DEO I

Teorija relacionih baza podataka

POGLAVLJE 1

Osnovni pojmovi

Dakle, šta je to mitsko biće, poznato pod imenom relaciona baza podataka (engl. *relational database*)? Ukratko, baza podatka je alatka koja omogućava skladištenje podataka i rad s njima, na efikasan i delotvoran način. „Efikasno i delotvorno“ znači da su podaci zaštićeni od nenamernog gubljenja ili oštećenja, da se za tu namenu ne koristi više resursa (ljudskih ili računarskih) nego što je zaista neophodno i da se podaci mogu učitavati u smislenim oblicima unutar prihvatljivih ograničenja performansi. Da bi se mogla kvalifikovati kao relaciona, baza podataka mora da realizuje relacioni model, što je način na koji se opisuje određeni aspekt stvarnog sveta, u skladu s pravilima koje je definisao dr E. F. Codd kasnih šezdestih godina.

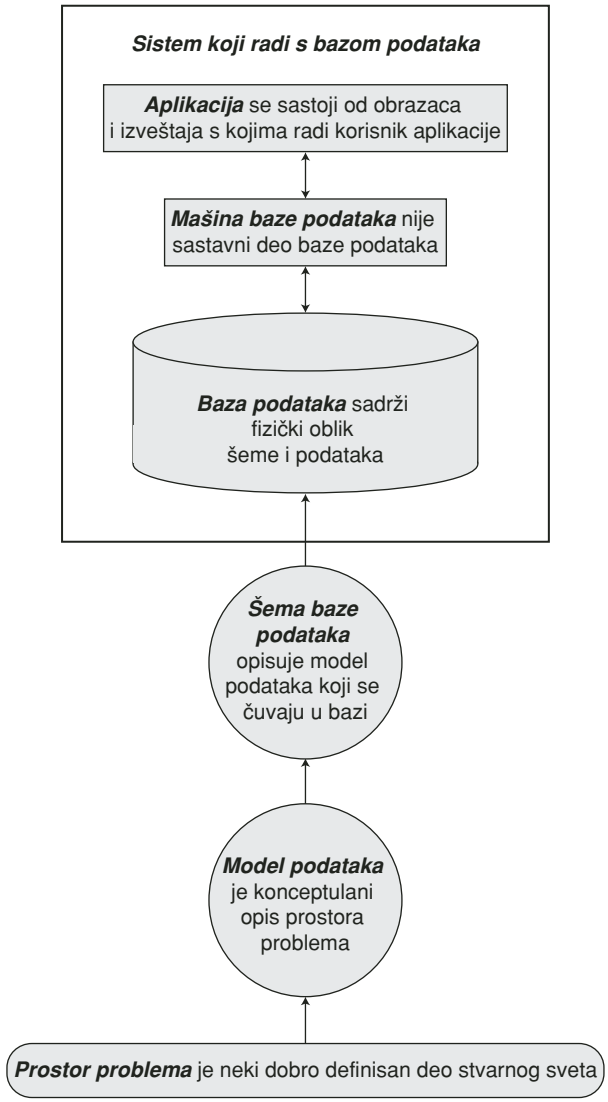
Teorijski, softver za relacionu bazu podataka može se napisati „od nule“, ali u praksi ćete uvek koristiti usluge određenog sistema za upravljanje bazama podataka (engl. *database management system*, DBMS). DBMS se ponekad naziva relacioni DBMS (RDBMS), ali tehnički gledano, da bi se jedan DBMS kvalifikovao kao relacioni, morao bi da ispuni nekih 300 uslova, a koliko ja znam, nijedan sistem koji postoji na tržištu nije u potpunosti kvalifikovan. Dva sistema za upravljanje relacionim bazama podataka koje ćemo razmatrati u ovoj knjizi, jesu Microsoftov Jet i Microsoftov SQL Server.

Već sam napomenula da je relaciona baza fizička realizacija relacionog modela (modela podataka); važno je da razlikujete ta dva pojma. Kao što ćete saznati u drugom delu knjige, dimenzionalni model može se realizovati i na relacionom DBMS-u, a ako brkate pojmovno (konceptualno) s fizičkim, bićete potpuno zbunjeni. (Da, *baš tako* govori moje iskustvo.)

Šta je to baza podataka?

Terminologija baza podataka gotovo je isto toliko neodređena koliko i izraz „objektno orijentisano programiranje“. Izraz „baza podataka“ koristi se za opisivanje svačega, u opsegu od obične grupe podataka, do složenog skupa alatki, kao što je SQL Server, a i svega između. Taj manjak preciznosti ne

mora obavezno da znači nešto loše – razlog je sama priroda jezika – ali pošto nije ni naročito koristan za naše namene, pokušaću da ovde budem preciznija. Na slici 1–1 prikazane su veze između izraza koje pominjemo u ovom poglavlju. Te izraze definišemo u ovom poglavlju, a detaljno ćemo ih razmotriti u preostalom delu knjige.



Slika 1–1 Terminologija relacionih baza podataka

Iako se za relacione baze podataka ne mogu naći analogije u stvarnom svetu, svrha većine baza podataka je da modeluju određeni aspekt iz stvarnog života. Taj delić stvarnog sveta nazvaću **prostor problema** (engl. *problem space*). Po samoj svojoj prirodi, prostor problema je zbrkan i složen – kada ne bi bio takav, ne bi ni trebalo da pravite njegov model. Međutim, za uspeh projekta od ključne je važnosti da sistem za koji projektujete bazu podataka bude ograničen na tačno definisan skup objekata i njihovih odnosa; jedino na taj način moći ćete da donosite pravilne odluke o opsegu koji sistem treba da obuhvati.

Izraz **model podataka** (engl. *data model*) koristiću za pojmovni opis prostora problema. Rad s relacionim modelom obuhvata definicije entiteta, njihovih atributa (na primer, Kupac je entitet, koji može imati atribute Prezime i Adresa) i ograničenja koja važe za atribute (kao što je, na primer, pravilo da polje ImeKupca ne može biti prazno). Kada radite s dimenzionalnim modelom, definicija modela podataka obuhvata činjenice i dimenzije, ali time ćemo se baviti u drugom delu knjige.

Model podataka takođe obuhvata opis veza ili odnosa između pojedinih entiteta, kao i ograničenja koja važe za te veze. Na primer, rukovodilac grupe ne može imati više od pet podređenih koji mu podnose izveštaje. Model podataka ništa ne govori o fizičkoj strukturi sistema.

Definicija fizičke strukture – tabela i prikaza koji će biti napravljeni – zove se **šema baze podataka** (engl. *database schema*) ili samo **šema**. To je preslikavanje pojmovnog modela u fizički oblik, koji se može realizovati pomoću nekog DBMS-a. Imajte u vidu da je šema takođe konceptualni pojam, a ne fizički. Šema nije ništa drugo do model podataka izražen pomoću elemenata koje **mašina baze podataka** (engl. *database engine*) prepoznaje, kao što su tabele, okidači i slična „bića“. Jedna od prednosti upotrebe mašine baze podataka jeste to što ne morate da se bavite fizičkim oblikom baze podataka; možete slobodno zanemariti B-stabla, čvorove na nivou listova stabla i druge fizičke koncepte nižeg nivoa.

Pošto mašini baze podataka objasnite kako želite da podaci izgledaju, pomoću SQL koda, ili u nekom interaktivnom okruženju, kao što je Microsoftov Access, ili SQL Server Enterprise Manager, mašina baze podataka pravi nekoliko fizičkih objekata (obično, ali ne uvek, negde na čvrstom disku) u koje ćete kasnije smestiti podatke. Tu kombinaciju strukture i podataka zvaću **baza podataka**. Takva baza podataka sadrži: fizičke tabele u kojima se čuvaju podaci; definisane prikaze, upite i uskladištene procedure koje omogućavaju učitavanje podataka na razne načine; pravila čije poštovanje obezbeđuje mašina baze podataka i time štiti podatke.

Izraz „baza podataka“ *ne* odnosi se na **aplikaciju** koja se sastoji od obrazaca i izveštaja s kojima radi korisnik aplikacije, niti se odnosi na bilo koju drugu vrstu softvera – kao što je posrednički sloj ili Internet Information Server – čija je namena da povezuje čeonu i pozadinsku komponentu sistema. Izraz „baza podataka“ takođe isključuje mašinu baze podataka. Prema tome, Accessova .mdb datoteka je baza podataka, dok je Microsofov Jet – mašina baze podataka. U stvari, .mdb datoteka može da sadrži i objekte aplikacije – na primer, obrasce i izveštaje – ali to je tema koju ćemo razmotriti kasnije.

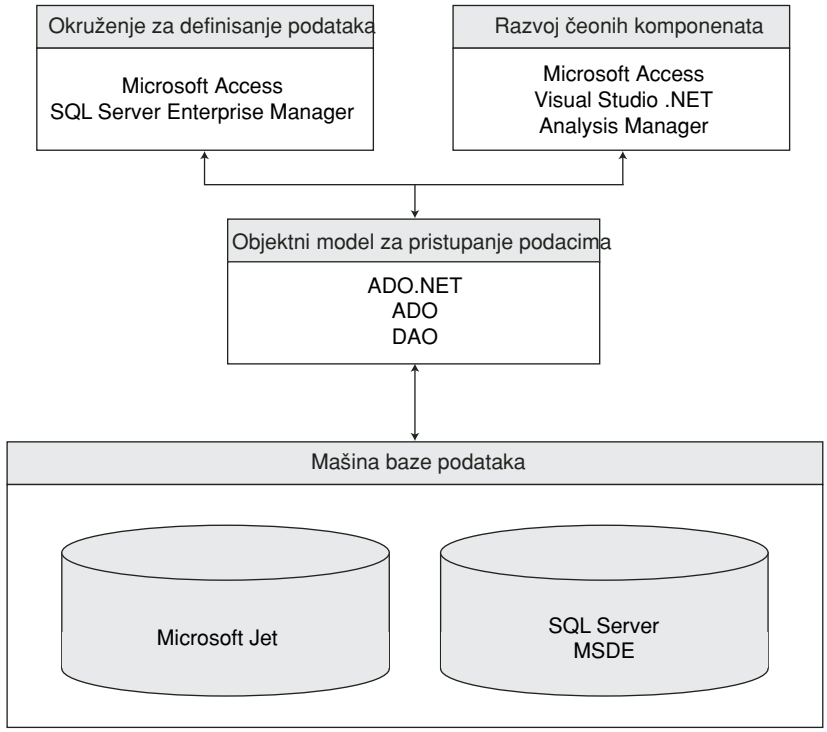
Da bih obuhvatila sve navedene komponente – aplikaciju, bazu podataka i mašinu baze podataka, koristiću izraz **sistem koji radi s bazom podataka** (engl. *database system*). Sve softverske komponente i svi podaci koji su neophodni za rad produkcionog sistema, čine sistem koji radi s bazom podataka.

Alatke za rad s bazama podataka

Iako je glavna tema ove knjige projektovanje a ne izrada baza podataka, apstraktna teorija nije naročito korisna ako ne znate kako da je primenite; zato ćemo u ovoj knjizi mnogo govoriti o izradi relacionih baza podataka pomoću Microsoftovih alatki. Na raspolaganju je veliki broj alatki te vrste, a izgleda da Microsoft svaki čas nudi neku novu; zato ćemo posvetiti malo vremena razmatranju o čemu se tu radi i gde sve to tačno spada. Na slici 1–2 prikazane su alatke koje ćemo razmatrati. Najlakše je da o njima razmišljate imajući na umu šta nama kao projektantima treba da bismo sistem preslikali iz apstraktnog modela u živi produkcioni sistem; tako su i alatke grupisane na slici.

Mašine baza podataka

Na najnižem nivou nalaze se **mašine baza podataka** (engl. *database engines*). Ponekad se nazivaju i „pozadinske komponente“ (engl. *back ends*), ali je to prilično loše, jer izraz „pozadinska komponenta“ zapravo opisuje fizičku arhitekturu, kao što ćemo videti u poglavlju 13. Zadatak tih alatki je fizičko manipulisanje podacima – smeštanje na disk i učitavanje s njega na zahtev. Razmotrićemo dve među njima: Jet i SQL Server. Možda vas iznenađuje što ovde ne pominjem Microsoftov Access. Tehnički gledano, Access je okruženje za razvoj čeonog dela aplikacije, koje za skladištenje podataka standardno koristi Jet ili SQL Server, a može da koristi i bilo koju drugu mašinu baze podataka koja podržava standard ODBC. Access radi s podacima koji se čuvaju u .mdb datotekama pomoću mašine Jet, a SQL Server (ili drugu ODBC mašinu baze podataka) koristi kada radi s podacima u .adp datotekama. Access je oduvek koristio mašinu Jet, koju Microsoft nije nudio kao zasebnu komponentu do izdavanja Visual Basic a 3.



Slika 1–2 Alatke za rad s bazama podataka koje ćemo razmatrati u ovoj knjizi

I Jet i SQL Server, mada veoma različiti, izuzetne su alatke za skladištenje podataka i rad s njima. Razlikuju se po arhitekturi i problemima za čije su rešavanje projektovane. Microsoftov Jet je „stona" mašina baze podataka, namenjena sistemima koji se po veličini mogu svrstati u opseg od malih do srednjih. (Molim vas da imate u vidu da to nikako ne znači da je mašina baze podataka Jet pogodna samo za trivijalne sisteme.) S druge strane, SQL Server koristi arhitekturu klijent/server i namenjen je sistemima u opsegu od srednjih do ogromnih, koji se potencijalno mogu smanjiti ili povećati za hiljade korisnika aplikacija od ključne važnosti za kompaniju. MSDE (akronim za **M**icro**s**oft **D**esktop **E**ngine) funkcionalno je ograničena verzija SQL Servera namenjena upotrebi u jednokorisničkom okruženju. Gledano iz ugla projektanta, razlika između MSDE-a i pune verzije SQL Servera zanemarljiva je i nećemo se baviti njome. U celoj ovoj knjizi bavićemo se razlikama između dve navedene mašine baze podataka, a u poglavlju 13 razmotraćemo kompromise između njihove dve arhitekture.

NAPOMENA O YUKONU: U vreme pisanja ove knjige (proleće 2004. godine), nova verzija SQL Servera, koja je imala radno ime „Yukon“, bila je u ranoj beta fazi. Bilo je poznato da će Yukon doneti značajnu novu funkcionalnost, ali se nije znalo šta će to tačno biti. Zbog te (razumljive) nepoznanice, ograničila sam primarni deo gradiva na mogućnosti SQL Servera 2000, koji se još uvek široko koristi. Mogućnosti za koje se očekuju značajne promene navela sam u napomenama kao što je ova.

Objektni modeli za pristupanje podacima

Microsoftov Access, a u manjoj meri i Visual Studio .NET, nude jednostavne mehanizme za povezivanje kontrola na obrascima direktno sa izvorom podataka, čime se izbegava potreba da programer radi neposredno s mašinom baze podataka. Međutim, iz raznih razloga koje ćemo razmotriti, to nije uvek ni moguće ni prikladno. U takvim slučajevima moraćete da koristite **objektni model za pristupanje podacima** (engl. *data access object model*) da biste s podacima radili pomoću programskog koda.

Objektni model za pristupanje podacima vrsta je „lepka“ između okruženja za programiranje aplikacija i mašine baze podataka; model stavlja na raspolaganje skup objekata, sa njihovim svojstvima i metodama koji se mogu koristiti u programskom kodu. Budući da se ova knjiga bavi projektovanjem više nego realizovanjem, nećemo detaljno razmatrati razlike između tih modela, ali smatram da je korisno da ih ovde ukratko pomenemo.

Microsoft (zasad) stavlja na raspolaganje tri objektna modela za pristupanje podacima: Data Access Objects (DAO), koji postoji u dve varijante (DAO/Jet i DAO/ODBCDirect), Microsoft ActiveX Data Objects (ADO) i ADO.NET.

DAO, najstariji od svih, standardni je interfejs za mašinu baze podataka Jet. Bez obzira na tvrdnje Microsoftove službe za marketing, to je najefikasniji objektni model za rad s Jetovim bazama podataka u Accessu. ADO koristi jednostavniju hijerarhiju objekata nego DAO jer se sastoji od samo četiri primarna objekta i donosi nekoliko značajnih proširenja osnovnog modela – na primer, podršku za rad s nepovezanim i hijerarhijskim skupovima podataka. Koristi se u Microsoftovom Accessu i u drugim softverskim proizvodima koji podržavaju upotrebu jezika VBA za rad sa svakom bazom podataka koja podržava ODBC interfejs, kao što su to i Jet i SQL Server. ADO.NET je, naravno, verzija modela ADO za rad unutar .NET Frameworka.

Okruženja za definisanje podataka

Microsoftov Jet i SQL Server obavljaju poslove fizičkog manipulisanja podacima umesto nas, a nama treba način da im opišemo kako da strukturiraju podatke. Microsoft nudi bezbroj metoda za tu namenu, ali ovde ćemo razmotriti samo tri: Access i SQL Server Enterprise Manager za relacione modele, i Analysis Manager za dimenzionalne modele. Postoje i druge alatke koje pružaju slične mogućnosti, ali najradije koristim navedene tri. Kada savladate principe, izabraćete alatku koja je najpogodnija za posao koji treba da obavite.

Strukturu baze podataka možete definisati i pomoću SQL koda; opisaću kako se to radi, mada u uobičajenim okolnostima to ne preporučujem. Osim kada je iz nekog razloga neophodno da izmenite strukturu podataka aplikacije dok se ona izvršava (mada nisam pobornik te prakse – ako šema baze podataka nije stabilna, verovatno niste dobro razumeli domen problema), interaktivne alatke su brže, lakše i zabavnije za upotrebu.

Razvoj čeone komponente aplikacije

Kada završite fizičku definiciju baze podataka, potrebne su vam alatke za izradu obrazaca i izveštaja s kojima će korisnici raditi. Razmotrićemo primere upotrebe dve takve alatke: Access i Visual Studio .NET (konkretno, Visual Basic .NET). U ovoj kategoriji takođe postoji bezbroj alatki za izradu čeonih komponenata, ali budući da su principi rada uvek isti, trebalo bi da budete u stanju da ono što iz ove knjige naučite primenite na alatku za koju se opredelite. U poglavlju 10 ukratko ćemo razmotriti čitače Weba, ali je sam HTML izvan opsega ove knjige.

Relacioni model

Relacioni model se zasniva na grupi matematičkih principa izvedenih iz teorije skupova i predikatne logike. Te principe je kasnih šezdesetih godina prvi primenio na oblast modelovanja podataka dr E. F. Codd, tada istraživač u kompaniji IBM, a njegovi radovi prvi put su objavljeni 1970. godine.[1] Pravila relacionog modela definišu oblik u kojem se podaci predstavljaju (struktura podataka), način na koji se podaci štite (integritet podataka) i operacije koje se mogu izvršavati nad podacima (manipulisanje podacima).

1. Codd, E. F. „A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, tom 13, broj. 6 (jun 1970).

Relacioni model nije jedini model koji postoji za skladištenje podataka i rad s njima. Ostale mogućnosti su hijerarhijski, mrežni i objektni modeli podataka. Svaki ima svoje pristalice i pruža neosporne prednosti za određene vrste poslova. Međutim, zbog svoje efikasnosti i prilagodljivosti, relacioni model je najpopularnija tehnika rada s bazama podataka i njega ćemo razmatrati u ovoj knjizi. I Jet i SQL Server realizuju relacioni model.

Razmotrićemo i dimenzionalni model, što je specijalna vrsta relacione šeme za skladištenje arhivskih podataka. Dimenzionalni model i njegove veze s relacionim modelom, detaljno ćemo obraditi u drugom delu knjige.

Uopšteno govoreći, sistemi relacionih baza podataka imaju sledeće odlike:

- Svi podaci se konceptualno predstavljaju organizovani u redove i kolone; skup tako organizovanih podataka zove se **relacija** (engl. *relation*).
- Sve vrednosti su **skalarne**. To znači da se na svakom mestu koje je određeno datim redom i kolonom, nalazi jedna i samo jedna vrednost.
- Sve operacije obavljaju se nad celom relacijom, a rezultat je takođe cela relacija. Taj koncept je poznat kao **celovitost** (engl. *closure*).

Ako ste dosad radili s bazama Microsoftovog Accessa, prepoznali ste „relaciju“ kao „skup zapisa“ (engl. *recordset*) ili, u SQL Serverovoj terminologiji, kao „skup rezultata“ (engl. *result set*). Kada je formulisao relacioni model, dr Codd je izabrao reč „relacija“ jer nije imala praktično nikakvo drugo značenje zavisno od konteksta, za razliku od, na primer, reči „tabela“. Uvreženo je pogrešno mišljenje da se relacioni model tako zove zbog veza (engl. *relationships*) koje uspostavljamo između tabela. Ime zapravo potiče od relacija na kojima se model zasniva.

Imajte u vidu to da model zahteva da podaci budu predstavljeni u obliku relacija samo na *konceptualnom* nivou; model ne propisuje fizički oblik u kojem se podaci čuvaju. Razdvajanje konceptualnog oblika od fizičkog, mada sada izgleda očigledno, bilo je velika inovacija pre 30 godina kada je programiranje baza podataka uglavnom značilo pisanje mašinskog koda za fizičko manipulisanje uređajima za skladištenje podataka.

U stvari, relacijama nije ni potreban fizički oblik. Određena relacija može se preslikati u stvarnu, fizičku tabelu koja se čuva na čvrstom disku, ali može se sastojati i od kolona koje potiču iz gomile različitih tabela, uz još nekoliko izračunatih kolona – koje fizički nigde ne postoje – dodatih tek da stvar bude složenija. Relacija se može nazvati relacijom ako su podaci organizovani u redove i kolone i ako su vrednosti podataka isključivo skalarne. Postojanje relacije potpuno je nezavisno od fizičkog oblika podataka.

Uslov da svi podaci od kojih se relacija sastoji budu skalarni, ponekad može navesti na pogrešne zaključke. Pojam „proste vrednosti“ subjektivan je sam po sebi i zavisi od semantike modela podataka. Na primer, „Ime osobe“ može biti prosta vrednost u jednom modelu, ali u nekom drugom okruženju ta vrednost će možda biti podeljena na elemente kao što su „Funkcija“, „Ime“ i „Prezime“, dok će treće okruženje možda zahtevati i „Očevo ime“ ili „Titulu“. Nijedno od navedenih rešenja nije ni bolje ni gore od drugog u apsolutnom smislu; sve zavisi od svrhe za koju će se podaci koristiti.

Princip celovitosti – tj. da se i tabele i rezultati operacija nad njima predstavljaju u obliku relacija – omogućava da se rezultati jedne operacije upotrebe kao ulazni podaci za drugu operaciju. Zahvaljujući tome, i Jet i SQL Server omogućavaju da rezultati jednog upita budu osnova za drugi upit. Ta činjenica pruža projektantima baza podataka mogućnost da operacije koje su složene ili se često obavljaju, izdvoje u zasebne celine koje se mogu ponovo izvršiti gde god i kad god je to potrebno.

Na primer, napravili ste upit koji ste nazvali UpitPunoIme; on nadovezuje sve elemente od kojih se sastoji ime osobe da bi dobio izračunato polje nazvano PunoIme. Možete zatim napraviti drugi upit čiji je izvor podataka UpitPunoIme i u kojem se izračunato polje PunoIme koristi na isti način kao da je u pitanju polje koje postoji u izvornoj tabeli. Nema potrebe da ponovo sastavljate elemente imena.

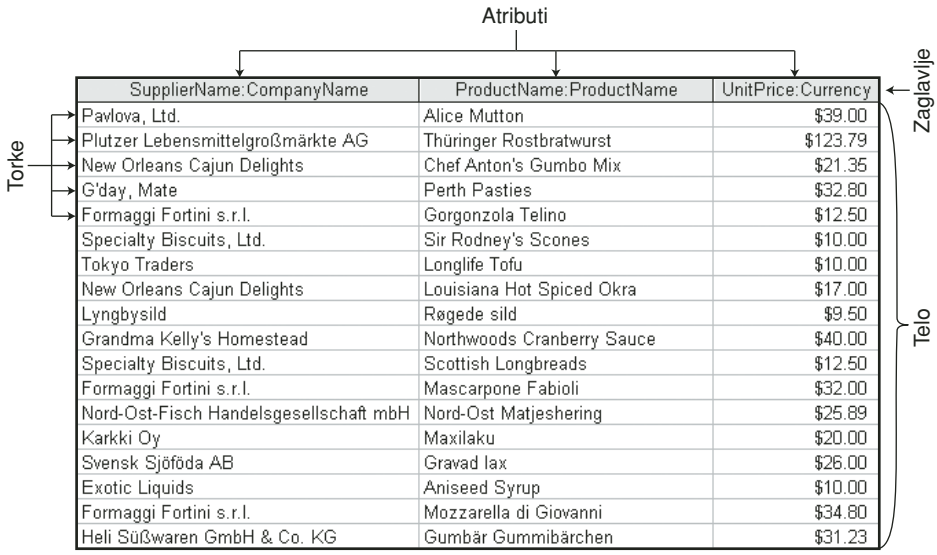
Razume se, ovo je jednostavan primer i verovatno nije sasvim očigledna neka značajna prednost koju bi pozivanje upita UpitPunoIme pružalo nad ponovnim sastavljanjem elemenata imena. Ali, kao što ćemo videti, jezik SQL omogućava sastavljanje izuzetno složenih upita, a kako raste složenost upita, tako postaju i sve vidljivije prednosti upotrebe postojećih upita kao ulaznih podataka za nove.

Relaciona terminologija

Na slici 1–3 prikazana je relacija na kojoj su označena formalna imena osnovnih komponenata. Čitaoci koji znaju nešto o projektovanju relacionih baza podataka, prepoznaće da relacija nije u normalnoj formi. To je sasvim u redu; i dalje se može nazvati relacijom zato što su podaci razmešteni u redove i kolone, a sve vrednosti su skalarne.

Kao što smo već naveli, cela struktura je **relacija**. Svaki red podataka zove se **torka** (engl. *tuple*). Tehnički gledano, svaki red je zapravo n-torka (npr. petorka, šestorka itd.), ali se „n-“ obično izostavlja. Ukupan broj torki u relaciji određuje **kardinalitet** (engl. *cardinality*) relacije. U primeru na

slici, kardinalitet je 18. Svaka kolona torke zove se **atribut** ili **obeležje** (engl. *attribute*). Ukupan broj atributa određuje **stepen** (engl. *degree*) relacije. U primeru na slici, stepen relacije je 3.



SupplierName:CompanyName	ProductName:ProductName	UnitPrice:Currency
Pavlova, Ltd.	Alice Mutton	$39.00
Plutzer Lebensmittelgroßmärkte AG	Thüringer Rostbratwurst	$123.79
New Orleans Cajun Delights	Chef Anton's Gumbo Mix	$21.35
G'day, Mate	Perth Pasties	$32.80
Formaggi Fortini s.r.l.	Gorgonzola Telino	$12.50
Specialty Biscuits, Ltd.	Sir Rodney's Scones	$10.00
Tokyo Traders	Longlife Tofu	$10.00
New Orleans Cajun Delights	Louisiana Hot Spiced Okra	$17.00
Lyngbysild	Røgede sild	$9.50
Grandma Kelly's Homestead	Northwoods Cranberry Sauce	$40.00
Specialty Biscuits, Ltd.	Scottish Longbreads	$12.50
Formaggi Fortini s.r.l.	Mascarpone Fabioli	$32.00
Nord-Ost-Fisch Handelsgesellschaft mbH	Nord-Ost Matjeshering	$25.89
Karkki Oy	Maxilaku	$20.00
Svensk Sjöföda AB	Gravad lax	$26.00
Exotic Liquids	Aniseed Syrup	$10.00
Formaggi Fortini s.r.l.	Mozzarella di Giovanni	$34.80
Heli Süßwaren GmbH & Co. KG	Gumbär Gummibärchen	$31.23

Slika 1–3 Komponente relacije

Relacija je podeljena na dva odeljka: **zaglavlje** i **telo**. Torke čine telo relacije dok se zaglavlje sastoji od ... zaglavlja. Obratite pažnju na to da se u relacionoj notaciji natpis u zaglavlju svakog atributa sastoji od dva dela, razdvojena dvotačkom – na primer, UnitPrice:Currency. Prvi deo natpisa je ime atributa, dok je drugi domen atributa. **Domen** (engl. *domain*) atributa je „vrsta“ podataka koje atribut predstavlja – u ovom slučaju to su novčani iznosi. Domen *nije* isto što i tip podataka. To pitanje ćemo detaljno razmotriti u narednom odeljku. Specifikacija domena često se izostavlja iz natpisa u zaglavlju.

Telo relacije sastoji se od neuređenog skupa jedne ili više torki. Tu imamo nekoliko važnih koncepata. Prvo, relacija nije uređena. Zamislite relaciju kao papirnatu kesu (ili skupocenu kinesku vazu, ako ste pesnički raspoloženi) u kojoj su torke nagomilane bez ikakvog posebnog redosleda. Redni brojevi zapisa, uobičajen mehanizam za pristupanje zapisima u nerelacionim bazama podataka, *ne* postoje u relacijama. Drugo, relacija bez ijedne torke (tzv. **prazna relacija**) takođe je relacija. Treće, relacija je skup. Svaki element skupa se, po definiciji, može nedvosmisleno identifikovati. Prema tome, da bi se određena tabela kvalifikovala kao relacija, svaki zapis mora da bude nedvosmisleno identifikovan i tabela ne sme da sadrži duplirane zapise.

Ako ste čitali Accessovu ili SQL Serverovu dokumentaciju, možda se sad pitate zašto niste tamo nailazili na ove reči. To su formalni izrazi koji se koriste u tehničkoj literaturi, ali ih Microsoft ne koristi. Ovde sam ih navela samo zato da se ne biste obrukali na nekom prijemu (ili barem ne ako bude reč o n-torkama trećeg stepena).

Nažalost, Microsoftovi proizvodi ne samo što nisu usklađeni s formalnom terminologijom, nego nisu usklađeni ni međusobno. Izrazi koji se koriste u dva Microsoftova proizvoda prikazani su u tabeli 1–1. U ovoj knjizi koristiću formalnu terminologiju.

Tabela 1–1 Relaciona terminologija koja se koristi u proizvodima razmotrenim u ovoj knjizi

Formalna terminologija			
Konceptualna	**Fizička**	**Microsoft Access**	**SQL Server**
relacija	tabela	tabela ili skup zapisa	tabela ili skup rezultata
atribut	polje	polje	kolona
torka	zapis	zapis	red

Model podataka

Najapstraktniji nivo projektovanja baze podataka jeste **model podataka**, što je konceptualni opis prostora problema. Modeli podataka sastoje se od elemenata koji mogu biti entiteti, atributi, domeni i veze. U preostalom delu ovog poglavlja pojedinačno razmatram te elemente.

Entiteti

Teško je osmisliti preciznu formalnu definiciju entiteta (engl. *entity*), ali je pojam relativno lako razumljiv: entitet je sve o čemu sistem treba da skladišti podatke.

Kada počnete da projektujete model podataka, neće vam biti teško da sastavite početnu listu entiteta. Kada vi (ili vaši klijenti) govorite o prostoru problema, većina imenica i glagola kandidati su za entitete. „Kupci kupuju robu. Prodavci prodaju robu. Dobavljači nam prodaju robu.“ Imenice „Kupci“, „Roba“, „Prodavci“ i „Dobavljači“ predstavljaju očigledne entitete.

Događaji predstavljeni glagolima „kupiti“ i „prodati“ takođe su entiteti, ali tu postoji nekoliko zamki. Prvo, glagol „prodati“ predstavlja dva različita

događaja: prodaju robe kupcu (Prodavac→Kupac) i nabavljanje robe za prodaju (Dobavljač→Kompanija). U ovom primeru, razlika je sasvim očigledna, ali je to zamka u koju se lako upada, naročito ako niste dobro proučili prostor problema.

Druga zamka je suprotna prvoj: dva različita glagola („kupuju" u prvoj rečenici i „prodaju" u drugoj) opisuju zapravo isti događaj, tj. da je kupac kupio određenu robu. Ponavljam, to nije uvek očigledno, osim kada dobro poznajete prostor problema. Ovaj problem se često teže uočava od prvog. Ako klijent koristi različite izraze da bi opisao nešto što vama na prvi pogled izgleda kao isti događaj, on možda govori o različitim vrstama događaja. Na primer, ako je vaš klijent proizvođač konfekcije, rezultat događaja „kupac kupuje odelo" i „kupac naručuje odelo" u oba slučaja je prodato odelo, ali u prvom slučaju to može biti kupovina već gotovog konfekcijskog odela, dok se u drugom slučaju radi o šivenju odela po meri. To su dve vrlo različite radnje koje se moraju modelovati na različite načine.

Osim razgovora s klijentima na osnovu kojih ćete sastaviti listu entiteta, korisno je i da proučite dokumente koji postoje u prostoru problema. Obrasci za unošenje raznih podataka, izveštaji i pisana uputstva, dobri su izvori kandidata za entitete. Međutim, morate biti oprezni kada radite s dokumentima. Pisani dokumenti mogu biti prilično zastareli – štampani obrasci za unošenje podataka mogu biti poprilično skupi i često ne prate promene poslovnih pravila i postupaka. Ako naiđete na entitet koji se ne pominje ni u jednom razgovoru, nemojte pretpostaviti da je klijent zaboravio da ga pomene. Verovatnije je da taj entitet više nije važan organizaciji. Moraćete sami da ispitate šta je tačno posredi.

Većina entiteta su modeli objekata ili događaja iz stvarnog života: kupci, roba, prodajne ponude. To su **konkretni entiteti**. Entiteti mogu biti i modeli apstraktnih koncepata. Najuobičajeniji primer **apstraktnog entiteta** jeste entitet koji modeluje vezu ili odnos (engl. *relationship*) između drugih entiteta – na primer, činjenicu da je određeni predstavnik kompanije odgovoran za određenog klijenta, ili da određeni student sluša predavanja iz određenog predmeta.

Ponekad je dovoljno da modelujete samo činjenicu da veza ili odnos postoji. U drugim slučajevima može biti potrebno da o toj vezi evidentirate i dodatne podatke, kao što je datum uspostavljanja veze, ili neku odliku te veze (ili odnosa). Na primer, odnos između pume i kojota je konkurentski, dok je odnos između pume i zeca takav da puma lovi zeca, što je korisno znati ako planirate da izgradite zoološki vrt bez kaveza.

Postoje različita mišljenja o tome da li bi veze između entiteta koje nemaju sopstvene atribute trebalo da se modeluju kao zasebni entiteti. Mislim da se time ništa ne dobija, a komplikuje se postupak izvođenja šeme iz

modela podataka. Međutim, shvatanje da su veze/odnosi između entiteta podjednako važni kao i sami entiteti, ključno je za projektovanje kvalitetnog modela podataka.

Atributi

Sistem koji projektujete moraće da evidentira određene činjenice o svakom entitetu. Kao što smo već videli, te činjenice su atributi entiteta. Na primer, ako u svom sistemu imate entitet Kupac, verovatno će vam biti važno da znate imena i adrese kupaca, a možda i njihova zanimanja. Ako modelujete događaj kao što je ZahtevZaServisiranje, verovatno ćete hteti da znate koji je klijent u pitanju, ko je zvao, kad je zvao i da li je problem rešen. Svi navedeni elementi su atributi.

Određivanje atributa koje ćete ugraditi u svoj model jeste semantički postupak, što znači da odluke morate donositi na osnovu značenja podataka i načina na koji će se oni koristiti. Pogledajmo čest primer: adresu. Da li ćete adresu modelovati kao jedan entitet (Adresa) ili kao grupu više entiteta (Ulica, Broj, Grad, PoštanskiBroj, Država)? Većina projektanata (među koje ubrajam i sebe) obično automatski razbija adresu na grupu atributa zbog opšteg principa da se lakše radi sa strukturiranim podacima, ali to nije uvek tačno, a svakako nije očigledno.

Kao primer, uzmimo lokalno udruženje muzičara amatera. Potrebno im je da evidentiraju adrese svojih članova kako bi mogli da štampaju nalepnice za koverte. Pošto je to jedina svrha za koje će se adrese koristiti, nema razloga da se adresa ikada tretira drugačije nego kao jedan blok od više redova teksta, koji se štampa na zahtev.

Ali šta je sa kompanijom koja se bavi prodajom robe u SAD putem Interneta? Zbog obračuna poreza, kompaniji je potrebno da zna u kojoj saveznoj državi živi svaki njen kupac. Iako se podatak o saveznoj državi može izdvojiti iz tekstualnog polja koje odgovara udruženju muzičara, to nije lako; prema tome, logično je da u ovom slučaju treba modelovati barem saveznu državu kao zaseban atribut. Šta je sa ostalim delom adrese? Da li bi trebalo da ga razbijemo na više atributa i koji bi to atributi bili? Možda biste pomislili da bi odgovarao skup atributa {Kućni broj, Ulica, Grad, Savezna država, Poštanski broj}? Međutim, može zatrebati i broj stana, broj poštanskog pretinca i APO adresa. Šta ćete uraditi ako je u pitanju uslužna adresa koja pripada nekom drugom? Pošto svet postaje sve manji, ali ne manje složen, šta će se desiti kada dobijete prvog klijenta u inostranstvu? Domaće adrese imaju prilično standardizovan format, ali to ne važi za porudžbine iz inostranstva.

Ne samo što morate da znate ciljnu državu i da tome prilagodite format poštanskog broja, nego ćete možda morati da menjate i redosled atributa. Na primer, za razliku od SAD, u većem delu Evrope kućni broj se piše iza imena ulice. To nije tako loše, lako ćete se setiti toga kada budete unosili podatke, ali koliko će vaših operatera znati da adresa 4/32 Griffen Avenue, Bondi Beach, Australija, znači stan broj 4, u zgradi na broju 32?

Suština u ovom slučaju nije to da je teško modelovati adresu (mada jeste), nego da ne možete unapred određivati kako ćete modelovati određenu vrstu podataka. Složen sistem koji razvijete za obradu međunarodnih porudžbina, biće potpuno neprikladan za udruženje muzičara.

Slikaru Matisu pripisuje se tvrdnja da je slika gotova tek kad joj se više ništa ne može ni dodati ni oduzeti. Projektovanje entiteta pomalo je slično tome. Kako znate da ste stigli u tu fazu? Nažalost, odgovor glasi da u to nikad ne možete biti potpuno sigurni (a čak i ako mislite da jeste, s vremenom ćete promeniti mišljenje). Pri tekućem stanju tehnologije, ne postoji način da projektujete strukturu baze podataka za koju se može dokazati da je potpuno ispravna. Možete dokazati da u određenoj strukturi ima propusta i grešaka, ali ne možete dokazati da ih u drugoj strukturi nema. To znači, otprilike, da ne možete dokazati svoju „nevinost". Kako se može rešiti taj problem? Nema strogih pravila, ali postoji nekoliko strategija.

Prva strategija: **Krenite od rezultata i nemojte praviti složeniju strukturu nego što je zaista potrebno**. Na koja pitanja vaša baza podataka mora davati odgovore? Pošto je u našem prvom primeru, udruženju muzičara, jedino pitanje bilo: „Na koju adresu treba poslati pismo za tu i tu osobu?", model s jednim atributom za adresu bio je dovoljan. U drugom primeru, kompaniji koja prodaje robu putem Interneta, bio je potreban i odgovor na pitanje: „U kojoj saveznoj državi živi ta i ta osoba?", pa nam je zato bila neophodna drugačija struktura adrese da bismo došli do zahtevanog rezultata.

Razume se, morate voditi računa i pokušati da obezbedite fleksibilnost koja će pružiti odgovore, i to ne samo na pitanja koja korisnici postavljaju sada, već i na ona koja možete predvideti da će se pojaviti u budućnosti. Na primer, kladila bih se da će u roku od godinu dana nakon puštanja sistema u redovnu upotrebu, udruženje muzičara zatražiti od vas da im sortirate adrese po poštanskom broju da bi dobili količinski popust.

Trebalo bi takođe da očekujete i pitanja koja bi korisnici postavili kada bi znali da mogu da ih postave, naročito kada automatizujete postojeći ručni sistem. Zamislite da rukovodilac biblioteke želi da zna koliko je iz fonda od četiri miliona knjiga bilo objavljeno u Novom Sadu pre 1900. godine. On ili ona bi vam pokazali orman s kartotekom i poželeli vam dobru zabavu. Međutim, u dobro projektovanom sistemu koji radi s bazom podataka, ta vrsta zahteva smatra se trivijalnim.

Jedan od zaštitnih znakova dobrih projektanata jeste temeljitost i kreativnost s kojima podstiču potencijalna pitanja. Neiskusni analitičari često tvrde da korisnici ne znaju šta tačno hoće. Naravno da ne znaju; vaš posao je da im pomognete da otkriju šta zapravo žele.

Međutim, pri tome morate biti oprezni. „Cena" fleksibilnosti često je veći stepen složenosti. Kao što smo videli na primeru adrese, što više „seckate" i usitnjavate podatke, više ćete imati posebnih slučajeva za obradu, a to će vas dovesti do tačke kada predloženo rešenje počinje da gubi smisao.

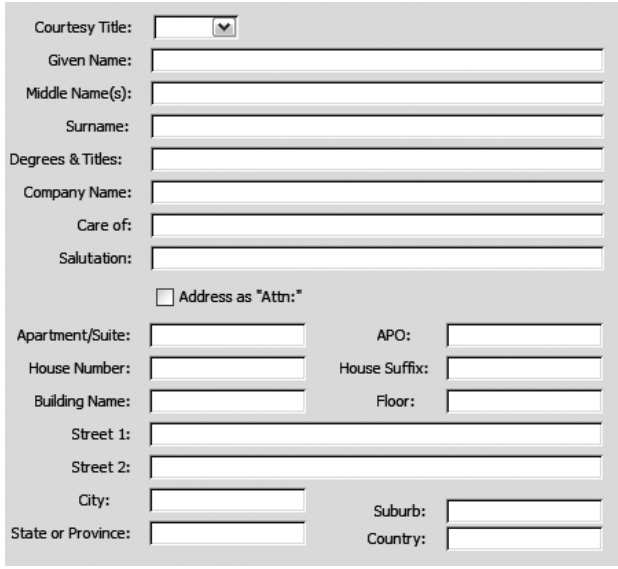
Tako stižemo do strategije broj dva: **Otkrijte izuzetke**. Ova strategija ima dva aspekta. Prvo, morate identifikovati sve izuzetke, i drugo, sistem morate projektovati tako da obrađuje što veći broj izuzetaka, ali da pri tome ne zbunjuje korisnike. Kao ilustraciju šta tačno znači ovaj princip, razmotrićemo još jedan primer: imena osoba.

Ako će namena sistema koji projektujete biti korespondencija, od ključne je važnosti da imena osoba budu tačno navedena. (Prikladan primer: svu poštu koju dobijam na kućnu adresu, a primalac je *gospodin* R. M. Riordan, uopšte i ne otvaram.) Većina imena osoba prilično je jasna sama po sebi. Ako je primalac „gospođa Jelena K. Jovanović", elementi imena su: Oslovljavanje, Ime, OčevoIme i Prezime, zar ne? Pogrešno. (Osetili ste da ima neka začkoljica, a?) Sigurnije je (i pravilnije po bontonu) koristiti UobičajenoIme[2] i Prezime. Dalje, šta ćete uraditi kad je primalac Sir James Peddington Smythe, Lord Dunstable? Peddington Smythe je u tom slučaju Prezime, ili je to OčevoIme? Šta je onda „Lord Dunstable"? A pevač Sting? Da li je to UobičajenoIme ili Prezime? A šta će se desiti Umetniku Ranije Poznatom Pod Imenom Prince? Da li vas to zaista zanima?

Poslednje pitanje nije tako besmisleno kao što se čini. Pismo čiji je primalac naveden kao Sir James Peddington Smythe, verovatno neće nikoga uvrediti. Ali ime pomenutog gospodina s plemićkom titulom nije Sir Smythe; u javnosti ga oslovljavaju sa Sir James ili možda Lord Dunstable. Međutim, budimo realni, koliko vaših klijenata ima titulu lorda? Lokalno udruženje muzičara sigurno vam neće zahvaljivati ako im napravite ekran nalik na onaj sa slike 1–4.

Imajte u vidu da morate napraviti kompromis između fleksibilnosti i složenosti. Mada je važno da obuhvatite što veći broj izuzetaka, potpuno je razumno da neke od njih zanemarite jer su suviše malo verovatno da bi opravdali „cenu" svog uključivanja u sistem.

2. U nekim kulturama je uobičajeno da ljudi dobijaju dva, tri ili više imena pri rođenju. U krštenici se navode sva imena, ali se u praksi koristi samo jedno, koje se zove *uobičajeno ime* (engl. *given name*). (Prim. prev.)



Slika 1–4 Previše složen ekran za unošenje adresa

Razlikovanje entiteta od atributa ponekad može biti teško. Reći će vam opet da je adresa dobar primer kao i da vaša odluka zavisi od prostora problema. Neki projektanti predlažu upotrebu samo jednog entiteta za adresu koji bi predstavljao sve vrste adresa koje se čuvaju u sistemu. Sa tačke gledišta praktične realizacije, to rešenje pruža nesporne prednosti u pogledu kapsuliranja i višekratne upotrebe koda. Međutim, sa tačke gledišta strukture baze podataka, imam određene rezerve.

Na primer, malo je verovatno da će se adrese zaposlenih i kupaca koristiti na isti način i za iste namene. Verovatnije je da će se cirkularne poruke zaposlenima slati putem interne e-pošte nego putem klasične pošte. Ako je tako, za drugi slučaj pravila i zahtevi su drugačiji. Grozan ekran za unošenje podataka prikazan na slici 1–4, ili nešto slično tome, može biti sasvim prikladan za adrese kupaca. Međutim, ako imate samo jedan entitet za adrese, morate koristiti isti ekran i za adrese zaposlenih; malo je verovatno da je to potrebno, a još manje da će se svideti korisnicima.

Domeni

Možda se sećate s početka ovog poglavlja da se u zaglavlju relacija nalaze parovi ImeAtributa:ImeDomena za svaki atribut. Rečeno je da definicija domena određuje vrstu podataka koje predstavlja atribut. Tačnije rečeno, **domen** (engl. *domain*) je skup svih prihvatljivih vrednosti koje atribut može imati.

Domeni se često brkaju s tipovima podataka; to su dva različita pojma. Tip podataka je fizički pojam, dok je domen logički pojam. „Broj“ je tip podataka; „Starost“ je domen. I „Ulica“ i „Prezime“ mogu biti predstavljeni poljima tekstualnog tipa, ali je očigledno da su u pitanju različite vrste tekstualnih polja, koja pripadaju različitim domenima.

Domen je uži pojam od tipa podataka jer definicija domena zahteva precizniji opis validnih podataka. Uzmite kao primer domen StručnaSprema, koji predstavlja stručnu spremu osobe. U šemi baze podataka, taj atribut se može definisati kao Text, ali to ne može biti bilo koji tekst, već samo element iz skupa {niža, srednja, viša, visoka, magistratura, doktorat}.

Razume se, ne možete sve domene definisati pojedinačnim navođenjem prihvatljivih vrednosti. Starost, na primer, sadrži otprilike stotinak vrednosti ako je u pitanju starost osoba, ali to mogu biti desetine hiljada različitih vrednosti kada su u pitanju muzejski eksponati. U takvim slučajevima, umesto u obliku liste vrednosti, lakše je definisati domen u obliku pravila koja utvrđuju da li određena vrednost pripada skupu prihvatljivih vrednosti. Na primer, StarostOsobe može se definisati kao „celobrojna vrednost u opsegu od 0 do 120“, dok bi StarostEksponata bila „celobrojna vrednost veća od 0“.

Možda ste sad pomislili da je domen kombinacija tipa podataka i pravila za ispravnost podataka. Ako tako mislite, nećete daleko stići. Pravila ispravnosti podataka isključivo su deo integriteta podataka, a ne opisa podataka. Na primer, pravilo ispravnosti za poštanske brojeve određuje da su prihvaljivi samo određeni brojevi, dok je domen za poštanske brojeve „petocifreni broj“.

Obratite pažnju na to da se u navedenim definicijama pominje vrsta podataka koji se evidentiraju (znakovni ili numerički). To puno podseća na tip podataka, ali ipak nije tako. Tipovi podataka su, kao što je već napomenuto, fizički pojam; oni se definišu i zadaju u obliku koji prepoznaje mašina baze podataka. Bilo bi pogrešno definisati domen kao varchar(30) ili Long Integer jer su to opisi specifični za određenu mašinu baze podataka (SQL Server).

Za svaka dva data domena, ako je moguće porediti atribute definisane u tim domenima (pa prema tome, obavljati i relacione operacije kao što je spajanje, što ćemo razmotriti u poglavlju 5), kaže se da su oni **kompatibilni po tipu** (engl. *type compatible*). Na primer, dve relacije prikazane na slici 1–5 mogu se povezati putem atributa EmployeeID = SalespersonID (šifra zaposlenog = šifra prodavca). To možete uraditi npr. da biste dobili spisak faktura koje je svaki zaposleni izdao. Domeni EmployeeID i SalespersonID kompatibilni su po tipu. Međutim, ako pokušate da kombinujete te dve relacije putem atributa EmployeeID = OrderID (šifra zaposlenog = broj porudžbine), verovatno nećete dobiti smislen rezultat, uprkos tome što su oba domena definisana sa istim tipom podataka.

Order ID	Customer	Salesperson ID	Order Date
10643	Alfreds Futterkiste	6	25-Aug-1997
10692	Alfreds Futterkiste	4	03-Oct-1997
10702	Alfreds Futterkiste	4	13-Oct-1997
10835	Alfreds Futterkiste	1	15-Jan-1998
10952	Alfreds Futterkiste	1	16-Mar-1998

Employee ID	Title Of Courtesy	Given Name	Surname	Title
1	Ms.	Nancy	Davolio	Sales Representative
2	Dr.	Andrew	Fuller	Vice President, Sales
3	Ms.	Janet	Leverling	Sales Representative
4	Mrs.	Margaret	Peacock	Sales Representative
5	Mr.	Steven	Buchanan	Sales Manager
6	Mr.	Michael	Suyama	Sales Representative
7	Mr.	Robert	King	Sales Representative
8	Ms.	Laura	Callahan	Inside Sales Coordinator
9	Ms.	Anne	Dodsworth	Sales Representative

Slika 1–5 Relacije Employees (zaposleni) i Orders (porudžbine)

Nažalost, ni mašina baze podataka Jet, niti SQL Server ne pružaju ugrađenu podršku za domene koja bi bila jača od obične provere tipova podataka. Čak i za tipove podataka, nijedna mašina baze podataka ne obavlja striktnu proveru tipa: obe će mirno konvertovati podatke u pozadini. Na primer, ako koristite Microsoftov Access i u tabeli Employees definisali ste da je EmployeeID tipa Long Integer, a u tabeli Invoices (fakture) atribut InvoiceTotal (ukupan zbir) definisan je kao tip Currency, možete napraviti upit koji povezuje te dve tabele kao EmployeeID = InvoiceTotal. Microsoftov Jet će vam ljubazno sastaviti spisak zaposlenih čije su šifre (EmployeeID) jednake ukupnim zbirovima određenih faktura. Ta dva atributa nisu kompatibilna po tipu, ali to Jet ne zna ili zanemaruje.

Zašto biste se onda uopšte petljali s domenima? Zato što su oni, kao što ćemo videti u trećem delu knjige, izuzetno korisne alatke pri projektovanju baza podataka. „Da li su ta dva atributa međusobno zamenjivi?“ „Postoje li pravila koja važe u jednom, a ne važe u drugom domenu?“ To su važna pitanja kada projektujete model podataka, a analiza domena vam pomaže da nađete odgovore.

Veze/odnosi između entiteta

Osim atributa svakog entiteta, model podataka mora da definiše i veze ili odnose koji postoje između entiteta. Na pojmovnom nivou, **veza** ili **odnos** (engl. *relationship*) jeste asocijacija između entiteta. Rečenica „kupci kupuju robu“ pokazuje da postoji veza između entiteta Kupci i Roba. Entiteti između kojih postoji veza ili odnos zovu se **učesnici veze** (engl. *praticipants*).

Broj učesnika određuje **stepen** veze. (Stepen veze je sličan, ali ne i jednak, stepenu relacije, što je ukupan broj atributa.)

Velika većina veza između entiteta je binarna, kao u primeru „kupci kupuju robu“, ali to nije pravilo. Uobičajene su i ternarne veze, tj. one s tri učesnika. Binarne veze „zaposleni prodaju robu“ i „kupci kupuju robu“ implicitno podrazumevaju postojanje ternarne veze „zaposleni prodaju robu kupcima“. Međutim, navedene dve binarne veze ne omogućavaju nam da saznamo koji su zaposleni koju robu prodali kojim kupcima; to se može saznati samo pomoću ternarne veze.

Specijalan slučaj binarne veze je entitet koju učestvuje u vezi sa samim sobom. To se često zove veza tipa sastavnice (engl. *bill of materials relationship*) i najčešće se koristi za predstavljanje hijerarhijskih struktura. Uobičajen primer je odnos između zaposlenih i nadređenih rukovodilaca. Svaki zaposleni može istovremeno i *biti* rukovodilac i *imati* sebi nadređenog rukovodioca.

Veza ili odnos između dva entiteta može biti tipa „jedan prema jedan“, „jedan prema više“ ili „više prema više“. Veze tipa „jedan prema jedan“ su retke, ali mogu biti korisne u nekim okolnostima.

Veze tipa „jedan prema više“ verovatno su najuobičajenija vrsta. Jedna faktura se sastoji od više artikala. Jedan prodavac izdaje više faktura. U oba primera imamo vezu tipa „jedan prema više“.

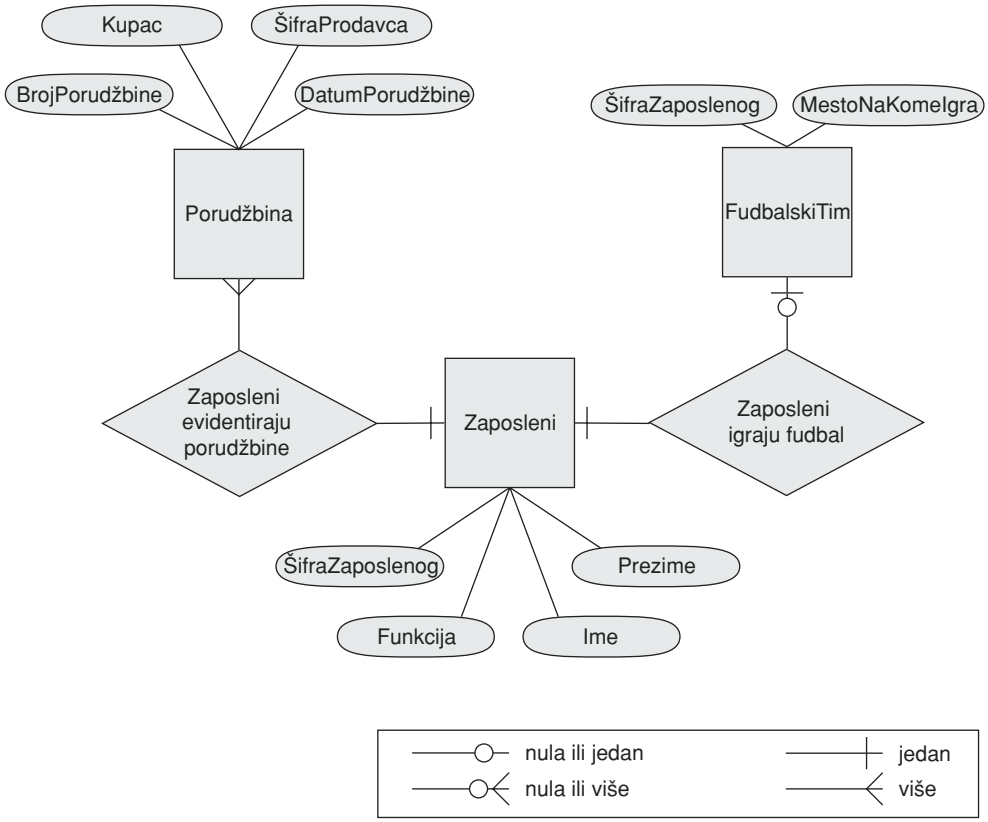
Iako nisu toliko česte kao veze tipa „jedan prema više“, veze tipa „više prema više“ nisu neuobičajene i mogu se naći brojni primeri. Jedan kupac kupuje više artikala, a isti artikal može kupiti više kupaca. Više studenata sluša predavanja jednog profesora, a isti student može slušati predavanja više profesora. Veze tipa „više prema više“ ne mogu se direktno predstaviti u relacionom modelu, ali je njihov indirektan oblik vrlo jednostavan, kao što ćemo videti u poglavlju 3.

Učestvovanje jednog entiteta u vezi može biti **delimično** ili **potpuno**. Ako entitet ne može da postoji ukoliko ne učestvuje u nekoj vezi, učestvovanje je potpuno; u suprotnom, učestvovanje je delimično. Na primer, podaci o određenom Prodavcu ne mogu logički da postoje ako ne postoji odgovarajući Zaposleni. Obrnuto nije tačno. Pošto zaposleni može biti i nešto drugo osim prodavca, može postojati zapis o Zaposlenom bez odgovarajućeg zapisa među Prodavcima. Prema tome, učestvovanje Zaposlenog u vezi je delimično, dok je učestvovanje Prodavca potpuno.

Trik je u tome da tvrdnja o delimičnom ili potpunom učestvovanju bude tačna za sve instance (primerke) entiteta, u svim slučajevima. Na primer, nije neuobičajeno da kompanija promeni dobavljača za određeni artikal. Ako je učestvovanje entiteta Artikal u vezi „dobavljači dobavljaju artikle“ definisano kao potpuno, tekućeg dobavljača nećete moći da izbrišete ako ne izbrišete i sve podatke o artiklima koje vam on dobavlja.

Dijagrami entiteta i veza

Model entiteta i veza, koji podatke opisuje izražene kao entitete, atribute i veze, uveo je Peter Pin Shan Chen 1976. godine.[3] Istovremeno, predložio je metodu predstavljanja pomoću dijagrama nazvanih Entity Relationship (E/R) dijagrami, koja je postala široko prihvaćena. E/R dijagrami se sastoje od pravougaonika, koji predstavljaju entitete, elipsi, koje predstavljaju atribute i rombova, koji predstavljaju veze između entiteta (slika 1–6).



Slika 1–6 Primer E/R dijagrama

3. Peter Pin Shan Chen, „The Entity Relationship Model – Toward a Unified View of Data?“, ACM TODS 1, broj 1 (mart 1976).

Priroda veza između entiteta („jedan prema jedan“, „jedan prema više“ ili „više prema više“) predstavlja se na više načina. Mnogi koriste simbole 1 i M ili 1 i ∞ (simbol za beskonačnost). Ja koristim oblik „vranina kandža“, prikazan na slici 1–6, koji smatram izražajnijim.

Velika prednost E/R dijagrama jeste to što se lako crtaju i razumeju. Ja obično crtam atribute na odvojenom dijagramu jer se mogu praviti dijagrami sa različitim nivoima detalja. Uglavnom, na dijagramu razmatramo ili entitete koji čine model i veze između njih, ili atribute datog entiteta, ali retko razmatramo oba aspekta istovremeno.

Sažetak

U ovom poglavlju razmotrili smo komponente sistema koji radi s bazom podataka i postavili osnovu za preostali deo gradiva u knjizi. Počeli smo od opisa prostora problema kao određenog, precizno definisanog dela stvarnog sveta. Konceptualni model podataka je opis prostora problema izražen pomoću entiteta, atributa i domena u kojima su definisani. Fizička struktura modela podataka je šema baze podataka, koja se realizuje u bazi podataka. I najzad, mašina baze podataka fizički manipuliše podacima na zahtev aplikacije sačinjene od obrazaca i izveštaja s kojima korisnici rade.

U narednom poglavlju, detaljnije ćemo proučiti strukturu baze podataka dok budemo razmatrali principe normalizacije.