljuju se naredbe CREATE TABLE za relacije KNJIGA, ČLAN, i POSUDBA. 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 5.9 SQL Server automatski stvara fizičku bazu gdje je svaka od pet relacija prikazana kao jedna datoteka. Koristi se jedna vrsta indeks-sekvencijalne organizacije: dakle zapisi datoteke raspoređeni su u blokove, a u datoteku je ugrađen primarni indeks koji osigurava svojstvo primarnog ključa i ubrzava pretraživanje po ključu.

Naredbe CREATE TABLE sa Slike 5.9 mogle su se napisati i bez navođenja primarnog ključa. U tom slučaju SQL Server 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 zahtijevalo bi sekvencijalno čitanje cijele datoteke.

```
CREATE TABLE KNJIGA (
ISBN CHAR(13) NOT NULL,
    NASLOV CHAR(80),
    AUTOR_ID INT,
    GODINA_IZDAVANJA INT,
    ZANR CHAR(30),
    BROJ_PRIMJERAKA SMALLINT,
    PRIMARY KEY(ISBN)
);

CREATE TABLE CLAN(
    CLANSKI_BROJ INT NOT NULL,
    PREZIME VARCHAR(30),
    IME VARCHAR(30),
    DATUM_UPISA DATE,
    VRSTA_CLANSTVA VARCHAR(15) CHECK (VRSTA_CLANSTVA
IN ('REDOVNI', 'STUDENTSKI', 'OBITELJSKI')),
    PRIMARY KEY (CLANSKI_BROJ)
);

CREATE TABLE POSUDBA(
   CLANSKI_BROJ INT NOT NULL,
    ISBN CHAR(13) NOT NULL,
    DATUM_POSUDBE DATE,
    DATUM_VRACANJA DATE,
    STATUS_POSUDBE VARCHAR(10) CHECK (STATUS_POSUDBE
IN ('POSUDENO', 'VRACENO')),
    PRIMARY KEY (CLANSKI_BROJ, ISBN)
);

CREATE TABLE AUTOR (
    ID_AUTORA INT NOT NULL,
    PREZIME VARCHAR(30),
    IME VARCHAR(30),
    GODINA_RODJENJA DATE,
    GODINA_SMRTI INT,
    PRIMARY KEY (ID_AUTORA)
);

CREATE TABLE ZANR(
    ID_ZANRA INT NOT NULL,
    NAZIV_ZANRA VARCHAR(30) NOT NULL,
    OPIS VARCHAR(160),
    PRIMARY KEY (ID_ZANRA)
);
```

Slika 5.9: Početna inačica fizičke sheme za bazu podataka o knjižnici

Neki DBMS-i, na primjer Oracle, omogućuju 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.

Zbog navođenja primarnih ključeva i uporabe primarnih indeksa, fizička shema sa Slike 5.9 osigurava da će se u svim datotekama traženje po primarnom ključu odvijati razmjerno brzo. No traženje po drugim podacima bit će sporo jer će zahtijevati sekvencijalno čitanje odgovarajućih datoteka. Pretraživanje po odabranim podacima koji nisu ključevi možemo ubrzati ako DBMS-u naredimo da sagradi odgovarajuće sekundarne indekse. U tu svrhu koriste se SQL-naredbe CREATE INDEX.

Slika 5.10 prikazuje dodatak shemi sa Slike 5.9 kojim se uvodi indeks za atribut NASLOV u relaciji KNJIGA, te indeksi za atribute ČLANSKI_BROJ i ISBN u relaciji POSUDBA. Stvaranjem tih sekundarnih indeksa, indeks-sekvencijalne datoteke za prikaz relacija KNJIGA i POSUDBA nadograđuju se na invertiranu organizaciju. Time postaje moguće brzo pronalaženje knjige po naslovu ili brzi ispis svih knjiga sortiranih po naslovu. Također, efikasno se pronalaze sve knjige koje je posudio zadani član, kao i svi članovi koji su posudili zadanu knjigu.

```
CREATE NONCLUSTERED INDEX KN_NASLOV_IND ON KNJIGA
(NASLOV);

CREATE NONCLUSTERED INDEX POS_CLAN_IND ON POSUDBA
(CLANSKI_BROJ);

CREATE INDEX POS_ISBN_IND ON POSUDBA (ISBN);
```

Slika 5.10: 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, jer 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.

5.2. Integritet baze

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 dopuštaju projektantu baze da definira takozvana *ograničenja* (*constraints*). Riječ je o uvjetima (pravilima) koje korektni i konzistentni podaci moraju zadovoljiti.

Projektant uvodi ograničenja tako da ih upiše u fizičku shemu baze. Uvedena će ograničenja DBMS 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 pogrešci. Sljedeća tri odjeljka opisuju tri vrste ograničenja i konkretne načine njihova uvođenja u fizičku shemu.

5.2.1. 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 je obično 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 knjižnici na Slici 5.9. Vidimo da smo za neke atribute uspjeli vrlo precizno odrediti tipove, čime smo osigurali integritet domene. Zaista, za atribut VRSTA_CLANSTVA u tablici ČLAN postavili smo tip ENUM, što znači da VRSTA_CLANSTVA može poprimati samo vrijednosti iz navedene liste: 'REDOVNI', 'STUDENTSKI', 'OBITELJSKI'. U skladu s tim, DBMS će spriječiti unos člana s nepostojećom vrstom članstva. Da smo VRSTA_CLANSTVA proglasili tekstualnim tipom, tada ne bismo mogli spriječiti takav pogrešan unos. U SQL Serveru tip ENUM nije podržan, ali isti efekt možemo postići korištenjem CHECK ograničenja.

Zamislimo sada da knjižnica koristi police označene troznamenkastim brojevima, gdje prva znamenka označava kat na kojem se polica nalazi. Pretpostavimo da knjižnica ima tri kata označena brojevima 1, 2 i 3. Tada ispravni brojevi polica mogu biti samo oni između 101 i 399. Međutim, ako atribut BROJ_POLICE ima tip NUMERIC(3) UNSIGNED, DBMS neće spriječiti unos vrijednosti izvan tog raspona, poput 050 ili 501. Dakle, samo pridruživanje tipa atributu ne osigurava potpuni integritet domene.

```
CREATE TABLE KNJIGA
(
   ISBN CHAR(13) NOT NULL,
   NASLOV VARCHAR(80),
   AUTOR_ID INT,
   GODINA_IZDAVANJA SMALLINT,
   ZANR VARCHAR(30),
   BROJ_POLICE SMALLINT,
   PRIMARY KEY (ISBN),
   CONSTRAINT CHK_BROJ_POLICE CHECK (BROJ_POLICE
BETWEEN 101 AND 399)
);
```

Slika 5.11: Osiguravanje integriteta domene za BROJ_POLICE

Budući da pridruživanje tipa atributu ne osigurava da će se u potpunosti čuvati integritet domene, mnogi DBMS-i dopuštaju da se u shemu ugradi i precizniji uvjet koji vrijednosti atributa moraju zadovoljavati. Takav uvjet može se uklopiti u naredbu CREATE TABLE ili se može pojaviti kao zasebna naredba.

Na Slici 5.11 prikazan je primjer kako se uvođenjem dodatnog uvjeta može osigurati integritet domene za atribut BROJ_POLICE. Dodatni uvjet zadan je u naredbi CREATE TABLE KNJIGA, koja zamjenjuje inačicu sa Slike 5.9. Slika 5.11 koristi sintaksu Microsoft SQL Servera. Slično rješenje postoji i u Oracleu. Dodali smo CHECK ograničenje nazvano CHK_BROJ_POLICE,

koje osigurava da vrijednost atributa BROJ_POLICE mora biti između 101 i 399. Time osiguravamo integritet domene za taj atribut.

5.2.2. Uvođenje ograničenja za čuvanje integriteta u relaciji

Ograničenja za čuvanje integriteta u SQL Serveru koriste se kako bi se osigurala ispravnost odnosa između atributa u tablici, primjerice kroz funkcionalne ovisnosti. Najvažniji primjer takvog ograničenja jest onaj koji zahtijeva da dva retka (n-torke) iste tablice ne smiju imati jednaku vrijednost primarnog ključa. Slično tome, možemo uvesti i ograničenje jedinstvenosti (UNIQUE) za stupac koji nije primarni ključ, ali predstavlja kandidata za ključ.

Ograničenje ključa u SQL Serveru najčešće se uvodi korištenjem klauzula PRIMARY KEY ili UNIQUE unutar naredbe CREATE TABLE. Na primjer, ako želimo osigurati da primarni ključ bude jedinstven, dodajemo PRIMARY KEY na definirane stupce. Ako određeni stupac treba biti jedinstven, ali nije primarni ključ, koristimo UNIQUE. Alternativno, jedinstvenost se može osigurati i stvaranjem jedinstvenog indeksa putem naredbe CREATE UNIQUE INDEX.

Ako pogledamo fizičku shemu za bazu podataka o knjižnici na Slici 5.9, vidimo da u svim naredbama CREATE TABLE postoje odgovarajuće klauzule PRIMARY KEY. Dakle, već u polaznoj inačici sheme osigurali smo da se čuva svojstvo ključa. Umjesto klauzule PRIMARY KEY mogli smo se koristiti i naredbom CREATE UNIQUE INDEX, no način sa Slike 5.9 smatra se čitljivijim.

```
CREATE TABLE KNJIGA
(
   ISBN CHAR(13) NOT NULL,
   NASLOV VARCHAR(80),
   AUTOR_ID INT,
   GODINA_IZDAVANJA INT,
   ZANR VARCHAR(30),
   BROJ_POLICE NUMERIC(3),
   CONSTRAINT PK_KNJIGA PRIMARY KEY (ISBN),
   CONSTRAINT UQ_BROJ_POLICE UNIQUE (BROJ_POLICE)
);
```

```
CREATE UNIQUE INDEX POLICA_UNIQUE_IND ON KNJIGA
(BROJ_POLICE);
```

Slika 5.12: Dva načina osiguravanja da BROJ_POLICE ima svojstvo ključa

Kao primjer, zamislimo pravilo da svaka polica u knjižnici ima jedinstven broj kako bi se knjige mogle jednoznačno locirati. Prema ovom zamišljenom pravilu, atribut BROJ_POLICE u relaciji KNJIGA postaje kandidat za ključ. Naime, ne smije se dogoditi da dvije knjige budu na istoj polici, jer svaka knjiga mora imati jasno definiranu lokaciju. Iako se takvo pravilo u stvarnosti rijetko primjenjuje (obično jedna polica sadrži više knjiga), ovaj primjer služi kao ilustracija koncepta kandidatskog ključa i jedinstvenosti vrijednosti unutar jednog stupca.

Shema prikazana na Slici 5.9, u izvornom obliku, ne čuva integritet u relaciji jer ne osigurava jedinstvenost vrijednosti atributa BROJ_POLICE. Na Slici

5.12 prikazana su dva načina kako se u SQL Serveru može uvesti dodatno ograničenje za osiguranje svojstva ključa:

1. Ugradnja UNIQUE ograničenja unutar CREATE TABLE naredbe:

Ovim pristupom u postojeću CREATE TABLE naredbu (za KNJIGA ili NASTAVNIK) umetnemo klauzulu UNIQUE uz odgovarajući atribut. Tako izmijenjena naredba zamjenjuje izvorni primjer sa Slike 5.9, osiguravajući da se već pri kreiranju tablice ne mogu unositi duplicirane vrijednosti u tom stupcu.

2. Stvaranje jedinstvenog indeksa pomoću CREATE UNIQUE INDEX naredbe:

Drugi pristup ne zahtijeva mijenjanje postojeće CREATE TABLE naredbe. Umjesto toga, dodajemo novu CREATE UNIQUE INDEX naredbu nad stupcem koji treba biti jedinstven. Time i dalje postižemo da svaka vrijednost u tom stupcu bude jedinstvena, bez izmjene postojeće definicije tablice.

U oba slučaja, ako se pokuša umetnuti ili ažurirati podatak koji bi prekršio jedinstvenost, SQL Server će vratiti poruku o pogrešci, potvrđujući ispravnost i djelotvornost postavljenog ograničenja. Iako ovaj zamišljeni primjer s policama ili sobama ne mora biti primjenjiv u stvarnoj situaciji, jasno ilustrira kako se koriste UNIQUE ograničenja i kandidatski ključevi za očuvanje integriteta podataka.

Na Slici 5.12 vide se dva načina kako se u SQL Serveru uvođenjem dodatnog ograničenja može osigurati svojstvo ključa za BROJ_POLICE. Prvi način svodi se na ubacivanje klauzule UNIQUE u naredbu CREATE TABLE KNJIGA – takva inačica naredbe trebala bi zamijeniti inačicu sa Slike 5.9. Drugi način svodi se na to da se shemi sa Slike 5.9 doda nova naredba CREATE UNIQUE INDEX, s time da naredba CREATE TABLE ostaje nepromijenjena.

5.2.3. Uvođenje ograničenja kojima se čuva referencijalni integritet

Ograničenja za očuvanje referencijalnog integriteta osiguravaju konzistentnost veza između relacija u bazi podataka. Najčešće se primjenjuju na strane ključeve, odnosno na atribute u jednoj relaciji koji istovremeno predstavljaju primarni ključ u drugoj relaciji. Svaka vrijednost takvog atributa u prvoj relaciji mora postojati i u drugoj relaciji.

Ako pogledamo našu bazu podataka knjižnice sa Slika 3.5 i 3.6, možemo zamisliti da relacija POSUDBA sadrži strani ključ ISBN koji odgovara primarnom ključu relacije KNJIGA. Također, relacija POSUDBA može imati strani ključ ČLANSKI BROJ koji odgovara primarnom ključu u relaciji ČLAN. Ove veze omogućuju praćenje koja je knjiga posuđena kojem članu. Fizička shema sa Slike 5.9 ne čuva integritet nijednog od tih stranih ključeva, jer u njoj nisu označeni 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 se navodi ime atributa koji predstavlja strani ključ u određenoj relaciji i ime relacije u kojoj taj isti atribut predstavlja strani ključ.

Slika 5.13 sadrži novu inačicu fizičke sheme za bazu podataka knjižnice, koju možemo koristiti kao zamjenu za shemu sa Slike 5.9. Zahvaljujući klauzulama FOREIGN KEY, ta nova inačica ima uvedena ograničenja za očuvanje integriteta stranih ključeva. Na primjer, DBMS neće dopustiti unos n-torke u relaciju POSUDBA s vrijednošću ISBN koja ne postoji u relaciji KNJIGA, niti s vrijednošću CLANSKI_BROJ koja ne postoji u relaciji ČLAN.

```
CREATE TABLE AUTOR(
   ID_AUTORA INT NOT NULL,
   PREZIME VARCHAR(30),
   IME VARCHAR(30),
   GODINA_RODJENJA DATE,
   GODINA_SMRTI DATE,
   PRIMARY KEY (ID_AUTORA)
);
CREATE TABLE ZANR(
   ID_ZANRA INT NOT NULL,
   NAZIV_ZANRA VARCHAR(30) NOT NULL,
   OPIS VARCHAR(160),
   PRIMARY KEY (ID_ZANRA)
);
CREATE TABLE KNJIGA(
   ISBN CHAR(13) NOT NULL,
   NASLOV CHAR(80),
   AUTOR_ID INT,
   ZANR_ID INT,
   GODINA_IZDAVANJA DATE,
   BROJ_PRIMJERAKA SMALLINT,
   PRIMARY KEY(ISBN),
   FOREIGN KEY (AUTOR_ID) REFERENCES AUTOR(ID_AUTORA),
   FOREIGN KEY (ZANR_ID) REFERENCES ZANR(ID_ZANRA)
);
CREATE TABLE CLAN(
   CLANSKI_BROJ INT NOT NULL,
   PREZIME VARCHAR(30),
   IME VARCHAR(30),
   DATUM_UPISA DATE,
   VRSTA_CLANSTVA VARCHAR(15) CHECK (VRSTA_CLANSTVA
IN ('REDOVNI', 'STUDENTSKI', 'OBITELJSKI')),
   PRIMARY KEY (CLANSKI_BROJ)
);
CREATE TABLE POSUDBA(
   CLANSKI_BROJ INT NOT NULL,
   ISBN CHAR(13) NOT NULL,
   DATUM_POSUDBE DATE,
   DATUM_VRACANJA DATE,
   STATUS_POSUDBE VARCHAR(10) CHECK (STATUS_POSUDBE
IN ('POSUDENO', 'VRACENO')),
   PRIMARY KEY (CLANSKI_BROJ, ISBN),
   FOREIGN KEY (CLANSKI_BROJ) REFERENCES
CLAN(CLANSKI_BROJ),
   FOREIGN KEY (ISBN) REFERENCES KNJIGA(ISBN)
);
```

Slika 5.13: Čuvanje integriteta za strane ključeve

Nakon realizacije sheme sa Slike 5.13, DBMS na primjer neće dopustiti da se u relaciju POSUDBA ubaci n-torka s vrijednošću ISBN koje nema u relaciji KNJIGA. Također, iz relacije ČLAN neće se moći brisati n-torka s CLANSKI_BROJ koja se pojavljuje u relaciji POSUDBA.

Primijetimo da je redoslijed naredbi CREATE TABLE na Slici 5.13 drukčiji nego na Slici 5.9. To je zato što se bilo koja relacija sa stranim ključem može stvoriti tek nakon što već postoji relacija koju taj strani ključ referencira. Na primjer, relacija POSUDBA mora se stvarati nakon relacija KNJIGA i ČLAN, jer njezini atributi ISBN i CLANSKI_BROJ referenciraju primarne ključeve tih relacija.

U SQL Serveru nisu potrebne specifične organizacije datoteka kao što je InnoDB u MySQL-u, niti se moraju eksplicitno deklarirati sekundarni indeksi uz svaki strani ključ. SQL Server automatski upravlja indeksiranjem povezanim sa stranim ključevima, što pojednostavljuje definiciju tablica. Ovo eliminira potrebu za dodatnim naredbama CREATE INDEX koje su bile potrebne na Slici 5.10.

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 su ograničenja stvarno potrebna, a koja se mogu zanemariti ili zamijeniti odgovarajućim provjerama u aplikacijama. Shema sa Slike 5.13 predstavlja svojevrsno pretjerivanje jer smo za pet relacija uveli pet referencijalnih integriteta. Vjerojatno bi bilo dovoljno provjeravati samo sekundarne ključeve u relaciji POSUDBA, jer se jedino tu radi o nešto većoj količini podataka.

5.3. Sigurnost baze

Baza podataka predstavlja dragocjen resurs. Njezin nastanak i održavanje iziskuju goleme količine ljudskog rada. Zato se od DBMS-a očekuje da u što većoj mjeri jamči sigurnost podataka. To znači da se ne smije dogoditi 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 načine kako da projektant te mehanizme ugradi u svoju fizičku shemu i tako postigne njihovo aktiviranje tijekom rada baze. Prvi se odjeljak bavi se tehničkim aspektom sigurnosti, dakle oporavkom baze u slučaju njenog većeg ili manjeg oštećenja. Ostali odjeljci bave se suptilnijim aspektom koji se tiče zaštite baze od neovlaštenih radnji korisnika.

5.3.1. 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*. Iako s korisničkog stajališta transakcija djeluje kao nedjeljiva cjelina, tehnički se ostvaruje kao niz elementarnih operacija unutar baze podataka.

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 i povećanje salda na drugom.

Osnovno svojstvo transakcije je da ona prevodi bazu iz jednog konzistentnog stanja u drugo. No međustanja 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 bila do kraja obavljena, morala bi biti *neutralizirana* – dakle svi podaci koje je ona do trenutka prekida promijenila, morali bi dobiti natrag 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 novac biti skinut s jednog računa, a neće još biti stavljen na drugi. Ili obratno: bit će stavljen na drugi račun prije nego što je bio skinut s prvog. Ako se transakcija prekine tijekom realizacije, novac će netragom nestati ili će se stvoriti niotkud.

Rekli smo već da se baza podataka u toku svojeg 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, ali nije bila obavljena do kraja, a nije bila ni neutralizirana. Daljnji su, ali znatno rjeđi razlozi koji također mogu dovesti do oštećenja baze: pogrešno sastavljena transakcija, softverske pogreške u DBMS-u ili operacijskom sustavu te hardverske pogreške, na primjer kvar diska.

Od suvremenog se DBMS-a očekuje 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 od aplikacije očekuje da eksplicitno najavi početak transakcije i 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 se 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 prečesto, već periodički u predviđenim terminima – na primjer jednom tjedno.
- **Održava se žurnal-datoteka.** Riječ je o datoteci gdje je zabilježena „povijest“ svake transakcije koja je mijenjala bazu nakon zadnjeg stvaranja rezervne kopije. Za pojedinu transakciju žurnal bilježi adresu svakog zapisa koji je transakcija promijenila, 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 utvrdi 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. Koristi 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 njezina 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 koristi postupak odmotavanja unaprijed (*roll-forward*). Dakle, žurnal-datoteka se čita od početka prema kraju i ponovo se za svaku potvrđenu transakciju u bazu unose sve zabilježene promjene podataka. 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 spada u nadležnost projektanta. Naime, uključivanje pojedinog mehanizma donosi veću sigurnost, ali opterećuje svakodnevni rad baze i smanjuje performanse. Projektant treba procijeniti kakve će se vrste transakcija izvršavati, koliki je stvarni rizik od oštećenja i u kojoj mjeri se isplati žrtvovati performanse zbog veće sigurnosti.

Na osnovi 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 pri oblikovanju fizičke građe, odnosno fizička shema projektanta mora biti usklađena s njegovim planovima za kasniju zaštitu.

U nastavku ćemo opisati kako se opisani mehanizmi i sredstva za oporavak baze realiziraju u SQL Server:

- SQL Server prema zadanim postavkama koristi *autocommit mode*, gdje se svaka SQL naredba smatra zasebnom transakcijom i automatski se izvršava. Za upravljanje transakcijama bez autocommita koriste se sljedeće naredbe:

  BEGIN TRANSACTION;

  Za potvrdu transakcije:

  COMMIT TRANSACTION;

  Za poništavanje transakcije:

  ROLLBACK TRANSACTION;

- Izrada backupa baze podataka u SQL Serveru obavlja se korištenjem T-SQL naredbi ili putem SQL Server Management Studio (SSMS). Primjer naredbe za izradu backupa putem T-SQL-a je:

```
BACKUP DATABASE [ImeBaze] TO DISK =
'LokacijaNaDisku\ImeBaze.bak';
```

SQL Server automatski održava transaction log za svaku bazu podataka, koji zapisuje sve promjene kako bi se omogućilo vraćanje baze u dosljedno stanje i podržao oporavak. Za kontrolu aspekata logiranja, konfiguracija se vrši na razini baze podataka, ali bez direktnih ekvivalenata za MySQL opciju --log-update=ime_datoteke. Za prilagodbu konfiguracije log datoteka koriste se opcije unutar SQL Server konfiguracijskih postavki.

Kao primjer upravljanja transakcijama u SQL Serveru na Slici 5.14 vidi se niz naredbi kojima se broj primjeraka knjige smanjuje za jedan kada je knjiga posuđena, a zapis o posudbi dodaje se u relaciju POSUDBA. Cijeli niz naredbi proglašen je nedjeljivom cjelinom koja se mora izvršiti u cijelosti ili se uopće ne smije izvršiti. Tako se čuva konzistencija u smislu da je broj dostupnih primjeraka uvijek točan, a posudba zabilježena u sustavu. Promjene se obavljaju u relacijama KNJIGA i POSUDBA, koje su opisane fizičkom shemom na Slici 5.13.

```
SELECT * FROM KNJIGA;
SELECT * FROM POSUDBA;

BEGIN TRANSACTION; -- Početak transakcijskog bloka.

-- Smanjivanje broja primjeraka knjige
UPDATE KNJIGA
  SET BROJ_PRIMJERAKA = BROJ_PRIMJERAKA - 1
  WHERE ISBN = '9781234567890';

-- Dodavanje zapisa o posudbi knjige
INSERT INTO POSUDBA (CLANSKI_BROJ, ISBN,
DATUM_POSUDBE, STATUS_POSUDBE)
  VALUES (12345678, '9781234567890', GETDATE(), 'POSUDENO');
-- GETDATE()  - koristi se za trenutačni datum u SQL Serveru

-- Potvrda transakcije ako su sve operacije uspješne
COMMIT TRANSACTION; -- Potvrđuje sve izmjene, ako je sve u redu.

-- Alternativno, ako dođe do pogreške u bilo kojoj od naredbi iznad
ROLLBACK TRANSACTION; -- Poništava sve promjene unutar
transakcijskog bloka.

--Na kraju ponovno ispisujemo sadržaj tablica KNJIGA i POSUDBA
kako bismo provjerili rezultate transakcije

SELECT * FROM KNJIGA;
SELECT * FROM POSUDBA;
```

Slika 5.14: Transakcija u SQL Serveru

5.3.2. 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, korisno je da projektant u fazi fizičkog projektiranja baze predvidi nekoliko tipičnih korisnika i predloži njihova ovlaštenja. Projektantovi naputci kasnije mogu poslužiti administratoru kao smjernice za uvođenje dodatnih korisnika.

U SQL Serveru, ovlaštenja se mogu dodijeliti na razini cijele baze, pojedine tablice ili čak na razini pojedinih stupaca unutar tablice. Uobičajena ovlaštenja su: SELECT, INSERT, DELETE, UPDATE, ALTER, CREATE, DROP, a pomoću naredbe ALL mogu se dodijeliti sva navedena ovlaštenja odjednom.

Slika 5.15 sadrži prvi primjer davanja ovlaštenja u SQL Serveru anonimnom korisniku za samo čitanje podataka iz određene baze podataka. Administrator može dodijeliti ovlaštenje naredbom sa slike. Anonimni korisnik može pregledavati podatke unutar baze podataka knjižnice, ali ne može unositi, brisati ili mijenjati podatke. Donji dio slike prikazuje moguću sesiju anonimnog korisnika. Korisnik se uspješno prijavljuje i pregledava sadržaj baze knjižnice, ali ne uspijeva izvršiti promjene.

```
USE knjiznicaDB;
GO
CREATE USER [anonimni_korisnik] WITHOUT LOGIN;
GO
```

```
GRAND SELECT ON SCHEMA::dbo TO [anonimni_korisnik];
GO
SELECT USER_NAME() AS TrenutniKorisnik;
-- Izvršavanje SELECT naredbe za pregled svih zapisa iz tablice 'KNJIGA'
> SELECT * FROM KNJIGA;
...
(ovdje se ispisuju rezultati upita)
...
-- Pokušaj dodavanja nove knjige u tablicu 'KNJIGA'
> INSERT INTO KNJIGA (Naslov, AUTOR_ID, GODINA_IZDANJA)
> VALUES ('Nova knjiga', 10, 2023);
...
(poruka o greški zbog nedostatka odgovarajućih ovlasti)
```

Slika 5.15: Davanje ovlaštenja neregistriranom korisniku

Slika 5.16 sadrži drugi primjer davanja ovlaštenja u SQL Serveru za knjižnični sustav. Administrator započinje kreiranjem baze podataka knjiznica_someuser za individualnu upotrebu korisnika "someuser".

Nadalje, stvara korisnički login s lozinkom, dodaje "someuser" u bazu knjiznica_someuser s ulogom db_owner, omogućujući mu puna prava nad tom bazom. U institucionalnoj bazi knjiznicaDB, "someuser" dobiva ograničena čitateljska prava, što mu omogućava pregled sadržaja bez mogućnosti izmjene. Ovaj pristup osigurava odgovarajuću razinu kontrole i sigurnosti za upravljanje knjižničnim resursima.

Na slici 5.17 vidi se primjer oduzimanja ovlaštenja u SQL Serveru. Gornji dio slike prikazuje naredbe kojima administrator korisniku someuser oduzima pravo čitanja relacija KNJIGA i ČLAN na bazi podataka knjižnice. Nakon oduzimanja prava, administrator može odlučiti dodijeliti prava čitanja na druge relacije unutar iste baze. Ovo omogućava korisniku someuser pristup određenim podacima koje je još uvijek ovlašten pregledavati.

```
CREATE DATABASE knjiznica_someuser;
CREATE LOGIN someuser WITH PASSWORD = 'loz000';
```

```
-- Dodavanje korisnika u bazu 'knjiznica_someuser' s db_owner ulogom
USE knjiznica_someuser;
CREATE USER someuser FOR LOGIN someuser;
ALTER ROLE db_owner ADD MEMBER someuser;

-- Dodavanje korisnika u bazu 'knjiznicaDB' s db_datareader ulogom
USE knjiznicaDB;
GO
CREATE USER someuser FOR LOGIN someuser;
ALTER ROLE db_datareader ADD MEMBER someuser;
GO  ...
```

Slika 5.16: Davanje ovlaštenja registriranom korisniku

ZA PREDAVAČA

Ako bude vremena, primjeri sa Slike 5.16 i 5.17 mogu se izvesti u SQL Serveru. Predavač, koji mora imati administratorske ovlasti, izvršava naredbe iz gornjeg dijela slike. Svaki od polaznika zasebno izvršava naredbe iz donjeg dijela slike. Naredbe su kratke pa se mogu izravno upisivati na komandnu liniju. Imena baza i korisnika mogu se prilagoditi.

```
USE knjiznicaDB;
REVOKE SELECT ON dbo.Knjiga FROM someuser;
REVOKE SELECT ON dbo.Autor FROM someuser;
```

```
GRANT SELECT ON dbo.Clan TO someuser;
GRANT SELECT ON dbo.Posudba TO someuser;
```

Slika 5.17: Oduzimanje ovlaštenja korisnicima

SQL Server, za razliku od MySQL-a, nudi mogućnost upravljanja pravima pristupa putem korisničkih skupina, poznatih kao uloge. Ove uloge omogućuju administratorima da efikasnije dodjeljuju ovlaštenja skupinama korisnika umjesto pojedinačno, što je korisno u okruženjima s mnogim korisnicima koji zahtijevaju slična prava pristupa. Svi korisnici unutar iste uloge nasljeđuju ovlaštenja koja su dodijeljena toj ulozi, što olakšava administraciju i održavanje dosljednosti prava pristupa.

Pored toga, SQL Server zahtjeva zaštitu na razini operacijskog sustava kako bi se osigurala sigurnost podataka. U ovom kontekstu, potrebno je konfigurirati operacijski sustav tako da samo SQL Server proces, točnije sqlserver.exe, ima ovlaštenja za pristup direktorijima i datotekama baze podataka. Ova mjera sprječava neautorizirani pristup i pomaže u očuvanju integriteta i sigurnosti podataka.

5.3.3. Uporaba pogleda kao mehanizma zaštite

U prvom poglavlju spomenuli smo poglede (podsheme) 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 znatno ograničene mogućnosti zlouporabe podataka.

U relacijskom modelu i globalna shema i pogled (podshema) 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 djelovala, potrebno je određenom korisniku još regulirati i 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 pristup do stvarnih relacija, ali da može pristupiti pogledima. Tako je korisnik prisiljen raditi samo s onim podacima koje smo obuhvatili pogledima, a ostali podaci su mu skriveni i nedostupni.

Definiranje pogleda za pojedine vrste korisnika moglo bi se prepustiti administratoru baze. No bolje je da se time bavi projektant, jer je to aktivnost koja zadire u logičku razinu projektiranja. Ako to nije učinio prije, projektant bi najvažnije poglede trebao definirati u sklopu svoje fizičke sheme navođenjem odgovarajućih naredbi CREATE VIEW.

Slika 5.18 sadrži primjer uporabe pogleda kao mehanizma zaštite u bazi podataka knjižnice sa Slike 5.13. Cilj koji se želi postići je skrivanje povjerljivog podatka o članovima knjižnice, poput ČLANSKOG BROJA. Naredba CREATE VIEW može kreirati pogled CLAN_VIEW1, koji prikazuje podatke o korisnicima (ime, prezime, članski broj) bez osjetljivih informacija, kao što je broj osobne iskaznice. Većina korisnika baze bit će ovlašteni za korištenje samo ovog pogleda, dok će stvarna relacija CLAN ostati dostupna samo administratorima.

```
CREATE VIEW CLAN_VIEW1
AS SELECT PREZIME, IME, CLANSKI_BROJ
FROM CLAN;
```

Slika 5.18: Pogled koji skriva povjerljive podatke o članovima knjižnice

Slika 5.19 prikazuje još jedan primjer zaštite baze podataka knjižnice pomoću pogleda. Knjižničar u Odjelu za beletristiku treba pristupiti samo podacima o knjigama u tom odjelu. Definiramo pogled KNJIGA_VIEW1 koji prikazuje podatke o knjigama iz tog odjela (naslov, autor, godina izdanja).

Knjižničar vidi samo „horizontalni“ segment relacije KNJIGA, ograničen prema odjelu.

```
CREATE VIEW KNJIGA_VIEW1
  AS SELECT  * FROM KNJIGA
  WHERE ZANR_ID= 1;
```

Slika 5.19: Pogled koji se ograničava na podatke o knjigama iz određenog odjela

```
CREATE VIEW CLAN_POSUDBE
AS SELECT POSUDBA.ISBN, KNJIGA.NASLOV,
POSUDBA.DATUM_POSUDBE, POSUDBA.DATUM_VRACANJA,
POSUDBA.STATUS_POSUDBE
FROM POSUDBA
JOIN KNJIGA ON POSUDBA.ISBN = KNJIGA.ISBN
WHERE POSUDBA.CLANSKI_BROJ = 123456; /*Ovdje zamijenite
123456 s pravim brojem člana*/
```

Slika 5.20: Pogled na prikaz individualnog statusa posudbi

Slika 5.20 sadrži treći primjer uporabe pogleda u bazi podataka knjižnice. Naredba CREATE VIEW pisana u SQL Serveru definira pogled POSUDBA_VIEW1, koji omogućuje članu da vidi naslov i datum povratka knjiga koje je posudio. Osobni podaci drugih članova i detalji njihovih posudbi također nisu mu dostupni. Na ovaj način pogledi ne samo omogućuju zaštitu podataka, nego i olakšavaju rad korisnicima tako što im pružaju personalizirane prikaze podataka, prilagođene njihovim ulogama i potrebama.

5.4. Vježbe

- **Zadatak 5.1.** Rješavanjem Zadatka 3.1 ili 4.1 dobili ste nadopunjenu relacijsku shemu baze podataka o knjižnici. Pretvorite tu relacijsku shemu u fizičku shemu za SQL Server. Osigurajte čuvanje integriteta za neke od stranih ključeva.
- **Zadatak 5.2.** Rješavanjem Zadatka 3.2 ili 4.2 dobili ste relacijsku shemu baze podataka o teretani. Pretvorite tu relacijsku shemu u fizičku shemu za SQL Server. Osigurajte čuvanje integriteta za neke od stranih ključeva.
- **Zadatak 5.3.** Razmislite koje bi se sve vrste (skupine) korisnika mogle koristiti bazom 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 SQL Servera.

Dodatni zadaci:

- **Zadatak 5.4.** Rješavanjem Zadatka 3.3 ili 4.8 dobili ste relacijsku shemu za bazu podataka iz svojeg područja interesa. Pretvorite tu relacijsku shemu u fizičku shemu za SQL Server. Osigurajte čuvanje integriteta za neke od stranih ključeva.
- **Zadatak 5.5.** 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 SQL Server. Usporedite svoje rješenje s onim u prilogu.
- **Zadatak 5.6.** 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 SQL Server. Usporedite svoje rješenje s onim u prilogu.
- **Zadatak 5.6.** Baza podataka za sustav upravljanja hotelom sadrži sljedeće tablice: Gost(ID_Gosta, Ime, Prezime, Kontakt), Soba(ID_Sobe, Tip, Cijena), Rezervacija(ID_Rezervacije, ID_Gosta, ID_Sobe, Datum_Dolaska, Datum_Odlaska). Predložite kako biste organizirali tablice u fizičkoj bazi podataka (npr. indeksiranje, particioniranje). Također, dodajte indeks za brzo pretraživanje prema ID_Gosta u tablici Rezervacija.
- **Zadatak 5.7.** Pretpostavite relaciju koja prati zalihe u skladištu: Zalihe(ID_Artikla, Naziv_Artikla, Količina, Cijena). Dodajte ograničenje koje osigurava da atribut Količina ne može imati negativnu vrijednost, te implementirajte ograničenje koje osigurava da atribut Cijena uvijek bude veća od nule.
- **Zadatak 5.8.** Baza podataka za sustav upravljanja studentskim ispitima uključuje sljedeće relacije: Student(ID_Studenta, Ime, Prezime, Smjer), Ispit(ID_Ispita, Naziv, Datum, ID_Studenta, Ocjena). Kreirajte SQL naredbe za dodavanje vanjskog ključa ID_Studenta u tablici Ispit koji referencira tablicu Student, te dodajte ograničenje koje sprječava brisanje studenta iz tablice Student ako je za njega evidentiran ispit.
- **Zadatak 5.9.** Tvrtka koristi bazu podataka za praćenje financijskih transakcija, gdje relacija Transakcija sadrži sljedeće atribute: Transakcija(ID_Transakcije, Datum, Iznos, Vrsta, ID_Korisnika).

Kreirajte pogled koji omogućuje korisnicima da vide samo Datum, Iznos i Vrsta za transakcije bez pristupa ostalim podacima.

- **Zadatak 5.10.** Sustav upravljanja knjižnicom koristi bazu podataka koja uključuje tablice Knjiga, Član i Posudba. Struktura: Knjiga(ID_Knjige, Naslov, Autor, Žanr), Član(ID_Člana, Ime, Prezime), Posudba(ID_Posudbe, ID_Člana, ID_Knjige, Datum_Posudbe, Datum_Povrata). Kreirajte SQL naredbe koje dodjeljuju ovlaštenje za pregled podataka o knjigama korisniku student, te pregled i uređivanje podataka o posudbama korisniku knjižničar. Oduzmite korisniku student pravo pregleda podataka o žanru knjige.

Prilozi

U prethodnim poglavljima opisali smo postupak projektiranja baze podataka i ilustrirali smo ga 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 i baza podataka o znanstvenoj konferenciji.

P.1. Projektiranje baze podataka o bolnici

U ovom studijskom primjeru bavimo se pojednostavnjenom i donekle neobičnom bolnicom. Projektiranje kreće od specifikacije koja je dobivena utvrđivanjem i analizom zahtjeva, a zatim se odvija u tri uobičajene faze: projektiranje na konceptualnoj, logičkoj, odnosno fizičkoj razini.

P.1.. Specifikacija za bolnicu

Utvrđivanjem i analizom zahtjeva dobili smo ovu specifikaciju. Ona govori o bolnici, njezinim prostorijama, liječnicima, osoblju i pacijentima te o odgovarajućim odnosima i pravilima.

Pacijenti koji zauzimaju sobe. Pacijent se obično prilikom dolaska u bolnicu smješta u bolničku sobu. Svaka soba može primiti više 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 su o jednoj operaciji: tip operacije, pacijent, kirurg, datum, vrijeme i mjesto.

Kirurzi koji obavljaju operacije. Jednu operaciju obavlja samo jedan kirurg, a za ostale se prisutne kirurge smatra da asistiraju pri operaciji. Kirurge nadgledaju stariji kirurzi, takozvani konzultanti, koji također mogu obavljati operacije ili asistirati. Informacije su 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 se operacija 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 projektiranja baze podataka o bolnici je projektiranje na konceptualnoj razini. Čitanjem prethodne specifikacije otkrivamo elemente od kojih se sastoji konceptualna shema naše baze, dakle entitete, veze i atribute. Konceptualnu shemu oblikujemo crtanjem reduciranog Chenova 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 kardinaliteta zadane su i funkcionalnosti veza te obaveznost članstva entiteta u vezama. Slika P.1 određuje popis atributa za pojedini tip entiteta i vezu.

P.1.3. Relacijska shema i rječnik podataka za bolnicu

Nastavak projektiranja baze podataka o bolnici je projektiranje na logičkoj razini. 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 njihova 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, jer 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 zbog svoje je funkcionalnosti M:M morala biti prikazana posebnom relacijom, koja sadrži ključne atribute od KIRURG i OPERACIJA 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. Kad bi većina sestara bila zadužena za sobe, tada bi možda bilo bolje vezu ZADUŽENA ZA SOBU prikazati stranim ključem ID SOBE u relaciji SESTRA, no tada bi u istu relaciju morali ugurati i dodatni atribut DATUM ZADUŽIVANJA.
- 1:M veza NADGLEDA zbog neobaveznosti je 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

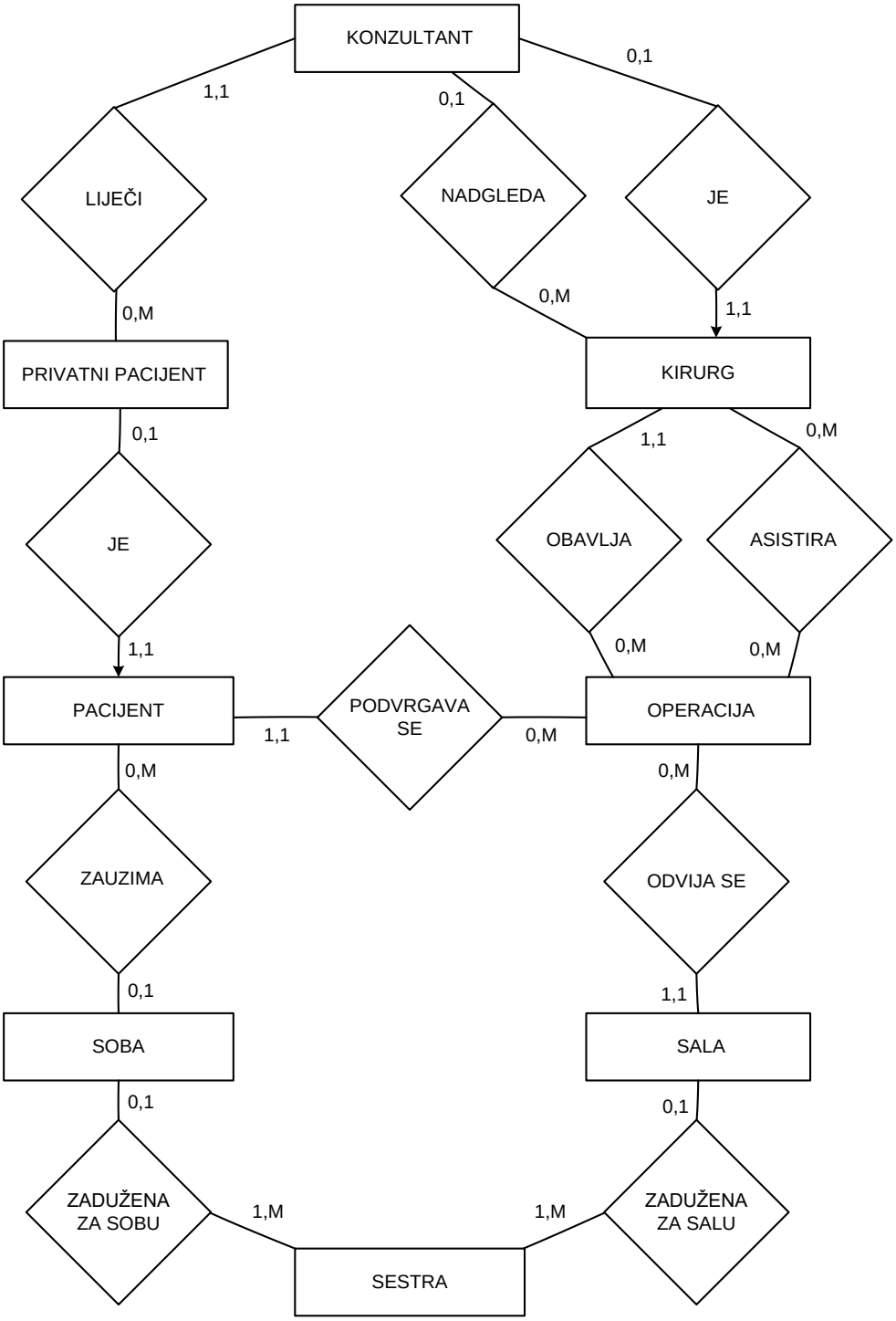
Zadnji dio projektiranja baze podataka o bolnici predstavlja projektiranje na fizičkoj razini. Relacijsku shemu naše baze pretvaramo u fizičku shemu tako da građu svake relacije sa Slike P.3 opišemo odgovarajućom SQL-naredbom CREATE TABLE. Pritom tipove atributa određujemo u skladu s rječnikom podataka sa Slike P.4. Ako se služimo sintaksom iz MySQL-a, tada dobivena fizička shema izgleda kao na Slikama P.5 i P.6.

Tekst sa Slike P.5 i P.6 treba smatrati početnom inačicom fizičke sheme koja se može dalje dotjerivati. Ta početna inačica osigurava integritet domena za atribute u onoj mjeri koliko to dopuštaju mogućnosti zadavanja tipova u MySQL-u. Također, osigurana je jedinstvenost vrijednosti primarnog ključa u svakoj relaciji.

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 i 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 podtip 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 podtip od PACIJENT,
ima dodatni atribut:
ID PRIVATNE SOBE.

Tip entiteta SESTRA ima atribute:
OIB, PREZIME, IME, STRUČNI STUPANJ.

Tip entiteta SOBA (misli se na sobu koja nije privatna) 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 u mreži
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	Kratki niz znakova	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 DECIMAL(11, 0) NOT NULL,
   PREZIME VARCHAR(20),
   IME VARCHAR(20),
   ADRESA VARCHAR(80),
   BROJ_TELEFONA VARCHAR(15),
   PRIMARY KEY (OIB)
);
CREATE TABLE KONZULTANT (
   OIB DECIMAL(11, 0) NOT NULL,
   SPECIJALNOST VARCHAR(40),
   PRIMARY KEY (OIB)
);
CREATE TABLE PACIJENT (
   OIB DECIMAL(11, 0) NOT NULL,
   PREZIME VARCHAR(20),
   IME VARCHAR(20),
   ID_SOBE CHAR(5),
   ADRESA VARCHAR(80),
   DATUM_RODJENJA DATE,
   SPOL VARCHAR(1) CHECK (SPOL IN ('M', 'Z')),
   PRIMARY KEY (OIB)
);
CREATE TABLE PRIVATNI_PACIJENT (
   OIB DECIMAL(11, 0) NOT NULL,
   OIB_KONZULTANTA DECIMAL(11, 0) NOT NULL,
   ID_PRIVATNE_SOBE CHAR(5),
   PRIMARY KEY (OIB)
);CREATE TABLE SESTRA (
   OIB DECIMAL(11, 0) NOT NULL,
   PREZIME VARCHAR(20),
   IME VARCHAR(20),
   STRUCNI_STUPANJ VARCHAR(4) CHECK (STRUCNI_STUPANJ IN
('SSS', 'VSHS', 'VSS')),
   PRIMARY KEY (OIB)
);
CREATE TABLE SOBA (
   ID_SOBE CHAR(5) NOT NULL,
   TIP_SOBE VARCHAR(20),
   BROJ_KREVETA DECIMAL(2, 0),
   PRIMARY KEY (ID_SOBE)
);
```

Slika P.5: Fizička shema za bazu podataka o bolnici (prvi dio)

```
CREATE TABLE SALA (
   ID_SALE CHAR(5) NOT NULL,
   TIP_SALE VARCHAR(20),
   PRIMARY KEY (ID_SALE)
);
CREATE TABLE OPERACIJA (
   ID_OPERACIJE DECIMAL(7, 0) NOT NULL,
   OIB_KIRURGA DECIMAL(11, 0) NOT NULL,
   ID_SALE CHAR(5) NOT NULL,
   OIB_PACIJENTA DECIMAL(11, 0) NOT NULL,
   TIP_OPERACIJE VARCHAR(20),
   DATUM DATE,
   VRIJEME TIME,
   PRIMARY KEY (ID_OPERACIJE)
);
CREATE TABLE ASISTIRA (
   ID_OPERACIJE DECIMAL(7, 0) NOT NULL,
   OIB_KIRURGA DECIMAL(11, 0) NOT NULL,
   ULOGA VARCHAR(40),
   PRIMARY KEY (ID_OPERACIJE, OIB_KIRURGA)
);
CREATE TABLE NADGLEDA (
   OIB_NADGLEDANOG DECIMAL(11, 0) NOT NULL,
   OIB_KONZULTANTA DECIMAL(11, 0) NOT NULL,
   PRIMARY KEY (OIB_NADGLEDANOG)
);
CREATE TABLE ZADUZENA_ZA_SOBU (
   OIB_SESTRE DECIMAL(11, 0) NOT NULL,
   ID_SOBE CHAR(5) NOT NULL,
   DATUM_ZADUZIVANJA DATE,
   PRIMARY KEY (OIB_SESTRE)
);
CREATE TABLE ZADUZENA_ZA_SALU (
   OIB_SESTRE DECIMAL(11, 0) NOT NULL,
   ID_SALE CHAR(5) NOT NULL,
   DATUM_ZADUZIVANJA DATE,
   PRIMARY KEY (OIB_SESTRE)
);
```

Slika P.6: Fizička shema za bazu podataka o bolnici (drugi dio)

P.2. Projektiranje baze podataka o znanstvenoj konferenciji

U ovom studijskom primjeru želimo projektirati bazu podataka koja će služiti kao podrška organizatorima neke znanstvene konferencije. Kao i u prethodnim primjerima, postupak kreće od specifikacije i nastavlja se kroz faze projektiranja na konceptualnoj, logičkoj i fizičkoj razini.

P.2.1. Specifikacija za znanstvenu konferenciju

Utvrđivanjem i analizom zahtjeva dobili smo ovu specifikaciju. Ona govori o znanstvenoj konferenciji, njezinim organizatorima i sudionicima i znanstvenim radovima koji će se izlagati na sjednicama. Opisani su postupci vezani uz pripremu, organizaciju i odvijanje konferencije.

Općenito o konferenciji. Computing Conference 2024 (kratica CC 2024) omogućuje prezentaciju novih rezultata u računarskim znanostima. Organizatori svoj poziv za sudjelovanje upućuju raznim fakultetima, institutima i kompanijama. Kao odgovor stiže nekoliko stotina radova (članaka). Recenzenti pregledavaju pristigle radove. Zbog ograničenja trajanja konferencije, samo 120 radova bit će prihvaćeno za prezentaciju na CC 2024. Svaki rad se svrstava u jednu od tema konferencije.

Raspored tema i sjednica. CC 2024 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 dva sata i posvećena je jednoj određenoj temi. Svaka tema ima dakle četiri 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 konferenciji CC 2024 sudjelovati nekoliko tisuća ljudi. O svakom sudioniku treba pamtiti nekoliko osobnih podataka (titula, ime i prezime, radno mjesto, poštanska adresa, adresa e-pošte itd). Također, sudionik se treba odlučiti kojim sve sjednicama namjerava prisustvovati. Zadnja informacija služi za računanje kotizacije.

Računanje kotizacije. Standardna kotizacija za jednu sjednicu je 50 EUR. Popusti su: 20 % za sudionika koji je i autor rada prihvaćenog za prezentaciju, 30 % za autora koji će i prezentirati rad, 40 % za sudionika koji je ujedno i predsjedavajući neke od sjednica.

Sudjelovanje na sjednicama. Svaki sudionik može prijaviti prisustvovanje većem broju 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 konferencije CC 2024 smatraju se konačnima.

Novčane doznake. Da bi podmirio kotizaciju, sudionik šalje organizatorima jednu ili više novčanih doznaka. Ako 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 projektiranja na konceptualnoj razini, 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 Chenova 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 i obaveznost članstva entiteta u vezama. Slika P.8 daje popis atributa za pojedini tip entiteta i vezu.

P.2.3. Relacijska shema i rječnik podataka za znanstvenu konferenciju

U prvom dijelu projektiranja na logičkoj razini na osnovi konceptualne sheme sa Slika P.7 i P.8 izravno se dolazi do početne inačice relacijske sheme i do rječnika podataka. Dobivena relacijska shema prikazana je na Slici P.9, a sastoji se od skupa relacija od kojih 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 ovisi o njenoj 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 što 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 ADRESA E-POŠTE 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 ADRESA E-POŠTE 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 projektiranja na logičkoj razini 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:

- 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 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. Zanima nas adresa na kojoj se nalazi određena 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 ADRESA E-POŠTE → 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 inačica relacijske sheme koja nastaje normalizacijom prikazana je na Slici P.11. U odnosu na prethodnu inačicu sa Slike P.9, u novoj inačici postoje samo dvije razlike:

- relacija SUDIONIK ima jednostavniju građu
- pojavila se nova relacija TARIFA.

Shema sa Slike P.11 u dovoljnoj je mjeri normalizirana. Naime, sve su njezine relacije 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 veza s funkcionalnošću M:1 između SUDIONIKA i TARIFE koja određuje koliko određeni sudionik plaća za 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

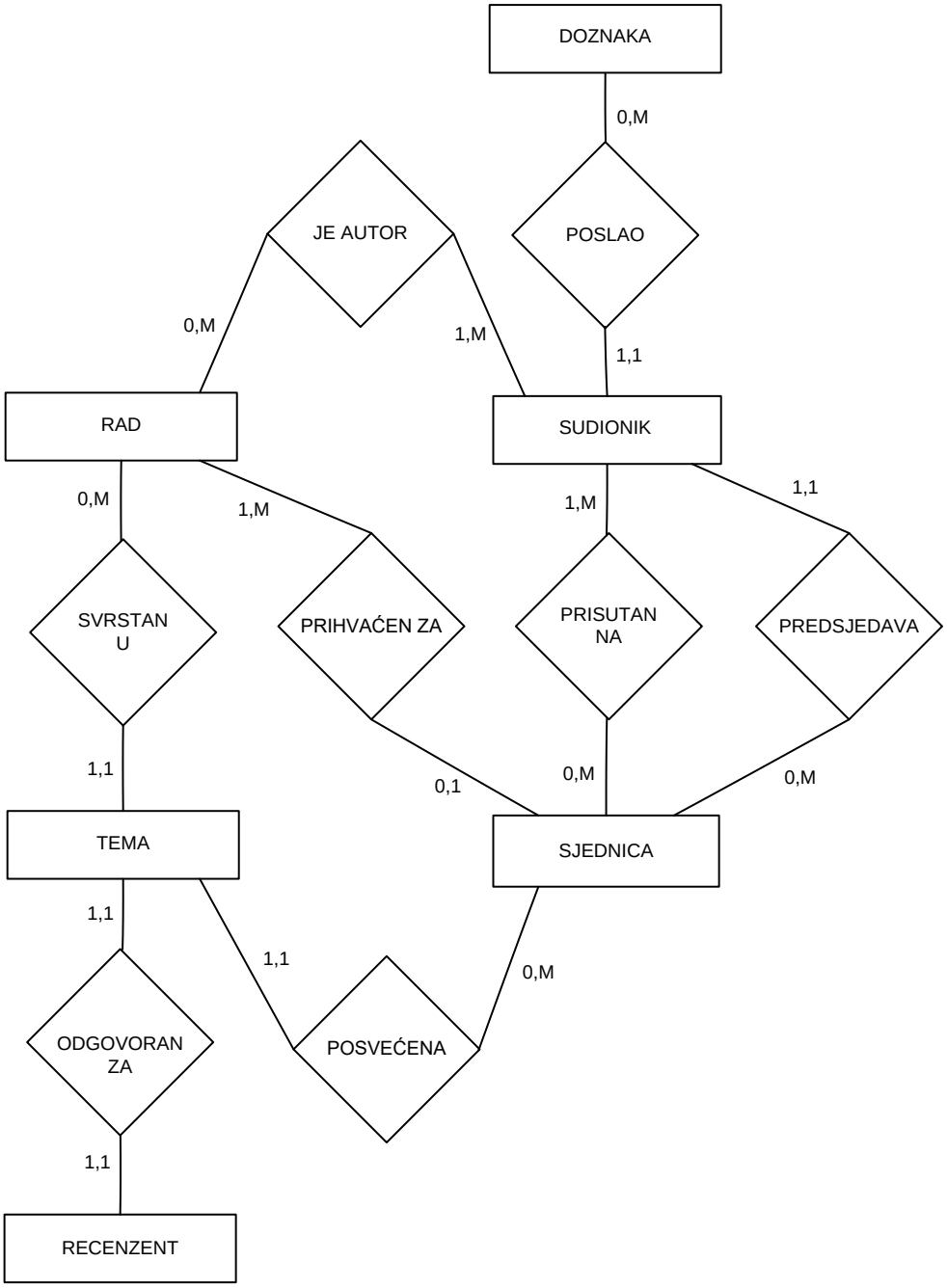
U sklopu projektiranja na fizičkoj razini, normaliziranu relacijsku shemu naše baze za znanstvenu konferenciju pretvaramo u fizičku shemu. To radimo tako da građu svake relacije sa Slike P.11 opišemo odgovarajućom SQL-naredbom CREATE TABLE. Pritom tipove atributa nastojimo što bolje uskladiti s rječnikom podataka sa Slike P.10. Dobivena fizička shema prikazana je na Slikama P.12 i P.13. Koristili smo se sintaksom SQL-a.

Kao i u prošlim primjerima, tekst sa Slika P.12 i P.13 tek je početna inačica fizičke sheme koja se može dalje dotjerivati. Naime, ta početna inačica 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
- ADRESA E-POŠTE SUDIONIKA u relaciji DOZNAKA
- OZNAKA TEME i ADRESA E-POŠTE PREDSJEDAVAJUĆEG u relaciji SJEDNICA
- BROJ RADA i ADRESA E-POŠTE AUTORA u relaciji AUTORSTVO
- BROJ RADA i OZNAKA SJEDNICE u relaciji PRIHVAĆANJE
- ADRESA E-POŠTE 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:
ADRESA E-POŠTE, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA,
STATUS (obični, autor, izlagač, predsjednik),
IZNOS KOTIZACIJE (po sjednici).

Tip entiteta RECENZENT ima atribute:
ADRESA E-POŠTE, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA.

Tip entiteta TEMA ima atribute:
OZNAKA TEME, NAZIV TEME, OPIS TEME.

Tip entiteta DOZNAKA ima atribute:
ADRESA E-POŠTE SUDIONIKA, DATUM, PLAĆENI IZNOS
(jedan sudionik u jednom danu može imati samo jednu doznaku).

Tip entiteta SJEDNICA ima atribute:
OZNAKA SJEDNICE (pon1, pon2, … cet8), 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 (ADRESA E-POŠTE, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA,
STATUS, IZNOS KOTIZACIJE).

RECENZENT (ADRESA E-POŠTE, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA,
OZNAKA TEME)

TEMA (OZNAKA TEME, NAZIV TEME, OPIS TEME)

DOZNAKA (ADRESA E-POŠTE SUDIONIKA, DATUM,
PLAĆENI IZNOS)

SJEDNICA (OZNAKA SJEDNICE, DATUM, VRIJEME OD,
VRIJEME DO, OZNAKA DVORANE, OZNAKA TEME,
ADRESA E-POŠTE PREDSJEDAVAJUĆEG)

AUTORSTVO (BROJ RADA, ADRESA E POŠTE AUTORA)

PRIHVAĆANJE (BROJ RADA, OZNAKA SJEDNICE)

PRISUSTVO (ADRESA E-POŠTE SUDIONIKA,
OZNAKA SJEDNICE)

Slika P.9: Početna inačica 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
ADRESA E-POŠTE	Niz znakova	Koristi se kao šifra koja jednoznačno određuje osobu
TITULA	Niz znakova	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	Niz znakova	Status sudionika koji određuje kolika će mu biti kotizacija
IZNOS KOTIZACIJE	Decimalni broj	Iznos kotizacije u eurima koji sudionik mora platiti za svaku sjednicu kojoj prisustvuje
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	Niz znakova	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 (ADRESA E-POŠTE, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA,
STATUS).

TARIFA (STATUS, IZNOS KOTIZACIJE)

RECENZENT (ADRESA E-POŠTE, TITULA, PREZIME, IME,
NAZIV USTANOVE, POŠTANSKA ADRESA,
OZNAKA TEME)

TEMA (OZNAKA TEME, NAZIV TEME, OPIS TEME)

DOZNAKA (ADRESA E-POŠTE SUDIONIKA, DATUM,
PLAĆENI IZNOS)

SJEDNICA (OZNAKA SJEDNICE, DATUM, VRIJEME OD,
VRIJEME DO, OZNAKA DVORANE, OZNAKA TEME,
ADRESA E-POŠTE PREDSJEDAVAJUĆEG)

AUTORSTVO (BROJ RADA, ADRESA E-POŠTE AUTORA)

PRIHVAĆANJE (BROJ RADA, OZNAKA SJEDNICE)

PRISUSTVO (ADRESA E-POŠTE SUDIONIKA,
OZNAKA SJEDNICE)

Slika P.11: Normalizirana inačica relacijske sheme za bazu podataka o znanstvenoj konferenciji

```
CREATE TABLE RAD (
   BROJ_RADA DECIMAL(4, 0) NOT NULL,
   NASLOV_RADA VARCHAR(160),
   BROJ_STRANICA DECIMAL(3, 0),
   PRIMARY KEY (BROJ_RADA)
);
CREATE TABLE TARIFA (
   STATUS VARCHAR(1) NOT NULL CHECK (STATUS IN ('O', 'A', 'I',
'P')),
   IZNOS_KOTIZACIJE DECIMAL(6, 2),
   PRIMARY KEY (STATUS)
);
CREATE TABLE SUDIONIK (
   ADRESA_E_POSTE VARCHAR(40) NOT NULL,
   TITULA VARCHAR(10) CHECK (TITULA IN ('prof.dr.sc', 'doc.dr.sc',
'dr.sc', 'mr.sc')),
   PREZIME VARCHAR(20),
   IME VARCHAR(20),
   NAZIV_USTANOVE VARCHAR(40),
   POSTANSKA_ADRESA VARCHAR(80),
   STATUS VARCHAR(1) NOT NULL CHECK (STATUS IN ('O', 'A', 'I',
'P')),
   PRIMARY KEY (ADRESA_E_POSTE),
);
CREATE TABLE TEMA (
   OZNAKA_TEME VARCHAR(3) NOT NULL,
   NAZIV_TEME VARCHAR(40),
   OPIS_TEME VARCHAR(160),
   PRIMARY KEY (OZNAKA_TEME)
);
CREATE TABLE RECENZENT (
   ADRESA_E_POSTE VARCHAR(40) NOT NULL,
   TITULA VARCHAR(10) CHECK (TITULA IN ('prof.dr.sc', 'doc.dr.sc',
'dr.sc', 'mr.sc')),
   PREZIME VARCHAR(20),
   IME VARCHAR(20),
   NAZIV_USTANOVE VARCHAR(40),
   POSTANSKA_ADRESA VARCHAR(80),
   OZNAKA_TEME VARCHAR(3) NOT NULL,
   PRIMARY KEY (ADRESA_E_POSTE),
);
```

Slika P.12: Fizička shema za bazu o znanstvenoj konferenciji (prvi dio)

```
CREATE TABLE DOZNAKA (
   ADRESA_E_POSTE_SUDIONIKA VARCHAR(40) NOT NULL,
   DATUM DATE NOT NULL,
   PLACENI_IZNOS DECIMAL(4, 0),
   PRIMARY KEY (ADRESA_E_POSTE_SUDIONIKA, DATUM)
);
CREATE TABLE SJEDNICA (
   OZNAKA_SJEDNICE VARCHAR(5) NOT NULL CHECK
(OZNAKA_SJEDNICE IN ('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')),
   DATUM DATE,
   VRIJEME_OD TIME,
   VRIJEME_DO TIME,
   OZNAKA_DVORANE VARCHAR(4),
   OZNAKA_TEME VARCHAR(3) NOT NULL,
   ADRESA_E_POSTE_PREDSJEDAVAJUCEG VARCHAR(40) NOT
NULL,
   PRIMARY KEY (OZNAKA_SJEDNICE)
);
CREATE TABLE AUTORSTVO (
   BROJ_RADA DECIMAL(4, 0) NOT NULL,
   ADRESA_E_POSTE_AUTORA VARCHAR(40) NOT NULL,
   PRIMARY KEY (BROJ_RADA, ADRESA_E_POSTE_AUTORA)
);
CREATE TABLE PRIHVACANJE (
   BROJ_RADA DECIMAL(4, 0) NOT NULL,
   OZNAKA_SJEDNICE VARCHAR(5) NOT NULL CHECK
(OZNAKA_SJEDNICE IN ('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')),
   PRIMARY KEY (BROJ_RADA, OZNAKA:SJEDNICE),
);
CREATE TABLE PRISUSTVO (
   ADRESA_E_POSTE_SUDIONIKA VARCHAR(40) NOT NULL,
   OZNAKA_SJEDNICE VARCHAR(5) NOT NULL CHECK
(OZNAKA_SJEDNICE IN ('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')),
   PRIMARY KEY (ADRESA_E_POSTE_SUDIONIKA,
OZNAKA_SJEDNICE),
);
```

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, 14th Edition. 2020.
- J.A.Hoffer, R. Venkataraman, H.Topi: Modern Database Management, 13th Edition, 2019 R.Elmasri, S.B.Navathe: Fundamentals of Database Systems, 7th Edition,2016

Udžbenici o projektiranju baza podataka:

- C. Churcher: Beginning Database Design – From Novice to Professional. 2nd Edition, 2012.
- M.J. Hernandez: Database Design for Mere Mortals: 25th Anniversary Edition4th Edition. 2020.
- C.Coronel, S.Morris: Database Systems: Design, Implementation, & Management, 13th Edition, 2018

Priručnici za jezik SQL:

- A. Beaulieu: Learning SQL, 3e: Generate, Manipulate, and Retrieve Data2020.
- D. Wade: Simple SQL: Beginner's Guide To Master SQL and Boost Career, 2022
- A. Molinaro, R. De Graaf: SQL Cookbook, 2E: Query Solutions and Techniques for All SQL Users 2020.

Bilješke: