Objektno-relacijske baze podataka

Kozina, Katarina

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:944263

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Kozina, Katarina

OBJEKTNO-RELACIJSKE BAZE PODATAKA

Diplomski rad

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Katarina Kozina

OBJEKTNO-RELACIJSKE BAZE PODATAKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Ovaj rad posvećujem najdražima koji su mi pomogli da postanem osoba koja sam danas.
Obitelji, hvala na podršci i razumijevanju tijekom studiranja.
Goranu, hvala za sve. Na svim savjetima i bodrenju.
Posebno, veliko hvala mentoru prof. dr. sc. Robertu Mangeru na strpljenju, pomoći i vodstvu pri izradi ovog diplomskog rada.

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Općenito o bazama podataka **3**
1.1 Osnovni pojmovi 3
1.2 Jezici za rad s bazama podataka 5
1.3 Modeliranje podataka 6

2 Relacijske baze podataka **11**
2.1 Modeliranje entiteta i veza 11
2.2 Relacijski model 16

3 Objektno-orijentirane baze podataka **21**
3.1 UML-ov dijagram klasa 23
3.2 Objektni model 26

4 Usporedba relacijskih i objektno-orijentiranih baza podataka **29**

5 Objektno-relacijske baze podataka **31**
5.1 Objektno-relacijski model podataka 31
5.2 SQL standard: objektno-relacijska proširenja 32
5.3 Oblikovanje objektno-relacijskih baza podataka 38

6 Studijski primjer **43**

Zaključak **55**

Bibliografija **57**

Uvod

Danas se kaže da su podaci novo zlato, te da su jedan od najvažnijih resursa u poslovnom svijetu. Nalaze se svuda oko nas i dolaze u raznim oblicima i količinama. Kako bi iz mora podataka dobili adekvatne i potrebne informacije važan nam je i informacijski sustav koji pohranjuje i obrađuje prikupljene podatke. Za to nam služe baze podataka. Relacijske baze podataka najzastupljenije su u današnjem svijetu, ali naglim porastom količine podataka, razvojem informacijske tehnologije i nemogućnosti relacijskih baza da uspješno pohranjuju nove složenije tipove podataka na scenu stupaju objektno-orijentirane baze podataka. Međutim, iako objektno-orijentirane baze podataka zadovoljavaju potrebe novijih tehnologija njihova upotreba nije odvela relacijske baze podataka u prošlost. Prednosti, a posebno nedostaci relacijskih i objektno-orijentiranih baza podataka, pokazuju da poslovan svijet zahtjeva sposobnosti oba modela podataka. Kao rezultat razvijena je objektno-relacijska baza podataka.

Ovaj diplomski rad sastoji se od šest poglavlja. Prvo poglavlje kreće od osnovnih pojmova vezanih općenito za baze podataka, zatim opisuje jezike za rad s bazama podataka i na kraju po koracima opisujemo proces modeliranja podataka. U drugom poglavlju opisat ćemo relacijske baze podataka, objasniti osnovne pojmove vezane za njih te na kraju opisati relacijski model. Zatim ćemo po istom principu u trećem poglavlju opisati objektno-orijentirane baze podataka. U četvrtom poglavlju opisujemo razlike, prednosti i nedostatke relacijskih i objektno-orijentiranih baza podataka. Zatim u petom poglavlju opisujemo objektno-relacijske baze podataka koje su nastale kako bi se dobilo ono najbolje iz relacijskih i objektno-orijentiranih baza podataka. Diplomski rad završava šestim poglavljem u kojem je prikazan studijski primjer oblikovanja i testiranja objektno-relacijske baze podataka.

Poglavlje 1

Općenito o bazama podataka

Baze podataka razvile su se 60-tih godina 20. stoljeća s ciljem stvaranja bolje tehnologije za upravljanje i rad s podacima. Njihov razvoj osigurao je veću kvalitetu, pouzdanost i produktivnost u razvoju aplikacija koje ovise o pohranjivanju i pretraživanju podataka.

1.1 Osnovni pojmovi

Baza podataka

Baza podataka (eng. Database - DB) je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijama [5]. Čitanje, upravljanje, mijenjanje i brisanje podataka vrši se pomoću posebnog softvera odnosno sustava za upravljanje bazom podataka.

Sustav za upravljanje bazom podataka

Sustav za upravljanje bazom podataka (eng. DataBase Management System - DBMS) je poslužitelj (server) baze podataka. Većina korisnika i tvrtki ne razvija samostano DBMS, već kupuju licence kako bi mogli koristiti neki od komercijalnih DBMS-ova. Mnoge tvrtke uz sam DBMS uključuju i dodatne alate za administriranje baze i slično [5]. DBMS ima niz funkcija:

- **Funkcije za definiranje baze podataka:** u skladu s logičkom strukturom oblikujemo fizički prikaz baze; koristimo standardni jezik za rad s bazom podataka.
- **Funkcije za manipulaciju podacima u bazi podataka:** koristimo neproceduralni jezik kao što je SQL ili dodajemo DML naredbe u standardne programske jezike.

- **Upravljačke funkcije:** funkcije sigurnosti podataka, odnosno zaštite od neovlaštenog korištenja, funkcije očuvanja integriteta baze podataka, funkcije statističkog praćenja baze podataka.

Navedimo neke, za ovaj rad bitne, vrste DBMS-a:

- **relacijski DBMS** (eng. Relational Database Management System - RDBMS)
  DBMS temeljen na relacijskom modelu podataka, a podaci su fizički i logički neovisni. Fizička neovisnost podataka, odnosno razdvojenost logičke definicije baze podataka od njenog fizičkog prikaza ostvarena je pomoću datoteka na disku, dok je logička neovisnost podataka osigurana razdvajanjem globalne logičke definicije baze podataka od lokalne logičke definicije aplikacije.

  Neki popularni primjeri RDMS-a su: MySQL (besplatan, sustav otvorenog koda; tvrtka MySQL AB), Oracle DBMS, DB2 (tvrtka IBM), Microsoft SQL Server.

- **objektno-orijentirani DBMS** (eng. Object-Oriented Database Management System - OODBMS)
  DBMS temeljen na objektno-orijentiranom modelu podataka. Objekt reprezentira i podatke (koji se nazivaju varijablama) i funkcionalnosti (koje se nazivaju metodama). OODBMS-ovi nisu jako popularni u industriji zbog svoje složenosti.

  Primjeri OODBMS-a su db4o: (tvrtka Versant), Caché (tvrtka Intersystems), GemStone/S (tvrtka GemTalk Systems).

- **objektno-relacijski DBMS** (eng. Object-Relational Database Management System - ORDBMS)
  ORDBMS koristi relacijski model proširen s objektno orijentiranim konceptima. Stoga dijele karakteristike i RDBMS-ova i OODBMS-ova. Većina relacijskih DBMS-ova kao što su Oracle, DB2, Microsoft SQL Server i PostgreSQL imaju objektno-relacijske ekstenzije.

Model podataka

Model podataka je skup pravila koja određuju kako može izgledati logička struktura baze podataka [5]. Model čini osnovu za oblikovanje i implementiranje baze. Potrebno je koristiti odgovarajući model podataka, koji podržava DBMS koji smo odabrali, da bi se podaci pohranili u bazu na odgovarajući način. Dakle podaci u bazi trebaju biti logički organizirani na način koji podržava odabrani DBMS. Svaki DBMS podržava samo neki od modela.

Najpoznatiji modeli podataka su:

- **mrežni model** - baza je predočena usmjerenim grafom, čvorovi su tipovi zapisa, a lukovi definiraju veze među tipovima zapisa;
- **hijerarhijski model** - specijalni slučaj mrežnog modela; baza je predočena jednim stablom ili skupom stabala, čvorovi su tipovi zapisa, a hijerarhijski odnos "nadređeni-podređeni" izražava veze među tipovima zapisa;
- **relacijski model** - zasnovan na matematičkom pojmu relacije; podaci i veze među podacima prikazuju se pomoću tablica (detaljnije ćemo opisati u 2.2);
- **objektni model** - baza je skup trajno pohranjenih objekata koji se sastoje od svojih internih podataka i "metoda" za rukovanje s tim podacima, a svaki objekt pripada nekoj klasi (detaljnije ćemo opisati u 3.2).

U 60-tim i 70-tim godinama 20. stoljeća u upotrebi su bili hijerarhijski i mrežni model. 80-tih godina definira se relacijski model koji tada prevladava te je sve do danas najčešće korišteni model. Očekivani prijelaz na objektni model, zbog njegove kompleksnosti, za sada se nije dogodio. Ovaj rad osvrnuti će se prvo na relacijski model, zatim na objektni model, te konačno i na objektno-relacijski model što je i tema ovoga rada.

1.2 Jezici za rad s bazama podataka

Jezici za rad s bazama podataka služe za komunikaciju korisnika odnosno aplikacijskog programa i DBMS-a. Svaki DBMS dolazi s jednim ili više jezika:

- **Jezik za opis podataka** (eng. Data Definition Language- DDL)
  Koristi ga projektant ili administrator baze podataka (eng. Database Administrator - DBA) kako bi definirao podatke i veze među njima, i to na logičkom nivou.
- **Jezik za manipuliranje podacima** (eng. Data Manipulation Language - DML)
  Koristi se za kreiranje, čitanje, promjenu i brisanje podataka, odnosno za tzv. CRUD (Create, Read, Update, Delete) operacije.
- **Jezik za postavljanje upita** (eng. Query Language - QL)
  Jezik koji podsjeća na govorni jezik, naredbe su neproceduralne te služe korisniku za interaktivno pretraživanje baze.

DDL, DML i QL nisu programski jezici, nužni su nam za povezivanje s bazom, ali oni nisu dovoljni za razvoj aplikacije. Danas je ovakva podjela zastarjela već se koriste sveobuhvatni jezici koji objedinjuju sva tri navedena jezika. Opisat ćemo sada dva primjera takvih integriranih jezika.

SQL

Jezik SQL (eng. Structured Query Language) razvio je IBM u sklopu projekta "System R". SQL je uglavnom zasnovan na relacijskom računu, ali je matematička notacija zamijenjena ključnim riječima nalik na engleski govorni jezik [5]. SQL je deklarativni programski jezik, dakle kažemo što želimo, a ne kako želimo nešto napraviti (kao u proceduralnim jezicima).

OQL

OQL (eng. Object Query Language) je za objektne baze podataka ono što je SQL za relacijske baze podataka. To je zapravo upitni jezik objektnih baza podataka koji je napravljen po uzoru na SQL. Razvijen je od strane „Object Data Management Group – ODMG". Iako je fleksibilan jako je kompliciran stoga ga niti jedan proizvođač nije u potpunosti implementirao.

Razlike između OQL i SQL:

- OQL radi sa pokazivačima, te je time brži i efikasniji u nekim upitima od SQL-a,
- OQL podržava referenciranje na objekte unutar tablica (objekti mogu ugnježđivati druge objekte),
- OQL ne podržava sve ključne riječi iz SQL,
- OQL podržava matematičke izračune unutar OQL izraza.

1.3 Modeliranje podataka

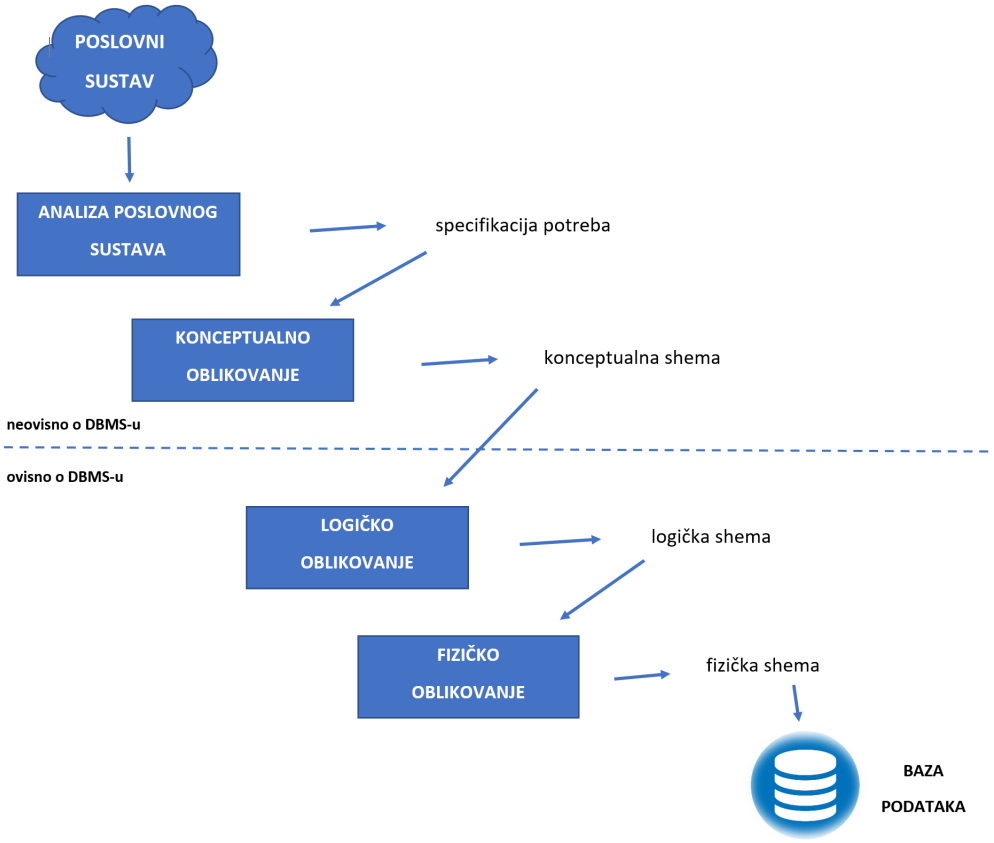
Modeliranje podataka shematski je prikazano na slici 1.1 gdje su prikazani svi koraci. Modeliranje počinje nakon analize poslovnog sustava, gdje je cilj prepoznati i razumjeti različite zahtjeve i podatkovne potrebe sustava. DBA u suradnji s poslovnim korisnikom razjašnjava i dogovara zahtjeve za bazu podataka te to dokumentiraju u specifikaciji. Zatim pomoću konceptualnog oblikovanja iz specifikacije razvijamo konceptualnu shemu, koja se u fazi logičkog oblikovanja razvija u logičku ili implementacijsku shemu, te se na kraju fizičkim oblikovanjem pretvara u fizičku shemu. Realizacijom fizičke sheme nastaje baza podataka.

Modeliranje podataka ima razne ciljeve:

- dokumentiranje informacijskih zahtjeva;

- izgradnja baze podataka koja ima maksimalnu konzistentnost podataka i integriranost, minimalnu redundanciju, odgovarajuću fleksibilnost i stabilnost i dobar pristup i iskoristivost;

- povećanje vrijednosti podatkovnih resursa.



Slika 1.1: Od poslovnog sustava do baze podataka

Konceptualno oblikovanje

Prvi korak u modeliranju podataka je konceptualno oblikovanje, a glavni cilj je stvoriti konceptualnu shemu. Kod relacijskih baza podataka obično je predstavljena korištenjem modela entiteta i veza (ER model) ili UML dijagramom klasa, u slučaju objektno-orijentiranih

baza podataka. Konceptualna shema služi kao sredstvo komunikacije između DBA i poslovnog korisnika. Obje strane nastoje formalizirati zahtjeve podataka tako da konceptualna shema bude jednostavna za razumijevanje poslovnom korisniku i dovoljno formalna za dizajnera baze podataka koji će ju koristiti u sljedećem koraku, gdje će biti dorađena u logičku shemu temeljenu na okruženju na koje će se implementirati. Konceptualna shema trebala bi biti i dovoljno fleksibilna tako da se novi ili promjenjivi zahtjevi podataka mogu lako dodati, odnosno izmijeniti. Konačno, mora biti i neovisna o implementaciji, odnosno o DBMS-u.

Ova konceptualna shema imat će i svoja ograničenja, koja treba jasno dokumentirati i pratiti tijekom razvoja aplikacije. U svakom koraku modeliranja bitno je pisati dokumentaciju radi kasnijeg lakšeg održavanja baze. Dokumentaciju u ovoj fazi uglavnom je grafička, oslanja se na dijagrame te opisuje konceptualnu shemu. Najčešći predlošci:

- **izvorni Chen-ov dijagram**
  Grafički elementi su pravokutnici (entiteti), rombovi (veze), „mjehurići“ (atributi) i spojnice među njima. Dijagram sadrži i imena atributa, entiteta i veza i oznake njihovih kardinalnosti.

- **reducirani Chenov dijagram**
  Nacrtani su pravokutnici (predstavljaju entitete), rombovi (predstavljaju veze) i spojnice među njima. Kao i u izvornoj verziji prikazana su imena entiteta i veza, te njihove kardinalnosti. Nedostatak informacije o atributima nadomještava se tekstom uz dijagram (detaljnije u poglavlju 2).

- **UML-ov dijagram klasa**
  UML je standardizirani grafički jezik koji koristimo u objektno-orijentiranim metodama. Dijagram klasa standardni je UML dijagram i on služi za prikaz klasa objekata i veza između tih klasa. Možemo ga koristiti za prikaz konceptualnog modela baze tako da entitet interpretiramo kao klasu koja ima atribute ali nema operacije. Kao i u Chenovim dijagramima entitete pikazujemo u obliku pravokutnika s imenom entiteta na vrhu dok se u sredini nalaze imena atributa. Veza se crta kao spojnica između pravokutnika, na sredinu upisujemo njeno ime, a na krajevima se nalaze oznake kardinalnosti. Dijagram sadrži svu potrebnu informaciju (detaljnije u poglavlju 3).

Logičko oblikovanje

Logičko oblikovanje slijedi nakon konceptualnog oblikovanja i predstavlja drugi korak modeliranja podataka. Cilj je kreirati shemu baze koja opisuje logičku strukturu baze podataka.

Nakon što su se sve strane složile oko konceptualne sheme, dizajner baze podataka može ju preslikati u logičku shemu. Ona se temelji na modelu podataka koji koristi traženi

DBMS. Iako se u ovoj fazi već zna koji će se tip DBMS-a (npr. RDBMS, OODBMS itd.) koristiti, o samom proizvodu (npr. Microsoft, IBM, Oracle) još nije odlučeno. Važno je napomenuti da logička shema ne rezultira razradom konačne fizičke strukture podataka.

Fizičko oblikovanje

Fizičko oblikovanje treća je faza modeliranja podataka. Cilj je stvoriti fizičku shemu odnosno detaljni opis fizičke građe baze podataka. U ovom završnom koraku, dizajner baze podataka preslikava logičku shemu u interni model podataka. DBA također može dati neke preporuke u vezi s performansama tijekom ovog koraka oblikovanja.

Fizička shema zapravo je skup naredbi napisanih u SQL-u ili nekom drugom jeziku koji odabrani DBMS razumije (u ovom koraku poznat je DBMS proizvod). Pokretanjem tih naredbi kreira se fizička baza podataka. Baza podataka se tada može popuniti podacima i spremna je za korištenje.

Poglavlje 2

Relacijske baze podataka

Relacijska baza podataka je baza podataka temeljena na relacijskom modelu podataka. Sastoji se od skupa povezanih tablica tj. relacija u obliku dvodimenzionalnih tablica. Tablica je osnovni objekt relacijske baze podataka i u njoj su pohranjeni podatci.

2.1 Modeliranje entiteta i veza

Prilikom oblikovanja baze, u oblikovanju konceptualne sheme najprije moramo otkriti entitete, veze i atribute. Za točno definiranje podataka i veza između njih koristimo tehniku **modeliranje entiteta i veza** (eng. Entity-Relationship Model - ER Model). U stvarnog svijetu prilično je teško direktno pogoditi relacijsku shemu. Kako bi oblikovali shemu za bazu podataka, usklađenu s pravilima relacijskog modela služimo se tom pomoćnom fazom, tzv. ER-model koji se dalje pretvara u relacijski model. Za daljnje shvaćanje ER modela potrebni su nam sljedeći pojmovi:

- **entiteti**: objekti ili događaji koji su nam od interesa,
- **veze**: odnosi među entitetima koji su nam od interesa,
- **atributi**: svojstva entiteta i veza koja su nam od interesa [5].

Entiteti i atributi

Entiteti su osnovni elementi za koje prikupljamo informacije za koje možemo odrediti neke karakteristike. Entitet je nešto što je u stanju postojati ili ne postojati, te se može identificirati. On može biti objekt ili biće (npr. auto, zaposlenik), odnosno događaj ili pojava (npr. kupovina auta, rukometna utakmica, praznik).

Nakon što smo odredili za koje entitete ćemo prikupljati informacije, potrebne su nam neke zajedničke karakteristike koje će ih opisivati. Entitet je opisan **atributima** (npr. atributi auta su: registracija, broj šasije, model, boja, . . .). Ako neki atribut istovremeno može imati više vrijednosti i/ili zahtijeva svoje atribute, tada ga smatramo novim entitetom. Tip entiteta određen je njegovim imenom i pripadnim atributima.

Jedna konceptualna shema ne smije imati različite tipove entiteta istog imena, a svi atributi jednog entiteta moraju imati različita imena. Dva entiteta smiju imati atribute s istim imenom (npr. entiteti POSLODAVAC i ZAPOSLENIK mogu imati atribut PREZIME).

Kandidat za ključ je atribut, ili skup atributa, čije vrijednosti jednoznačno određuju primjerak entiteta zadanog tipa. Dakle, ne mogu postojati dva različita primjerka entiteta istog tipa s istim vrijednostima kandidata za ključ [5]. Objasnimo bolje na primjeru: za tip entiteta AUTO, atribut REGISTRACIJA je kandidat za ključ. Ako jedan tip entiteta ima više od jednog kandidata za ključ, tada izabiremo jedan, takav ključ zovemo **primarnim ključem**.

Veze

Veze predstavljaju asocijacije između entiteta. Veze se uspostavljaju između dva ili više tipova entiteta (npr. veza ZAPOSLEN između tipova entiteta ZAPOSLENIK i TVRTKA) tako izražavamo činjenicu da se ti entiteti nalaze u nekom odnosu. Veze možemo podijeliti prema više kriterija:

- **kardinalnost**
  - 1:1 (jedan na jedan)
  - 1:N (jedan na više)
  - N:M (više na više)
- **obaveznost članstva**
  - obavezno
  - opcionalno
- **brojnost**
  - unarne
  - binarne
  - ternarne

Ukratko ćemo objasniti i dati primjere veza (1:1, 1:N, N:M):

Jedan na jedan (1:1): Jedan primjerak prvog tipa entiteta može biti u vezi s najviše jednim primjerkom drugog tipa entiteta, isto tako jedan primjerak drugog tipa može biti u vezi s najviše jednim primjerkom prvog tipa. Na primjer veza JE_RAVNATELJ između tipova entiteta NASTAVNIK i ŠKOLA [5].

Jedan na više (1 : N): Jedan primjerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primjeraka drugog tipa entiteta, no jedan primjerak drugog tipa može biti u vezi s najviše jednim primjerkog prvog tipa. Na primjer veza PREDAJE između tipova entiteta NASTAVNIK i PREDMET [5].

Više na više (M : N): Jedan primjerak prvog tipa entiteta može biti u vezi s 0, 1 ili više primjeraka drugog tipa entiteta, isto tako jedan primjerak drugog tipa može biti u vezi s 0, 1 ili više primjeraka prvog tipa [5]. Na primjer veza SLUŠA između tipova entiteta UČENIK i PREDMET.

Podjela prema obaveznosti članstva je zapravo davanje atributima, koji povezuju tablice, posebna ograničenja (not null, unique i sl.). Na primjer u asocijaciji „tvrtka-zaposlenik“ veze prema obaveznosti članstva izgledale bile bi:

- obavezno - jedna tvrtka mora imati barem jednog zaposlenika,
- opcionalno - jedan zaposlenik može biti zaposlen u tvrtki, ali ne mora ako ne želi.

Kažemo da tip entiteta ima obavezno članstvo ako svaki primjerak entiteta nekog tipa mora sudjelovati u toj vezi, u suprotnom tip entiteta ima opcionalno članstvo. Ima li veza obavezno ili opcionalno članstvo često je stvar dogovora odnosno odluke projektanta.

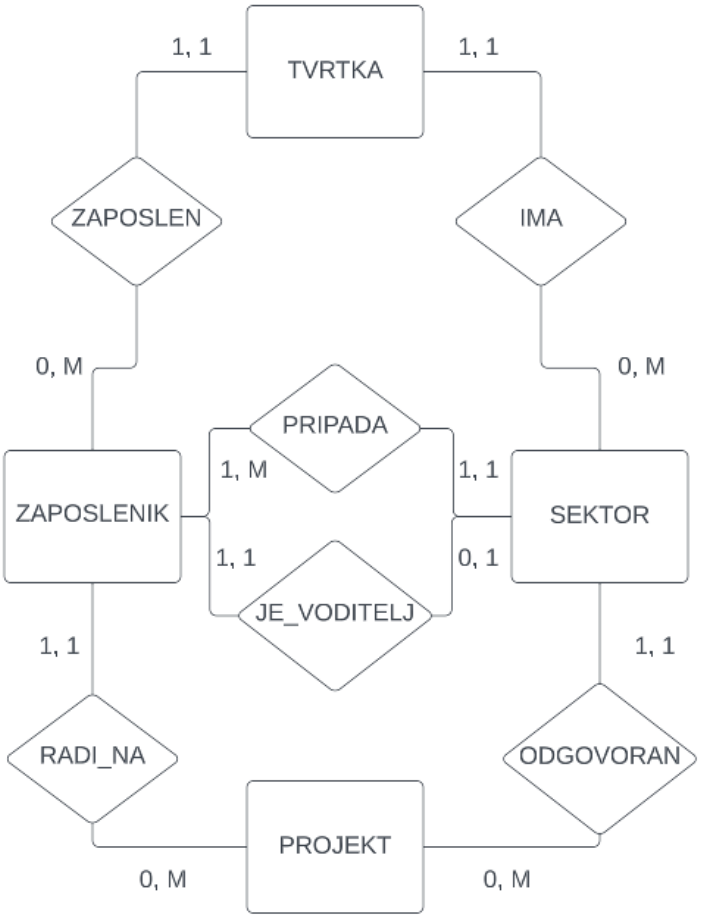
Podjela prema brojnosti govori koliko tablica se povezuje:

- unarna - entitet se povezuje sam sa sobom,
- binarna - veza između dva entiteta,
- ternarna - veza između tri entiteta.

I veza može imati svoje atribute, njih ne možemo pripisati ni jednom od tipova entiteta (npr. veza ZAPOSLEN može imati atribut DATUM_ZAPOSLENJA). Vrijednost DATUMA_ZAPOSLENJA pridružuje se konkretnom paru primjeraka ZAPOSLENIK i TVRTKA koji su povezani. Unutar iste sheme sve veze moraju imati različita imena.

Također, atributi koji pripadaju istoj vezi ne smiju imati ista imena.

Otkrivanje entiteta, veza i atributa može zapeti ako je specifikacija potreba nepotpuna ili nejasna. Tada projektant treba ponovo razgovarati s poslovnim korisnicima da bi popunili rupe i razjasnili sve nedoumice. Nakon što smo otkrili entitete, veze i atribute, za svaki od njih potrebno je utvrditi koji atributi ih opisuju. Za svaku vezu treba odrediti kardinalnost, obaveznosti članstva i brojnost, a za svaki entitet potrebno je odabrati primarni ključ.
Idući korak u konceptualnom oblikovanju je te elemente prikazati na dijagramu. To ćemo napraviti pomoću reduciranog Chen-ovog dijagrama:

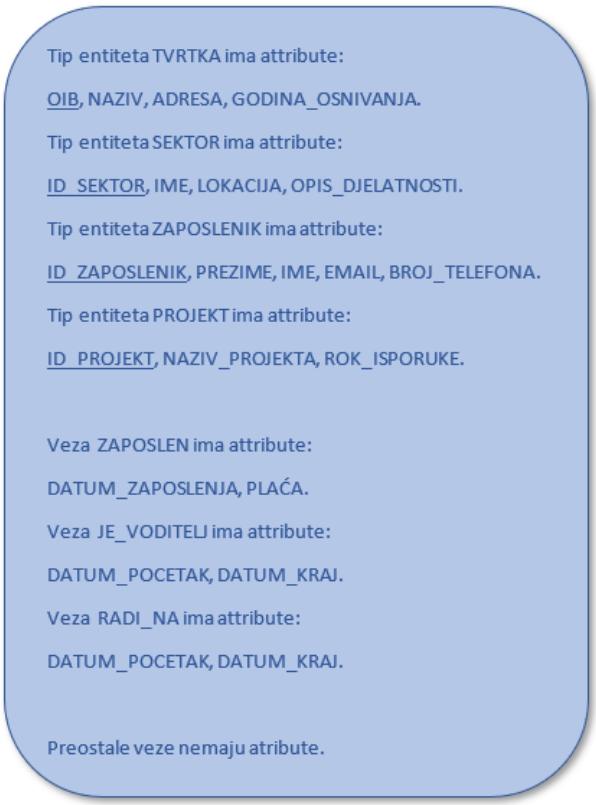


Slika 2.1: ER model baze podataka o tvrtci

Tipove entiteta crtamo kao pravokutnike, a veze kao rombove. Imena tipova entiteta ili veza upisuju se u odgovarajuće pravokutnike odnosno rombove. Romb je povezan spoj-

nicama s odgovarajućim pravokutnicima. Na krajevima spojnica upisujemo oznake kardinalnosti veze, tako da kardinalnost u smjeru od jednog do drugog tipa entiteta piše bliže drugom tipu.

Reduciranim Chen-ovim dijagramom opisani su samo entiteti i veze, on ne sadrži informacije o atributima. Razlog tome je želja da dijagram bude što jednostavniji i pregledniji. Dakle sam dijagrama ne opisuje u potpunosti konceptualnu shemu. Konceptualno oblikovanje završava sastavljanjem teksta koji prati dijagram. Taj tekst sadrži informacije o atributima, tj. one informacije koje smo izostavili iz dijagrama. U tom popratnom tekstu navodimo popis svih atributa za svaki tip entiteta i za svaku vezu koja ima atribute. Za svaki tip entiteta podvlačenjem označavamo primarni ključ.



Slika 2.2: Popratni tekst uz dijagram sa slike 2.1

Osim najnužnijih informacija, tekst uz dijagram može uključiti i dodatne sadržaje: objašnjenje projektanta zašto je uveo novi atribut koji nije u specifikaciji, zašto je odabrao primarni ključ na određeni način, zašto je pretpostavio da neka veza ima tu kardinalnost i

dr. U tekst možemo uključiti i rječnik podataka, gdje za svaki atribut objašnjavamo njegovo značenje i tip. Rječnik nije obavezan u fazi konceptualnog oblikovanja, već se ostavlja za logičko oblikovanje.

2.2 Relacijski model

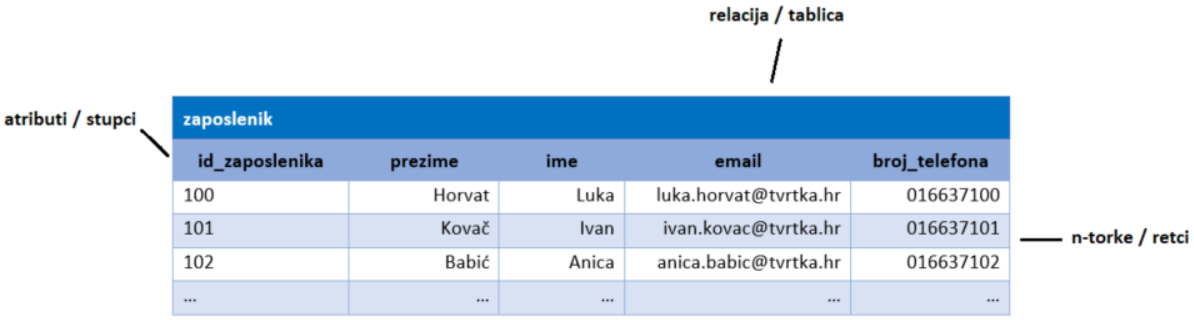
Relacijski model zasnovan je na matematičkom pojmu relacije. I podaci i veze među podacima prikazuju se "pravokutnim" tabelama [5]. Prvi ga je definirao E. Codd krajem 60-tih godina 20. stoljeća. To je jedan od logičkih modela baza podataka, koje smo već opisali u 1.1. Logičku shemu definiramo u drugoj fazi oblikovanja baze podataka. Logička shema opisuje logičku strukturu baze u skladu, kod relacijskih baza podataka, s pravilima relacijskog modela. Podaci u ovom modelu su „u relaciji" jedni s drugima. U ovoj fazi oblikovanja entiteti i veze pretvaraju se u tablice. Dobiveni relacijski model obično još nije u svom konačnom obliku. On se podvrgava postupku normalizacije koji će biti opisan malo kasnije u ovom poglavlju. Neki od osnovnih pojmova vezanih uz relacijski model (slika 2.3):

- **atribut** (stupac; eng. attribute) - karakteristika entiteta ili veze,
- **n-torka** (redak; eng. tuple) - sadrži vrijednosti koje odgovaraju atributima iz relacije,
- **relacija** (tablica; eng. relation) - predstavlja konačan skup n-torki.

Terminologija potječe iz matematike. Jedan stupac relacije sadrži vrijednost jednog atributa (za entitet ili vezu), zato stupac poistovjećujemo s atributom i obratno. Vrijednosti jednog atributa su podaci iste vrste. Definiran je tip ili skup dozvoljenih vrijednosti za atribut, koji se zove i domena atributa. Jedan redak relacije opisuje primjerak entiteta ili bilježi vezu između dva ili više primjeraka. Redak nazivamo n-torka. U jednoj relaciji ne smiju postojati jednake n- torke [5]. Relaciju tumačimo kao skup n-torki.

Tablice se povezuju preko stupca koji predstavlja jedinstveni podatak unutar tablice i nalazi se u obje tablice. U glavnoj tablici naziva se **primarni ključ** (eng. Primary Key - PK), dok se u podređenoj tablici taj stupac naziva **strani ključ** (eng. Foreign Key - FK). U relacijskom modelu primarni ključ je atribut ili skup atributa koji jednoznačno definiraju redak tablice. Primarni ključ mora zadovoljavati sljedeće:

- primarni ključ jedinstveno identificira svaki redak tablice,
- uvjet minimalnosti (ako izbacimo neki atribut, narušavamo gornje svojstvo),
- primarni ključ ne smije imati NULL vrijednost (svaka tablica mora imati upisanu vrijednost).



Slika 2.3: Pojmovi vezani uz relacijski model

Osnovni ciljevi relacijskog modela su:

- omogućiti nezavisnost podataka,
- dati teorijske temelje za postupanje s podacima i za rješavanje redundancije podataka,
- omogućiti razvoj jezika za obradu podataka,
- dati bogat model podataka za opis i obradu jednostavnih i kompleksnih podataka.

Mrežni i hijerarhijski model nisu ostvarili prva dva cilja, korištenjem jednostavnog tabličnog prikaza podataka i razrađenom teorijom normalizacije podataka, ostvaruje ih tek relacijski model. Relacijski model uporabom operacija relacijske algebre ostvario je i treći cilj. Za sada četvrti cilj nije ostvaren na odgovarajući način.

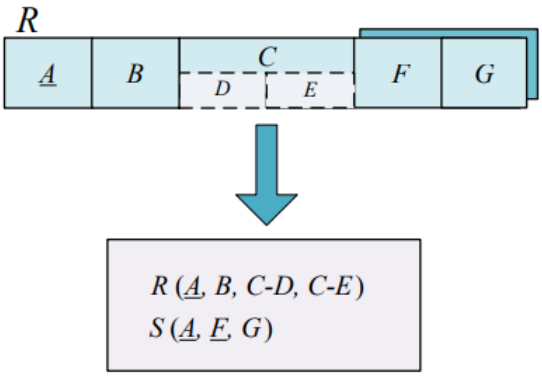
Normalizacija relacijske sheme

Normalizacija ili daljnje dotjerivanje dio je druge faze modeliranja podataka, odnosno dio logičkog oblikovanja. Normalizacija je potrebna jer polazni relacijski model može sadržavati nepravilnosti. Njih je potrebno ukoniti prije fizičkog oblikovanja. Zahvaljujući normalizaciji, postupak modeliranja podataka postaje samokorigirajući proces, greške napravljene u ranijim fazama uspješno ispravljamo u kasnijim fazama. Izvor greške obično se nalazi već u oblikovanju konceptualne sheme, dakle u pogrešnom prepoznavanju entiteta, veza i atributa.

Pravila normalizacije zapravo su formalni opis intuitivno prihvatljivih principa o pravilnom i prirodnom oblikovanju entiteta, veza i atributa. Provedemo li oblikovanje na

ispravan način, odmah ćemo dobiti relacije u visokim normalnim formama i neće biti potrebe za normalizacijom. Teorija normalizacije zasnovana je na pojmu **normalnih formi**. Svaka normalna forma predstavlja „zahtjev na kvalitetu“ relacije. Što je normalna forma viša, zahtjevi su stroži. Postoji više normalnih forma, a mi ćemo navesti samo one koje se najčešće koriste u praksi.

Prva normalna forma - 1NF
Vrijednost atributa unutar relacije mora biti jednostruka i jednostavna, odnosno atribut ne smije sadržavati skupove podataka i relacija ne smije sadržavati ponavljajuće grupe stupaca. 1NF zapravo ne predstavlja nikakav posebni zahtjev na relaciju, to svojstvo već je ugrađeno u sam relacijski model. U relacijskoj bazi podataka ne može postojati relacija koja ne bi bila u 1NF. Pojam 1NF izmišljen je zbog drugih modela podataka gdje podaci ne moraju biti normalizirani [5].



Slika 2.4: Shematski prikaz prevođenja u 1NF [6]

Na slici 2.4 vidi se pretvorba jednog podzapisa i jedne ponavljajuće skupine. Ako imamo više podzapisa ili više ponavljajućih skupina, tada se zahvati prikazani na slici 2.4 moraju ponavljati.

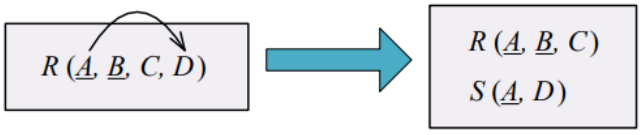
Pojasnimo pojam **funkcionalne ovisnosti**. Za zadanu relaciju R, atribut B od R je funkcionalno ovisan o atributu A od R (oznaka: $A \rightarrow B$) ako vrijednost od A jednoznačno određuje vrijednost od B. Dakle ako u isto vrijeme postoje u R dvije n-torke s jednakom vrijednošću A, tada te n-torke moraju imati jednaku vrijednost B. Analogna definicija primjenjuje se i za slučaj kad su A i B složeni atributi [5].

Druga normalna forma - 2NF

Relacija je u 2NF ako je u 1NF i ako je svaki neprimarni atribut potpuno funkcionalno ovisan o PK-u [5]. Neključni atributi moraju potpuno funkcijski ovisiti o ključu (odnosno, relacija ne može sadržavati atribute koji ovise samo o dijelu ključa). Shematski prikaz prevođenja u 2NF prikazan je na slici 2.5. 2NF odnosi se na relacije s kompozitnim ključevima, na primjer:
$R1$: (OIB, ak.godina, šifra predmeta) → (naziv predmeta, ocjena) treba postati:

$$R1 : (\text{OIB, ak.godina, šifra predmeta}) \rightarrow (\text{ocjena})$$

$$R2 : (\text{šifra predmeta}) \rightarrow (\text{naziv predmeta})$$



Slika 2.5: Shematski prikaz prevođenja u 2NF [6]

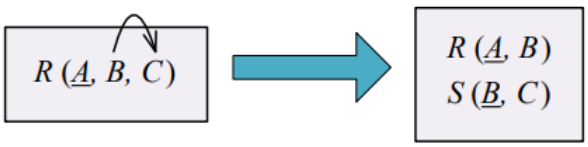
Treca normalna forma - 3NF

Relacija je u 3NF ako je u 2NF i ako ne sadrži tranzitivne ovisnosti. Shematski prikaz prevođenja u 3NF prikazan je na slici 2.6. Na primjer, poštanski broj mjesta rođenja ovisi o OIB, ali naziv mjesta ovisi o poštanskom broju, pa se ta dva atributa sele u zasebnu tablicu.

$R1$: (OIB) → (ime, prezime, poštanski broj mjesta, naziv mjesta) treba postati:

$$R1 : (\text{OIB}) \rightarrow (\text{ime, prezime, poštanski broj mjesta})$$

$$R2 : (\text{poštanski broj mjesta}) \rightarrow (\text{naziv mjesta})$$



Slika 2.6: Shematski prikaz prevođenja u 3NF [6]

Poglavlje 3

Objektno-orijentirane baze podataka

Objektno-orijentirana baza podataka (eng. Object Oriented Database - OODB) je baza podataka u kojoj su informacije predstavljene u obliku objekata. Nastala je 80-ih godina kao odgovor na nedostatke relacijske baze podataka. Objektno-orijentirani sustavi za upravljanje bazom (OODBMS) kombiniraju mogućnosti baze s mogućnostima objektno-orijentiranog programiranja. Budući da je baza podataka integrirana s programskim jezikom, programer može lakše upravljati s bazom jer programski jezik i OODBMS koriste isti model reprezentacije.

Kako bi bolje razumijeli objektno-orijentirane baze podataka, prvo ćemo objasniti značajke objektno-orijentiranog programiranja:

- klasa (razred),
- objekt,
- identitet objekta,
- enkapsulacija,
- hijerarhija klasa i nasljeđivanje,
- polimorfizam [3].

Klasa (razred)

Koristeći klase objektno-orijentirani jezici opisuju objekte. Nakon što definiramo klasu, iz nje možemo kreirati objekte. Klasa se sastoji od svojstava i metoda. Pomoću svojstava definiramo klasu za koju želimo instancirati objekte, a metode su funkcije koje su definirane unutar klase. Klasa je u objektno-orijentiranom modelu ono što je tablica u relacijskom modelu.

Objekt

Objekt je apstraktni koncept koji općenito predstavlja entitet od interesa. Svaki objekt je instanca klase koja sadrži opis svih karakteristika objekta. Sastoji se od atributa (određuju stanje objekta) i metoda (određuju ponašanje objekta). Atributi su podaci koji određuju karakteristike objekta, oni mogu biti jednostavni tipovi podataka (integer, string) ili reference na složenije tipove podataka. Metoda sadrži naziv metode te imena i vrste parametara metode. Objekti koji imaju isto stanje (opisano atributima) i ponašanje opisani su istom klasom [9]. Stanje objekta opisuje unutarnju strukturu objekta, tj. karakteristike objekta. Ponašanje objekta je skup metoda koje se koriste za stvaranje, pristup i manipuliranje objektom.

Objasnimo primjerom, promotrimo osobu kao objekt, stanje objekta može sadržavati informacije kao što su OIB, ime, prezime i adresa. Objekt osobe, na primjer, može imati metode za stvaranje i brisanje objekta, te metode za izmjenu stanja objekta. Objekt također može imati metode za povezivanje objekta s drugim objektima, kao što je upisivanje osobe na tečaj ili dodjeljivanje osobi instruktora tečaja.

Objekti mogu biti jednostavni i kompleksni. Jednostavan objekt može biti temelj za kreiranje kompleksnijih objekata, on je opisan jednostavnim atributima. Objekt čiji atributi mogu biti i sami objekti nazivamo kompleksnim objektom.

Identitet objekta

Svaki objekt ima jedinstveni i nepromjenjivi identifikator objekta (eng. Object Identifier - OID) koji se koristi za jedinstvenu identifikaciju objekta. Dva objekta ne mogu imati isti identifikator, a svaki objekt ima samo jedan identifikator. Jednom kada je objekt kreiran OID ostaje isti sve dok objekt postoji. OID ne ovisi o stanju objekta, stanje objekta je promjenjivo što znači da se vrijednosti svojstava objekta mogu mijenjati. Vrijednosti svojstava se stoga ne koriste kao jedinstveni identifikatori za objekt u bazi podataka.

Enkapsulacija

Enkapsulacija ili učahurivanje odnosi se na mogućnost stvaranja klase kao apstraktnog tipa podatka, koji ima sučelje i implementaciju. Sučelje se sastoji od popisa metoda i definira ponašanje apstraktnog tipa podatka na konceptualnoj razini dok implementacija definira realizaciju tog ponašanja na razini programskog jezika. Implementacija se temelji na atributima objekta i definicijama metoda. Glavna ideja je učiniti atribute privatnima za klasu, tako da im se ne može izravno pristupiti izvan klase. Time programer ima kontrolu nad time kako i kada se može pristupiti varijablama.

Enkapsulacija nudi nekoliko prednosti koje ju čine standardnom preporučenom praksom. Sučelje i implementacije strogo su odvojene, stoga se korištenjem koncepta enkap-

sulacije, implementacija klase može promijeniti bez utjecanja na sučelje koje klasa pruža ostatku aplikacije. OODBMS podržavaju enkapsulaciju specificiranjem tipova definiranih od strane korisnika.

Hijerarhija klasa i nasljeđivanje

Svaka novo kreirana klasa može biti roditelj ili dijete nekih drugih klasa (eng. parent, child). Klasa koja je roditelj neke druge klase naziva se nadklasa, dok klasa koja je dijete neke klase naziva se podklasa. Na primjer, učenik je osoba, stoga klasa UČENIK može biti podklasa klase OSOBA. Srednjoškolac i osnovnoškolac su učenici, pa možemo kreirati nove klase SREDNJOŠKOLAC i OSNOVNOŠKOLAC koje će biti podklase klase UČENIK kojoj je klasa OSOBA nadklasa. Svaka podklasa nasljeđuje atribute i metode od nadklase. Bitno je napomenuti da kada neka klasa nasljeđuje neku drugu klasu, tada ona ne nasljeđuje sve atribute i metode nego samo one koji su dozvoljeni.

Polimorfizam

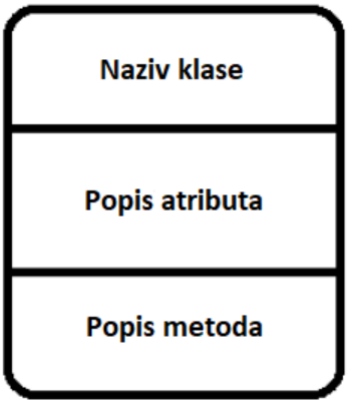
Polimorfizam je sposobnost objekata da različito reagiraju na istu metodu. To je ključni koncept u objektno-orijentiranom programiranju i usko je povezan s nasljeđivanjem. Ista metoda, definirana za nadklasu, može se drukčije implementirati u dvjema različitim podklasama iste nadklase. Npr. metoda kojom se računa površina nekog geometrijskog lika drukčije je implementirana za krugove odnosno za kvadrate.

3.1 UML-ov dijagram klasa

UML (Unified Modeling Language) je standardizirani grafički jezik koji se rabi u objektno-orijentiranim metodama, a dijagram klasa je standardni UML dijagram koji služi za prikaz klasa objekata i veza između tih klasa. Već smo spomenuli da ga najčešće koristimo kako bi grafički prikazali konceptualnu shemu baze, pa ćemo sada detaljnije opisati i kako. UML-ov dijagram klasa sastoji se od klasa, atributa, metoda i asocijacija. Također podržava specijalizaciju/generalizaciju i agregaciju.

Za prikaz konceptualne sheme baze upotrebljava se tako da se objekt (vrlo sličan entitetu u ER modelu) interpretira kao klasa koja ima atribute i metode. Dok u relacijskom modelu entitet sadrži atribute, ovdje objekt osim atributa sadrži i metode. Klasa se crta kao pravokutnik s imenom klase na vrhu, u sredini se nalaze imena svih atributa, a na dnu imenima metoda (vidi sliku 3.1). Asocijacija se crta kao spojnica između pravokutnika, u sredini se nalazi njeno ime, a na krajevima oznake kardinalnosti. Dijagram može sadržavati

i asocijacijske klase. Riječ je o kvazi-klasama koje sadrže atribute vezane uz neku asocijaciju. Prikazuju se kao i klase te su isprekidanom linijom povezane za asocijaciju čije atribute sadrže. Dijagram sadržava sve potrebne informacije te nema potrebe za dodatnim tekstom kao što je bio slučaj kod reduciranog Chenovog dijagrama.



Slika 3.1: Izgled klase

Atributi s jedinstvenim vrijednostima (slično primarnim ključevima u ER modelu) izravno nisu podržani u UML-u. Razlog je taj jer se pretpostavlja da se UML dijagram klasa implementira korištenjem OODBMS-a, time je svakom stvorenom objektu dodijeljen OID (već smo ga objasnili u 3).

Interakcije između klasa nazivaju se veze u ER modelu i asocijacije u UML dijagramu. Asocijacija je označena kao spojnica koja povezuje klase uključene u asocijaciju. Brojevi naznačeni na rubovima koji povezuju odnos s njegovim pridruženim klasama pokazuju koliko objekata iz svake klase potencijalno sudjeluje u odnosu.

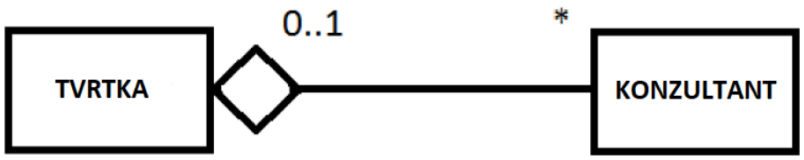
Asocijacija

Asocijacija je strukturna veza kojom se određuje povezanost instance jedne klase s instancama druge ili iste klase. Najčešće vrste asocijacije u UML-u su:

- agregacija (slaba agregacija),
- kompozicija (jaka agregacija).

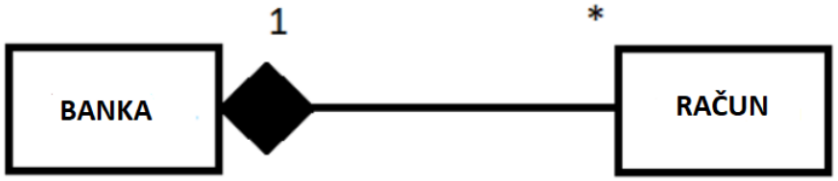
Kod agregacije, objekt može istovremeno pripadati raznim složenim objektima, a može se pojaviti i samostalno, bez pripadnosti nekoj cjelini. Agregacija stoga predstavlja prilično

labavu vezu između klasa. Na primjer, imamo agregaciju između TVRTKA i KONZULTANT. Konzultant može raditi za više tvrtki i kada se tvrtka obriše, svi konzultanti koji su radili za nju neće se obrisati. Agregacija se prikazuje praznim rombom, što možemo vidjeti i na slici 3.2.



Slika 3.2: Agregacija

Kod kompozicije, svaki objekt u jednom trenutku može biti član samo jedne cjeline. Kompozicija predstavlja čvrstu vezu između klasa. Ako se obriše cjelina, s njom se brišu i svi njezini dijelovi, dok se dio cjeline može obrisati, a cjelina će ostati. Na primjer, imamo kompoziciju između BANKA i RAČUN. Račun je usko povezan samo s jednom bankom. Kada se banka obriše, nestaju i svi njezini računi, dok se jedan račun unutar banke može obrisati bez da se u banci nešto promijeni. Na slici 3.3 prikazana je kompozicija i vidimo da se ona prikazuje pomoću ispunjenog romba.



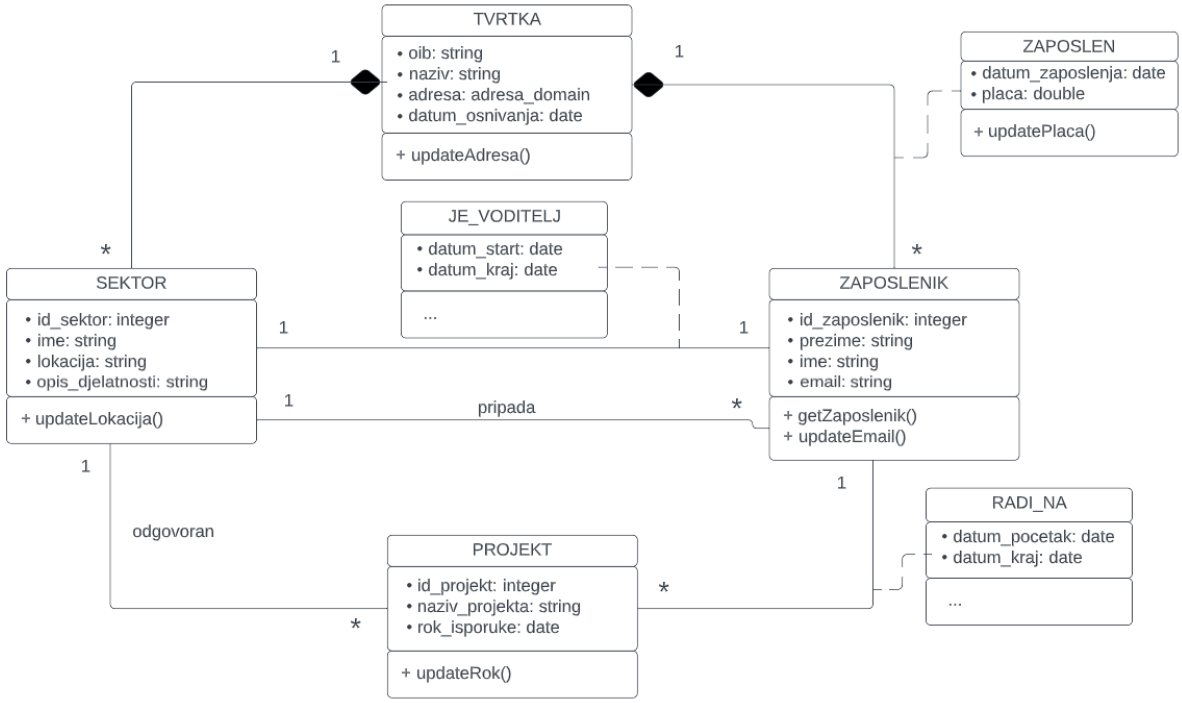
Slika 3.3: Kompozicija

Generalizacija/specijalizacija

Generalizacija je postupak prepoznavanja nadklase neke druge klase. Dok je specijalizacija obrnuti postupak. Prisjetimo se hijerarhije klasa, klasa može imati nadklasu i podklasu. Nadklase su generalizacija podklasa, a podklase su specijalizacija nadklasa. Specijalizaciju u UML dijagramu označavamo praznim trokutom. UML također podržava višestruko

nasljeđivanje gdje podklasa može naslijediti atribute, metode i asocijacije iz više nadklasa.

Slika 3.4 prikazuje UML-ov dijagram klasa na primjeru tvrtke.



Slika 3.4: Primjer UML-ovog dijagrama klasa

3.2 Objektni model

Objektni model inspiriran je objektno-orijentiranim programskim jezicima. Baza je skup objekata koji se sadrže interne podatke i operacije (metode) za rukovanje s tim podacima. Svaki objekt pripada nekoj klasi. Između klasa se uspostavljaju veze nasljeđivanja, agregacije, odnosno međusobnog korištenja operacija što smo objasnili u prošlom poglavlju [5].

Za razliku od relacijskog modela, koji je teoretski uveden prije implementacije komercijalnih proizvoda relacijske baze podataka, objektni model razvio se dodavanjem postojanosti objektima u objektno-orijentiranim programskim jezicima. Nekoliko proizvoda objektno-orijentirane baze podataka pojavilo se prije razvoja ODMG standarda. Prvi prototip OODBMS pojavio se 80-ih godina.

ODMG standard

ODMG (Object Data Management Group) standard utemeljen je 1991.godine kako bi definirali skup pravila koji omogućuju korisnicima razvoj aplikacija koje koriste objektne baze podataka, slično ulozi koju SQL ispunjava prema RDBMS-ovima. Njegov naziv promijenjen je u iz Object Database Management Group u Object Data Management Group 1998.godine kako bi obuhvaćao i OODBMS i standarde objektno-relacijskog mapiranja. ODMG standard sastoji se od:

- objektnog modela,
- jezika za specificiranje objekata,
- objektnog upitnog jezika,
- veze na programske jezike,
- jezika za specificiranje objekata koji je ovisan o odabranom programskom jeziku,
- pružanja aplikacijskog programskog sučelja za preslikavanje tipova podataka.

Kod objektnog modela treba pripaziti da se modeli objekata iz stvarnog svijeta ne rastavljaju na dijelove već da čine jednu smislenu cjelinu. Dizajner baze podataka koristi jezik za specificiranje objekata (eng. Object Definition Language - ODL) za određivanje tipova objekata u aplikaciji prema ODMG objektnom modelu, koji je neovisan o određenoj bazi podataka ili prodavaču DBMS-a. Nakon što je shema definirana u ODL-u, objektni upitni jezik (eng. Object Query Language - OQL) je jezik za postavljanje upita u objektnoj bazi podataka. OQL koristi se za opisivanje upita preko ODMG objektnog modela i on je neovisan o bazi podataka.

Jedna od najvažnijih karakteristika ODMG standarda je da se za dohvaćanje informacija iz OODBMS-a koristi izričito OQL, ali se stvaranje, izmjena i brisanje tih podataka vrši izravno putem glavnog programskog jezika, koji je uvijek objektno-orijentirani programski jezik.

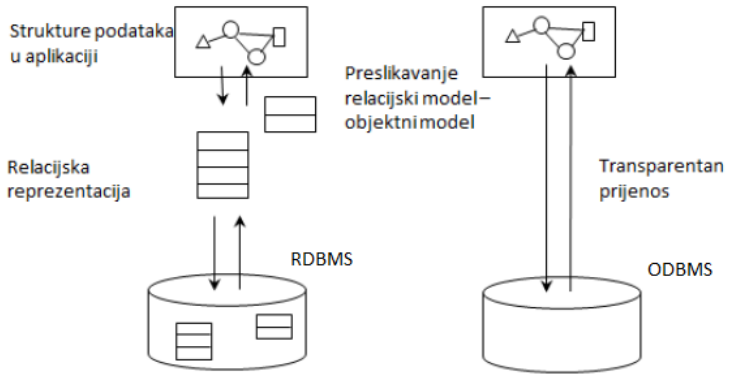
Poglavlje 4

Usporedba relacijskih i objektno-orijentiranih baza podataka

Unatoč popularnosti relacijskog modela u industriji, pojava aplikacija koje koriste složenije ili nove tipove podataka (npr. multimedija, geoprostorni informacijski sustavi (GIS), slike itd.) razotkrila je njegove nedostatke. Prvi od nedostataka odnosi se na normalizaciju. Zbog normalizacije, relacije se mogu povezati samo korištenjem relacija primarni-strani ključ. Ovo stavlja veliki teret na performanse aplikacija jer je potrebno puno operacija za defragmentaciju podataka prije nego što se mogu uspješno koristiti. Koncepti modeliranja kao što su specijalizacija, generalizacija i agregacije ne mogu se izravno podržati što očito negativno utječe na učinkovitost koda i na samo održavanje.

Još jedan nedostatak relacijskih baza podataka je taj da se kompleksni tipovi podataka ne mogu jednostavno spremati u bazu jer to zahtijeva razbijanje kompleksnih informacija u jednostavne podatke koji se spremaju u nekoliko relacija. To zahtjeva puno vremena i u većini slučajeva puno koda. Taj problem prikazan je na slici 4.1 i nazivamo ga objektno relacijska neusklađenost (eng. object relational impedance mismatch).

Za aplikacije koje koriste složenije tipove podataka prikladnije su objektne baze podataka jer tada koristimo jedan podatkovni model, odnosno objekti u bazi jednaki su objektima u aplikaciji, stoga nema objektno relacijske neusklađenosti. Prednost objektnih baza je i u tome da omogućuju bolju povezanost aplikacije i baze podataka jer se za pristup podacima i bazi za aplikaciju koristi isti programski jezik. Kako je tipični OODBMS svojim programskim sučeljem vezan za isključivo jedan programski jezik, o njemu je i u potpunosti ovisan. OODBMS brže i bolje upravljaju sa složenim objektima i vezama stoga brže pristupaju podacima. Razlog tome je što objekti ne moraju dohvaćati podatke iz više tablica svaki put kada ih želimo koristiti. Za razliku od relacijskih, objektne baze podataka podržavaju hijerarhiju, klase i nasljeđivanje. Kod njih nema potrebe za primarnim ključevima, identifikacija objekata skrivena je od korisnika (objektima se automatski ge-



Slika 4.1: Objektno relacijska neusklađenost

nerira i dodjeljuje OID). Pristup temeljen na identitetu omogućuje poboljšanu izvedbu pri izvođenju složenih upita koji uključuju više međusobno povezanih objekata, izbjegavajući puno spajanja koja smanjuju performanse.

Nakon što smo naveli nekoliko prednosti objektnih baza, dotaknuti ćemo se i njihovih nedostataka. Nedostatak logičke neovisnosti podataka jedna je od velikih mana objektnih baza podataka, odnosno izmjene na bazi podataka zahtijevaju izmjene u aplikaciji i obrnuto, što kod relacijskih baza nije slučaj. Njihov najveći nedostatak ipak je nedostatak dogovorenih standarda, tj. postojeći standard (ODMG) nije implementiran potpuno, za razliku od relacijskih baza podataka koje imaju potpuno implementiran i široko rasprostranjen standard. Zbog toga objektne baze nisu kompatibilne s velikim brojem alata i mogućnostima koje koristimo u SQL-u. Također one nisu u mogućnosti međusobno razmjenjivati podatke između sebe i relacijske baze, koje su i dalje više zastupljene. Sigurnost objektne baze podataka dovodi se u pitanje jer većina OODBMS-ova ne podržava autorizacije. Na kraju, možda i najvažnije, objektne baze ne leže na čvrstim matematičkim temeljima kao relacijake baze podataka.

Zaključujemo da je uspjeh OODBMS-ova ograničen na specijalizirane aplikacije koje koriste složene, ugniježđene strukture podataka i gdje se metoda rada koja se temelji na identitetu, umjesto na vrijednosti, isplati. Međutim, pokazalo se da je široku upotrebu i izvedbu RDBMS-ova teško zamijeniti: formulacija (ad-hoc) upita i postupci optimizacije OODBMS-a često su inferiorni u odnosu na relacijske baze, koje koriste SQL kao svoj primarni jezik u kombinaciji sa snažnim optimizatorom upita. U usporedbi s RDBMS-ovima, OODBMS-ovi nisu tako dobro razvijeni u smislu robusnosti, sigurnosti, skalabilnosti i tolerancije na pogreške.

Poglavlje 5

Objektno-relacijske baze podataka

Objektno-relacijske baze podataka (eng. Object-relational Database - ORDB) pokušaj su iskorištavanja najboljeg iz relacijskh i objektno-orijentiranih baza podataka. Ideja je da se u baze podataka uvedu objektno-orijentirani koncepti preuzeti iz objektno-orijentiranih programskih jezika poput C++, *C#*, PHP, Java i sl. Tako su ORDB nastale kao kompromis između relacijskih i objektnih baza jer sadrže karakteristike oba modela.

5.1 Objektno-relacijski model podataka

Objektno-relacijski model podataka vrlo je sličan relacijskom modelu, ali sadrži karakteristike i objektnog modela. Točnije, to je relacijski model koji je proširen objektnom-orijentiranim konceptima koji omogućuju rad s novim tipovima podataka. Neke od karakteristika objektno-relacijskog modela su:

- temeljen je na relacijskom modelu podataka,
- sačuvane su relacijske karakteristike (deklarativan pristup podacima),
- zadržana kompatibilnost s postojećim relacijskim jezicima,
- dozvoljava da atributi u n-torkama imaju složene vrijednosti, uključujući i ugnježdene relacije (narušena 1NF),
- omogućuje nasljeđivanje tipova i tablica,
- znatno uvećane mogućnosti modeliranja podataka čime je proširen opseg primjene.

Bitno je istaknuti da ne postoji jedinstveni objektno-relacijski model, odnosno modeli se razlikuju u količini objektnih proširenja koja uključuju. Objektno-relacijski koncepti nisu

potpuno implementirani niti u jednom DBMS-u, ali neka proširenja su ugrađena u sve glavne komercijalne DBMS-ove. Neka od navedenih proširenja relacijskog modela su:

- apstraktni tipovi podataka,
- identifikatori objekta i reference,
- učahurivanje,
- tipizirane tablice,
- nasljeđivanje tablica i tipova,
- ugnježdene relacije (složeni atributi, kolekcije).

sifOsoba	ime	prezime	adresa			slika	zivotopis
			ulica	mjesto			
				postBr	nazMjesto		
11001	Hrvoje	Novak	Ilica 25	10000	Zagreb		Rođen je .....
78936	Ana	Kolar	Marmontova 18	21000	Split		

Slika 5.1: Primjer ugnježdene relacije - relacija osoba nije u 1NF

Ova proširenja postala su dio SQL standarda od 1999. godine, ta verzija naziva se SQL:1999. Također su uključena i u verziji SQL:2003. Ove verzije SQL standarda predstavljaju stalni pokušaj standardizacije proširenja relacijskog modela. Neka proširenja smo već pojasnili kada smo objašnjavali koncepte objektno-orijentiranih sustava. Ona proširenja koja nismo još spomenuli sada ćemo opisati detaljnije.

5.2 SQL standard: objektno-relacijska proširenja

Tipovi podataka dijele se na:

- **unaprijed definirane tipove** (eng. Predefined Types)
  - atomaran tip - njihova vrijednost nije izgrađena od vrijednosti drugih podatkovnih tipova, primjeri: boolean, character, integer, float, ...

- **izgrađene tipove** (eng. Constructed Types)
  - izgrađeni atomarni tipovi (eng. Constructed Atomic Types)
  - izgrađeni kompozitni tipovi (eng. Constructed Composite Types)
- **korisnički definirane tipove** (eng. User-defined Types - UDT)
  - distinct type
  - strukturirani tip (eng. Structured Type)

Izgrađeni tipovi

Kao što smo već naveli izgrađeni tipovi dijele se na atomarne i kompozitne. Naziv tipa definiran je standardom i specificira se pomoću konstruktora tipa (REF, ARRAY, ROW).

Primjer atomarnog tipa je referenca. Referenca (eng. reference type - REF type) je tip čija vrijednost pokazuje na lokaciju na kojoj je pohranjena vrijednost referenciranog tipa [1].

Kod kompozitnih tipova svaka vrijednost je složena od jedne ili više vrijednosti koje pripadaju istim (kolekcija) ili mogu pripadati različitim podatkovnim tipovima (ROW) [1]. Izgrađeni kompozitni tipovi mogu se koristiti za definiranje složenih atributa relacije. Primjer je dan na slici 5.2, gdje je prikazano kreiranje tablice koja sadrži ugniježđenu relaciju sa slike 5.1.

```
CREATE TABLE osoba (
        sifOsoba  INTEGER,
        ime       VARCHAR(25),
        prezime   VARCHAR(25),
        adresa    ROW ( ulica VARCHAR(50),
                        mjesto ROW ( postBr      INTEGER,
                                     nazMjesto   VARCHAR(40))
                      )
);
```

Slika 5.2: Kreiranje tablice pomoću ROW tipa

ROW tip je jedan od izgrađenih kompozitnih tipova, to je sekvenca od jednog ili više elemenata, a element ROW tipa definiran je parom *(ime_elementa, tip_podatka)*. Vrijednost ROW tipa sadrži po jednu vrijednost za svaki element tog ROW tipa i ta vrijednost mora odgovarati definiranom podatkovnom tipu tog elementa [1].

Spomenuli smo **kolekciju**, dakle one sadržavaju vrijednosti koje pripadaju istim podatkovnim tipovima za razliku od ROW tipa. Kolekciju možemo podijeliti na:

- polje (ARRAY) - jednodimenzionalno polje s maksimalnim brojem elemenata,
- multiskup (MULTISET) - neuređena kolekcija koja dozvoljava duplikate,
- skup (SET) - neuređena kolekcija koja ne dozvoljava duplikate,
- lista (LIST) - uređena kolekcija koja dozvoljava duplikate [1].

Korisnički definirani tipovi

Korisnički definirani tipovi (eng. User-defined Types – UDT) ili apstraktni tipovi podataka (eng. Abstract Data Types), kako ih neki nazivaju, definirani su i imenovani od strane korisnika. UDT dijele se na distinct i strukturirani tip.

Distinct tip temelji se na unaprijed definiranom tipu koji je jedan od atomarnih tipova i naziva se izvorni tip (eng. source type). Tom postojećem atomarnom tipu dodjeljujemo posebno značenje. Time sprječavamo miješanje logički nekompatibilnih vrijednosti (npr. usporedbu godina s težinom). Njegove se vrijednosti ne mogu izravno miješati u operacijama s tim izvornim tipom ili s drugim distinct tipovima koji se temelje na istom izvornom tipu. Nad DISTINCT tipovima moguće je definirati metode, ali nije moguća hijerarhija jer ne podržava nasljeđivanja tipova [7].

Strukturirani tip omogućava definiranje novog tipa podataka koji može biti složeniji od SQL-ovih atomarnih tipova. Iskazan je kao lista atributa kojima navodimo ime i podatkovni tip. Kada definiramo strukturirani UDT, moramo definirati ne samo elemente podataka koje sadrži, već i njegovu semantiku - ponašanja koja se izazivaju korištenjem sučelja za pozivanje metode. Vrijednost strukturiranog tipa sastoji se od vrijednosti atributa. Strukturirani tipovi mogu se koristiti kao:

- tip atributa drugog strukturiranog tipa,
- tip parametra funkcija, metoda i procedura,
- tip SQL varijable,
- tip atributa i tip n-torke relacije [1].

LOB tip

Mnoge baze podataka multimedijskih aplikacija koriste velike podatkovne objekte kao što su audio, video, fotografije, tekstualne datoteke, karte itd. Relacijske baze podataka ne

```
CREATE TYPE mjestoT AS (
        postBr      INTEGER,
        nazMjesto   VARCHAR(40)
) NOT FINAL;

CREATE TYPE adresaT AS (
        ulica       VARCHAR(50),
        mjesto      mjestoT
) NOT FINAL;
```

```
CREATE TABLE osoba (
        sifOsoba    INTEGER
        ime         VARCHAR(25),
        prezime     VARCHAR(25),
        adresa      adresaT
);
```

Slika 5.3: Kreiranje tablice s atributima (strukturirani tip)

pružaju odgovarajuću podršku za to. ORDBMS uvode velike objekte (eng. Large Object type - LOB) za rad s takvim podacima. Dakle, LOB tipovi omogućuju pohranu velikih vrijednosti. ORDBMS-ovi obično podržavaju različite vrste LOB podataka kao što su:

- BLOB (eng. Binary Large Object) - niz binarnih podataka, promjenjive duljine, čija je interpretacija prepuštena vanjskoj aplikaciji;
- CLOB (eng. Character Large Object) - niz jednobajtnih znakova promjenjive duljine;
- DBCLOB (eng. Double Byte CLOB) - niz dvobajtnih znakova promjenjive duljine.

Mnogi ORDBMS-ovi također pružaju prilagođene SQL funkcije za LOB podatke. Primjerice funkcije za pretraživanje slikovnih ili video podataka [9].

```
CREATE TABLE osoba (
        sifOsoba    INTEGER,
        ime         VARCHAR(25),
        prezime     VARCHAR(25),
        zivotopis   CLOB (50K),
        slika       BLOB (2M)
);
```

Slika 5.4: Kreiranje tablice koja sadrži LOB tipove

Tipizirane tablice

Tipizirana tablica (eng. Typed Table) novi je tip tablice koja je definirana na temelju preciziranog strukturiranog tipa. Tada svaki atribut strukturiranog tipa postaje stupac tablice, te

tipizirana tablica sadrži dodatni stupac koji sadrži jedinstveni identifikator objekta, poznat kao referenca, za svaki redak tablice. Taj stupac zovemo samoreferencirajući stupac (eng. self-referencing column). Identifikator objekta je jedinstven samo unutar tipizirane tablice. Koriste se za modeliranje veza između entiteta i ponašanje entiteta.

Kako bi bolje objasnili i shvatili tipizirane tablice, na primjeru ćemo pokazati kreiranje jedne. Ta tablica bit će tipa *mjestoT*, stoga prvo moramo njega kreirati (vidi sliku 5.5).

```
CREATE TYPE mjestoT AS (postBr     INTEGER,
                        nazMjesto  VARCHAR(40))
                        INSTANTIABLE
                        NOT FINAL
                        REF IS SYSTEM GENERATED;
```

Slika 5.5: Kreiranje strukturiranog tipa *mjestoT*

Na slici 5.5 crvenom bojom označena je mogućnost direktnog kreiranja instance tipa (INSTANTIABLE - za tip postoji constructor metoda ili NOT INSTANTIABLE - za tip ne postoji constructor metoda) i način generiranja reference na objekt - REF IS klauzula. Načini generiranja vrijednosti reference na objekt su:

- SYSTEM GENERATED - generiranje obavlja sustav,
- USING - kreiranje vrijednosti reference obavlja korisnik,
- FROM - korisnik specificira listu atributa iz strukturiranog tipa, koja će biti korištena za izvođenje jedinstvene reference na objekt [1].

Nakon što smo definirali strukturirani tip, možemo kreirati svoju tipiziranu tablicu (vidi sliku 5.6). OF klauzula sadrži strukturirani tip nad kojim je definirana tipizirana tablica, što je u našem slučaju tip *mjestoT*. Kao i kod kreiranja strukturiranog tipa, REF IS klauzula služi za određivanje načina generiranja vrijednosti reference i mora biti konzistentna s načinom generiranja u korištenom strukturiranom tipu.

```
CREATE TABLE mjesto OF mjestoT
(PRIMARY KEY (postBr),
 REF IS mjestoID SYSTEM GENERATED,
 postBr WITH OPTIONS CONSTRAINT dozvoljeniPbr
                     CHECK (postBr BETWEEN 10000 AND 99999));
```

Slika 5.6: Kreiranje tipizirane tablice tipa *mjestoT*

Upisivanje zapisa i prikaz tog zapisa u tablici *mjesto* prikazano je na slici 5.7.

```
INSERT INTO mjesto VALUES (10000, 'Zagreb');
```

mjestoID	postBr	nazMjesto
1023456734	10000	Zagreb

Slika 5.7: Upisivanje zapisa u tablicu *mjesto*

Pohranjene procedure

Pohranjene procedure mogu biti pozvane iz SQL koda (eng. SQL-invoked routines). SQL razlikuje tri tipa pohranjenih procedura:

- procedura - ima ulazne i izlazne parametre, poziva se CALL naredbom;
- funkcija - ima samo ulazne parametre (izlazni se vraća kao ”vrijednost” funkcije), poziva se korištenjem notacije *imeFunkcije*(*parametri*);
- metoda - specijalni slučaj funkcije, čvrsto je vezana uz jedan strukturirani tip.

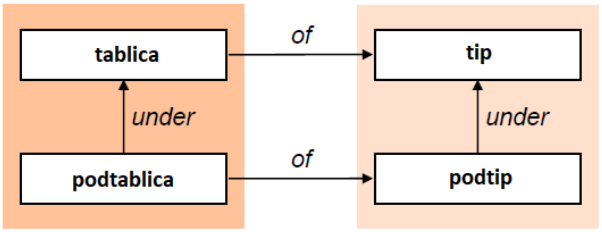
Sa strukturiranim tipom manipuliramo koristeći metode koje su definirane na njemu. Osim ugrađenih metoda korisnici za strukturirane tipove mogu definirati i vlastite metode, to su tzv. korisnički-definirane metode. Postoje tri vrste ugrađenih metoda: constructor, observer i mutator. Metoda constructor, istog je imena kao tip, koristi se za kreiranje instanci tipa i implicitno se definira u trenutku kreiranja tipa. To je funkcija bez argumenata koja vraća pretpostavljene (default) ili NULL vrijednosti atributa, prilikom poziva koristi se izraz *new* [1]. Metode observer i mutator koriste se po jedna za svaki atribut strukturiranog tipa, istog su imena kao atribut. Pozivaju se korištenjem dot notacije (*varijabla.imeFunkcije*). Dok metoda observer vraća vrijednosti atributa strukturiranog tipa, metoda mutator modificira vrijednosti tog atributa.

Nasljeđivanje

Nasljeđivanje možemo podijeliti na nasljeđivanje tipova i nasljeđivanje tablica.

Nasljeđivanje tipova moguće je samo za strukturirane tipove. Podtip nasljeđuje atribute i metode nadređenog tipa. Hijerarhija strukturiranih tipova definira se u definiciji tipa korištenjem klauzule UNDER.

Nasljeđivanje tablica moguće je samo na tipiziranim tablicama. Tipovi podtablica moraju biti podtipovi tipa nadređene tablice i svaki atribut koji postoji u nadređenoj tablici postoji i u podtablicama. Svaka n-torka iz podtablice implicitno postoji i u nadređenoj tablici, odnosno n-torka u podtablici odgovara n-torki u nadređenoj tablici, ako ima iste vrijednosti svih naslijeđenih atributa. Tada te korespondentne n-torke u podtablici i nadređenoj tablici



Slika 5.8: Nasljeđivanje tipova i tablica

predstavljaju isti entitet. Tako nasljeđivanje tablica omogućava više tipova istog objekta, dozvoljavajući istovremeno postojanje entiteta u više od jedne tablice [1].

5.3 Oblikovanje objektno-relacijskih baza podataka

Pojava objektno-relacijskih baza podataka na tržištu komercijalnih baza podataka potaknula je stručnjake na traženje alata za njihovo oblikovanje. ER model koji se koristi za konceptualno oblikovanje relacijskih baza podataka ne može predočiti značajke objektno-relacijskih baza podataka zbog toga što je statičan i ne nudi sredstva za opisivanje dinamičkih aspekata ORDB-ova. Kako bi u potpunosti mogli predočiti sve aspekte ORDB-ova pomoći će nam objektno-orijentirane metode.

Objektno-orijentirane metode koje se koriste za oblikovanje sustava s objektno-relacijskim bazama podataka temelje se na konceptima klasa i objekata te omogućavaju korištenje tri različita modela za oblikovanje objektno-relacijskih baza podataka, ti modeli su:

- statički model- modeliraju se objekti i odnosi među njima,
- dinamički model - opisuju se interakcije između objekata,
- funkcionalni model - vrijednosti podataka mijenjaju se pomoću operacija [4].

UML nudi veliki potencijal za ORDB modeliranje podataka, njegova verzija 2 definira više od desetak dijagrama, kao što su dijagram slučajeva uporabe (eng. use case), dijagram klasa, dijagram aktivnosti, dijagram sekvenci, dijagram stanja i dijagram komunikacije. Navedeni dijagrami idealni su za predstavljanje višestrukih perspektiva ORDB-ova pružajući različite tipove grafičkih dijagrama tijekom različitih faza razvoja ORDB-a. Ne samo da ovi dijagrami mogu modelirati statičke i dinamičke aspekte ORDB-ova, već i nasljeđivanje, enkapsulaciju i druge objektno-orijentirane značajke ORDB-ova. Postoje mnoge tehnike za transformaciju UML modela u ORDB sustave. Iako se niti jedna tehnika

nije pokazala vodećom, svaka ima neke prednosti i nedostatke stoga izbor tehnike ovisi o tipu sustava koji modeliramo.

Kako bi pokrili sve situacije modeliranja, autori UML-a daju mogućnost proširenja sintakse i semantike UML jezika kroz stereotipe, komentare i ograničenja. Definiranje skupa stereotipa, komentara i ograničenja, koji proširuju postojeći tip dijagrama, kako bi se postigao određeni cilj, naziva se profil. Zajedno, ta proširenja omogućuju stvaranje UML ekstenzija prilagođenih specifičnom projektu. Moguće je dodati nove stavke, promijeniti postojeće specifikacije, pa čak i izmijeniti njihovu semantiku. Naravno, važno je da se ta proširenja rade na kontroliran način. UML proširenje mora sadržavati kratki opis, popis i opis stereotipa, komentare, ograničenja, kao i skup pravila oblikovanja vezana uz model podataka. Standardni skup UML notacija i dijagrama može se proširiti korištenjem sljedećih aspekata:

- namjena: objektno-relacijske baze podataka dizajnirane su posebno za pohranu složenih podataka (kao što su XML, multimedijski, prostorni) i moraju se optimizirati za postizanje različitih upita nad njima;
- podatkovni model: ORDBMS implementiraju objektno-relacijski model, hibridni model koji kombinira značajke standardnih podatkovnih modela (relacijskih i objektno-orijentiranih);
- tipične operacije: korištenje objektno-relacijskih baza podataka uključuje operacije s multimedijskim podacima, operacije nad prostornim podacima i operacije sa strukturiranim ili nestrukturiranim XML podacima.

Knjiga [8] opisuje tri razine oblikovanja objektno-relacijskih baza podataka (vidi sliku 5.9):

- konceptualna razina: odnosi se na razvoj modela neovisno o bilo kakvom razmatranju izgleda podataka, kao rezultat analize baze podataka i modeliranja;
- logička razina: odnosi se na razvoj objektno-relacijskog modela podataka, ali neovisno o odabranom DBMS-u;
- fizička razina: odnosi se na implementaciju objektno-relacijske baze podataka, uključuje aspekte koji se odnose na osiguranje sigurnosti podataka, kojima se upravlja kroz određeni DBMS.

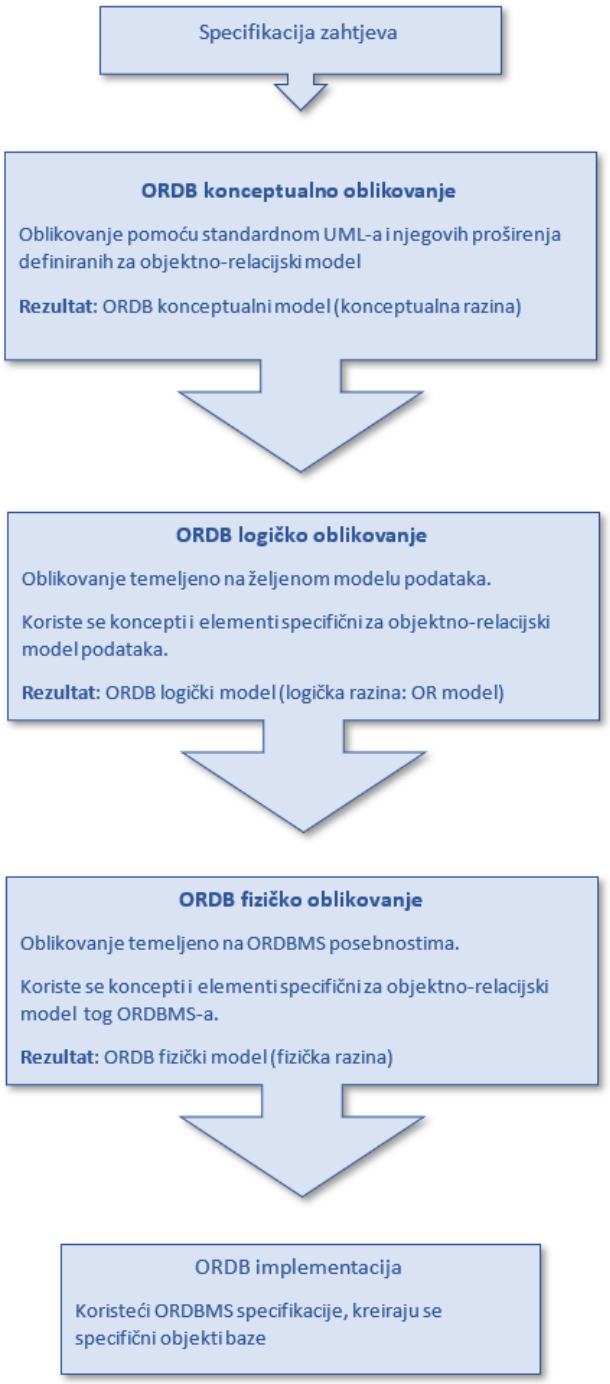
Slika 5.10 prikazuje korake koje je potrebno napraviti da bi se oblikovala objektno-relacijska baza podataka. U tom smislu, neophodan je proces preslikavanja modela dobivenog iz UML dijagrama klasa u standardni objektno-relacijski model (neovisan o proizvodu), a zatim naknadna implementacija u DBMS. Mapiranje objektnog modela specifičnog za UML jezik u objektno-relacijski model koji se može implementirati u bazu

Slika 5.9: Razine u ORDB oblikovanju

podataka zahtijeva, implementaciju identiteta, klasa, asocijacija i odnosa generalizacije, agregacije i kompozicije. Prema [10], transformacija UML dijagrama klasa u objektno-relacijski model nastavlja se na mapiranje logičke strukture podataka i njihovog ponašanja, što nije moguće s tradicionalnim ER modelom. Za razliku od relacijskog modela, tipovi objekata podržavaju enkapsulaciju, a definicija metoda može se eksplicitno povezati s definiranjem tipova objekata.

Konceptualna razina UML klasa (konceptualna shema baze podataka) zadovoljava zahtjeve sustava i fokusira se na klase, asocijacije, atribute, stanja, operacije. Mogu se identificirati različite vrste asocijacija između klasa, kao što su agregacija, kompozicija, hijerarhija, itd. Objektno-relacijska razina (logička shema baze podataka) sadrži elemente predložene standardom SQL:2003, kao što su korisnički definirani tipovi, strukturirani tipovi, reference, ROW tipovi podataka i zbirke podataka. Logičko oblikovanje posebno je važno u oblikovanju objektno-relacijskih baza podataka jer svaki proizvod implementira poseban objektno-relacijski model. Stoga možemo identificirati objektno-relacijsku specifikaciju, neovisnu o proizvodu, koja se može lako odrediti pomoću standarda SQL:2003. Fizička razina (fizička shema baze podataka) zapravo su tablice od kojih su neke kreirane iz prethodno definiranih tipova objekata. Osim tablica, ova razina sadrži i druge specifične relacijske elemente, kao što su ograničenja, podatkovna polja, odnosi između tablica.

Nakon oblikovanja slijedi faza implementacije sustava u kojoj se specificiraju glavne komponente i njihova distribucija na postojeće fizičke resurse. U ovoj fazi pišu se programi za definiranje i manipuliranje objektima u ORDBMS-u koji podržava programske jezike visoke razine, kao i standardizirani deklarativni jezik SQL. Također, sustav se testira kako bi se napravila poboljšanja u njegovom radu i ispravile potencijalne greške.



Slika 5.10: Oblikovanje objektno-relacijske baze podataka

Poglavlje 6

Studijski primjer

U ovom studijskom primjeru korišteni su sljedeći alati:

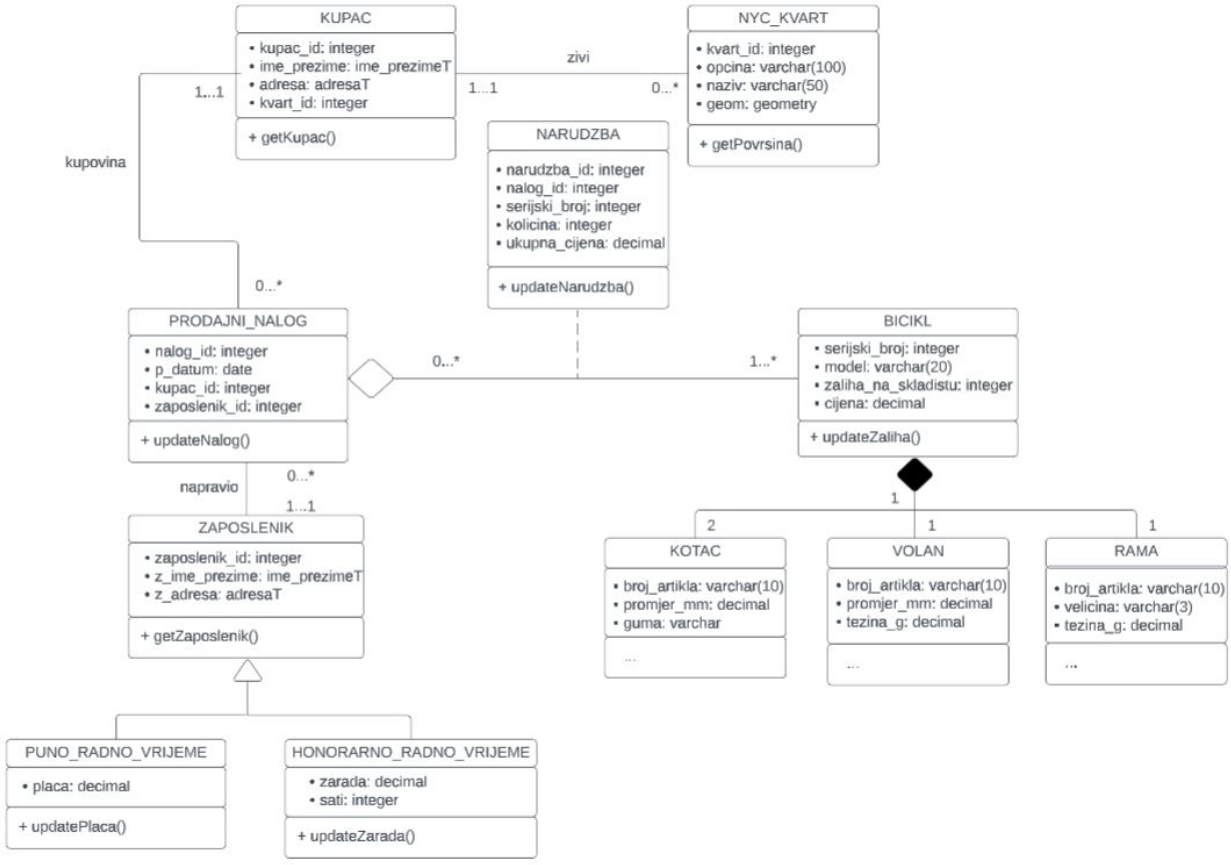
- PostgreSQL (verzija 15) - besplatan sustav za upravljanje objektno-relacijskim bazama podataka otvorenog koda, sustav je proširljiv i drži se većine SQL:2011 standarda;
- PostGIS (verzija 3.3.2) - softverski program otvorenog koda koji dodaje podršku za geografske objekte objektno-relacijskoj bazi podataka PostgreSQL;
- DBeaver (verzija 22.3.3) - SQL klijentska softverska aplikacija i alat za administraciju baze podataka.

Cilj nam je kreirati objektno-relacijsku bazu podataka i pokazati što više značajki objektno-relacijskih baza podataka. Objektno-relacijsku bazu u ovom studijskom primjeru nazvat ćemo *prodaja_NYC* i ona će sadržavati podatke o prodaji bicikla na području New Yorka.

Konceptualno oblikovanje

Prvi korak u svakom oblikovanju baze podataka je konceptualno oblikovanje. Konceptualnu shemu grafički prikazujemo UML dijagramom klasa (vidi sliku 6.1).

Kao što smo opisali u 3.1 klase su prikazane u obliku pravokutnika s imenom klase na vrhu, u sredini se nalaze imena svih atributa, a na dnu imena metoda. Prazan romb označava agregaciju između klasa PRODAJNI_NALOG i BICIKL, tu se nalazi i asocijacijska klasa NARUDZBA koja sadrži atribute vezane uz tu agregaciju. Kod klase BICIKL vidimo i ispunjeni romb koji označava kompoziciju između klasa BICIKL i KOTAC, VOLAN i RAMA. Prazan trokut označava nasljeđivanje između klase ZAPOSLENIK koja je nadklasa klasama PUNO_RADNO_VRIJEME i HONORARNO_RADNO_VRIJEME koje su njene podklase. Uz navedene klase na UML dijagramu klasa možemo vidjeti da imamo



Slika 6.1: UML dijagram klasa objektno-relacijske baze *prodaja_NYC*

još dvije klase koje smo nazvali KUPAC i NYC_KVART. Atribute vezane uz klase možemo vidjeti na slici 6.1.

Logičko oblikovanje

Nakon konceptualnog oblikovanja slijedi nam logičko oblikovanje gdje nam je cilj preslikavanje konceptualne sheme, prikazane UML dijagrama klasa, u standardni objektno-relacijski model. U ovom koraku definirat ćemo korisnički definirane tipove (UDT) koji su nam potrebni. U ovom primjeru svi UDT su strukturirani. Lista potrebnih UDT-a s listom atributa i njihovih podatkovnih tipova:

- ime_prezimeT - ime (VARCHAR) i prezime (VARCHAR);

- mjestoT - postBr (INTEGER) i nazMjesto(VARCHAR);
- adresaT - ulica (VARCHAR) i mjesto (mjestoT);
- kotacT - oznaka_artikla (VARCHAR), promjer_mm (decimal) i marka_gume (VARCHAR);
- volanT - oznaka_artikla (VARCHAR), velicina (VARCHAR) i tezina_g (DECIMAL);
- ramaT - oznaka_artikla (VARCHAR), velicina (VARCHAR) i tezina_g (DECIMAL);
- biciklT - serijski_broj (INTEGER), model (VARCHAR), zaliha_na_skladistu (INTEGER), cijena (DECIMAL), prednji_kotac (kotacT), zadnji_kotac (kotacT), volan (volanT) i rama (ramaT);
- zaposlenikT - zaposlenik_id (INTEGER), ime_prezime (ime_prezimeT) i adresa (adresaT).

Klasa NYC_KVART ima atribut tipa geometry, no njega ne trebamo kreirati jer je dio PostGIS-a, tj. kreira se u trenutku kada kreiramo PostGIS proširenje na bazi podataka.

Popis ugnježdenih relacija, tj. relacija koje ne zadovoljavaju 1NF:

- kupac,
- zaposlenik,
- puno_radno_vrijeme,
- honorarno_radno_vrijeme,
- bicikl.

Zaposlenik će biti i tipizirana tablica definirana na temelju strukturiranog UDT-a zaposlenikT, ona će također biti nadređena tablica, a njezine podtablice biti će puno_radno_vrijeme i honorarno_radno_vrijeme.

Fizičko oblikovanje

Slika 6.2 prikazuje SQL naredbe za kreiranje UDT-ova koje smo naveli u prethodnom koraku (6). Prilikom pisanja naredbi pazili smo na poredak, bitno je prvo kreirati UDT-ove koji se koriste kao tip atributa drugog strukturiranog tipa kako ne bi dobili grešku prilikom izvršavanja naredbi.

```
CREATE TYPE ime_prezimeT AS (
    ime VARCHAR(20),
    prezime VARCHAR(40)
);

CREATE TYPE mjestoT AS (
    postBr INTEGER,
    nazMjesto VARCHAR(40)
);

CREATE TYPE adresaT AS (
    ulica VARCHAR(50),
    mjesto mjestoT
);

CREATE TYPE zaposlenikT AS (
    zaposlenik_id INTEGER,
    ime_prezime ime_prezimeT,
    adresa adresaT
);
```

```
CREATE TYPE kotacT AS (
    oznaka_artikla VARCHAR(30),
    promjer_mm DECIMAL,
    marka_gume VARCHAR(20)
);

CREATE TYPE volanT AS (
    oznaka_artikla VARCHAR(30),
    velicina VARCHAR(3),
    tezina_g DECIMAL
);

CREATE TYPE ramaT AS (
    oznaka_artikla VARCHAR(30),
        velicina VARCHAR(3),
    tezina_g DECIMAL
);

CREATE TYPE biciklT (
    serijski_broj INTEGER,
    model VARCHAR(50),
    zaliha_na_skladistu INTEGER,
    cijena DECIMAL,
    prednji_kotac kotacT,
    zadnji_kotac kotacT,
    volan volanT,
    rama ramaT
);
```

Slika 6.2: SQL naredbe za kreiranje UDT-ova

Nakon što smo kreirali potrebne UDT-ove možemo kreirati potrebne tablice. SQL naredbe prikazane su na slici 6.3. Kod kreiranja tablica isto smo pazili na poredak jer smo ovdje odmah prilikom kreiranja tablica kreirali i potrebna ograničenja, tj. strane ključeve. Ako imamo veliki broj tablica prvo možemo kreirati tablice bez ograničenja onda ne trebamo paziti na redoslijed kreiranja tablica. Zatim nakon što kreiramo sve tablice naredbom "ALTER TABLE naziv_tablice ADD CONSTRAINT naziv_fkey FOREIGN KEY (naziv_stupca) REFERENCES ime_tablice2 (naziv_stupca2);" trebamo kreirati potrebna ograničenja.

Izvršavanjem naredbi kreirana je fizička baza podataka *prodaja_NYC*. Njezina implementacija u DBeaver-u prikazna je na slici 6.4. Na navedenoj slici vidimo da se naša baza sastoji od osam tablica, koje ćemo sada ukratko opisati:

- nyc_kvart - sadrži podatke o kvartovima u New Yorku, budući da se prodaja u našem primjeru odvija jedino na području New Yorka;
- kupac - sadrži podatke o kupcima koji su kupili neki bicikl iz ponude, a žive na području New Yorka;

```
CREATE TABLE nyc_kvart (
    kvart_id INTEGER NOT NULL,
    naziv VARCHAR(100),
    opcina VARCHAR(50),
    geom public.geometry,
    CONSTRAINT nyc_kvart_pkey PRIMARY KEY (kvart_id)
);
CREATE INDEX nyc_neighborhoods_geom_idx ON public.nyc_neighborhoods USING gist (geom);

CREATE TABLE kupac (
    kupac_id INTEGER NOT NULL,
    ime_prezime ime_prezimeT,
    adresa adresaT,
    kvart_id integer,
    CONSTRAINT kupac_pkey PRIMARY KEY (kupac_id),
    CONSTRAINT kupac_kvart_fkey FOREIGN KEY (kvart_id) REFERENCES nyc_kvart(kvart_id)
);

CREATE TABLE zaposlenik OF zaposlenikT (
    CONSTRAINT zaposlenik_pkey PRIMARY KEY (zaposlenik_id)
);

CREATE TABLE puno_radno_vrijeme (placa decimal)
    INHERITS (zaposlenik);

CREATE TABLE honorarno_radno_vrijeme (zarada decimal, sati integer)
    INHERITS (zaposlenik);

CREATE TABLE bicikl OF biciklT (
    CONSTRAINT bicikl_pkey PRIMARY KEY (serijski_broj)
);

CREATE TABLE prodajni_nalog (
    nalog_id INTEGER NOT NULL,
    p_datum DATE NULL,
    kupac_id INTEGER NOT NULL,
    zaposlenik_id INTEGER NOT NULL,
    CONSTRAINT prodajni_nalog_pkey PRIMARY KEY (nalog_id),
    CONSTRAINT nalog_kupac_fkey FOREIGN KEY (kupac_id) REFERENCES kupac(kupac_id),
    CONSTRAINT nalog_zaposlenik_fkey FOREIGN KEY (zaposlenik_id) REFERENCES zaposlenik(zaposlenik_id)
);

CREATE TABLE narudzba (
    narudzba_id INTEGER NOT NULL,
    nalog_id INTEGER,
    serijski_broj INTEGER,
    kolicina INTEGER,
    ukupna_cijena_eur DECIMAL,
    CONSTRAINT narudzba_pkey PRIMARY KEY (narudzba_id),
    CONSTRAINT narudzba_nalog_fkey FOREIGN KEY (nalog_id) REFERENCES prodajni_nalog(nalog_id),
    CONSTRAINT narudzba_bicikl_fkey FOREIGN KEY (serijski_broj) REFERENCES bicikl(serijski_broj)
);
```

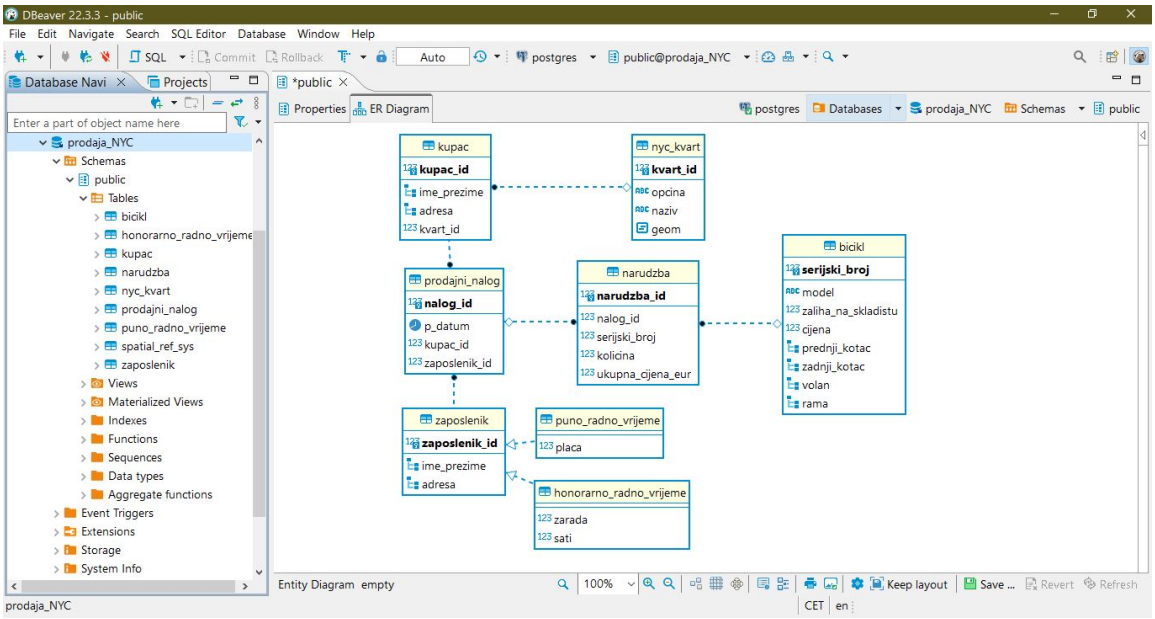
Slika 6.3: SQL naredbe za kreiranje tablica

- prodajni_nalog - sadrži osnovne podatke o prodaji (poput datum prodaje, prodavač i kupac)
- narudzba - sadrži više podataka o prodaji (poput koji bicikl je kupljen, količina i ukupna cijena);
- bicikl - sadrži podatke o biciklima (poput informacija o dijelovima bicikla, stanju zaliha i njegovoj cijeni);
- zaposlenik - sadrži osnovne podatke o svim zaposlenicima, odnosno prodavačima;

- puno_radno_vrijeme - uz osnovne podatke o zaposlenicima sadrži i informaciju o plaći prodavača koji su zaposleni na puno radno vrijeme;

- honorarno_radno_vrijeme - uz osnovne podatke o zaposlenicima sadrži informacije o zaradi i broju odrađenih sati onih prodavača koji su nisu zaposleni na puno radno vrijeme;

Ostale podatke poput imena atributa, tip podataka i primarne ključeve možemo vidjeti na slici 6.4. Sada bazu možemo popuniti podacima kako bi ju mogli testirati i provjerili radi li sve kako treba te kako bi ispravili eventualne pogreške koje smo pronašli testirajući bazu.



Slika 6.4: Implementacija fizičkog podatkovnog modela u DBeaver-u

Testiranje baze podataka

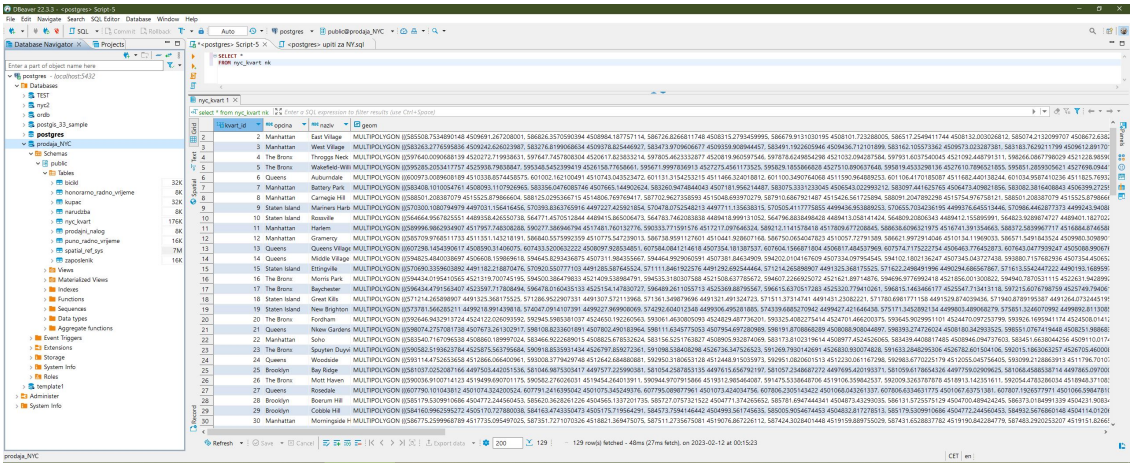
Nakon što smo kreirali bazu podataka trebamo ju popuniti podacima. Najviše podataka koji su ujedno i najkompleksniji nalazi se u tablici NYC_kvart, oni su preuzeti s adrese [2]. Slika 6.5 prikazuje primjer upisivanja zapisa u tablicu NYC_kvart.

Motivacija iza kreiranja ove tablice bila je pokazati osnove rada s geografskim podacima u PostgreSQL-u, za što nam je potrebno i proširenje PostGIS. DBeaver omogućuje

```
INSERT INTO public.nyc_kvart VALUES
    ('Brooklyn','Bensonhurst','SRID=26918;MULTIPOLYGON (((582771.4257198056 4495167.427365481, 584651.2943549604 4497541.64252024,
       585422.2807558172 4496946.878818268, 586886.9685500104 4495817.449966315, 585588.6556524017 4495630.326774346,
       584465.0125448668 4494198.883010179, 584332.4288763426 4494179.2236701315, 584199.7371319681 4494374.26086377,
       584033.4428052086 4494549.90133414, 583829.2610803288 4494765.574784693, 583573.8426533843 4494910.491737136,
       583299.143872625 4495034.007512174, 583134.0036559393 4495034.322667328, 583027.5867306096 4495034.527694217,
       582926.2536556575 4495088.579070913, 582904.7205756706 4495172.444347731, 582771.4257198056 4495167.427365481)))');
```

Slika 6.5: Primjer upisivanja geografskih podataka

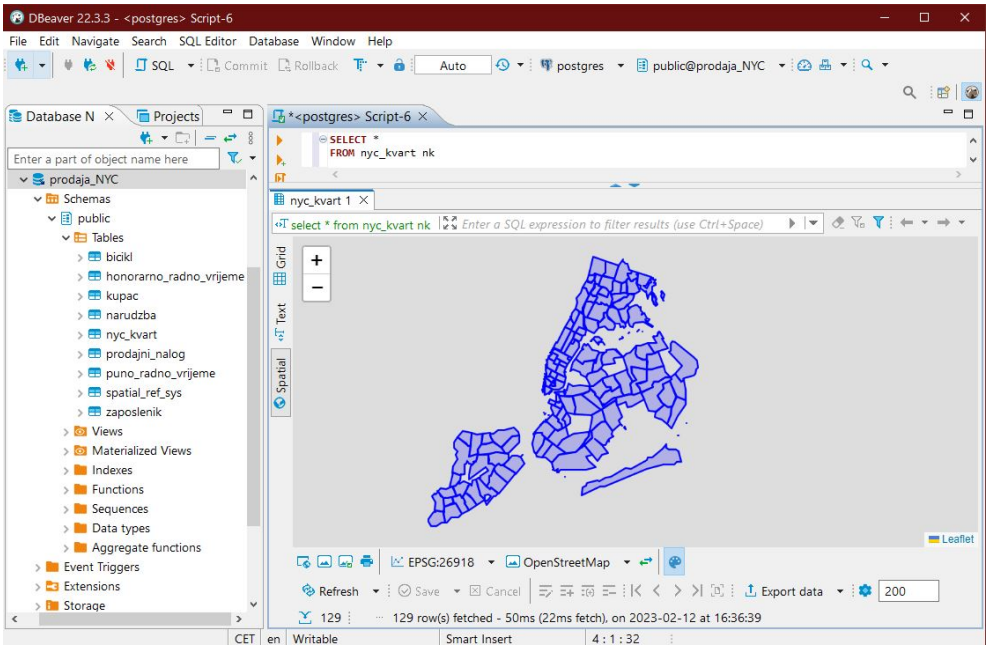
prikaz rezultata postavljenog upita u raznim oblicima. Slika 6.6 prikazuje rezultat dohvata svih zapisa tablice NYC_kvart u tabličnom prikazu. Tablični prikaz ujedno je najpoznatiji i najčešće se koristi, ali za geografske podatke baš i nije koristan. DBeaver nam omogućuje i poseban oblik za geografske podatke, ako za rezultate odaberemo opciju prikaza ”Spatial” podaci će nam se prikazati na karti što je jako korisno za provjeru upita postavljenih nad geografskim podacima. Slika 6.7 prikazuje takav prikaz rezultata upita koji dohvaća sve zapise tablice NYC_kvart, tj. svih kvartova grada New Yorka.



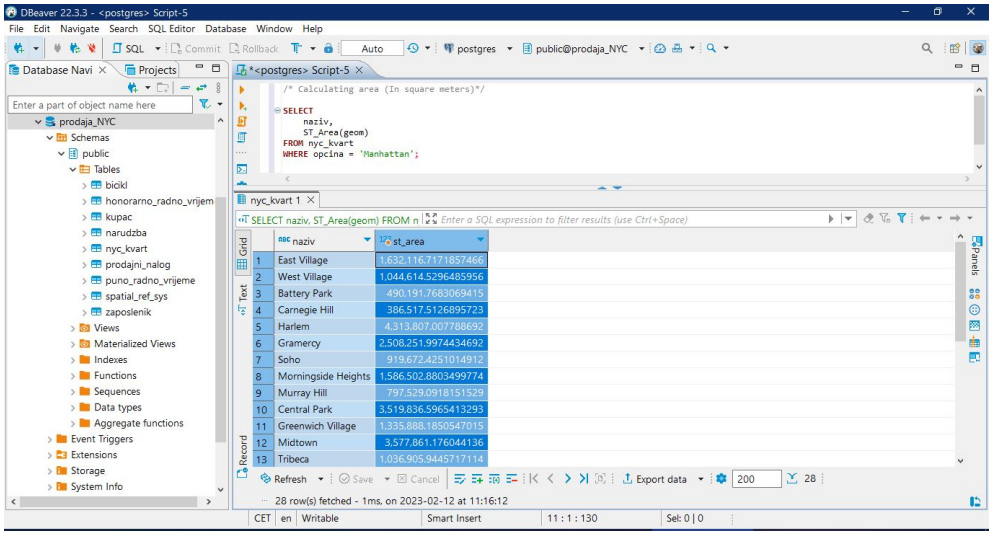
Slika 6.6: Primjer tabličnog prikaza rezultata upita

PostGIS ima i funkcije specifične za geografske podatke (npr. ST_Contains, ST_Covers, ST_Intersects, ST_Overlaps, ...). Slika 6.8 prikazuje primjer korištenja funkcije ST_Area koja prima geografske podatke i vraća njihovu površinu izračunatu u kvadratnim metrima. Funkciju smo iskoristili kako bi izračunali površinu svakog kvarta New York-a koji se nalazi u općini Manhattan. Na slici je prikazano 13 od 28 rezultata koje nam je upit vratio.

Sada ćemo pokazati i primjer upisivanja zapisa u neku od ugniježđenih relacija. Za primjer ćemo uzeti tablicu zaposlenik, jer nam je ona ujedno i tipizirana tablice i nadređena



Slika 6.7: Primjer geografskog prikaza rezultata upita



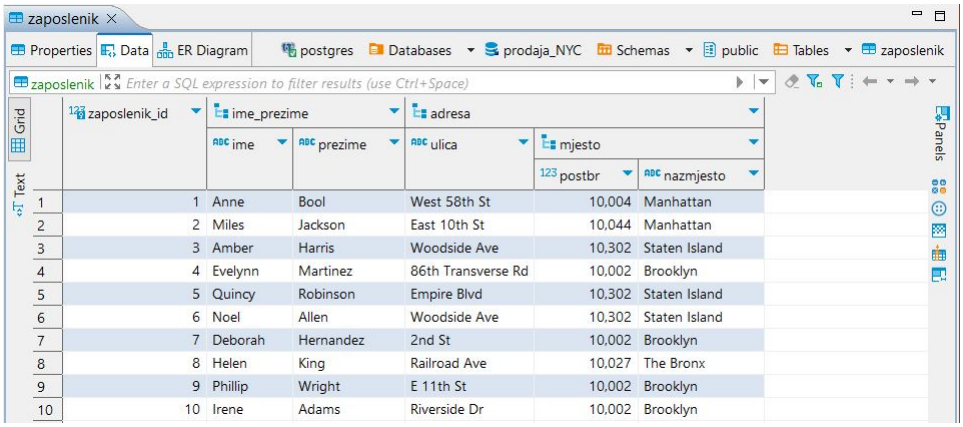
Slika 6.8: Primjer korištenja PostGIS funkcije ST_Area

tablica podtablicama puno_radno_vrijeme i honorarno_radno_vrijeme. Slika 6.9 prikazuje primjere upisivanja zapisa u podtablice puno_radno_vrijeme i honorarno_radno_vrijeme.

```
/* zaposlenik - puno */

INSERT INTO public.puno_radno_vrijeme VALUES
    (1, ('Anne', 'Bool'), ('West 58th St', (10004, 'Manhattan')), 2000),
    (2, ('Miles', 'Jackson'), ('East 10th St', (10044, 'Manhattan')), 2500),
    (3, ('Amber', 'Harris'), ('Woodside Ave', (10302, 'Staten Island')), 1875);

/* zaposlenik - honorarno*/

INSERT INTO public.honorarno_radno_vrijeme VALUES
    (6, ('Noel', 'Allen'), ('Woodside Ave', (10302, 'Staten Island')), 1000, 80),
    (7, ('Deborah', 'Hernandez'), ('2nd St', (10002, 'Brooklyn')), 1525, 70),
    (8, ('Helen', 'King'), ('Railroad Ave', (10027, 'The Bronx')), 1250, 80);
```

Slika 6.9: Primjer upisivanja zapisa u ugnježdenu relaciju

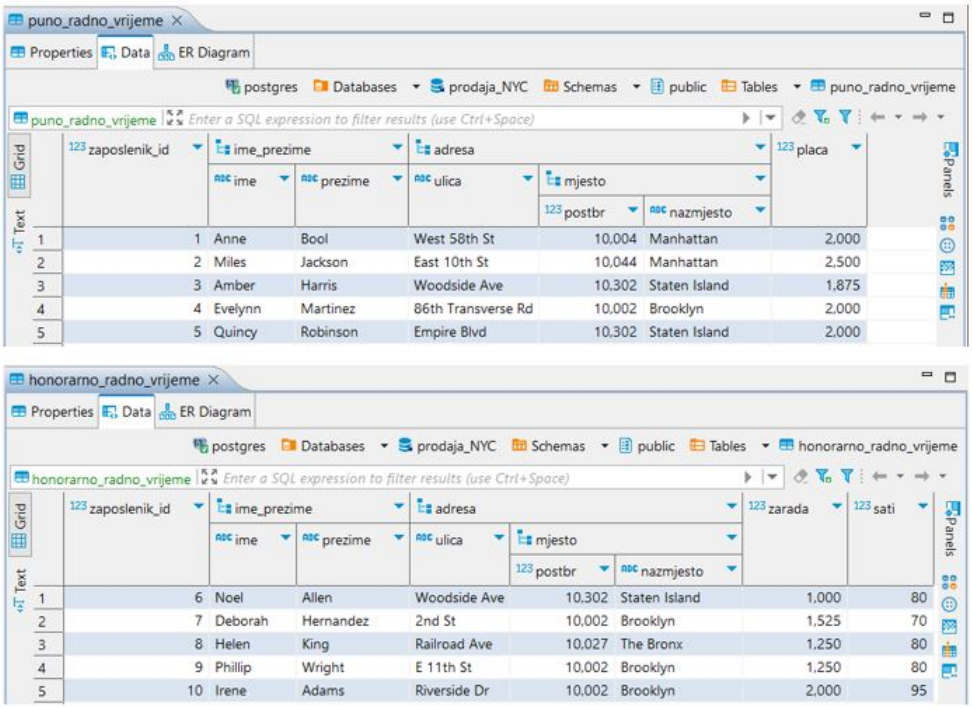
Slika 6.10 prikazuje zapise svih zaposlenika koji se nalaze u nadtablici zaposlenik, dok slika 6.11 prikazuje zapise o zaposlenicima u podtablicama puno_radno_vrijeme i honorarno_radno_vrijeme.



	zaposlenik_id	ime_prezime		adresa		
		ime	prezime	ulica	mjesto	
					postbr	nazmjesto
1	1	Anne	Bool	West 58th St	10,004	Manhattan
2	2	Miles	Jackson	East 10th St	10,044	Manhattan
3	3	Amber	Harris	Woodside Ave	10,302	Staten Island
4	4	Evelynn	Martinez	86th Transverse Rd	10,002	Brooklyn
5	5	Quincy	Robinson	Empire Blvd	10,302	Staten Island
6	6	Noel	Allen	Woodside Ave	10,302	Staten Island
7	7	Deborah	Hernandez	2nd St	10,002	Brooklyn
8	8	Helen	King	Railroad Ave	10,027	The Bronx
9	9	Phillip	Wright	E 11th St	10,002	Brooklyn
10	10	Irene	Adams	Riverside Dr	10,002	Brooklyn

Slika 6.10: Zapisi o zaposlenicima u nadtablici zaposlenik

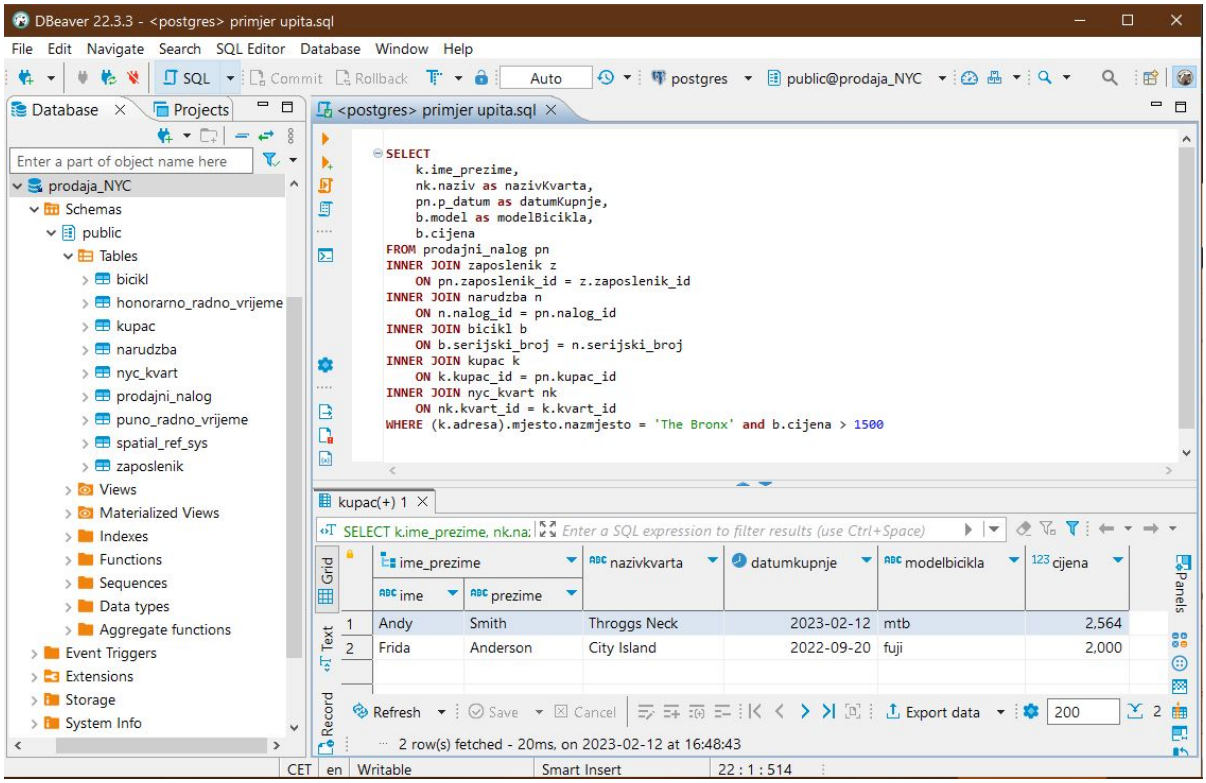
Iz slika vidimo da svaki stupac koji postoji u nadređenoj tablici postoji i u podtablicama, te svaki zapis iz podtablice implicitno postoji i u nadređenoj tablici, ali bez dodatnih stupaca koji su specifični za podtablice. Ako želimo dohvatiti samo zapise koji su direktno upisani u nadtablicu uz naziv tablice u FROM dijelu SELECT naredbe trebamo koristiti ključnu riječ ONLY. U našem slučaju takav upit nam ne bi dao rezultate jer nijedan zapis



Slika 6.11: Zapisi o zaposlenicima u podtablicama

nismo upisivali direktno u nadtablicu. Također ako obrišemo zapis o nekom zaposleniku iz nadtablice, obrisat će se i zapis o tom zaposleniku i iz odgovarajuće podtablice.

Sada ćemo napraviti jedan složeniji upit koji zahtijeva dohvat podataka iz nekoliko tablica. Tražili smo podatke o osobama koje žive na području The Bronx-a i koje su kupile bicikl čija cijena je veća od 1500 €. Informacije o tim kupcima koje smo htjeli dobiti su ime i prezime, naziv kvarta u kojem žive, kada su kupili bicikl, koji model bicikla su kupili i koja je njegova cijena. Na slici 6.12 možemo vidjeti opisani upit i njegove rezultate. Kao rezultat smo dobili dva kupca čije podatke možemo iščitati s navedene slike.



Slika 6.12: Primjer složenijeg upita

Zaključak

U ovom smo se diplomskom radu najprije susreli sa značajkama relacijskih i objektno-orijentiranih baza podataka. Zatim smo se fokusirali na objektno-relacijske baze podataka koje su nastale kako bismo dobili najbolje iz relacijskog i objektnog modela podataka. Objektno-relacijske baze podataka jednostavnije su za korištenje od objektno-orijentiranih baza podataka i mogu pohranjivati složene tipove podataka za razliku od relacijskih. Ipak i one imaju svojih nedostataka. I one su puno složenije za korištenje od relacijskih baza podataka, a izgubljena je jednostavnost i čistoća relacijskog modela. Pobornici relacijskog modela, a ni pobornici objektno-orijentiranog modela nisu zadovoljni ovim kompromisom. Teško je zadovoljiti sve potrebe trenutnih informacijskih sustava jer ne postoji jedinstveno savršeno rješenje. I sa svojim nedostacima smatram da su objektno-relacijske baze podataka bolje rješenje od relacijskih i objektno-orijentiranih baza podataka. Lakše se mogu prilagoditi potrebama sustava i imaju karakteristike oba svoja prethodnika. Njihov potencijal je prepoznat u poslovnom svijetu i mnogi stručnjaci i dalje rade na njihovim poboljšanjima. Zbog toga smatram da će u budućnosti baze s određenim objektno-orijentiranim proširenjima biti najzastupljenije baze podataka.

Bibliografija

[1] *Napredni modeli i baze podataka*, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2014., https://www.fer.unizg.hr/_download/repository/3._OOBP_i_ORBP.pdf.

[2] PostGIS podaci, dostupno na, (prosinac 2022.), http://s3.cleverelephant.ca/postgis-workshop-2020.zip.

[3] S.W. Dietrich, *Fundamentals of Object Databases-Object-Oriented and Object-Relational Design*, Morgan Claypool Publishers, 2011.

[4] I. Botha M. Velicanu, *Solutions for the Object-Relational Databases Design*, Database Systems Journal vol. II (2011.), br. 4/2011.

[5] R. Manger, *Baze podataka*, Element, Zagreb, 2014.

[6] _____, *Osnove projektiranja baza podataka*, Sveučilišni računski centar, Sveučilište u Zagrebu.

[7] J. Melton, *Advenced SQL 1999-Understanding Object-Relational and Other Advenced Features*, Morgan Kaufmann, San Francisco CA, 2002.

[8] C. Begg T. Connolly, *Database Systems - A Practical Approach to Design, Implementation, and Management*, Addison Wesley - Pearson Education, 2005.

[9] B. Baesen W. Lemahieu, S. vanden Broucke, *Principles of Database Management - The Practical Guide to Storing, Managing an Analyzing Big and Small Data*, Cambridge University Press, Cambridge UK, 2018.

[10] M. Wang, *Using UML for Object-Relational Databases Systems Development: A Framework*, (2008.), https://iacis.org/iis/2008/S2008_1108.pdf.

Sažetak

Tema ovog rada su objektno-relacijske baze podataka. Rad je podijeljen na šest poglavlja.

Prvo poglavlje sadrži osnovne pojmove vezane općenito za baze podataka. Na kraju prvog poglavlja opisujemo proces modeliranja podataka na koji ćemo se vraćati kod svake vrste baze podataka u ovom radu.

U drugom poglavlju opisujemo relacijske baze podataka, osnovne pojmove vezane za njih i zatim detaljno opisujemo modeliranje entiteta i veza (ER model). Prvo poglavlje završava relacijskim modelom, gdje su opisane njegove značajke.

U trećem poglavlju zbog boljeg razumijevanja prvo navodimo značajke objektno-orijentiranog programiranja. Zatim opisujemo UML-ov dijagram klasa, a onda po istom principu kao u prethodnom poglavlju, opisujemo objektni model podataka.

Četvrto poglavlje daje usporedbu relacijskih i objektno-orijentiranih baza podataka. Ističemo njihove prednosti i nedostatke.

U petom poglavlju opisujemo objektno-relacijske baze podataka koje su nastale kao kompromis relacijskih i objektno-orijentiranih baza podataka i kao takve imaju značajke oba prethodno opisana modela podataka.

Diplomski rad završava šestim poglavljem u kojem je prikazan studijski primjer oblikovanja i testiranja objektno-relacijske baze podataka.

Summary

The topic of this thesis is object-relational databases. The thesis is divided into six chapters.

The first chapter covers the basics of databases in general. At the end of the first chapter, we describe the data modeling process, which we will refer to for each database type.

In the second chapter, we describe the relational databases, the basic terms associated with them, and then we describe the entity–relationship model (ER model) in detail. The first chapter concludes with the relational model, the characteristics of which are described.

In the third chapter, for better understanding, we first list the features of object-oriented programming. Next, we describe the UML class diagram and, using the same principle as in the previous chapter, we describe the object data model.

The fourth chapter is a comparison of relational and object-oriented databases. We highlight their advantages and disadvantages.

In the fifth chapter, we describe an object-relational database. It was created as a compromise between relational and object-oriented databases and thus has the characteristics of the two previously described data models.

The thesis concludes with the sixth chapter, which presents a case study of the modeling and testing of an object-relational database.

Životopis

Zovem se Katarina Kozina i rođena sam 26. prosinca 1993. u Zagrebu, Republika Hrvatska. Pohađala sam Osnovnu školu Zapruđe, a 2012. sam završila opću gimnaziju "I. gimnazija" u Zagrebu. Po završetku srednje škole upisala sam, a 2017. završila preddiplomski studij matematike na Prirodoslovno-matematičkom fakultetu (Matematički odsjek) Sveučilišta u Zagrebu. Nakon nekoliko godina na diplomskom studiju Primijenjena matematika, na Prirodoslovno-matematičkom fakultetu u Zagrebu, odlučila sam promijeniti diplomski studij te sam upisala nastavnički smjer Matematika.

Nakon završetka preddiplomskog studija, točnije u travnju 2018. zaposlila sam se u Procjeni kreditnog rizika u Privrednoj banci Zagreb. Nakon toga sam od travnja 2019. do rujna 2022. radila u Combisu kao Database Developer, nakon čega sam se zaposlila u Syntio-u kao Associate Data Engineer, gdje se uskoro i planiram zaposliti na puno radno vrijeme. Slobodne vikende i lijepo vrijeme volim iskoristiti za bijeg u prirodu i planinarenje.