UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Milica Krasić

ORODJA IN TEHNIKE PRI MODELIRANJU PODATKOVNIH BAZ

Diplomska naloga

Ljubljana, 2016

ZAHVALA

Zahvaljujem se mentorju mag. Matiji Lokarju za pomoč, nasvete in koristne napotke pri izdelavi diplomske naloge ter potrpežljivost pri pisanju popravkov. Zahvaljujem se tudi družini in vsem prijateljem, ki so me tako ali drugače podpirali in mi stali ob strani v času študija in izdelavi diplomske naloge.

KAZALO

KAZALO SLIK

KAZALO TABEL

PROGRAM DELA

V diplomski nalogi predstavite orodja in tehnike pri modeliranju podatkovnih baz. Predstavite postopek od izgradnje konceptualnega modela do logičnega in fizičnega modela. Pri vsakem koraku postopka predstavite ustrezen računalniški program, ki nam pomaga pri izgradnji.

Osnovna literatura:

- Connolly, T. M. and Begg, C. E. 2005. *Database Systems (A Practical Approach to Design, Implementation and Management).* London: Addison-Wesley, 2005.
- Mohorič, T., Podatkovne baze. Ljubljana, BI-TIM 2002.
- Ramakrishnan, R., Gehrke, J., Database management systems-third edition. New York, McGraw-Hill 2003.
- Teorey, T., Lightstone, S., Nadeau, T. 2006. *Database Modeling & Design: Logical Design - Fourth Edition.* San Francisco: Morgan Kaufmann Publishers, 2006.

Ljubljana, april 2016 Viš. pred. mag. Matija Lokar

POVZETEK

V diplomski nalogi so predstavljena nekatera orodja in tehnike pri modeliranju relacijskih podatkovnih baz. Diplomska naloga poleg uvodnega in zaključnega dela obsega še štiri poglavja. V prvem poglavju je na kratko predstavljena relacijska podatkovna baza, sistem za upravljanje podatkovnih baz (SUPB), načrtovanje in faze načrtovanja podatkovnih baz, modeliranje podatkov in orodja CASE. Ostala tri poglavja so razdeljena glede na faze načrtovanja relacijske podatkovne baze. Drugo poglavje torej govori o konceptualnem modelu in opisuje gradnjo konceptualnega modela, ki sloni na metodi entitetno-relacijskega modela (ER-model). V tretjem poglavju je predstavljen logični model in izgradnja logičnega modela, ki temelji na relacijskem podatkovnem modelu. Četrto poglavje pa je namenjeno opisu in izgradnji fizičnega modela s pomočjo Oraclovega sistema za upravljanje podatkovnih baz. Vsi trije omenjeni modeli so opisani in zgrajeni na primeru poenostavljene podatkovne baze knjižnice, ki je predstavljen v prej omenjenem prvem poglavju.

Math. Subj. Class. (2010): 68P15

ComputingReview Class.System (1998): D.2.1, D.2.2, D.2.10, D.3.1, D.3.2, D.3.3, E.1

Ključne besede: relacijske podatkovne baze, načrtovanje podatkovne baze, modeliranje podatkov, konceptualni model, logični podatkovni model, fizični model, SUPB, orodja CASE

Keywords: relational database, database design, data modeling, conceptual model, logical data model, physical model, DBMS, CASE tools

1 UVOD

Podatkovne baze so danes nepogrešljive povsod, kjer je treba na urejen način shranjevati podatke in zagotavljati njihovo celovitost. Zato bomo v tej diplomski nalogi opisali in predstavili razvoj podatkovne baze. Na podatkovno bazo lahko gledamo kot na zbirko podatkov, ki jo upravlja sistem za upravljanje podatkovnih baz (SUPB). Po svoji vsebini pa predstavlja sliko iz dela realnega sveta. Poznamo več vrst podatkovnih baz, ki jih delimo glede na to, na katerem modelu temeljijo. Tako poznamo hierarhične, mrežne, relacijske in objektne. Prevladujejo relacijske podatkovne baze in prav načrtovanje in modeliranje teh bo obravnavala ta diplomska naloga.

Osnovni namen diplomske naloge je predstaviti in prikazati nekatera orodja in tehnike pri načrtovanju in modeliranju relacijskih podatkovnih baz. Drugi cilj je bralcu na čim bolj preprost način prikazati postopek izdelave relacijske podatkovne baze. Postopek izdelave relacijske podatkovne baze je v diplomski nalogi prikazan na zgledu poenostavljene podatkovne baze knjižnice. Na omenjenem zgledu tudi temelji večina primerov, ki so v diplomski nalogi predstavljeni.

Za izdelavo primera knjižnice bomo uporabili tri različna orodja. Za konceptualni model bomo uporabili orodje DIA, za logični model orodje DBDesigner, za izdelavo fizičnega modela pa razvojno okolje Oracle SQL Developer, preko katerega se bomo povezali na Oraclovo bazo XE. Tako orodji DIA in DBDesigner kot tudi razvojno okolje Oracle SQL Developer so prostodostopni. Omenjena orodja bodo v diplomski nalogi na kratko predstavljena tako, da jih bomo uporabili na praktičnem primeru, ki ga bomo ilustrirali z več slikami.

2 NAČRTOVANJE IN MODELIRANJE RELACIJSKE PODATKOVNE BAZE

2.1 Relacijska podatkovna baza

Dandanes se s podatkovnimi bazami srečujemo povsod, kjer na urejen način shranjujemo podatke, zagotavljamo njihovo dostopnost in ščitimo njihovo celovitost. Podatkovna baza ali baza podatkov je v knjigi (Connolly, in drugi, 2005) definirana kot urejena zbirka logično povezanih podatkov in opisov le-teh, načrtovana pa je tako, da zadovoljuje informacijske potrebe organizacije. Po svoji vsebini predstavlja model oziroma podatkovno sliko določenega dela realnega sveta. Pojem podatkovna baza je v različnih literaturah definirana na različne načine. V knjigi (Mohorič, 2002) je podatkovna baza definirana z naslednjima definicijama:

1. *Podatkovna baza je zbirka med seboj pomensko povezanih podatkov, ki so shranjeni v računalniškem sistemu, dostop do njih je centraliziran in omogočen s pomočjo sistema za upravljanje podatkovnih baz.*
2. *Podatkovna baza je mehanizirana, večuporabniška, formalno definirana in centralno nadzorovana zbirka podatkov.*

Iz definicij lahko povzamemo, da se v podatkovni bazi shranjujejo podatki, ki se nanašajo na določen del sveta, v katerem so vse stvari in dogajanja med seboj povezana; da je računalniško podprta; da obstajajo centralni mehanizmi za nadzor in dostop do podatkov (SUPB); da obstaja predpis, kakšni in kateri podatki se smejo nahajati v podatkovni bazi. Sisteme za upravljanje podatkovnih baz bomo podrobneje predstavili v naslednjem razdelku (poglavje 2.2).

Podatkovna baza je po priporočilih ANSI/SPARC[1] (Wikipedia) sestavljena iz treh nivojev, na katerih lahko opišemo podatke:

1. **zunanji (uporabniški) nivo** – predstavlja uporabniški pogled na podatkovno bazo,
2. **konceptualni (logični) nivo** – predstavlja, kakšni podatki so shranjeni v podatkovni bazi in kakšne so povezave med podatki,
3. **notranji (fizični) nivo** – predstavlja, kako so podatki shranjeni v podatkovni bazi in kakšen je dostop do podatkov.

Glede na to, kako so podatkovne baze strukturirane oziroma na katerem podatkovnem modelu temeljijo, jih delimo na hierarhične, mrežne, relacijske in objektne. Prevladujejo relacijske podatkovne baze, na katere se bomo v diplomski nalogi tudi omejili.

Relacijske podatkovne baze temeljijo na relacijskem podatkovnem modelu. Vpeljal ga je Edgar F. Codd leta 1970 (Mohorič, 2002). Relacijski podatkovni model temelji na relacijski teoriji, izpeljani iz teorije množic. Matematična osnova relacijskega podatkovnega modela je relacija. Relacija je predstavljena s tabelo, ki je sestavljena iz vrstic in stolpcev. Ker je relacijski podatkovni model predstavljen z množico tabel, je človeku dobro razumljiv. Je preprost, fleksibilen in ne vsebuje elementov fizičnega shranjevanja podatkov, s čimer je zagotovljena fizična podatkovna neodvisnost. Fizična podatkovna neodvisnost pomeni, da spremembe na notranjem nivoju, npr. sprememba zunanjega pomnilnika, datotečnega sistema itd. ne povzročajo sprememb na predhodnem (konceptualnem) nivoju.

[1] ANSI-SPARC: American National Standards Institute, Standards Planning And Requirements Committee.

Več o relacijskem podatkovnem modelu bomo povedali v poglavju 3, kjer bomo obravnavali logični model.

2.2 Sistem za upravljanje podatkovnih baz (SUPB)

Podatkovno bazo ustvarimo in vzdržujemo s pomočjo sistemov za upravljanje podatkovnih baz ali okrajšano SUPB. **SUPB** je programska oprema, ki omogoča definiranje, ustvarjanje in vzdrževanje podatkovne baze ter hkrati zagotavlja nadzorovan dostop nad uporabo podatkov v podatkovni bazi (Projekt Colos). SUPB torej zagotavlja učinkovito, varno in uporabnikom prijazno okolje. Namenjen je shranjevanju, dostopu in vzdrževanju velikih količin medsebojno povezanih podatkov. V knjigi (Mohorič, 2002) je SUBP predstavljen kot posrednik med podatkovno bazo in njenimi uporabniki. Njegova naloga je upravljati podatkovno bazo v skladu s potrebami uporabnikov.

SUPB je namenjen raznolikim uporabnikom, od načrtovalca podatkovne baze, programerja, skrbnika do končnega uporabnika. Pri izbiri SUPB je tako treba sodelovati tudi z uporabniki oziroma z naročnikom podatkovne baze. SUPB torej izvaja številne naloge. Med najpomembnejše naloge, ki jih SUPB omogoča, prištevamo (povzeto po (Projekt Colos)):

- **Ustvarjanje podatkovne baze in definiranje njene strukture**
  Ustvarjanje tabel in podatkovne baze nam omogoča jezik za definiranje podatkov, ki mu pravimo DDL (Data Definition Language). Kot bomo videli v nadaljevanju, DDL predstavlja en del oziroma je podmnožica povpraševalnega jezika SQL (Structured Query Language). DDL nam omogoča ustvarjanje, spreminjanje in brisanje tabel.
- **Vzdrževanje podatkovne baze** (učinkovito poizvedovanje in posodabljanje podatkov)
  Poizvedovanje, vstavljanje, brisanje in posodabljanje podatkov v podatkovni bazi nam omogoča jezik za upravljanje s podatki, ki mu pravimo DML (Data Manipulation Language). DML je ravno tako podmnožica jezika SQL.
- **Zaščita podatkov** (varovanje podatkov pred neavtoriziranimi dostopi in nesrečami)
  SUPB mora imeti vgrajen varnostni mehanizem za zaščito podatkov, s katerim omogoča vpogled podatkov le privilegiranim uporabnikom. Skrbi tudi za podvojen zapis podatkov, ki so potrebni pri obnavljanju podatkovne baze v primeru nesreč.
- **Zagotavljanje integritete podatkov** (preverjanje vrednosti vhodnih podatkov, nadzor nad sočasnim dostopom do podatkov)
  Pri izvedbi kakršne koli spremembe podatkov, npr. pred dodajanjem ali spreminjanjem podatkov, mora SUPB preveriti, ali so vrednosti teh podatkov skladne glede na določeno domeno oziroma zalogo vrednosti. Na primer, pri vnosu podatka v tabelo s poljem »Datum« ne moremo vnesti vrednosti 10.000, ker ta vrednost ne predstavlja veljavne vrednosti za datumski tip podatka.
- **Zagotavljanje razpoložljivosti podatkov**
  SUPB mora zagotoviti sočasen in učinkovit dostop vsem uporabnikom do vseh vrst podatkov, ki jih potrebujejo pri svojem delu in do katerih smejo dostopati.
- **Izvajanje transakcij** (operacije nad podatki)
  Vsaka sprememba na podatkih, naj bo to dodajanje, spreminjanje ali brisanje zapisa v tabeli v podatkovni bazi, pomeni v »ozadju« določeno transakcijo. Transakcije so torej osnovne enote izvrševanja v podatkovni bazi, preko katerih spreminjamo, brišemo, ali dodajamo podatke v podatkovno bazo. Transakcija je lahko sestavljena iz ene ali več operacij. SUPB implementira transakcije, s katerimi omogoča izvedbo sklopa operacij na atomaren način (po principu »vse ali nič«). To pomeni, da se transakcija mora izvesti do konca. Če se transakcija prekine med

delovanjem bodisi pričakovano bodisi nepričakovano, se transakcija razveljavi. Hkrati pa se razveljavijo vse operacije, ki sestavljajo transakcijo. To pomeni, da je baza v takem stanju, kot je bila pred začetkom transakcije. Na primer, da se nam sistem poruši pred koncem izvedbe transakcije, nam SUPB zagotovi, da se ob naslednjem zagonu SUPB-ja sistem obnovi v stanje, kot je bilo pred izvedbo transakcije – vse spremembe se torej prekličejo oziroma razveljavijo. V primeru, da se sistem poruši po izvedbi transakcije (stanje transakcije je torej izvršeno), nam SUPB zagotavlja, da so vse spremembe shranjene v podatkovni bazi.

Nekatere funkcije podpirajo uporabo podatkovne baze in so namenjene splošnim uporabnikom, npr. poizvedovanje po podatkih in posodabljanje podatkov v podatkovni bazi. Funkcije, ki podpirajo gradnjo podatkovne baze, npr. ustvarjanje podatkovne baze, reorganizacija podatkovne baze, pa so namenjene upravitelju podatkovne baze. Ostale funkcije skrbijo za zaščito podatkov in zagotavljajo nadzor nad uporabo podatkov.

Izbira SUPB je odvisna od več meril. Poleg stroškovnih meril moramo pri izbiri upoštevati predvsem: povezljivost z obstoječimi sistemi, zmogljivost in nadgradljivost sistema, odzivni čas sistema, ali so funkcionalnosti, ki jih sistem ponuja, skladne s potrebami, in možnosti izobraževanja. Običajno je glavni cilj izbire najustreznejšega SUPB-a čim večje zadovoljstvo uporabnikov z delovanjem podatkovne baze. Torej sta pomembna predvsem odzivni čas in zmogljivost sistema, ki jo merimo v številu izvedenih transakcij v časovni enoti.

Pravi razvoj sistemov za upravljanje podatkovnih baz se je začel v začetku 70. let s predstavitvijo mrežnega in relacijskega podatkovnega modela. Danes prevladujejo relacijski sistemi za upravljanje podatkovnih baz. Najpogosteje uporabljeni sistemi za upravljanje podatkovnih baz so:

- Oracle (http://www.oracle.com/index.html),
- MySQL (https://www.mysql.com/),
- Microsoft SQL Server (https://www.microsoft.com/sqlserver),
- PostgreSQL (https://www.postgresql.org/),
- IBM DB2 (https://www.ibm.com/analytics/us/en/technology/db2/),
- Microsoft Access (https://products.office.com/sl-si/access),
- SQLite (https://www.sqlite.org/),
- Informix (https://www-01.ibm.com/software/data/informix/),
- Firebird (http://www.firebirdsql.org/).

Zgoraj našteti sistemi so razvrščeni glede na njihovo množično uporabo oziroma glede na njihovo priljubljenost med uporabniki (vir: http://db-engines.com/en/ranking). Od naštetih sistemov so odprtokodni MySQL, PosgreSQL, Firebird, SQLite. Vsi sodobni sistemi podpirajo bodisi relacijski bodisi objektno-relacijski podatkovni model.

2.3 Načrtovanje in faze načrtovanja podatkovne baze

Vse, kar je okoli nas, vključno z nami, imenujemo svet. Vsak človek si v svetu, v katerem živi, ustvarja svojo podobo. Da lahko podobo sveta preslikamo v podatkovni bazi, jo je treba ustrezno strukturirati in identificirati njene osnovne sestavine (Mohorič, 2002). Načrtovanje podatkovne baze je postopek opredelitve in razvoja strukture podatkovne baze. V času načrtovanja podatkovne baze naredimo formalni model nekaterih vidikov realnega sveta (»mini svet«). Mera za pravilnost načrtovanega modela podatkovne baze je realni svet. To pomeni, da mora podatkovna baza odražati podatke, pravila in izjeme iz realnega sveta (Projekt Colos).

Načrtovanje podatkovne baze je kompleksen proces, ki zahteva odločitve na različnih ravneh. Pri načrtovanju podatkovne baze se zato velikokrat srečujemo z vrsto težav, kot so:

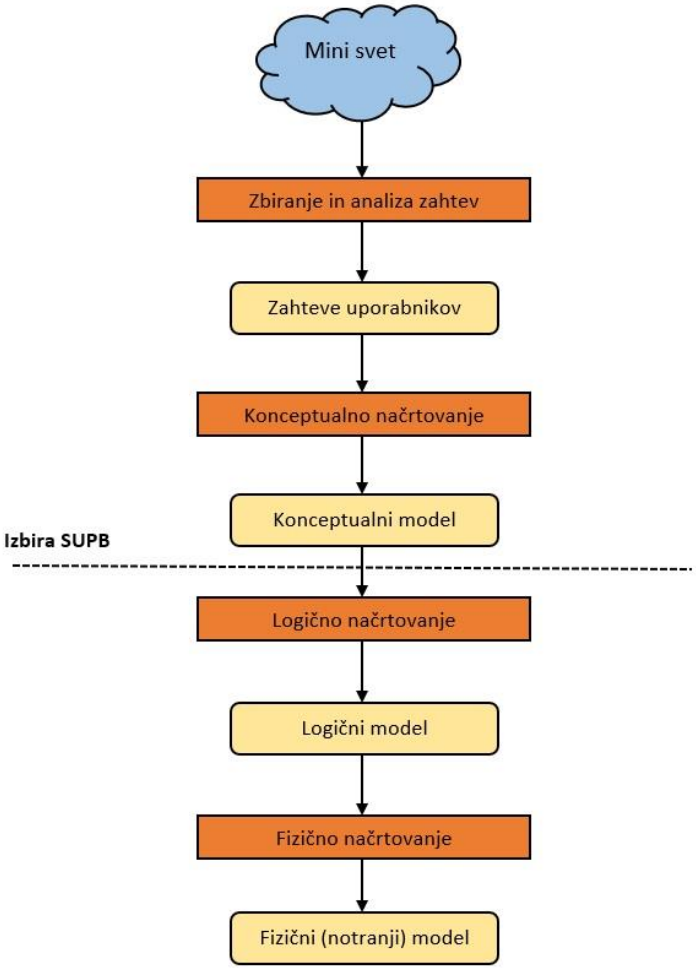
- nepoznavanje področja,
- upoštevanje vseh pravil in izjem realnega sveta in
- velikost oziroma kompleksnost načrta podatkovne baze.

Kot smo omenili, se s podatkovnimi bazami srečujemo povsod, kjer je treba podatke shranjevati na urejen način. Področja, kjer na urejen način shranjujemo podatke, so zelo raznolika. V diplomski nalogi bomo načrtovanje podatkovne baze prikazali na poenostavljenem zgledu knjižnice. Načrtovalec velikokrat ne pozna problemskega področja, zato se mora najprej dobro seznaniti in dobro spoznati domeno problema. Šele nato lahko začne z reševanjem dejanskega problema.

V realnem svetu obstajajo tako pravila kot tudi izjeme. Tako v našem primeru knjižnice obstaja pravilo, da lahko član sočasno opravi le 3 rezervacije. Ker sta realni svet in njegovo okolje dinamična sistema, se pogosto spreminjata. Zato morajo načrtovalci pri svojem delu upoštevati vsa pravila in vse izjeme. Zagotoviti pa morajo tudi dovolj fleksibilen model podatkovne baze, ki bo prilagojen morebitnim prihodnjim spremembam. Spreminjanje že izdelane podatkovne baze, ki temelji na slabem oziroma površnemu načrtovanju, je namreč vse prej kot preprosto. Velikokrat je lahko problem, ki ga načrtovalec rešuje, zelo velik. Zato so načrti podatkovne baze pogosto zelo kompleksni in za načrtovalca težje obvladljivi.

Osnovno pravilo dobrega in uspešnega načrtovanja je, da morajo načrtovalci tesno sodelovati z uporabniki podatkovne baze.

Postopek načrtovanja in izdelave podatkovne baze poteka po fazah, kar prikazuje Slika 1. Kot je razvidno iz slike, sta prvi dve fazi neodvisni od izbire SUPB, medtem ko sta preostali dve odvisni od izbranega SUPB.



Slika 1: Faze načrtovanja podatkovne baze (povzeto po (Remžgar, 2012))

Faze načrtovanja podatkovne baze so:

1. **Zbiranje in analiza zahtev**
   V prvi fazi načrtovanja podatkovne baze spoznavamo del realnega sveta (»mini svet«), ki ga bomo predstavili z modelom. Zbirati in analizirati moramo informacije o podatkih, ki jih bomo hranili v podatkovni bazi. Poznati in razumeti moramo, katere podatke bomo hranili v podatkovni bazi, kako so ti podatki povezani med seboj in katere operacije se izvajajo nad podatki. Zelo pomembno je tudi vedenje, kako bodo uporabniki podatkovno bazo uporabljali. Zato je v tej fazi tesno sodelovanje načrtovalcev z uporabniki ključnega pomena. Vendar pa morajo načrtovalci in uporabniki govoriti isti jezik, v nasprotnem primeru zelo hitro pride do razlik med željami uporabnikov in razumevanjem načrtovalcev. Želje in pričakovanja uporabnikov se z razumevanjem načrtovalcev običajno ne uskladijo že na prvem srečanju. Zato je potrebno večkratno srečanje načrtovalcev z uporabniki podatkovne baze. Načrtovalci nato na podlagi zahtev in želj uporabnikov pripravijo dokument, imenovan analiza zahtev. Dokument vsebuje natančno opisane zahteve uporabnikov in predstavlja temelj za nadaljnji postopek načrtovanja podatkovne baze.
2. **Konceptualno načrtovanje**
   Konceptualno načrtovanje je predstavitev uporabniških zahtev s pomočjo konceptualnega modela. Cilj konceptualnega načrtovanja podatkovne baze je izdelava enostavnega konceptualnega modela, ki bo razumljiv tudi uporabnikom, ki nimajo tehničnega znanja o

podatkovnih bazah. Konceptualni model je predstavljen kot strnjena predstavitev zahtev uporabnikov. V osnovi vključuje objekte, njihove lastnosti in povezave med objekti. Poznamo več vrst konceptualnih modelov, npr. entitetno-relacijski model, ORM-model, UML-objektni model, UML-razredni model. Izgradnja konceptualnega modela poteka po določenih pravilih, ki jih določa posamezni konceptualni model. V diplomski nalogi bomo pri izdelavi konceptualnega modela sledili pravilom oziroma korakom entitetno-relacijskega modela. Konceptualno načrtovanje je v primerjavi z logičnim in fizičnim načrtovanjem, ki mu sledita, zdaleč najbolj kritično. Posledice dobrega in slabega načrta v začetni fazi se prenesejo tudi v vse nadaljnje faze. Dober model, ki bo hkrati primeren za nadaljnji razvoj, lahko zagotovimo le s tesnim sodelovanjem z uporabniki. Zato se je treba z uporabniki srečati večkrat.

3. **Logično načrtovanje**

   Pogoj za prehod v fazo logičnega načrtovanja je izbira ustreznega sistema za upravljanje podatkovni baz (SUPB). Izbiro ciljnega SUPB izvede načrtovalec v sodelovanju z uporabniki. Cilj logičnega načrtovanja podatkovne baze je izdelava logičnega modela podatkov, ki upošteva ciljno podatkovno bazo. V našem primeru je to relacijska podatkovna baza. V tej fazi se izvede preslikava konceptualnega modela v logični model, ki ga podpira izbrani SUPB. Ker smo se v diplomski nalogi omejili na relacijske podatkovne baze in pri konceptualnem načrtovanju na entitetno-relacijski model, bomo pri logičnem načrtovanju obravnavali pretvorbo entitetno-relacijskega modela v relacijski podatkovni model. Pred prehodom na naslednjo fazo načrtovanja je treba opraviti podroben pregled dobljenega logičnega modela in preveriti, ali so podatki normalizirani. Treba je torej ugotoviti, če obstaja nepotrebno podvajanje podatkov in če so prisotne funkcionalne odvisnosti med atributi. Normalizacija je postopek, ki pomaga pri doseganju boljše kakovosti načrta podatkovne baze. Vsako relacijo v logičnem modelu vodimo skozi zaporedje »testov«. Z vsakim »testom« preverimo, ali so izpolnjeni določeni pogoji, ki jih zahteva posamezna normalna oblika. Pogoji se nanašajo na razmerja med atributi. Če relacija ustreza določenim pogojem, pravimo, da je v določeni normalni obliki. Če pa ne zadošča pogojem, jo razdelimo na več relacij tako, da dobljene relacije ustrezajo predpisanim pogojem.

4. **Fizično načrtovanje**

   Logičnemu načrtovanju sledi načrtovanje fizične podatkovne baze. Logični načrt podatkovne baze, ki obsega množico normaliziranih relacij, je treba »spraviti v življenje«. V ta namen moramo ustvariti fizični model, ki vsebuje vse z logičnim načrtom predvidene relacije (tabele), opremljene z indeksi, razpršilnimi strukturami in integritetnimi omejitvami (Mohorič, 1997). S tem omogočimo hiter dostop do posameznih podatkov v relacijah in skrbimo za njihovo konsistentnost. Torej v tej fazi na podlagi logičnega modela, dobljenega v prejšnji fazi, ustvarimo fizični model podatkovne baze. Ta obsega prazne objekte (tabele), ki jih nato napolnimo s podatki. Prav tako določimo, kateri uporabnik oziroma skupina uporabnikov lahko dostopa do katerih delov podatkovne baze in katere operacije lahko izvaja nad posamezno tabelo. Nato je treba preveriti še ustreznost in zmogljivost podatkovne baze glede na potrebe in zahteve uporabnikov. Navadno uporabniki podajo tudi zahteve, kot so hitrost poizvedovanja, hitrost posodobitve tabel ali koliko transakcij se mora izvesti v sekundi. Zato moramo preveriti odzivni čas in zmogljivost ciljne podatkovne baze, za kar pa je potrebno dobro poznavanje delovanja ciljnega SUPB-a.

2.4 Modeliranje podatkov

Modeliranje je nepogrešljiv proces pri načrtovanju podatkovnih baz. Namen modeliranja je, da od problemov realnega sveta preidemo do predstavitev z bazo podatkov. Modeliranje sveta s podatkovno bazo je odvisno od potreb in zahtev uporabnikov. Pri modeliranju gre za snovanje, izdelavo in uporabo nekega modela. Model je slika oziroma grafična predstavitev realnega sveta, ki odraža predstavo ali pogled na stvarnost (Dolžan, 2001). Omogoča nam boljšo predstavitev, opredelitev in s tem razumevanje obravnavanega problema.
Modeliranje podatkov je proces, s katerim ugotovimo povezave med podatki, ki jih moramo upoštevati pri zasnovi nekega modela (Gradišar, in drugi, 1994). Cilj modeliranja podatkov je torej oblikovanje modela podatkov. **Model podatkov** je abstraktna, vendar lahko razumljiva predstava potrebnih podatkov neke organizacije oziroma njenega dela (Gradišar, in drugi, 1994).

Pri načrtovanju podatkovne baze za predstavitev strukture podatkov v konceptualni in logični fazi načrtovanja podatkovne baze uporabljamo podatkovne modele. Strukture podatkov vključujejo podatkovne objekte in povezave med objekti. Bistvo izgradnje modela je, da opredelimo, kateri podatki so potrebni in kako bodo organizirani. Če želimo z modelom doseči učinkovit rezultat, mora biti model čim bolj preprost. Zato je pomembno, da prikazuje samo tiste podatke, ki so za posamezni problem pomembni, in zanemari nepomembne. Struktura podatkov, ki jo model prikazuje, mora biti jasna tudi uporabnikom podatkovne baze. Torej je modeliranje pomembno tako za načrtovalce podatkovnih baz kot tudi za uporabnike. Na podlagi preglednega in preprostega modela lahko načrtovalci v sodelovanju z uporabniki hitro odkrijejo in odpravijo morebitne pomanjkljivosti in napake. Le na ta način lahko zagotovimo dober model, ki hkrati predstavlja tudi pomemben vir pri vzdrževanju in morebitnem spreminjanju podatkovne baze.

Tako kot na drugih področji, kjer uporabljajo modele, so se tudi pri podatkovnih bazah uveljavila določena pravila in tehnike ali metode za modeliranje. Katero metodo bomo uporabili, je v veliki meri odvisno od vrste problema, ki ga rešujemo, in od ustreznih izkušenj načrtovalca. Prav izbira ustrezne metode včasih vpliva na uspeh načrtovanja in razvoja podatkovne baze. S pomočjo modelirnih tehnik je mogoče določene faze v razvoju podatkovne baze avtomatizirati s pomočjo orodij CASE[2]. Orodja CASE bomo podrobneje predstavili v naslednjem razdelku.

Za načrtovanje podatkovne baze uporabljamo različne metode. V diplomski nalogi se bomo omejili na relacijske podatkovne baze, zato bomo predstavili eno od metod za načrtovanje le-teh, in sicer metodo za načrtovanje z entitetno-relacijskim modelom.

Kot smo omenili, bomo načrtovanje in modeliranje podatkovne baze predstavili na primeru načrtovanja poenostavljene podatkovne baze knjižnice. Zahteve, ki jih mora podatkovna baza knjižnice zadovoljiti, strnimo v kratek opis.
Knjižnica izposoja knjige, o katerih hranimo naslednje podatke: naslov, avtor, ključne besede, založnik in leto izida. Vsaka knjiga je označena z mednarodno standardno knjižno številko (ISBN). ISBN-številko sestavlja največ 13 števk (najprej je ISBN-številko sestavljalo 10 števk, od leta 2007 pa jo sestavlja 13 števk). Shematsko to zapišimo s ***KNJIGA (ISBN, Naslov, Avtor, Ključne besede, Založnik, Leto izida).***

Poglejmo si še konkreten primer opisa:

KNJIGA (9616046128, Podatkovne baze, Tomaž Mohorič, baze, BI-TIM, 2002)

[2] CASE: Computer-Aided Software engineering.

Knjiga ima lahko enega ali več avtorjev. Enako velja tudi za ključne besede. Torej je mogoč tudi sledeč opis:

KNJIGA (9789612400675, Temelji elektronskega poslovanja, {Andrej Kovačič, Aleš Groznik, Miroslav Ribič}, {poslovni proces, poslovne informacije, trženje}, Ekonomska fakulteta, 2009)

Knjige so na voljo v enem ali več izvodih. Za vsak izvod hranimo signaturo, status (prost, izposojen) in datum vrnitve. Signatura je število, ki nam pove, na katerem mestu (oddelku) v knjižnici najdemo posamezen izvod knjige. Če je knjiga prosta, datuma vrnitve ni. Shema opisa je torej ***IZVOD (Signatura, Status, Datum vrnitve),*** primera tovrstnih podatkov pa

IZVOD (004, Izposojen, 27.03.2016) in
IZVOD (087, Prost, Null)

Pri zadnjem primeru smo z Null označili, da podatka nimamo.

Izvode knjig si izposojajo člani, o katerih knjižnica hrani naslednje podatke: ime, priimek, ulica, poštna številka in potek članstva. Vsakemu članu je ob prvem vpisu dodeljena članska številka, ki ga enolično določa.

ČLAN (Članska številka, Ime, Priimek, Ulica, Poštna številka, Potek članstva)
ČLAN (0020868, Maja, Novak, Srebrničeva 10, 5000 Nova Gorica, 10.02.2017)

Pri tem smo predpostavili, da je ulica nedeljiva celota. Problem, ki ga modeliramo, je torej tak, da ne potrebujemo torej samo imena ulice ali samo hišne številke. Enako velja tudi za poštno številko. Če pa bi želeli, da npr. izvajamo tudi povpraševanja v obliki »v kateri ulici živi največ članov« ali omogočili, da bi izvedli npr. preimenovanje neke ulice v drugo (pri ohranjenem številčenju), pa bi relacijo ČLAN spremenili v:

ČLAN (Članska številka, Ime, Priimek, Ulica, Hišna številka, Poštna številka, Kraj, Potek članstva)
ČLAN (0020868, Maja, Novak, Srebrničeva, 10, 5000, Nova Gorica, 10.02.2017)

Če je izvod, ki si ga član želi izposoditi, izposojen, lahko član opravi rezervacijo, na podlagi katere si zagotovi želeno knjigo. Pri rezervaciji beležimo datum rezervacije. Vsak član si lahko izposodi več izvodov in lahko opravi več rezervacij. Poglejmo primer podatkov o rezervaciji:

REZERVACIJA (Datum rezervacije)
REZERVACIJA (08.03.2016)

Na podlagi opisa sistema bomo v nadaljevanju izdelali konceptualni, logični in fizični model. Pri tem bomo uporabili različna orodja, katerih uporabo bomo tudi opisali.

2.5 Orodja CASE

Proces načrtovanja in izdelave podatkovne baze si danes težko predstavljamo brez uporabe orodja. Uvedba avtomatizacije v proces načrtovanja podatkovnih baz je načrtovalcem zelo olajšala delo. Pred tem so načrtovalci proces načrtovanja in izdelave podatkovne baze izvajali ročno in se zanašali predvsem na svoje znanje in izkušnje. Orodjem, ki omogočajo računalniško podprto načrtovanje in razvoj podatkovne baze, pravimo CASE-orodja. Po (Projekt Colos) povzemimo definicijo, ki pravi, da so orodja CASE (Computer Aided Software Engineering) tista programska orodja, ki omogočajo avtomatizacijo procesa razvoja in vzdrževanja programske opreme in pripadajoče dokumentacije sistemov.

CASE-orodja združujejo vrsto funkcij (Projekt Colos), kot so risanje diagramov, oblikovalniki diagramov, sestavljanje podatkovnih slovarjev, preverjanje specifikacij diagramov, generiranje kode in dokumentacije. Z oblikovalniki ustvarjamo obrazce, poročila, specifikacije itd. V podatkovnem slovarju so opredeljeni podatkovni elementi diagrama. Preverjanje specifikacij omogoča samodejno odkrivanje nepopolnih in sintaktično nepravilnih in nedoslednih specifikacij. Generatorji kode omogočajo samodejno generiranje kode na osnovi diagrama. Potrebno tehnično in uporabniško dokumentacijo pa pridobimo z generatorji dokumentacije.

CASE-orodja omogočajo, da je razvoj podatkovne baze hitrejši. Hkrati v posamezni fazi načrtovanja podatkovnih baz pomagajo pri odkrivanju morebitnih napak, še preden preidemo v naslednjo fazo. Namen CASE-orodij je torej, da v čim večji meri avtomatizirajo razvojne faze podatkovne baze. Načrtovalcu omogočajo izris konceptualnega diagrama v neki specifični notaciji. Dobljeni konceptualni model znajo nato s posebnim algoritmom preslikati v logični podatkovni model v nekaterih specifičnih SUPB-ih.

Nekatera CASE-orodja pa omogočajo tudi izvedbo normalizacije. S pomočjo funkcionalnih odvisnosti znajo izvesti dekompozicijo relacij tako, da ustrezajo določeni normalni obliki. Dekompozicija je postopek, ki razdeli eno relacijo na dve ali več manjših relacij.

Pri zadnjem koraku načrtovanja podatkovne baze pa orodja na osnovi grafične predstavitve logičnega podatkovnega modela omogočajo avtomatično generiranje SQL-kode za izbrani SUPB. S to kodo nato ustvarimo fizični model.

Za načrtovanje podatkovne baze imamo na voljo različna CASE-orodja. Pogosto uporabljena CASE-orodja so:

- Case Studio 2 (demo verzija dostopna na naslovu: http://download.cnet.com/Case-Studio-2/3001-10254_4-10517119.html?hlndr=1),
- PowerDesigner (http://go.sap.com/product/data-mgmt/powerdesigner-data-modeling-tools.html),
- DBDesigner (http://fabforce.net/dbdesigner4/),
- Oracle Designer (http://www.oracle.com/technetwork/developer-tools/designer/overview/index-082236.html),
- MySQL Workbench (https://www.mysql.com/products/workbench/),
- DeZing for Databases (http://www.datanamic.com/dezign/).

V diplomski nalogi si bomo ogledali uporabo orodja DBDesigner4. Z omenjenim orodjem bomo v nadaljevanju izdelali logični podatkovni model.

3 KONCEPTUALNI MODEL

Konceptualnemu modelu bi lahko rekli pojmovni model. Vendar je v literaturi povsod uporabljen izraz konceptualni model, zato bomo ta izraz uporabljali tudi v diplomski nalogi.
Konceptualno načrtovanje je opredelitev in predstavitev uporabnikovih podatkovnih potreb oziroma zahtev s pomočjo konceptualnega modela. Konceptualni model vsebuje opis podatkov in povezav med podatki ter predstavlja osnovo za ustvarjanje podatkovne baze (Mohorič, 1997). Dober konceptualni model lahko zgradimo zgolj ob tesnem sodelovanju z uporabniki podatkovne baze. Omogoča in olajša komunikacijo načrtovalcev z uporabniki. S tem, da uporabniki točno opišejo zahteve, dosežemo, da je model kakovostnejši in razumljivejši in da načrtovanje poteka hitreje. Konceptualni model zagotavlja boljšo kakovost končnega izdelka (v našem primeru podatkovne baze). Konceptualni model, predstavljen s konceptualno shemo, služi tudi kot del dokumentacije projekta, saj natančno opisuje vsebino podatkovne baze.

Podatkovna baza kot model sveta ni statična in enkrat za vselej zgrajena. Občasno jo je treba dopolnjevati in spreminjati v skladu s spreminjajočimi se zahtevami okolja (Mohorič, 1997). Konceptualni model ima pri dopolnitvah in spremembah podatkovne baze pomembno vlogo. Vse potrebne dopolnitve in spremembe najprej preverimo in izvedemo nad konceptualnim modelom, šele nato jih uveljavimo v dejanski podatkovni bazi. Torej se vedno, ko je treba podatkovno bazo dopolniti ali npr. zaradi menjave programske opreme zgraditi na novo, vračamo h konceptualnemu modelu.

Zelo pomemben del izgradnje konceptualnega modela je pogovor z uporabniki. Kot smo omenili že prej, morajo uporabniki in načrtovalci govoriti »isti« jezik. Če ne bodo govorili »istega« jezika, lahko zelo hitro pride do večjih razhajanj med dejanskimi in realiziranimi zahtevami. Pogovor z uporabniki ni enkraten proces. Z uporabniki se moramo večkrat srečati – tako med procesom izdelave konceptualnega modela kot tudi skozi vse nadaljnje faze načrtovanja podatkovne baze. Napake, ki jih naredimo pri konceptualnem modelu, prenesemo tudi na vse nadaljnje faze razvoja podatkovne baze, zato je pomembno, da smo večkrat v stiku z uporabniki podatkovne baze. Le na ta način bomo lahko odpravili vsa morebitna neskladja in napake ter zagotovili dober konceptualni model.
Izhodišče konceptualnega načrtovanja so uporabniške zahteve. Te so lahko podane na različne načine. Trije najpogostejši načini specifikacij uporabniških zahtev so:

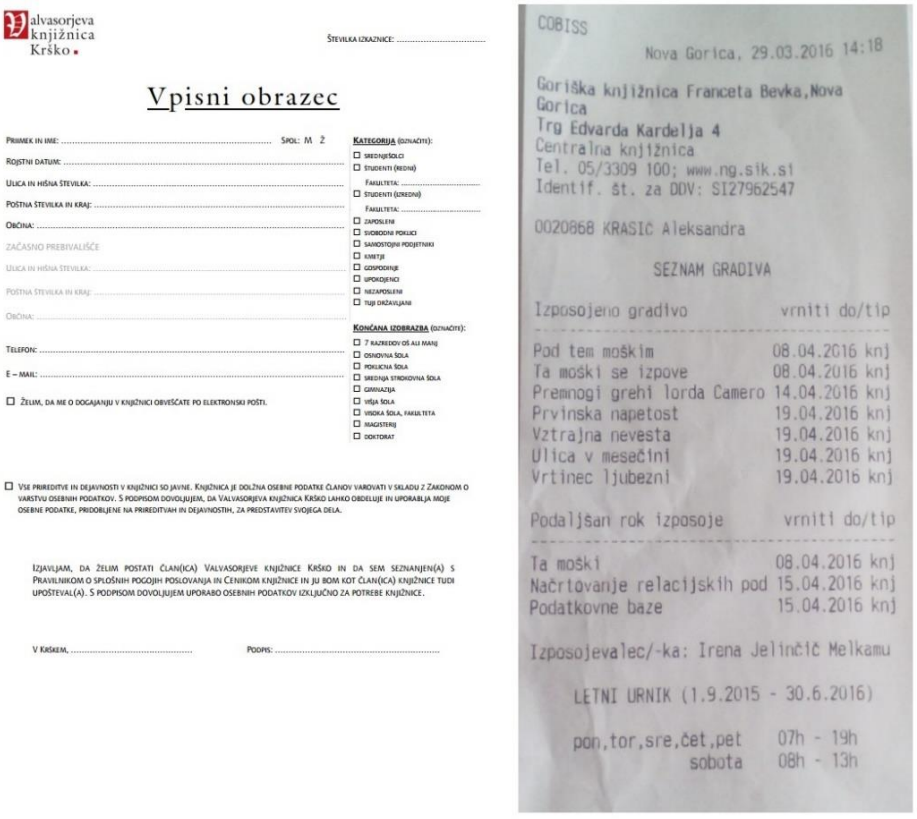
- opis problema v naravnem jeziku (opis v ustni ali pisni obliki),
- obrazci (dokumentacija) in
- datotečni formati (datotečna struktura).

Najobičajnejši način predstavitve problema oz. uporabniških zahtev je opis v naravnem jeziku. Uporabniki na papir zapišejo (ali povedo ustno) vse svoje zahteve in želje, ki naj bi jih bodoča podatkovna baza vsebovala. V okoljih (npr. v administrativnem okolju, v univerzitetnem okolju), kjer se veliko uporabljajo različni obrazci (npr. obrazec za odmero dohodnine, obvestilo o učnem uspehu), se lahko pri načrtovanju podatkovne baze naslonimo na podatke, ki so na posameznih obrazcih. V nekaterih naprednejših okoljih, kjer so podatke že shranjevali v računalniško podprtih datotekah, si lahko pri načrtovanju podatkovne baze pomagamo z opisi datotečnih zapisov oziroma z datotečno strukturo.

Na slikah Slika 2, Slika 3 in Slika 4 vidimo primer uporabniških zahtev (v različnih oblikah) za izdelavo podatkovne baze knjižnice.

V podatkovni bazi knjižnice so shranjeni podatki o knjigah in članih. Knjige so predstavljene z naslovom, avtorjem, zvrstjo (učbenik, priročnik, roman, kriminalka, znanstvena fantastika, zgodovina,...), ključnimi besedami, založnikom in letom izida. Vsaka knjiga je označena z mednarodno standardno knjižno številko (tako imenovana ISBN številka). Knjiga lahko ima več avtorjev in ključnih besed. Knjige so na voljo v enem ali več izvodih. Za vsak izvod se v bazi hrani status (prosta, izposojena, rezervirana). V primeru, da je knjiga izposojena, hranimo tudi datum vrnitve. Izvode knjig si izposojajo člani. O članih hranimo naslednje podatke: ime, priimek, naslov stalnega bivališča, spol, starost in status (zaposlen, brezposeln, dijak, študent, upokojenec, predšolski otrok). Poleg navedenih podatkov o članih hranimo še telefonske številke preko katerih jih knjižnica kontaktira. Vsak član si lahko izposodi več izvodov. V kolikor je izvod, ki si ga član želi izposojen, lahko opravi rezervacijo. Na podlagi rezervacije si član zagotovi željeno knjigo. Za vsako rezervacijo hranimo datum rezervacije, ime in priimek člana in naslov knjige. Član lahko opravi največ tri rezervacije in si sočasno lahko izposodi največ deset izvodov. Dodatne zahteve uporabnikov: vpogled v podatke koliko časa je nekdo član, vpogled v podatke o knjigah na podlagi naslova, avtorja ali zvrsti, vpogled v podatke koliko rezervacij ima posamezna knjiga.

Slika 2: Primer zapisanih uporabniških zahtev



Valvasorjeva knjižnica Krško

ŠTEVILKA IZKAZNICE: .....

Vpisni obrazec

PRIIMEK IN IME: ..... SPOL: M Ž

ROJSTNI DATUM: .....

ULICA IN HIŠNA ŠTEVILKA: .....

POŠTNA ŠTEVILKA IN KRAJ: .....

OBČINA: .....

ZAČASNO PREBIVALIŠČE

ULICA IN HIŠNA ŠTEVILKA: .....

POŠTNA ŠTEVILKA IN KRAJ: .....

OBČINA: .....

TELEFON: .....

E – MAIL: .....

☐ ŽELIM, DA ME O DOGAJANJU V KNJIŽNICI OBVEŠČATE PO ELEKTRONSKI POŠTI.

KATEGORIJA (OZNAČITI):

☐ SREDNJEŠOLCI
☐ ŠTUDENTI (REDNI)
FAKULTETA: .....
☐ ŠTUDENTI (IZREDNI)
FAKULTETA: .....
☐ ZAPOSLENI
☐ SVOBODNI POKLICI
☐ SAMOSTOJNI PODJETNIKI
☐ KMETJE
☐ GOSPODINJE
☐ UPOKOJENCI
☐ NEZAPOSLENI
☐ TUJI DRŽAVLJANI

KONČANA IZOBRAZBA (OZNAČITI):

☐ 7 RAZREDOV OŠ ALI MANJ
☐ OSNOVNA ŠOLA
☐ POKLICNA ŠOLA
☐ SREDNJA STROKOVNA ŠOLA
☐ GIMNAZIJA
☐ VIŠJA ŠOLA
☐ VISOKA ŠOLA, FAKULTETA
☐ MAGISTERIJ
☐ DOKTORAT

☐ VSE PRIREDITVE IN DEJAVNOSTI V KNJIŽNICI SO JAVNE. KNJIŽNICA JE DOLŽNA OSEBNE PODATKE ČLANOV VAROVATI V SKLADU Z ZAKONOM O VARSTVU OSEBNIH PODATKOV. S PODPISOM DOVOLJUJEM, DA VALVASORJEVA KNJIŽNICA KRŠKO LAHKO OBDELUJE IN UPORABLJA MOJE OSEBNE PODATKE, PRIDOBLJENE NA PRIREDITVAH IN DEJAVNOSTIH, ZA PREDSTAVITEV SVOJEGA DELA.

IZJAVLJAM, DA ŽELIM POSTATI ČLAN(ICA) VALVASORJEVE KNJIŽNICE KRŠKO IN DA SEM SEZNANJEN(A) S PRAVILNIKOM O SPLOŠNIH POGOJIH POSLOVANJA IN CENIKOM KNJIŽNICE IN JU BOM KOT ČLAN(ICA) KNJIŽNICE TUDI UPOŠTEVAL(A). S PODPISOM DOVOLJUJEM UPORABO OSEBNIH PODATKOV IZKLJUČNO ZA POTREBE KNJIŽNICE.

V KRŠKEM, ... PODPIS: .....

COBISS

Nova Gorica, 29.03.2016 14:18

Goriška knjižnica Franceta Bevka, Nova Gorica
Trg Edvarda Kardelja 4
Centralna knjižnica
Tel. 05/3309 100; www.ng.sik.si
Identif. št. za DDV: SI27962547

0020868 KRASIĆ Aleksandra

SEZNAM GRADIVA

Izposojeno gradivo	vrniti do/tip
Pod tem moškim	08.04.2016 knj
Ta moški se izpove	08.04.2016 knj
Premnogi grehi lorda Camero	14.04.2016 knj
Prvinska napetost	19.04.2016 knj
Vztrajna nevesta	19.04.2016 knj
Ulica v mesečini	19.04.2016 knj
Vrtinec ljubezni	19.04.2016 knj

Podaljšani rok izposoje	vrniti do/tip
Ta moški	08.04.2016 knj
Načrtovanje relacijskih pod	15.04.2016 knj
Podatkovne baze	15.04.2016 knj

Izposojevalec/-ka: Irena Jelinčič Melkamu

LETNI URNIK (1.9.2015 - 30.6.2016)

pon,tor,sre,čet,pet 07h - 19h
sobota 08h - 13h

Slika 3: Primer obrazcev

```
type član = record
                štČlana: intiger;
                ime: string;
                priimek: string;
                naslov: string;
            end;

type knjiga = record
                štKnjige: intiger;
                naslov: string;
                avtor: string;
                založnik: string;
                letoIzida: string;
            end;

type izvod = record
                štIzvoda: intiger;
                status: string;
                štKnjige: intiger;
                štČlana: intiger;
            end;

type rezervacija = record
                štRezervacije: intiger;
                datumRezervacije: string;
                ŠtKnjige: intiger;
                ŠtČlana: intiger;
            end;
```

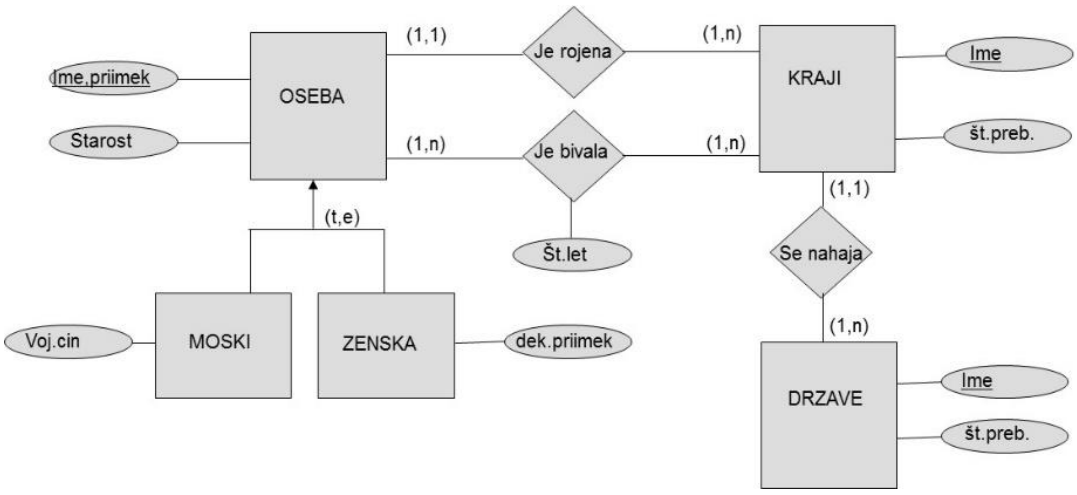
Slika 4: Primer datotečnih zapisov

Konceptualni model je univerzalen, saj nam lahko en model služi za gradnjo različnih podatkovnih baz in ni odvisen od sistema za upravljanje podatkovnih baz.

Obstaja več vrst konceptualnih modelov. V nadaljevanju se bomo osredotočili na entitetno-relacijski model, ki je v praksi eden najpogostejše uporabljenih modelov.

3.1 Entitetno-relacijski model

Entitetno-relacijski model, krajše ER-model, je v praksi ena izmed najpogosteje uporabljenih tehnik za predstavitev konceptualnih podatkovnih modelov. Na sliki Slika 5 vidimo primer takega modela.



Slika 5: Primer ER-modela (Vir: https://ucilnica.fmf.uni-lj.si/mod/resource/view.php?id=17602)

Razvil ga je Peter Pin-Shan Chen leta 1976 (Wikipedia; Chen, 1976). Je relativno preprost za razumevanje. Z njim na poljuden in razumljiv način predstavimo strukturo podatkov. ER-model je grafično predstavljen v obliki entitetno-relacijskega diagrama. Sestavljajo ga pravokotniki, ki določajo tipe podatkov (entitetni tipi) s svojimi atributi (predstavljeni z ovali), ki jih povezujejo razmerja (poimenovana znotraj romba), ki predstavljajo odnose med podatkovnimi tipi. Grafična predstavitev, ki jo imenujemo ER-diagram, omogoča boljši pregled nad strukturo podatkov. Kot vidimo na sliki Slika 5, ER-model ne vključuje podrobnosti, zato ga razumejo tudi uporabniki, ki nimajo tehničnega znanja o podatkovnih bazah. Nazorno prikazuje preslikavo med realnim svetom in podatkovnim modelom. To omogoča dobro komunikacijo z uporabniki in lažje vključevanje le-teh v proces izdelave podatkovne baze. Tako lahko iz diagrama na sliki Slika 5 razberemo kar precej lastnosti problema, ki ga modeliramo, čeprav ne poznamo podrobnosti. Zlahka lahko razberemo, da imamo osebe, o katerih zbiramo naslednje podatke: ime, priimek, starost, kraj bivanja in kraj rojstva. Osebe so lahko moškega in ženskega spola. Za moške nas zanima še vojaški čin, za ženske pa dekliški priimek. Za vsak kraj rojstva osebe in kraj bivanja osebe nas zanima tudi država, v kateri se kraj nahaja. Poleg imena kraja/države rojstva in bivanja osebe zbiramo tudi podatke o številu prebivalcev kraja/države. Razberemo tudi, da zbiramo tudi podatke o tem, koliko let je oseba bivala v posameznem kraju.

ER-model je neodvisen od sistema za upravljanje podatkovnih baz. Možno ga je pretvoriti v različne podatkovne modele, kot so relacijski, mrežni in hierarhični podatkovni model.

3.2 Orodja za izdelavo ER-diagramov

V primeru preprostejših in manjših ER-modelov lahko ER-diagram izdelamo ročno, torej s pomočjo papirja in svinčnika. Pri izdelavi kompleksnejših in večjih ER-modelov je ročno risanje diagramov nepraktično in težko obvladljivo, saj model vsebuje številne entitetne tipe (o njih bo govora v nadaljevanju) in razmerja. Zato je bolje ER-diagram izdelati s pomočjo CASE-orodij. Kaj so CASE-orodja, smo povedali že v drugem poglavju (Načrtovanje in modeliranje relacijske podatkovne baze, podpoglavje CASE-orodja). CASE-orodja vključujejo orodja, ki načrtovalcu omogočajo izris konceptualnega modela.
Orodja, ki omogočajo izris ER-diagramov, lahko razdelimo v dve skupini. V prvi so orodja, ki podpirajo le risanje ER-diagramov, in v drugi orodja, kjer je risanje ER-diagrama le ena od faz načrtovanja podatkovne baze.
Orodja, ki podpirajo le risanje ER-diagramov, omogočajo le konceptualno modeliranje. Ne omogočajo preslikave ER-modela v nadaljnjo fazo načrtovanja podatkovne baze. Primer tovrstnih orodji so:

- DIA Diagram Editor (https://wiki.gnome.org/Apps/Dia/),
- Creately (http://creately.com/),
- Draw.io (https://www.draw.io/),
- MS Visio (https://products.office.com/sl-si/Visio),
- Tiny Modeler (http://tinymodeler.com/).

Večina teh orodij ne omogoča le risanja ER-diagramov, ampak so namenjena pripravi različnih diagramov, denimo algoritemskih za prikaz strukture podjetja, omrežnih diagramov za prikaz omrežne strukture podjetja, organizacijskih diagramov za grafično ponazoritev organizacijske strukture itd.

Orodja, kjer je priprava ER-modela le ena od faz načrtovanja podatkovne baze, omogočajo tudi preslikavo ER-modela v nadaljnjo fazo razvoja podatkovne baze. Ta orodja vsebujejo algoritem, ki lahko konceptualno model (ER-model) pretvorijo oziroma preslikajo v podatkovni model v nekaterih specifičnih SUPB. Med tovrstna orodja uvrščamo orodja, kot so:

- DBDesigner (http://fabforce.net/dbdesigner4/),
- Case Studio 2 (demo verzija dostopna na: http://download.cnet.com/Case-Studio-2/3001-10254_4-10517119.html?hlndr=1),
- Open Model Sphere (http://www.modelsphere.com/org/),
- TOAD Data Modeler (http://www.casestudio.com/enu/products.aspx).

V diplomski nalogi bomo za izdelavo ER-diagrama uporabili orodje DIA Diagram Editor (v nadaljevanju DIA). Izbrali smo ga, ker je preprost za uporabo, a vseeno dovolj zmogljiv. Večina ER-diagramov v diplomski nalogi je pripravljenih prav s tem orodjem.

3.3 Osnovni gradniki ER-modela

Pri gradnji ER-modela se bomo srečali z določenimi pojmi. Med pomembnejšimi so entiteta in entitetni tip, atribut, razmerje med entitetnimi tipi, ključ entitetnega tipa (identifikator), šibki in močni entitetni tip ter generalizacija in specializacija. Na kratko jih bomo predstavili v nadaljevanju.

3.3.1 Entiteta in entitetni tip

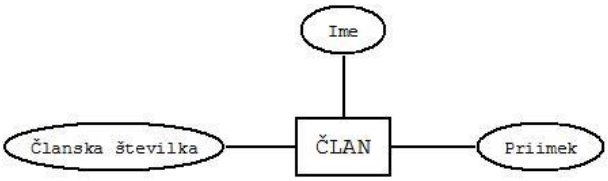
Entitete so posamezni objekti iz realnega sveta, o katerih zbiramo, obdelujemo in hranimo podatke ter jih lahko ločimo od drugih objektov (Projekt Colos). Entiteta je lahko oseba (npr. profesor, študent), objekt (npr. hiša, šola), prostor (npr. razred, pisarna), predmet (npr. avto, knjiga), dogodek (npr. koncert, prireditev), koncept (npr. predmet na urniku, projekt) itd. Vsaka entiteta mora imeti neko lastnost (ali skupek lastnosti), po kateri jo lahko enolično opredelimo. Ta lastnost mora biti med tistimi podatki, ki jih zberemo za vse tovrstne objekte, saj bomo na ta način ločili med posameznimi objekti.

O entitetah obstaja določena predstava, predstava o tem, kaj so in kakšne lastnosti posedujejo ali bi jih utegnile posredovati. Na osnovi takih predstav je možno entitete razvrstiti. Za entitete, ki ustrezajo določeni predstavi, pravimo, da pripadajo določenemu entitetnemu tipu (Mohorič, 1997). Na primer, vsak posamezen član knjižnice je entiteta. O vsakem članu zbiramo enake podatke, npr. ime, priimek, naslov, datum včlanitve. Ti podatki predstavljajo lastnosti, s katerimi opišemo posameznega člana. Na podlagi omenjenih lastnosti pravimo, da člani knjižnice pripadajo entitetnemu tipu »ČLAN«. Entitetni tip je torej množica entitet, ki imajo enake lastnosti (atribute).
Vsakemu entitetnemu tipu moramo določiti ime. Imena entitetnih tipov pišemo z velikimi črkami. Običajno je to samostalnik v ednini, ki je kratek, hkrati pa entiteto dovolj dobro in nedvoumno opisuje oziroma predstavlja njeno vlogo in pomen. Entitetni tip je v modelu ER predstavljen s pravokotnikom z vpisanim imenom tipa.
Primer entitetnega tipa je predstavljen na sliki Slika 6. Entitetni tip »ČLAN« predstavlja člane knjižnice. Vsak posamezni član knjižnice je entiteta, ki je tipa »ČLAN«. To pomeni, da o vseh članih zbiramo iste podatke. Seveda se vrednosti teh podatkov pri posameznih članih razlikujejo. Tako ima na primer vsak član svoje ime, priimek, datum včlanitve, vpisno številko … Dva člana imata lahko enaki vrednosti določene lastnosti (denimo ime, morda tudi priimek ali datum včlanitve), spet v drugih lastnostih pa se razlikujeta. Spomnimo se, da mora imeti vsaka entiteta (torej vsak član) neko lastnost, po kateri lahko razpoznamo (enolično določimo) posamezno entiteto te vrste (v našem primeru bi lahko to bila vpisna številka).

Slika 6: Entitetni tip "ČLAN"

3.3.2 Atribut in tip podatkovnega elementa

Atributi so lastnosti, ki opisujejo entiteto ali razmerja med entitetami. Atribut je v modelu ER predstavljen z elipso, v kateri je zapisano ime atributa. Ime atributa bomo vedno začeli z veliko črko, ostale pa bodo male. Vrednost atributa je podatkovni element nekega podatkovnega tipa. Vsak podatek pripada določenemu tipu, bodisi znakovnemu, številčnemu, datumskemu, slikovnemu itd. Vsakemu atributu moramo določiti domeno vseh možnih vrednosti. Denimo v našem primeru, predstavljenem na sliki Slika 7, je domena atributa »Članska številka« lahko množica števil, domena atributa »Ime« in atributa »Priimek« pa množica znakov. Kot bomo videli, imamo pri posameznem atributu lahko več možnosti, kako si izbrati njegovo domeno. Npr. lahko bi rekli, da je domena atributa »Članska številka« množica vseh 7 mestnih nizov, sestavljenih iz samih števk.



Slika 7: Atributi entitetnega tipa "ČLAN"

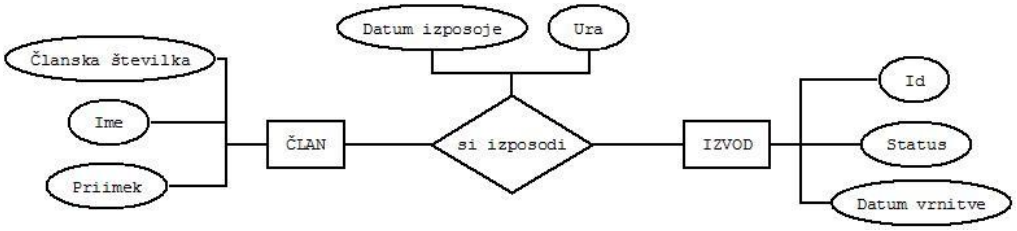
Poznamo več vrst atributov.

- **Enostavni** – so atributi, ki jih ni mogoče razdeliti na več manjših atributov. Tak primer atributa bi bil lahko atribut »Starost«.
- **Sestavljeni** – so atributi, ki jih je mogoče razdeliti na več manjših enostavnih atributov. Na primer atribut »Naslov« lahko razdelimo na naslednje enostavne atribute: »Ulica«, »Hišna številka«, »Poštna številka« in »Kraj«. Sestavljene atribute uporabljamo takrat, kadar se želimo sklicevati na posamezno komponento atributa, npr. samo na ulico ali samo kraj, ne pa na celotno vrednost atributa. V primeru, da pri sklicevanju na atribut uporabljamo celotno vrednost atributa, ni potrebe, da atribut obravnavamo kot sestavljenega. Preprosto ga obravnavamo kot enostavnega. Če npr. ne bomo nikoli uporabljali le ulice ali kraja, potem lahko naslov jemljemo kot enostavni atribut.
  Delitev na enostavne in sestavljene atribute torej ni absolutna kategorija, ampak je odvisna od domene in dejanske situacije, ki jo modeliramo. V knjižnici bomo npr. datum vedno uporabljali kot celoto. Zato ga bomo obravnavali kot enostavnega. Ampak če je problem, ki ga rešujemo, tak, da predvidevamo, da bomo želeli prešteti člane knjižnice, vpisane v določenem letu, se model spremeni. Tu bomo datum obravnavali kot sestavljen in ga tako razbili na komponente dan, mesec in leto.
- **Enovrednostni** – če ima posamezna entiteta v določenem trenutku lahko samo eno vrednost atributa, je ta atribut enovrednostni. Primera tovrstnih atributov sta atribut »Kraj« in atribut »Datum rojstva«. Vsak član namreč v danem trenutku stanuje le na enem naslovu, zato je v danem trenutku vrednost atributa »Kraj« ena sama. Prav tako ima vsaka oseba vedno le en datum rojstva, torej je tudi »Datum rojstva« enovrednostni atribut.
- **Večvrednostni** – če ima posamezna entiteta v določenem trenutku več vrednosti atributa, je ta atribut večvrednostni. Na primer vsaka knjiga ima vsaj enega avtorja, obstajajo pa knjige, ki imajo lahko več avtorjev. Torej je atribut »Avtor« večvrednostni.
  Pri relacijski podatkovni bazi večvrednostne atribute praviloma izločimo v posebne entitete. Na primer, večvrednostni atribut »Avtor« lahko izločimo iz entitetnega tipa »KNJIGA« in ga predstavimo kot novi entitetni tip »AVTOR«. Entitetni tip »AVTOR« nato z razmerjem povežemo z entitetnim tipom »KNJIGA«.

- **Opcijski** – so atributi, ki so lahko tudi brez zabeležene vrednosti. V relacijskih bazah takim atributom damo posebno vrednost, ki ji pravimo Null. S tem označimo, da vrednost atributa posamezne entitete še ni znana oziroma zabeležena v bazi.

Poznamo tudi atribute, ki opisujejo razmerja med entitetnimi tipi. Ti atributi beležijo informacije o razmerju. Če bi pri izposoji izvoda knjige želeli izvedeti kje, kdaj in ob kateri uri si je posamezen član izposodil izvod, potem bi lastnosti kraj, datum in ura zapisali kot atribute razmerja »si izposodi« med entitetnima tipoma »ČLAN« in »IZVOD«.

Na sliki Slika 8 sta prikazana entitetna tipa »ČLAN« in »IZVOD«. O članih beležimo naslednje podatke: članska številka, ime, priimek. Za posamezen izvod beležimo: zaporedno številko, status in datum vrnitve. Med njima je razmerje »si izposodi«, ki povezuje omenjena entitetna tipa. Ko si član izposodi izvod, zabeležimo datum in uro izposoje.



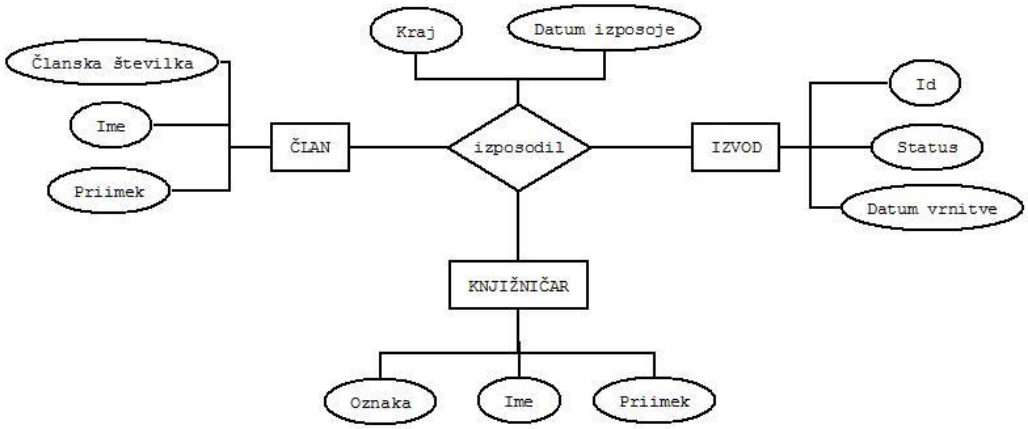
Slika 8: Primer atributov, ki opisujejo razmerje "si izposodi"

3.3.3 Razmerje med entitetnimi tipi in številnost razmerja

Entitete nastopajo v medsebojnih povezavah. Povezave med entitetami v modelu ER obravnavamo kot razmerja med entitetnimi tipi (Mohorič, 1997). Razmerje je torej povezava med entitetnimi tipi. Kot smo omenili, imajo lahko tudi razmerja atribute, ki beležijo informacije o razmerju. V modelu ER razmerja med entitetnimi tipi predstavimo z rombom, v katerem je vpisano ime razmerja, in z ustreznimi povezavami z entitetnimi tipi. Ime razmerja predstavlja vlogo entitetnega tipa glede na drugi entitetni tip, ki sodeluje v tem razmerju.

Slika 9: Razmerje "si izposodi"

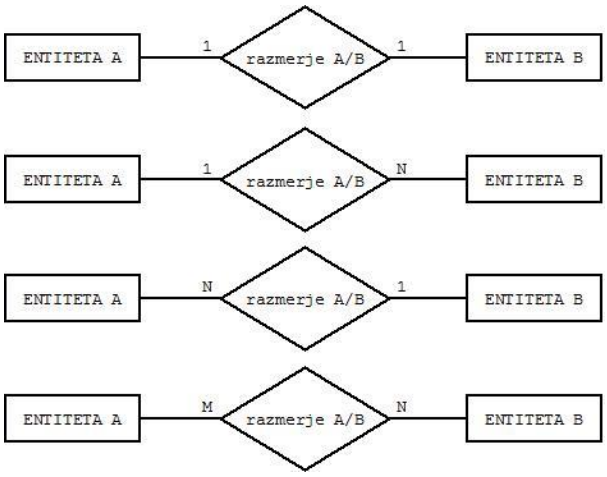
V posameznih razmerjih lahko nastopata dva ali več entitetnih tipov. Število entitetnih tipov, ki sodelujejo v razmerju, je predstavljeno s stopnjo razmerja. Večina CASE orodji praviloma omogoča le binarna razmerja, kar pomeni, da lahko z enim razmerjem povežemo le dva entitetna tipa. Na sliki Slika 9 je predstavljeno binarno razmerje »si izposodi«. Razmerja višjih stopenj uporabljamo redkeje, saj so zapletenejša.
Primer razmerja višje stopnje lahko vidimo na sliki Slika 10. Razmerje »izposodil« ima dva atributa: »Kraj« in »Datum izposoje«, ki določata, v katerem kraju in katerega dne si je član pri knjižničarju izposodil izvod. Razmerje »izposodil« je stopnje tri, saj povezuje med seboj tri entitetne tipe, in sicer: »ČLAN«, »IZVOD« in »KNJIŽNIČAR«.



Slika 10: Razmerje "izposodil" stopnje 3

Razmerja imajo običajno pravila oziroma omejitve, ki določajo, koliko-krat se lahko neka entiteta pojavi oziroma nastopa v konkretnem razmerju. Omejitve razmerij določimo glede na dejansko stanje v realnem svetu, ki ga posamezno razmerje predstavlja. Na primer za razmerje »si izposodi« na sliki Slika 9 obstaja pravilo, ki določa, da si vsak izvod lahko sočasno izposodi le en član. Takšna pravila predstavimo s številostjo razmerja. **Številost razmerja** označuje število pojavitev entitet posameznega entitetnega tipa v sodelujočem razmerju. Številost torej opredelimo za vsak entitetni tip posebej.

Številost bomo določili zgolj za binarna razmerja. Kot smo omenili, so to razmerja med dvema entitetnima tipoma. Grafična predstavitev številosti razmerij je prikazana na sliki Slika 11.

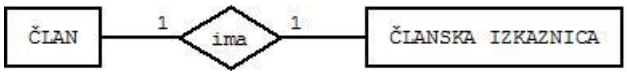


Slika 11: Značilne številosti razmerja

Značilne številosti razmerij so:

- **ena proti ena** (1 : 1),
- **ena proti mnogo** (1 : N) ali **mnogo proti ena** (N : 1),
- **mnogo proti mnogo** (M : N).

Številost *1 : 1* pomeni, da je vsak primerek entitete *A* povezan z enim primerkom entitete *B* in obratno. Povejmo takoj, da 1 običajno pomeni 1 ali 0 (da je torej en primerek entitete A povezan z največ enim primerkom entitete B, lahko pa tudi z nobenim). Primer številosti 1 : 1 je razmerje »ima« med entitetnima tipoma »ČLAN« in »ČLANSKA IZKAZNICA«, predstavljeno na sliki Slika 12. Vsak član knjižnice ima lahko le eno člansko izkaznico, vsaka članska izkaznica pa pripada natanko enemu članu.



Slika 12: Razmerje s številostjo 1 : 1

Številost *1 : N* pomeni, da je vsak primerek entitete *A* povezan z enim ali več primerki entitete *B*, vsak primerek entitete *B* pa je povezen z natančno enim primerkom entitete *A*. Tudi tu pogosto tako 1 kot N lahko "vsebujeta" tudi 0. **Številost *N : 1*** je obratna številosti 1 : N. Primer številosti 1 : N je razmerje »ima« na sliki Slika 13. V knjižnici imamo lahko več izvodov posamezne knjige, vsak izvod pa je primerek točno določene knjige.

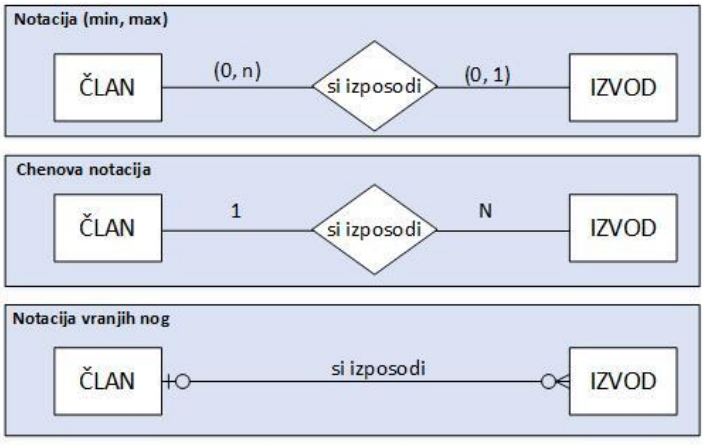
Slika 13: Razmerje s številostjo 1 : N

Številost *M : N* pomeni, da je vsak primerek entitete *A* povezan z enim ali več primerki entitete *B* in vsak primerek entitete *B* povezan z enim ali več primerki entitete *A*. Primer številosti M : N je razmerje »ima«, predstavljeno na sliki Slika 14. Posamezno knjigo je lahko napisalo več avtorjev, določen avtor pa je lahko napisal več knjig.

Slika 14: Razmerje s številostjo M : N

Razmerje pri zgledu na sliki Slika 9 je 1 : N, saj si vsak član lahko izposodi več izvodov, posamezen izvod pa si lahko sočasno izposodi le en član. Pri tem zgledu pa tudi vidimo, kako pomembno je, da razmerja ustrezno poimenujemo, oziroma, da res predstavljajo, kar želimo modelirati. Mi smo pri svojem zgledu namreč predpostavili, da opisuje trenutno stanje glede izposoje posameznega izvoda. Če pa bi to razmerje opisovalo »zgodovino izposoj«, bi bilo razmerje številosti M : N, saj si bo sčasoma posamezni izvod verjetno izposodilo več članov.

Pri ER-modelu poznamo več različnih oblik oziroma notacij za predstavitev številosti. Najpogostejše uporabljene oblike so: notacija (*min, max*) (Elmasri, in drugi, 2003), Chenova notacija (Chen, 1976) in notacija »vranjih nog« (Dybka, 2014), obstajajo pa tudi druge, npr. Bachmanova notacija, IDEF1X, Martinova notacija, notacija UML (Dybka, 2014; Ambler; Teorey, in drugi, 2006). Na sliki Slika 15 je isto razmerje predstavljeno v vseh treh najpogostejših notacijah.



Slika 15: Različne notacije številosti razmerij

Notacija (*min, max*) je prikaz številnosti, kjer je *min* najmanjše število pojavitev entitete v razmerju, *max* pa največje število pojavitev entitete v razmerju. Pri tem takoj omenimo, da običajno mej ne določamo točno, ampak za minimum uporabljamo le 0 in 1, za zgornjo mejo pa n (kar pomeni 2 ali več).

Slika 16: Notacija (min, max)

Vzemimo za zgled sliko Slika 17. Na sliki imamo entitetna tipa »KNJIGA« in »IZVOD«, ki ju povezuje razmerje »ima«. V razmerju velja pravilo, da ima vsaka knjiga lahko vsaj en, lahko pa tudi več izvodov, posamezen izvod pa pripada natanko eni knjigi.

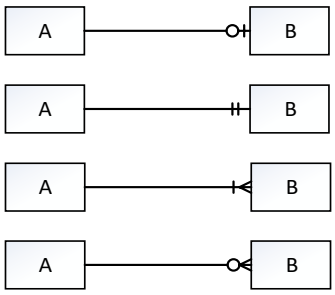
Slika 17: Številnost z notacijo (min, max)

Pri Chenovi notaciji so minimalne vrednosti izpuščene in številnosti sta glede na (*min*, *max*) zapis zamenjani. Označujemo jo z oznakami *1* in mnogo (mnogo predstavimo z oznako *N* ali *M)* poleg razmerja. Če v primeru razmerja »ima« na sliki Slika 17 izpustimo minimalne vrednosti in zamenjamo položaj navedbe številnosti glede na entitetna tipa, dobimo zapis, kot ga prikazuje Slika 18. V knjižnici je vsaka knjiga lahko v več (N) izvodih, posamezen izvod pa pripada le eni (1) knjigi.

Slika 18: Številnost s Chenovo notacijo

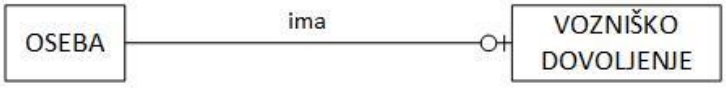
Številnost je pri notaciji vranjih nog predstavljena na naslednje načine:

- 0 ali 1
- 1 in natanko 1
- 1 ali več
- 0 ali več



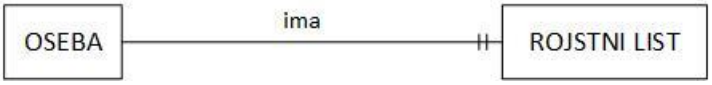
Zgoraj so prikazani le »polovični« zapisi notacije »vranjih nog«. To pomeni, da je številnost razmerja opredeljena le za en entitetni tip (v našem primeru za entitetni tip *A*), ki sodeluje v razmerju.

Številnost 0 ali 1 pomeni, da je vsak primerek entitete *A* povezan z nič ali enim primerkom entitete *B*. Primer številnosti 0 ali 1 je na sliki Slika 19 prikazana za entitetni tip »OSEBA«, ki sodeluje v razmerju »ima« z entitetnim tipom »VOZNIŠKO DOVOLJENJE«. Vsaka oseba ima lahko nič (je brez vozniškega dovoljenja) ali eno vozniško dovoljenje.



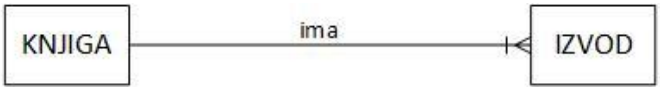
Slika 19: Prikaz številnosti 0 ali 1 z notacijo "vranjih nog"

Številost 1 in natanko 1 pomeni, da je vsak primerek entitete *A* povezan z enim in natanko enim primerkom entitete *B*. Primer številosti 1 in natanko 1 je prikazana na sliki Slika 20 za entitetni tip »OSEBA«, ki je v razmerju z entitetnim tipom »ROJSTNI LIST«. Vsaka oseba ima natanko en rojstni list.



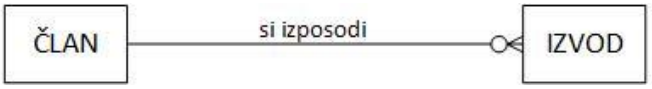
Slika 20: Prikaz številosti 1 in natanko 1 z notacijo "vranjih nog"

Številost 1 ali več pomeni, da je vsak primerek entitete *A* povezan z enim ali več primerki entitete *B*. Primer tovrstne številost prikazuje Slika 21. Na sliki je prikazana številost za entitetni tip »KNJIGA«, ki sodeluje v razmerju z entitetnim tipom »IZVOD«. Vsaka knjiga, ki jo vodimo v knjižnici, je vsaj v enem izvodu, lahko pa je določena knjiga na voljo tudi v več izvodih. Spet omenimo, da je pri vsakem predstavljenem konkretnem primeru treba upoštevati določen predpostavljeni model. Na primer, v pravkar opisanem zgledu smo predpostavili, da v bazi ne vodimo načrtovanih nakupov knjig. Takrat bi lahko imeli primerek entitete knjiga, ki pa bi bil v relaciji z 0 izvodi, saj knjige še ne bi kupili ali dobili v knjižnico.



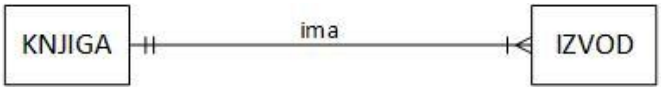
Slika 21: Prikaz številosti 1 ali več z notacijo "vranjih nog"

Številost 0 ali več pomeni, da je vsak primerek entitete A povezan z nič ali več primerki entitete B. Primer številosti 0 ali več je prikazana na sliki Slika 22 za entitetni tip »ČLAN«, ki sodeluje v razmerju z entitetnim tipom »IZVOD«. V knjižnici si lahko vsak član izposodi več knjig, ki so na voljo v več izvodih. Vsak član si lahko izposodi več izvodov. Imamo pa tudi primerke entitete član, ki so v relaciji z nič (0) izvodi, saj ni nujno, da si vsak član mora izposoditi izvod posamezne knjige. Vseh pravil ne moremo določiti v samem modelu oziroma jih predpisati s samo shemo. Na primer pravila, da si član ne more izposoditi več izvodov iste knjige, ne moremo določiti znotraj modela, vendar ga moramo kljub temu upoštevati.



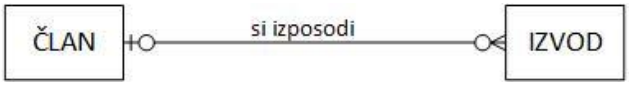
Slika 22: Prikaz številosti 0 ali več z notacijo "vranjih nog"

Poglejmo si, kako bi bil videti »polni« zapis številosti z notacijo »vranjih nog« za razmerje »ima« med entitetnima tipoma »KNJIGA« in »IZVOD« in za razmerje »si izposodi« med entitetnima tipoma »ČLAN« in »IZVOD«. To pomeni, da bomo številost opredelili za oba entitetna tipa, ki sodelujeta v omenjenih razmerjih.
Na sliki Slika 23 je predstavljeno razmerje »ima«, ki povezuje entitetna tipa »KNJIGA« in »IZVOD«. Vsaka knjiga, ki jo vodimo v knjižnici, je vsaj v enem izvodu, lahko pa je določena knjiga na voljo tudi v več izvodih. Posamezen izvod pa pripada natanko eni knjigi.



Slika 23: Prikaz zapisa številosti za razmerje "ima" z notacijo "vranjih nog"

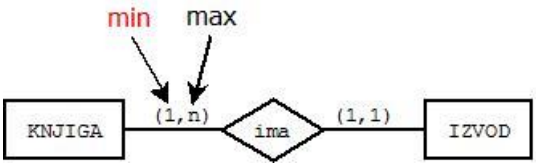
Na sliki Slika 24 je predstavljeno razmerje »si izposodi«, ki povezuje entitetna tipa »ČLAN« in »IZVOD«. Vsak član knjižnice si lahko izposodi več izvodov (tudi 0) različnih knjig, posamezen izvod pa si sočasno lahko izposodi le en član (lahko pa je v danem trenutku tudi na polici v knjižnici - neizposojen).



Slika 24: Prikaz zapisa številnosti za razmerje "si izposodi" z notacijo "vranjih nog"

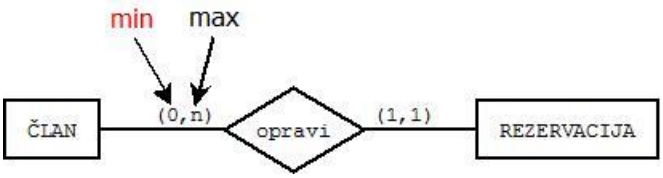
Ko govorimo o določenemu problemu, se včasih srečamo tudi s pojmom »**udeležba**«. S tem bi radi povedali, če so vsi primerki nekega entitetnega tipa v razmerju z drugim entitetnim tipom. Na primer, radi bi povedali, da mora vsak član imeti člansko izkaznico, ali da mora biti vsaka v sistemu zabeležena knjiga v knjižnici vsaj v enem izvodu, ali da nekateri člani knjižnice v danem trenutku nimajo izposojene nobene knjige. To prikažemo oziroma povemo tako, da razmerju določimo omejitev, ki jo imenujemo udeležba.
Udeležba je omejitev razmerja, ki določa, ali so vse ali samo nekatere entitete določenega entitetnega tipa udeležene v določenem razmerju. Če so vse entitete udeležene v razmerju, pravimo, da je udeležba obvezna, v nasprotnem primeru pa je neobvezna. Udeležba entitet v razmerju je predstavljena z najmanjšim številom pojavitev entitete v razmerju. Kot smo omenili pri notaciji (*min*, *max*), najmanjše število pojavitev entitete v razmerju označimo z min. Obvezna udeležba razmerja ima za min vrednost 1, neobvezna udeležba pa vrednost 0.

Primer obvezne udeležbe bomo ponazorili na razmerju »ima«, ki povezuje entitetna tipa »KNJIGA« in »IZVOD«. To razmerje je predstavljeno na sliki Slika 25. Vsaka knjiga, ki jo vodimo v knjižnici, je vsaj v enem izvodu. Torej entiteta »KNJIGA« (primerek entitetnega tipa »KNJIGA«) obstaja zgolj, če je udeležena v razmerju »ima« z entiteto »IZVOD«. Pravimo, da ima entiteta »KNJIGA« obvezno udeležbo v tem razmerju.



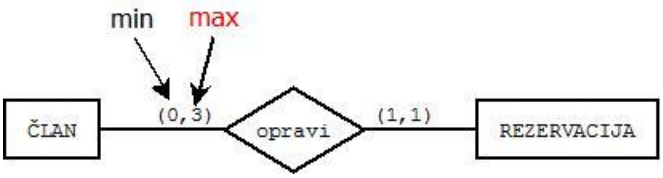
Slika 25: Razmerje "ima"

V primeru razmerja »opravi« na sliki Slika 26 pa seveda velja, da v danem trenutku nimajo vsi člani knjižnice rezervirane kake knjige. Zato pravimo, da ima entiteta »ČLAN« neobvezno udeležbo v tem razmerju.



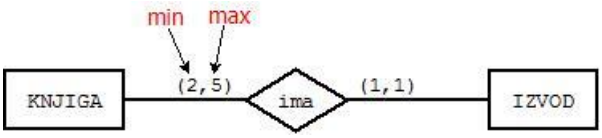
Slika 26: Razmerje "opravi"

Vidimo, da udeležbo z notacijo (*min*, *max*) in z notacijo »vranjih nog« lahko opišemo, s Chenovo notacijo pa ne.
Včasih je številnost N, ki pomeni 2 ali več, preveč ohlapna, da bi ustrezno predstavili konkretno stanje. Tako ima razmerje lahko tudi omejitev, ki omejuje največje število pojavitve entitete v tem razmerju. Denimo, da v knjižnici obstaja pravilo, da lahko član opravi največ 3 rezervacije. Potem bi številnost pri razmerju »opravi« na sliki Slika 26 opredelili drugače. Na mesto maksimalne vrednosti mnogo (N) bi napisali vrednost 3, kot prikazuje Slika 27.



Slika 27: Razmerje "opravi" z maksimalno omejitvijo številnosti

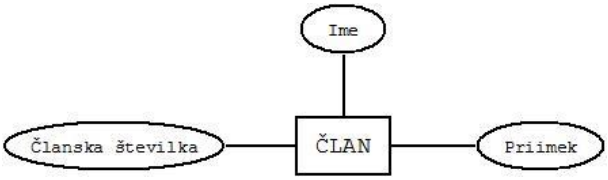
Prav tako moramo včasih modelirati stanje, kjer mora biti vsaka entiteta v razmerju z vsaj 2 ali več primerki nekega entitetnega tipa, ki sodeluje v tem razmerju. Tako imamo lahko tudi razmerje, ki omejuje najmanjše in hkrati največje število pojavitve entitete v tem razmerju. Denimo, da v knjižnici velja, da morajo vsako knjigo imeti vsaj v 2 in največ v 5 izvodih. Potem bi pri razmerju »ima« številnost opredelili drugače. Minimalna vrednosti je torej 2, maksimalna vrednosti pa je 5, kot prikazuje Slika 28.



Slika 28: Razmerje "ima" z minimalno in maksimalno omejitvijo številnosti

3.3.4 Entitetni ključ

Kot smo omenili, mora imeti vsaka entiteta neko lastnost, po kateri jo lahko enolično določimo. Tej lastnosti pravimo entitetni identifikator ali ključ entitete. Ta lastnost je lahko en sam atribut ali pa je skupek več atributov. V posameznem entitetnem tipu lahko obstaja več ključev.



Slika 29: Entitetni tip "ČLAN" z atributi

Oglejmo si nekaj primerov. Na sliki Slika 29 je še enkrat predstavljen entitetni tip »ČLAN« z atributi »Članska številka«, »Ime« in »Priimek«, ki smo ga uporabili že na sliki Slika 7. Ključ entitetnega tipa »ČLAN« predstavljenega na sliki Slika 29, je sestavljen iz enega atributa, atributa »Članska številka«, saj na podlagi lastnosti »Članska številka« lahko identificiramo posameznega člana. Vsakega člana knjižnice pa bi lahko identificirali tudi na podlagi lastnosti ime in priimek, seveda ob predpostavki, da v knjižnici ni dveh članov z enako kombinacijo imena in priimka. V tem primeru bi entitetni tip »ČLAN« imel še en ključ, in sicer takega, ki je sestavljen iz dveh atributov (»Ime« in »Priimek«). Vzemimo za zgled še entitetni tip »KNJIGA«, ki ga opisujejo naslednje lastnosti: »ISBN«, »Avtor«, »Naslov«, »Založnik« in »Leto izida«. Vsaka knjiga v knjižnici je označena z mednarodno standardno knjižno številko (ISBN). Vsaka knjiga ima torej različno ISBN-številko, zato lahko posamezno knjigo identificiramo na podlagi le-te. Atribut, ki določa ključ entitetnega tipa »KNJIGA«, je tako sestavljen iz enega atributa, atributa »ISBN«. Vsako knjigo pa bi lahko identificirali tudi z atributoma »Avtor« in »Naslov«, saj v knjižnici ne obstaja knjiga, ki bi hkrati imela enakega avtorja in naslov. V tem primeru bi vsako knjigo identificirali s skupkom dveh atributov (»Avtor« in »Naslov«). Samo atribut »Naslov« pa ne more predstavljati ključ entitetnega tipa »KNJIGA«, saj ima lahko več knjig isti naslov. Enako velja za atribut »Avtor«, saj posamezni avtor lahko napiše tudi več knjig.

Vsakemu entitetnemu tipu moramo tako določiti ključ, ki bo enolično določal enititete tega entitetnega tipa. Definiranje ključa entitete je v različnih literaturah predstavljeno na različne načine. Za določitev ključa entitete bomo sledili definicijam, ki so navedene v knjigi (Connolly, in drugi, 2005).
Minimalna množica atributov, ki enolično določa vsak primerek entitetnega tipa, je **kandidat za ključ**. Kandidat za ključ je torej skupek minimalnega števila atributov, katerih vrednost enolično določa posamezno entiteto. Na primer, atribut »Članska številka« je že kandidat za ključ entitetnega tipa »ČLAN«. Vsak član ob vpisu dobi svojo člansko številko. To pomeni, da je vrednost atributa »Članska številka« za vsakega člana različna. Torej posameznega člana knjižnice lahko identificiramo na podlagi članske številke.
Nekateri entitetni tipi imajo lahko več kandidatov za ključ. Če predpostavimo, da v knjižnici nimamo dveh ali več članov z istim imenom in priimkom, potem lahko vsakega člana knjižnice identificiramo tudi na podlagi imena in priimka. V tem primeru ima entitetni tip »ČLAN« dva kandidata za ključ. Prvi kandidat je prej omenjen atribut »Članska številka«. Drugega kandidata za ključ pa predstavljata atributa »Ime« in »Priimek«. Samo atribut »Ime« pa ne more biti kandidat za ključ entitetnega tipa »ČLAN«, saj je lahko v knjižnici včlanjenih več članov z enakim imenom. To pomeni, da na podlagi atributa »Ime« ne moremo enolično določiti posameznega člana knjižnice. Enako velja za atribut »Priimek«. Prav tako kombinacija »Ime« + »Članska številka« ne mora biti kandidat za ključ entitetnega tipa »ČLAN«, saj lahko posameznega člana identificiramo samo na podlagi članske številke. V tem primeru bi kršili pravilo minimalnosti, ki ga določa kandidat za ključ. Iz omenjene kombinacije torej lahko odstranimo atribut »Ime«, saj vsak član ob vpisu dobi svojo člansko številko, ki enolično določa posameznega člana.
Izmed vseh kandidatov za ključ izberemo enega in ga označimo kot **primarni ključ**. Primarni ključ je torej tisti kandidat za ključ, ki je izbran za enolično določitev posameznega primerka entitetnega tipa. Na primer, pri entitetnem tipu »ČLAN« lahko za primarni ključ izberemo atribut »Članska številka«.

Obstajajo tudi entitetni tipi, ki nimajo ključev ali pa imajo ključ sestavljen iz atributov drugega entitetnega tipa. Takim entitetnim tipom pravimo šibki entitetni tip. Podrobneje jih bomo predstavili v naslednjem razdelku.

Pri izbiri primarnega ključa moramo upoštevati več dejavnikov, kot so: dolžina atributa in minimalno število atributov, ki sestavljajo kandidat za ključ. Če pri izbiri primarnega ključa upoštevamo omenjene dejavnike, bomo tako v bazi imeli krajše zapise in bo baza delovala hitreje. Kadar imamo na izbiro dva kandidata za ključ, oba sestavljena le iz enega atributa, potem za primarni ključ raje izberemo tisti atribut, ki ima manjšo dolžino oziroma je sestavljen iz manj znakov oziroma števk. Podobno velja v primeru, ko je en kandidat za ključ sestavljen iz enega atributa, drugi kandidat za ključ pa iz npr. dveh atributov. Potem za primarni ključ raje izberemo kandidat za ključ, ki je sestavljen le iz enega atributa. V našem primeru je kandidat za ključ »Ime« in »Priimek« sestavljen iz dveh atributov, medtem ko je kandidat za ključ »Članska številka« sestavljen iz enega. Zato smo za primarni ključ raje izbrali atribut »Članska številka«.

Atribute, ki sestavljajo primarni ključ, v ER-diagramu podčrtamo. Primarni ključ ni nikoli opcijski atribut. Kot smo omenili, je atribut opcijski, kadar za posamezno entiteto ni nujno, da poznamo vrednost tega atributa. V primeru, da v entitetnem tipu ni primernih kandidatov za ključ (npr. šibki entitetni tipi), lahko uvedemo za primarni ključ dodaten atribut, na podlagi katerega bomo enolično določali vsak primerek entitetnega tipa. Pogosto atribut, ki ga uporabimo za primarni ključ, predstavlja zaporedno številko in ga označimo z oznako »Id«.

Ključem, ki so sestavljeni iz dveh ali več atributov, pravimo **sestavljen ključ**. Tako je kandidat za ključ »Ime in Priimek« sestavljen iz dveh atributov, zato mu pravimo sestavljeni kandidat za ključ. Načeloma

se sestavljenim ključem izogibamo, ker dolgi ključi zahtevajo daljše zapise v bazi, kar povzroča počasnejše delovanje podatkovne baze.

Oglejmo si še nekaj zgledov, kako entitetnim tipom določimo primarni ključ.
Recimo, da imamo entitetni tip »IZVOD«, ki ga opisujejo naslednji atributi: »Številka izvoda«, »Inventarna številka«, »Signatura« in »Status«. Atribut »Številka izvoda«, ki predstavlja šestmestno zaporedno številko izvoda, je kandidat za ključ entitetnega tipa »IZVOD«. Vsak izvod ima namreč svojo številko, na podlagi katere ga razlikujemo od drugih izvodov. Atribut »Inventarna številka« označuje osemmestno številko, pod katero je vpisan vsak kos knjižničnega gradiva. Vsak izvod v knjižnici ima tako različno inventarno številko. Zato lahko posamezen izvod knjige identificiramo tudi na podlagi inventarne številke. Tudi atribut »Inventarna številka« je kandidat za ključ entitetnega tipa »IZVOD«. Atribut »Signatura« predstavlja število, ki nam pove, na katerem mestu v knjižnici je izvod knjige. Na primer signatura 004 označuje mesto v knjižnici, kjer se nahajajo vsi izvodi knjig s področja računalništva. Torej na eni signaturi lahko najdemo več izvodov knjig. Zato atribut »Signatura« ni kandidat za ključ entitetnega tipa »IZVOD«. Z atributom »Status« označujemo, ali je izvod prost ali izposojen. V knjižnici imamo lahko več izvodov, ki so prosti. Imamo lahko tudi več izposojenih izvodov. Zato na podlagi statusa izvoda ne moremo identificirati posameznega izvoda knjige. Torej atribut »Status« ne predstavlja kandidata za ključ entitetnega tipa »IZVOD«. Prav tako ni kandidat za ključ kombinacija »Signatura« + »Status«, saj tudi ta ne enolično določa izvoda. Kombinacije »Številka izvoda« + »Signatura«, »Številka izvoda« + »Inventarna številka«, »Številka izvoda« + »Inventarna številka« + »Signatura« in podobne sicer enolično določajo izvod, vendar ne zadoščajo pravilu minimalnosti, ki ga določa kandidat za ključ. Zato omenjene kombinacije atributov niso kandidati za ključ entitetnega tipa »IZVOD«.
Ugotovimo torej, da ima entitetni tip »IZVOD« dva kandidata za ključ, atribut »Številka izvoda« in atribut »Inventarna številka«. Med tema kandidatoma izberemo enega in ga označimo za primarni ključ. Če pa upoštevamo še dejstvo, da je dolžina atributa »Številka izvoda« manjša od dolžine atributa »Inventarna številka«, za primarni ključ entitetnega tipa »IZVOD« izberemo atribut »Številka izvoda«.

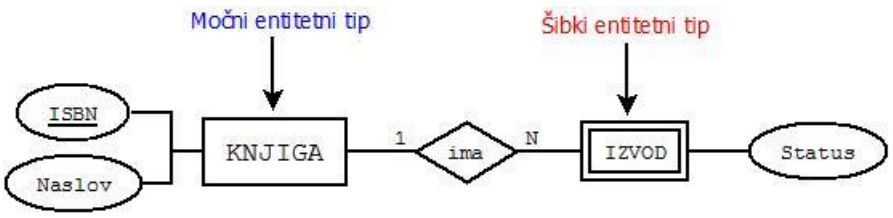
Za drugi zgled vzemimo entitetni tip »KNJIŽNIČAR«. Vsakega knjižničarja v knjižnici opišemo z naslednjimi atributi: »Številka delavca«, »EMŠO«, »Ime«, »Priimek«, »Spol« in »Naslov«. Atribut »Številka delavca« je trimestno število, ki označuje zaposlenega knjižničarja v knjižnici. Vsak knjižničar ima svojo številko delavca. To pomeni, da dva knjižničarja ne moreta imeti enake številke. Na podlagi številke delavca torej lahko enolično določimo posameznega knjižničarja. Zato je atribut »Številka delavca« kandidat za ključ entitetnega tipa »KNJIŽNIČAR«. Ravno tako ima vsak knjižničar različen EMŠO, zato je tudi atribut »EMŠO« kandidat za ključ. Atribut »Ime« ne more biti kandidat za ključ, saj lahko imamo v knjižnici dva ali več knjižničarjev z enakim imenom. Enako velja za atribut »Priimek«. Če pa predpostavimo, da v knjižnici nimamo knjižničarjev z istim imenom in priimkom hkrati, lahko posameznega knjižničarja identificiramo na podlagi atributa »Ime« in atributa »Priimek«. Tudi atributa »Ime« in »Priimek« skupaj sta kandidat za ključ entitetnega tipa »KNJIŽNIČAR«. V tem primeru je kandidat za ključ sestavljen iz dveh atributov (»Ime« in »Priimek«), zato mu pravimo sestavljen kandidat za ključ. Atribut »Spol« ne more biti kandidat za ključ, saj je v knjižnici lahko zaposlenih več knjižničarjev ženskega ali moškega spola. Enako velja za atribut »Naslov«. Namreč na istem naslovu lahko bivata dva ali več knjižničarjev. Prav tako niso kandidati za ključ kombinacije »Ime« + »Spol«, »Ime« + »Naslov«, »Priimek« + »Spol«, »Priimek« + »Naslov«, »Spol« + »Naslov«, saj tudi te ne enolično določajo knjižničarja. Kombinacije »Številka delavca« + »Ime«, »Številka delavca« + »EMŠO«, »Številka delavca« + »Priimek«, »Številka delavca« + »Ime« + »Priimek«, »Številka delavca« + »EMŠO« + »Ime« »Številka delavca« + »EMŠO« + »Ime« + »Priimek« + »Spol« in podobne sicer enolično določajo knjižničarja, vendar niso kandidati za ključ, saj kršijo pravilo minimalnosti. Pri entitetnem tipu

»KNJIŽNIČAR« imamo tri kandidate za ključ, in sicer: atribut »Številka delavca«, atribut »EMŠO« in atributa »Ime« in »Priimek«. Za primarni ključ entitetnega tipa »KNJIŽNIČAR« izberemo kandidat za ključ »Številka delavca«, saj je le-ta sestavljen iz enega samega atributa, hkrati pa je njegova dolžina omejena na največ 3 števke.

3.3.5 Močni in šibki entitetni tipi

Včasih imamo pri določanju ključev določenega entitetnega tipa težave. Vzemimo za zgled entitetna tipa »KNJIGA« in »IZVOD«. V modelu smo določili, da ima entitetni tip »KNJIGA« atributa »ISBN« in »Naslov«, entitetni tip »IZVOD« pa le atribut »Status«, ki ponazarja prost oziroma izposojen izvod. Za tip »KNJIGA« ni težav z določanjem ključa, saj ima vsaka knjiga svojo ISBN-številko. Te so enolično določene, torej lahko vsako knjigo identificiramo na podlagi ISBN-številke. Pri tipu »IZVOD« pa so težave, saj nam obstoječi atribut ne omogoča identifikacije izvoda posamezne knjige v množici vseh izvodov, torej entitetnemu tipu »IZVOD« ne moremo določiti primarnega ključa. Ena rešitev je, da bi uvedli dodaten atribut z oznako »Id«, na podlagi katerega bi lahko identificirali posamezen izvod. Po drugi strani pa posamezen izvod lahko identificiramo ob sklicevanju na entitetni tip »KNJIGA«. Če se sklicujemo na entitetni tip »KNJIGA«, potem bomo v primeru, ko imamo več izvodov iste knjige, imeli pri entitetnem tipu »IZVOD« več enakih zapisov (vrstic). Načeloma se pri relacijskih podatkovnih bazah izogibamo podvajanju vrstic (shranjevanju enakih zapisov v bazi), ampak pri šibkem entitetnem tipu se temu ne moremo izogniti: če imamo 5 izvodov neke knjige, potem bomo imeli pri entitetnem tipu »IZVOD« v bazi 5 enakih zapisov, kjer vsak zapis opisuje posamezen izvod iste knjige.

Glede na to, ali lahko posamezno entiteto znotraj množice entitet nekega entitetnega tipa ločimo od drugih samo s pomočjo vrednosti njenih atributov, delimo entitetne tipe na močne in šibke. Močni entitetni tip je tip, katerega identifikator (ključ) tvorijo zgolj njegovi lastni atributi. Šibki entitetni tip je tip, ki za identifikacijo potrebuje poleg lastnih atributov tudi atribute (identifikatorje) drugih močnih entitetnih tipov, s katerimi je v razmerju. Med šibkim in močnim entitetnim tipom vedno obstaja razmerje s številostjo 1 : N (ena proti mnogo). Vsak primerek šibkega entitetnega tipa je vedno povezan z natanko enim primerkom močnega entitetnega tipa, medtem ko je primerek močnega entitetnega tipa lahko povezan z več primerki šibkega entitetnega tipa. Na primer, vsak izvod pripada natanko eni knjigi, medtem ko ima posamezna knjiga lahko več izvodov. Šibki entitetni tip ne more obstajati brez svoje starševske entitete.
Vzemimo za zgled sliko Slika 30, ki prikazuje entitetna tipa »KNJIGA« in »IZVOD«. Vsak izvod je povezan z natanko eno knjigo, vsaka knjiga pa ima lahko enega ali več izvodov. Izvod ne more obstajati brez knjige. Entitetni tip »IZVOD« z atributom »Status«, ki ponazarja prost oziroma izposojen izvod, nam ne omogoča identifikacije izvoda posamezne knjige v množici vseh izvodov. Identificiramo ga lahko zgolj ob sklicevanju na entitetni tip »KNJIGA«. Ključ entitetnega tipa »IZVOD« je tako šibki entitetni tip z identifikatorjem {ISBN}.
Kot vidimo na sliki Slika 30, šibki entitetni tip označimo s pravokotnikom z dvojnim robom.

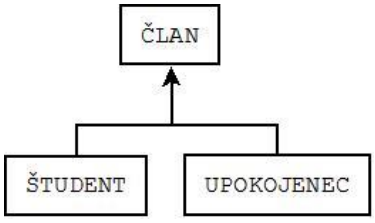


Slika 30: Primer močnega in šibkega entitetnega tipa

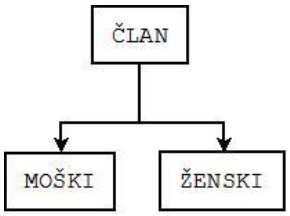
3.3.6 Generalizacija in specializacija

V določenih situacijah, ki jih modeliramo, lahko ugotovimo, da imamo entitetne tipe, ki imajo določene skupne lastnosti, ali da moramo določen entitetni tip dopolniti z dodatnimi lastnostmi, ki pripadajo le delu entitet tega tipa.

V primeru skupnih lastnosti tipov lahko za te entitetne tipe uvedemo generalizacijo in jim določimo skupen nadtip. V drugem primeru, ko moramo dodajati lastnosti, pa lahko uvedemo specializacijo, kjer ustvarimo novi entitetni (pod)tip, ki vsebuje le k nekemu tipu dodane lastnosti.
Oglejmo si primer. V knjižnico so vpisani občani z različnim statusom. Njene člane ločimo na predšolske otroke, dijake, študente, upokojence, zaposlene in brezposelne. Zato bi lahko uvedli sledeče entitetne tipe: »PREDŠOLSKI OTROK«, »DIJAK«, »ŠTUDENT«, »UPOKOJENEC«, »ZAPOSLEN« in »BREZPOSELN«. Ugotovimo pa, da imajo vsi omenjeni entitetni tipi določene skupne lastnosti. Te so »Članska številka«, »Ime«, »Priimek«, »Datum rojstva«, »Naslov« in »Potek članstva«. Na podlagi le-teh skupnih lastnosti lahko določimo za vse entitetne tipe skupni nadtip »ČLAN«, ki bo nosilec skupnih lastnosti.
Ker je višina članarine odvisna od statusa, seveda podatek, ki je bil prej podan z entitetnim tipom, hranimo v tem nadtipu.
Po drugi strani pa ugotovimo, da moramo pri članih, ki so mladoletni (člani, ki so mlajši od 18 let), hraniti tudi podatke o skrbnikih (ime in priimek skrbnika), saj mladoletna oseba ne more biti odgovorna za poravnavo zamudnine. Torej bi bilo smiselno, da bi uvedli ustrezen podtip, ki bi opisoval mladoletne osebe.
Generalizacija in specializacija sta postopka, ki omogočata oblikovanje hierarhije entitet. Z generalizacijo je predstavljen odnos tip–nadtip med entitetnimi tipi. Na primer, na sliki Slika 31 entitetnima tipoma »ŠTUDENT« in »UPOKOJENEC« lahko priredimo skupni nadtip »ČLAN«. To pomeni, da je npr. Miha študent in hkrati član knjižnice, Jože pa upokojenec in hkrati član knjižnice.
Specializacija je generalizaciji nasproten postopek in predstavlja odnos tip–podtip med entitetnimi tipi. Običajno temelji na vrednosti določene lastnosti atributov, ki pripadajo nadtipu. Na primer, na sliki Slika 32 entitetnemu tipu »ČLAN« lahko na podlagi lastnosti spol priredimo podtipa »MOŠKI« in »ŽENSKI«.
Vsak tip deduje vse lastnosti svojega nadtipa. V primeru, ko ima posamezen tip več nadtipov, deduje lastnosti od vseh po vrsti. Na sliki Slika 32 torej entitetna tipa »MOŠKI« in »ŽENSKI« dedujeta vse lastnosti nadtipa »ČLAN«.



Slika 31: Primer generalizacije

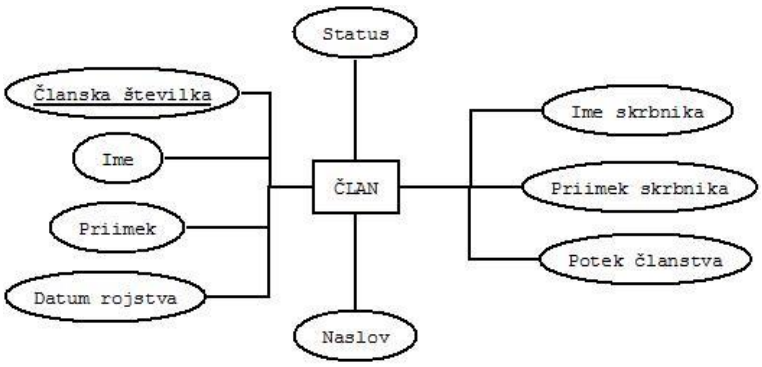


Slika 32: Primer specializacije

Generalizacijo oziroma specializacijo je smiselno uporabiti, ko npr. obstaja atribut, ki je značilen samo za določeno podmnožico primerkov entitet določenega tipa.

Nadaljujmo z zgledom o članih knjižnice, ki smo ga omenili na začetku razdelka. Zbiramo torej naslednje podatke o članih: »Članska številka«, »Ime«, »Priimek«, »Datum rojstva«, »Naslov« in »Potek članstva«. V knjižnici imamo mladoletne in polnoletne člane. Za mladoletne člane nas zanima še ime in priimek skrbnika, za polnoletne člane pa status. Status predstavlja lastnost, s katero določimo, ali je član dijak, študent, upokojenec, zaposlen ali brezposeln. Glede na opis bi lahko vse omenjene lastnosti članov prepisali enemu entitetnemu tipu, tipu »ČLAN«.

ČLAN (Članska številka, Ime, Priimek, Datum rojstva, Naslov, Potek članstva, Ime skrbnika, Priimek skrbnika, Status)



Slika 33: Entitetni tip "ČLAN" pred uvedbo specializacije

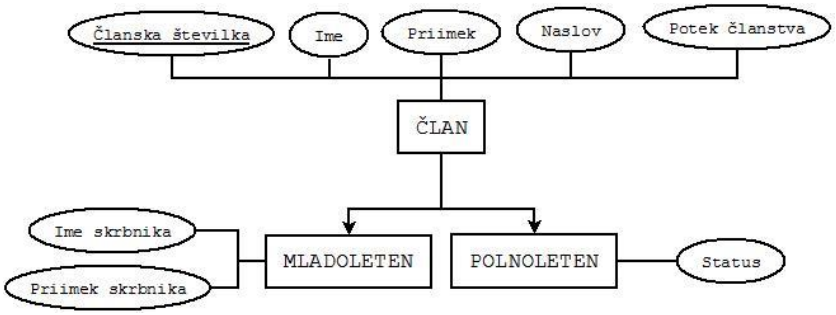
Ker ime in priimek skrbnika pripadata samo mladoletnim članom, bodo polnoletni člani imeli za atribut »Ime skrbnika« in atribut »Priimek skrbnika« v bazi vneseno vrednost Null. Kot smo omenili, z Null označimo, da vrednost atributa ni znana, oziroma za določen primerek entitetnega tipa ne obstaja ali ni smiselna. Enako bi imel atribut »Status« vrednost Null za vse mladoletne člane.

Članska številka	Ime	Priimek	Datum rojstva	Naslov	Potek članstva	Ime skrbnika	Priimek skrbnika	Status
1020868	Tina	Kovač	30.10.1981	Srebrničeva ulica 10	12.09.2016	Null	Null	Zaposlen
1080869	Jure	Novak	01.06.2008	Martinuči 1d	15.10.2016	Tina	Novak	Null
1029870	Nejc	Kos	11.05.1966	Erjavčeva ulica 2a	25.10.2016	Null	Null	Zaposlen
1030871	Tina	Novak	18.12.1988	Ulica talcev 12	13.12.2016	Null	Null	Brezposeln
1020222	Marko	Zimic	25.12.2004	Partizanska ulica 1	04.01.2017	Franc	Zimic	Null
1021153	Tia	Tinta	01.01.2007	Gradnikova ulica 25	12.09.2016	Mateja	Tinta	Null
0016024	Tomi	Frfolja	08.01.2010	Gregorčičeva ulica 3	01.10.2016	Ana	Frfolja	Null
2010895	Maja	Žužek	15.01.1955	Erjavčeva ulica 2a	25.12.2016	Null	Null	Upokojenec
1005766	Jure	Kavčič	25.12.1990	Bazoviška ulica 8	13.08.2016	Null	Null	Študent
1000637	Luka	Medved	29.03.1985	Prvomajska ulica 54	04.01.2017	Null	Null	Zaposlen
0995508	Saša	Stanič	06.11.1997	Cankarjeva ulica 18b	02.02.2017	Null	Null	Dijak
0190379	Dejan	Golob	13.02.2006	Rejčeva ulica 12	15.11.2016	Stanka	Golob	Null
0052501	Maja	Gorjanc	20.03.1988	Erjavčeva ulica 25	08.09.2016	Null	Null	Zaposlen
1234567	Majda	Šuligoj	27.02.1994	Šolska ulica 33	02.01.2017	Null	Null	Študent
2654321	Uroš	Simčič	06.08.2010	Vedrijan 18	28.01.2017	Tanja	Simčič	Null
0969863	Pia	Novak	13.03.2003	Laze 25	02.09.2016	Ana	Novak	Null
0883988	Nika	Šuligoj	20.03.1984	Cesta 18a	28.10.2016	Null	Null	Zaposlen
1114586	Marko	Konjedic	28.02.1970	Bilje 21	25.12.2016	Null	Null	Brezposeln
0111216	Jože	Velišček	04.07.1946	Ulica talcev 12	28.12.2016	Null	Null	Upokojenec
0036587	Stanko	Kos	11.04.1950	Lokavška cesta 11	05.01.2017	Null	Null	Upokojenec

Slika 34: Primerki entitetnega tipa "ČLAN" pred uvedbo specializacije

Kot vidimo na sliki Slika 34 bi v bazi lahko imeli veliko zapisov z vrednostjo Null. Zato je bolj smiselno entitetni tip »ČLAN« razdeliti glede na datum rojstva na podtipa »MLADOLETEN« in »POLNOLETEN«. Nadtip »ČLAN« bi tako imel atribute, ki so skupni vsem trem tipom, to so: »Članska številka«, »Ime«, »Priimek«, »Naslov« in »Potek članstva«, entitetna podtipa »MLADOLETEN« in »POLNOLETEN« pa samo tiste atribute, ki so značilni za posamezen podtip. Z uvedbo specializacije bi bil opis sledeč:

ČLAN (Članska številka, Ime, Priimek, Naslov, Potek članstva)
Podtip MLADOLETEN (Ime skrbnika, Priimek skrbnika)
Podtip POLNOLETEN (Status)



Slika 35: Entitetni tip "ČLAN" po uvedbi specializacije

Pri podtipih identifikatorja (ključa) nista postavljena, ker podtipa dedujeta vse lastnosti od nadtipa »ČLAN«, torej tudi ključ.

Prej omenjeno tabelo (Slika 34) bi torej lahko predstavili s tremi tabelami, kot prikazuje Slika 36.

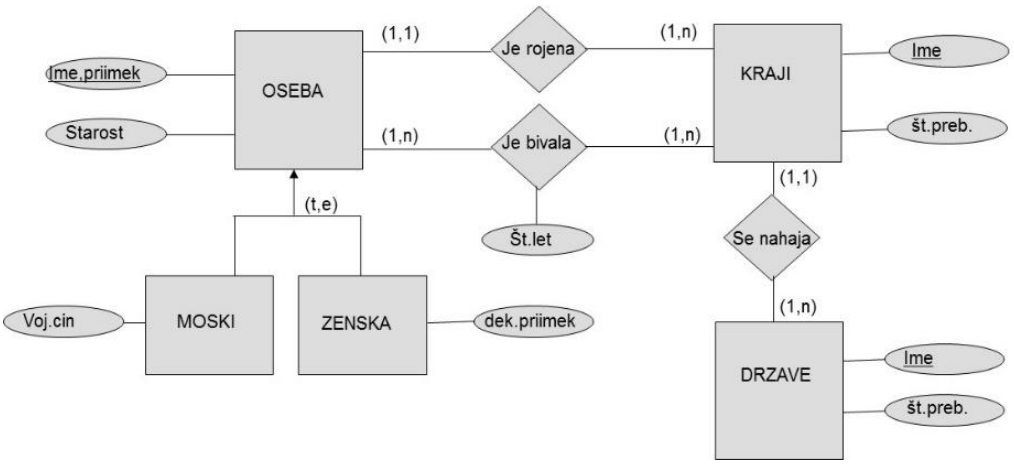
Članska številka	Ime	Priimek	Datum rojstva	Naslov	Potek članstva
1020868	Tina	Kovač	30.10.1981	Srebrničeva ulica 10	12.09.2016
1080869	Jure	Novak	01.06.2008	Martinuči 1d	15.10.2016
1029870	Nejc	Kos	11.05.1966	Erjavčeva ulica 2a	25.10.2016
1030871	Tina	Novak	18.12.1988	Ulica talcev 12	13.12.2016
1020222	Marko	Zimic	25.12.2004	Partizanska ulica 1	04.01.2017
1021153	Tia	Tinta	01.01.2007	Gradnikova ulica 25	12.09.2016
0016024	Tomi	Frfolja	08.01.2010	Gregorčičeva ulica 3	01.10.2016
2010895	Maja	Žužek	15.01.1955	Erjavčeva ulica 2a	25.12.2016
1005766	Jure	Kavčič	25.12.1990	Bazoviška ulica 8	13.08.2016
1000637	Luka	Medved	29.03.1985	Prvomajska ulica 54	04.01.2017
0995508	Saša	Stanič	06.11.1997	Cankarjeva ulica 18b	02.02.2017
0190379	Dejan	Golob	13.02.2006	Rejčeva ulica 12	15.11.2016
0052501	Maja	Gorjanc	20.03.1988	Erjavčeva ulica 25	08.09.2016
1234567	Majda	Šuligoj	27.02.1994	Šolska ulica 33	02.01.2017
2654321	Uroš	Simčič	06.08.2010	Vedrijan 18	28.01.2017
0969863	Pia	Novak	13.03.2003	Laze 25	02.09.2016
0883988	Nika	Šuligoj	20.03.1984	Cesta 18a	28.10.2016
1114586	Marko	Konjedic	28.02.1970	Bilje 21	25.12.2016
0111216	Jože	Velišček	04.07.1946	Ulica talcev 12	28.12.2016
0036587	Stanko	Kos	11.04.1950	Lokavška cesta 11	05.01.2017

Ime skrbnika	Priimek skrbnika
Tina	Novak
Franc	Zimic
Mateja	Tinta
Ana	Frfolja
Stanka	Golob
Tanja	Simčič
Ana	Novak

Status
Zaposlen
Brezposeln
Upokojenec
Študent
Dijak

Slika 36: Primerki entitetnega tipa "ČLAN" po uvedbi specializacije

Oglejmo si še en zgled. Na sliki Slika 37 je še enkrat prikazan model, ki smo ga že uporabili na sliki Slika 5.



Slika 37: Primer ER-modela

Zbiramo podatke o osebah. Osebe opišemo z imenom, priimkom in starostjo. Osebe so lahko ženskega in moškega spola. Za ženske nas zanima še dekliški priimek, za moške pa vojaški čin, če ga imajo. Glede na opis bi lahko vse omenjene lastnosti oseb pripisali enemu entitetnemu tipu, tipu »OSEBA«.

OSEBA (Ime, Priimek, Starost, Dekliški priimek, Vojaški čin)

Ker dekliški priimek pripada samo osebam ženskega spola, bodo moški imeli za atribut »Dekliški priimek« v bazi vneseno vrednost Null, enako bi veljalo pri ženskah za atribut »Vojaški čin«. Torej bi v takem primeru imeli veliko zapisov z vrednostjo Null. Zato je bolj smiselno entitetni tip »OSEBA« razdeliti glede na spol na podtipa »MOŠKI« in »ŽENSKA«. Nadtip »OSEBA« ima atribute, ki so skupni vsem trem tipom, to so: ime, priimek in starost. Entitetna podtipa »ŽENSKA« in »MOŠKI« pa imata atribute, ki jih nadrejeni tip nima.

OSEBA (Ime, Priimek, Starost)
Podtip MOŠKI (Vojaški čin)
Podtip ŽENSKA (Dekliški priimek)

3.4 Postopek izdelave ER-modela

Oglejmo si, kako izdelamo ER-model. Uporabili bomo primer poenostavljene podatkovne baze knjižnice. Zahteve, ki jim mora zadoščati podatkovna baza knjižnica, smo navedli v razdelku 2.4 (Modeliranje podatkov).

Gradnja ER-modela običajno poteka po korakih. Postopek gradnje lahko do neke mere formaliziramo z uporabo naslednjih standardnih korakov:

1. Opredelitev entitetnih tipov.
2. Opredelitev povezav med entitetnimi tipi.
3. Opredelitev kardinalnosti (številnosti) razmerja.
4. Opredelitev atributov entitetnim tipom.
5. Opredelitev (morebitnih) atributov povezavam.
6. Določitev primarnega ključa (entitetni identifikator).
7. Specializacija oz. generalizacija entitetnih tipov (po potrebi).
8. Risanje ER-diagrama.
9. Konzultacija z uporabniki (preverjanje ustreznosti modela).

Pri sami izdelavi ER-diagramov imamo na razpolago več možnih odločitev, saj postopek načrtovanja ER-diagrama dopušča precej »svobode«. Na primer, ko opredeljujemo atribute entitetnim tipom, ne vemo zagotovo, ali je bolje neko lastnost predstaviti kot atribut nekega entitetnega tipa ali kot samostojni entitetni tip. Neustrezna odločitev za eno ali drugo možnost lahko kasneje povzroči mnogo težav, kot je npr. nepotrebno podvajanje podatkov. Seveda je možno model kasneje še spreminjati in tako te težave odpraviti ali vsaj omiliti. Oglejmo si zgled, ko imamo na voljo različne odločitve glede tega, ali podatek uporabimo kot atribut ali kot samostojni entitetni tip.

Če bomo sestavljali podatkovno bazo za neko knjižnico, bomo zagotovo morali opisati knjige. Definirali bomo torej entitetni tip »KNJIGA«. Knjiga ima običajno avtorja. Potrebujemo torej lastnost avtor. To lahko dodamo bodisi kot atribut entitetnemu tipu »KNJIGA« bodisi zanj ustvarimo nov entitetni tip »AVTOR«. Oglejmo si najprej, kako bi bil videti del modela, če bi uporabili avtorja kot atribut.

KNJIGA (ISBN, Naslov, Avtor, Založnik, Leto izida)

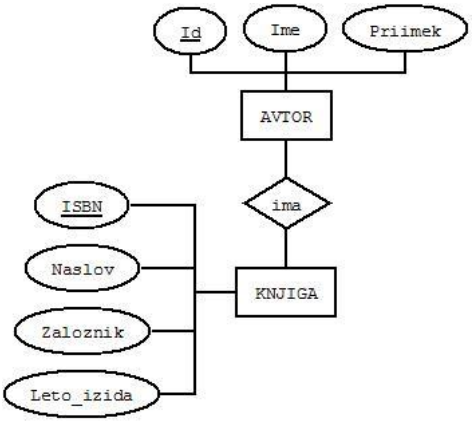
Slika 38: Avtor kot atribut entitetnega tipa "ČLAN"

Nekaj primerkov, ki ustrezajo temu delu modela prikazuje Slika 39.

ISBN	Naslov	Avtor	Založnik	Leto izida
9789610118176	Srečni človek	Arto Paasilinna	Mladinska knjiga	2012
8681049810	Informatika	Miro Gradišar	Moderna organizacija	1994
8681049810	Informatika	Gortan Resinovič	Moderna organizacija	1994
9616046055	Načrtovanje relacijskih podatkovnih baz	Tomaž Mohorič	BI-TIM	1997
9789612400675	Temelji elektronskega poslovanja	Andrej Kovačič	Ekonomska fakulteta	2009
9789612400675	Temelji elektronskega poslovanja	Aleš Groznik	Ekonomska fakulteta	2009
9789612400675	Temelji elektronskega poslovanja	Miroslav Ribič	Ekonomska fakulteta	2009
9789610027300	Skoraj popolna	Susan Mallery	Učila International	2015
9619075501	Informatika	Mirko Vintar	Bons	1999
9780136086208	Fundamentals of database systems	Ramez Elmasri	Addison Wesley	2011
9780136086208	Fundamentals of database systems	Shamkant B. Navathe	Addison Wesley	2011
9789610204954	SQL	Boštjan Brumen	DZS	2013
9616046128	Podatkovne baze	Tomaž Mohorič	BI-TIM	2002

Slika 39: Primerki entitetnega tipa "ČLAN", kjer je avtor predstavljen kot atribut

Če ustvarimo nov entitetni tip AVTOR, bo model izgledal tako, kot prikazuje Slika 40. V tem primeru je seveda potrebno entitetna tipa »KNJIGA« in »AVTOR« povezati z razmerjem.



Slika 40: Avtor kot samostojni entitetni tip

Ta model bi v primeru relacijske podatkovne baze, ki jo bomo uporabljali mi, na prej predstavljenih podatkih predstavili s tremi tabelami, kot prikazuje Slika 41.

KNJIGA (ISBN, Naslov, Založnik, Leto izida)
AVTOR (Id, Ime, Priimek)
IMA (ISBN, Id)

ISBN	Naslov	Založnik	Leto izida
9789610118176	Srečni človek	Mladinska knjiga	2012
8681049810	Informatika	Moderna organizacija	1994
9616046055	Načrtovanje relacijskih podatkovnih baz	BI-TIM	1997
9789612400675	Temelji elektronskega poslovanja	Ekonomska fakulteta	2009
9789610027300	Skoraj popolna	Učila International	2015
9619075501	Informatika	Bons	1999
9780136086208	Fundamentals of database systems	Addison Wesley	2011
9789610204954	SQL	DZS	2013
9616046128	Podatkovne baze	BI-TIM	2002

Id	Ime	Priimek
1	Arto	Paasilinna
2	Miro	Gradišar
3	Gortan	Resinovič
4	Tomaž	Mohorič
5	Andrej	Kovačič
6	Aleš	Groznik
7	Miroslav	Ribič
8	Susan	Mallery
9	Mirko	Vintar
10	Ramez	Elmasri
11	Shamkant B.	Navathe
12	Boštjan	Brumen

ISBN	Id
9789610118176	1
8681049810	2
8681049810	3
9616046055	4
9789612400675	5
9789612400675	6
9789612400675	7
9789610027300	8
9619075501	9
9780136086208	10
9780136086208	11
9789610204954	12
9616046128	4

Slika 41: Primerki entitetnega tipa "ČLAN", ko je avtor predstavljen kot entitetni tip

Ker vemo, da ima lahko posamezna knjiga več kot enega avtorja, je smiselno, če značilnost »Avtor« opredelimo kot nov entitetni tip. Na ta način se izognemo, da bi tip »KNJIGA« imel večvrednostni atribut. Hkrati se bomo s tem izognili nepotrebnemu podvajanju podatkov, saj bodo podatki o posameznem avtorju v bazi shranjeni le enkrat. Kot vidimo na sliki Slika 39, se lahko v primeru, da ima knjiga več kot enega avtorja, vrednosti vseh ostalih lastnosti (atributov) ponovijo za vsakega posameznega avtorja. V podatkovni bazi bi na ta način hranili podvojene zapise. Če pa atribut »Avtor« izločimo v samostojni entitetni tip, bomo v bazi hranili te podatke le enkrat.

Prav tako je smiselno, da če ima naša knjižnica le knjige z enim avtorjem, vendar o avtorju hrani več podatkov (npr. še letnico rojstva, državo rojstva, psevdonim itd.), za avtorja ustvarimo nov entitetni tip »AVTOR«. Oglejmo si najprej primerke entitetnega tipa »ČLAN«, če bi uporabili avtorja kot atribut (Slika 42).

KNJIGA (ISBN, Naslov, Avtor {Ime in priimek, Datum rojstva, Država rojstva, Psevdonim}, Založnik, Leto izida)

ISBN	Naslov	Avtor	Založnik	Leto izida
9789610118176	Srečni človek	Arto Paasilinna	Mladinska knjiga	2012
9789610118176	Srečni človek	20.04.1942	Mladinska knjiga	2012
9789610118176	Srečni človek	Finska	Mladinska knjiga	2012
9789610118176	Srečni človek	Null	Mladinska knjiga	2012
9789610114123	Solzice	Lovro Kuhar	Mladinska knjiga	2011
9789610114123	Solzice	10.08.1893	Mladinska knjiga	2011
9789610114123	Solzice	Slovenija	Mladinska knjiga	2011
9789610114123	Solzice	Prežihov Voranc	Mladinska knjiga	2011
9789610204954	SQL	Boštjan Brumen	DZS	2013
9789610204954	SQL	13.10.1972	DZS	2013
9789610204954	SQL	Slovenija	DZS	2013
9789610204954	SQL	Null	DZS	2013
9789616623179	Črna vrtnica	Eleanor Marie Robertson	Meander	2007
9789616623179	Črna vrtnica	10.10.1950	Meander	2007
9789616623179	Črna vrtnica	ZDA	Meander	2007
9789616623179	Črna vrtnica	Nora Roberts	Meander	2007

Slika 42: Primerki, ko je avtor predstavljen kot atribut

Sedaj pa si poglejmo, kako bi podatke predstavili, če avtorja izločimo v posebni entitetni tip (Slika 43).

KNJIGA (ISBN, Naslov, Založnik, Leto izida)
AVTOR (Id, Ime, Priimek, Datum rojstva, Država rojstva, Psevdonim)

ISBN	Naslov	Založnik	Leto izida
9789610118176	Srečni človek	Mladinska knjiga	2012
9789610114123	Solzice	Mladinska knjiga	2011
9789610204954	SQL	DZS	2013
9789616623179	Črna vrtnica	Meander	2007

Id	Ime	Priimek	Datum rojstva	Država rojstva	Psevdonim
1	Arto	Paasilinna	20.04.1942	Finska	Null
2	Lovro	Kuhar	10.08.1893	Slovenija	Prežihov Voranc
3	Boštjan	Brumen	13.10.1972	Slovenija	Null
4	Eleanor Marie	Robertson	10.10.1950	ZDA	Nora Roberts

Slika 43: Primerki, ko je avtor predstavljen kot entitetni tip

V primeru, da hranimo več podatkov o avtorju in imajo knjige lahko več avtorjev, dileme sploh ni, saj nujno potrebujemo dva entitetna tipa.
Podoben razmislek kot pri avtorjih velja za lastnost »Ključne besede« pri entitetnem tipu »KNJIGA«.

Poglejmo si postopek izdelave ER-modela na primeru opisa knjižnice, predstavljenega v razdelku 1.4. Iz opisa opredelimo entitetne tipe KNJIGA, AVTOR, KLJUČNA BESEDA, IZVOD, ČLAN, POŠTA in REZERVACIJA.

Nato opredelimo povezave oziroma razmerja med entitetnimi tipi, kar prikazuje Tabela 1.

	KNJIGA	AVTOR	KLJUČNA BESEDA	IZVOD	ČLAN	POŠTA	REZERVACIJA
KNJIGA		ima	opisuje	ima			
AVTOR							
KLJUČNA BESEDA							
IZVOD							
ČLAN				si izposodi		pripada	opravi
POŠTA							
REZERVACIJA	zagotovi						

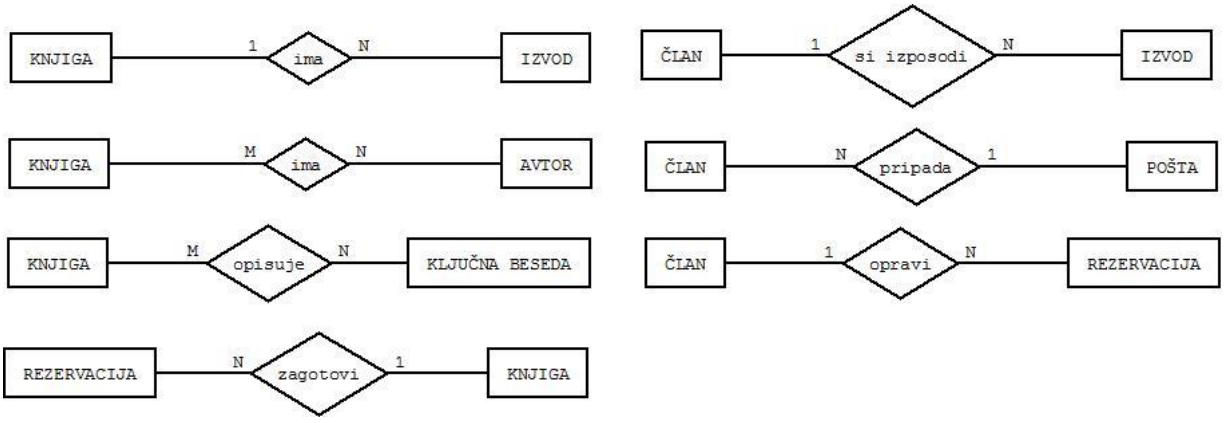
Tabela 1: Prikaz razmerij med entitetnimi tipi

Opredelimo številnost razmerij. Pri tem bomo seveda izhajali iz uporabniških zahtev. Te so:

- knjiga ima lahko več izvodov,
- knjiga ima lahko več avtorjev,
- knjigo lahko opisuje več ključnih besed,
- vsak član si lahko izposodi več izvodov,
- član lahko opravi več rezervacij,
- vsak član pripada eni pošti,
- vsaka rezervacija zagotovi eno knjigo.

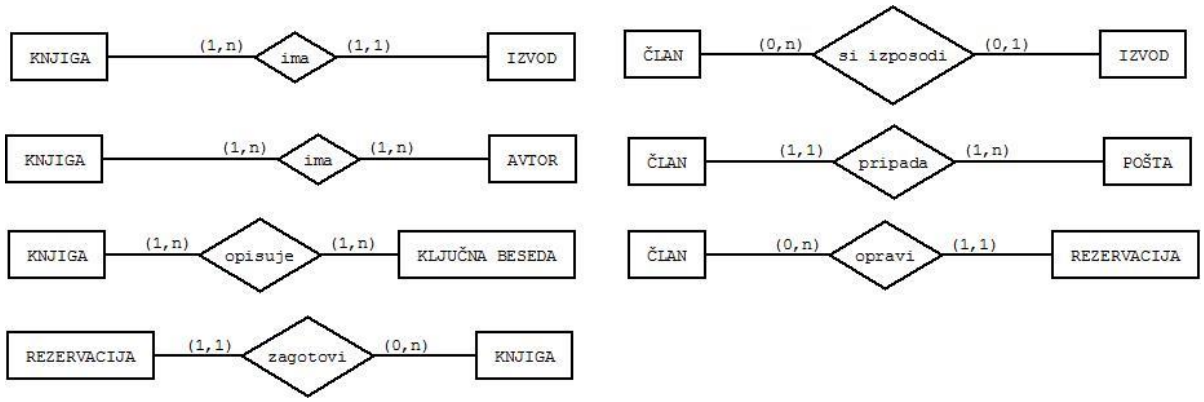
Kot smo omenili, obstaja več načinov zapisov številnosti v ER-modelu. Orodje DIA, s katerim bomo izrisali konceptualni model, uporablja bodisi notacijo (*min*, *max*) bodisi Chenovo notacijo. Mi bomo številnosti prikazali najprej s Chenovo notacijo, nato pa še z notacijo (*min*, *max*).

Opredeljene številnosti razmerij s Chenovo notacijo:



Slika 44: Opredelitev številnosti – Chenova notacija

Opredeljene številnosti razmerij z notacijo (min, max):



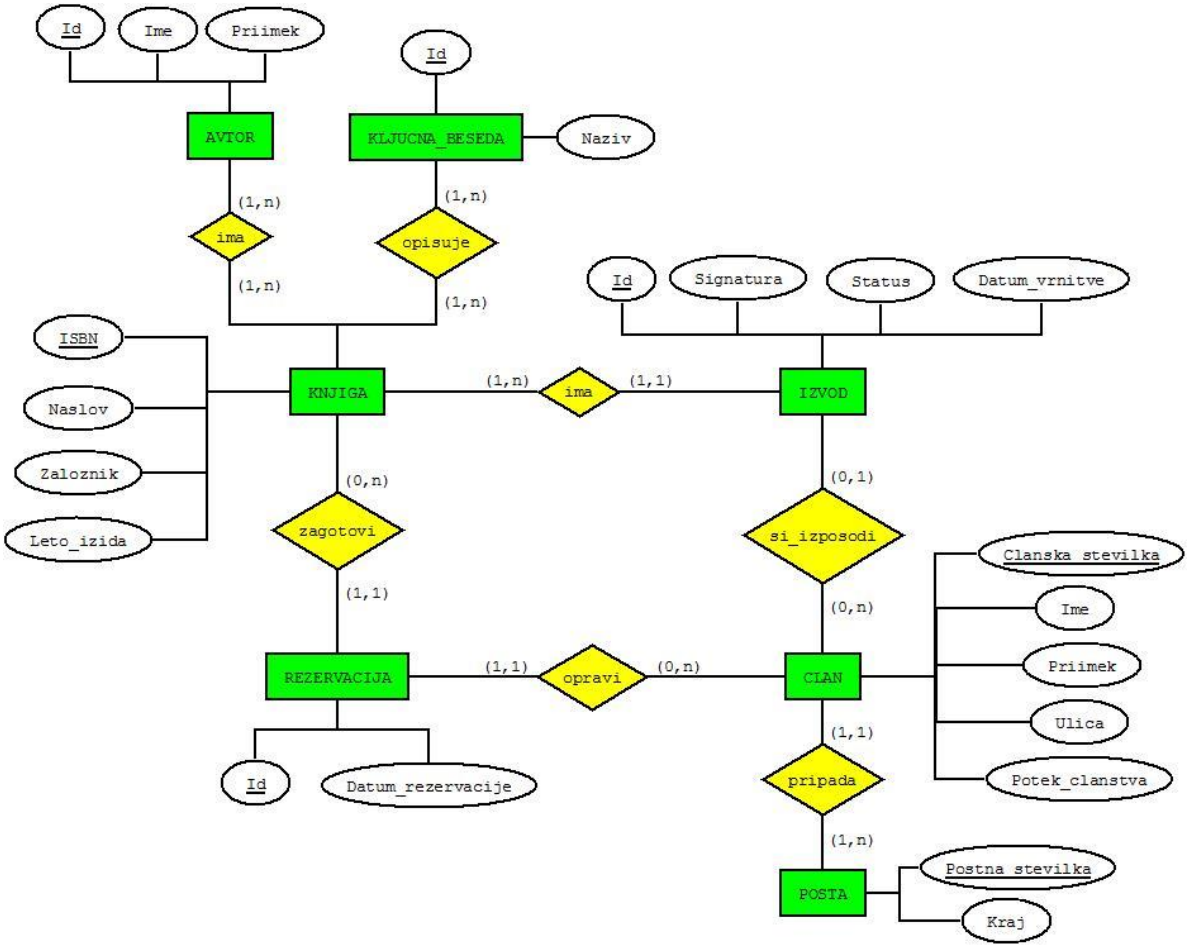
Slika 45: Opredelitev številnosti – notacija (min, max)

V nadaljevanju bomo za določitev številnosti uporabljali notacijo (*min*, *max*).

Opredelimo atribute entitetnim tipom in določimo primarne ključe. Atribut, ki označuje primarni ključ, je podčrtan.
Pri večini tipov je izbira ključev očitna. Pri tipu "REZERVACIJA" dodamo atribut »Id«, ker na podlagi atributa »Datum rezervacije«, ki ga ima omenjen tip, ne moremo identificirati posamezne rezervacije. Iz enakega razloga entitetnemu tipu »IZVOD« dodamo atribut »Id". Kot smo omenili, je signatura število, ki označuje mesto oziroma lokacijo, kje v knjižnici najdemo posamezni izvod. Na posamezni lokaciji imamo lahko več izvodov knjig, zato posameznega izvoda ne moremo identificirati na podlagi atributa »Signatura«. Enako velja, da na podlagi atributa »Status« in »Datum vrnitve« ne moremo identificirat posameznega izvoda. Zato dodamo atribut »Id«, na podlagi katerega bomo lahko identificirali posamezen izvod.

KNJIGA (ISBN, Naslov, Založnik, Leto izida)
AVTOR (Id, Ime, Priimek)
KLJUČNA BESEDA (Id, Naziv)
IZVOD (Id, Signatura, Status, Datum vrnitve)
ČLAN (Članska številka, Ime, Priimek, Ulica, Potek članstva)
POŠTA (Poštna številka, Kraj)
REZERVACIJA (Id, Datum rezervacije)

Naslednji korak je risanje ER-diagrama. Dobimo diagram, ki je predstavljen na sliki Slika 46.



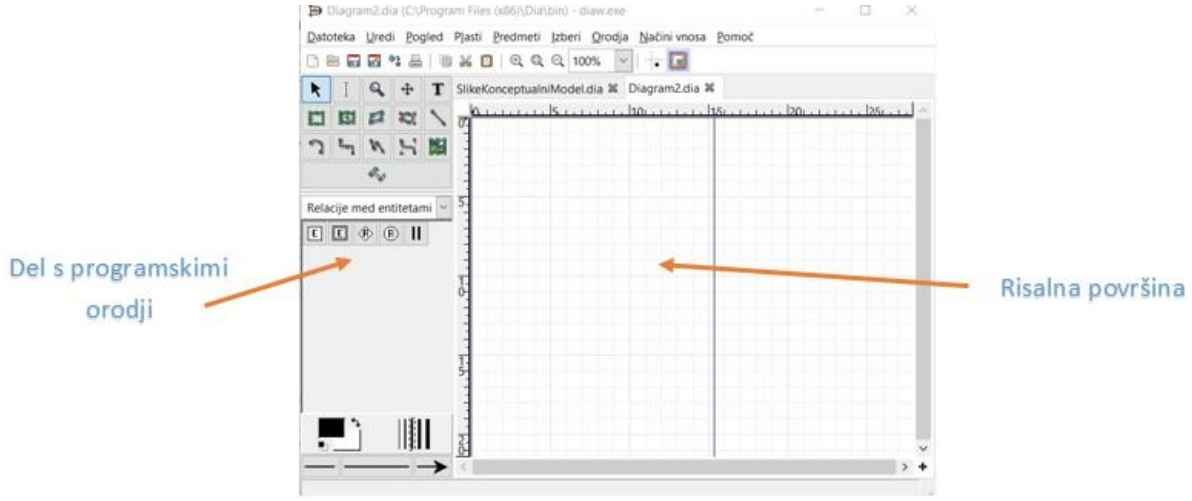
Slika 46: ER-diagram

Postopek risanja ER-diagrama z orodjem DIA Diagram Editor bomo predstavili v razdelku 2.5.

3.5 Izdelava ER-diagrama s pomočjo orodja DIA

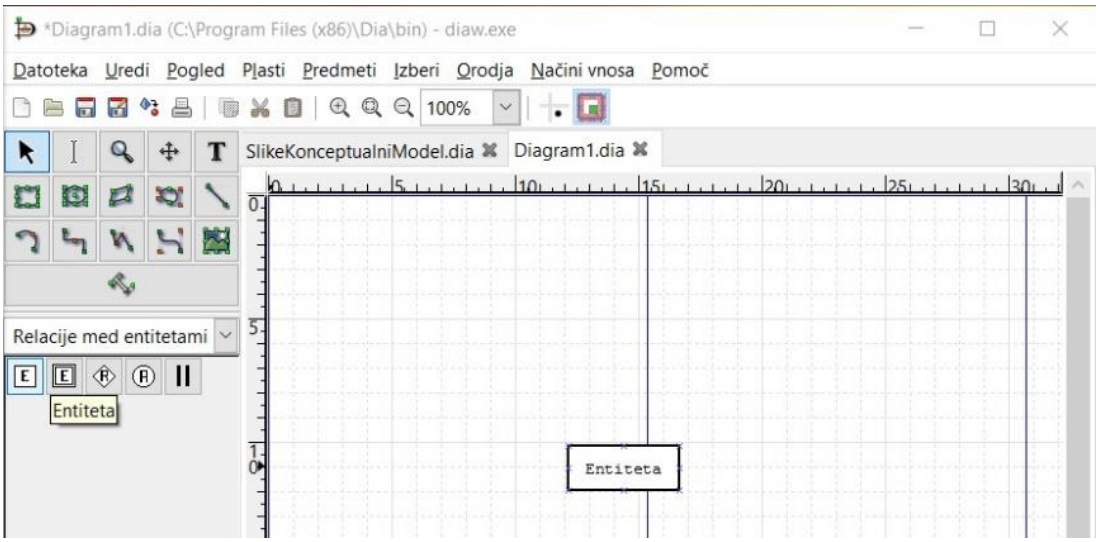
DIA je program za risanje strukturiranih diagramov. Razvil ga je Alexander Larsson. Ponuja več paketov z različnimi oblikami, kar omogoča risanje različnih tipov diagramov, kot so: diagram poteka, omrežni diagram, ER-diagram, UML-diagram, diagram vezij itd. Narisane diagrame lahko preprosto shranimo, natisnemo ali izvozimo v različnih formatih (oblikah), vključno z EPS, SVG, XFIG, WMF, PNG, JPEG itd. Izdelava ER-diagrama v orodju DIA Diagram Editor poteka na enak način kot sam postopek izdelave ER-modela. Najprej narišemo entitetne tipe, nato povežemo entitetne tipe, ki so v določenem razmerju in določimo številnosti razmerij. Entitetnim tipom dodamo atribute in primarne ključe. Pri risanju diagramov se praviloma izogibamo šumnikom in presledkom med besedami (presledek nadomestimo s podčrtajem).

Orodje DIA je sestavljeno iz dveh delov. V prvem (levem) delu so zapisana programska orodja za izdelavo diagramov. V drugem (desnem) je risalna površina, kjer izdelujemo in oblikujemo diagrame.



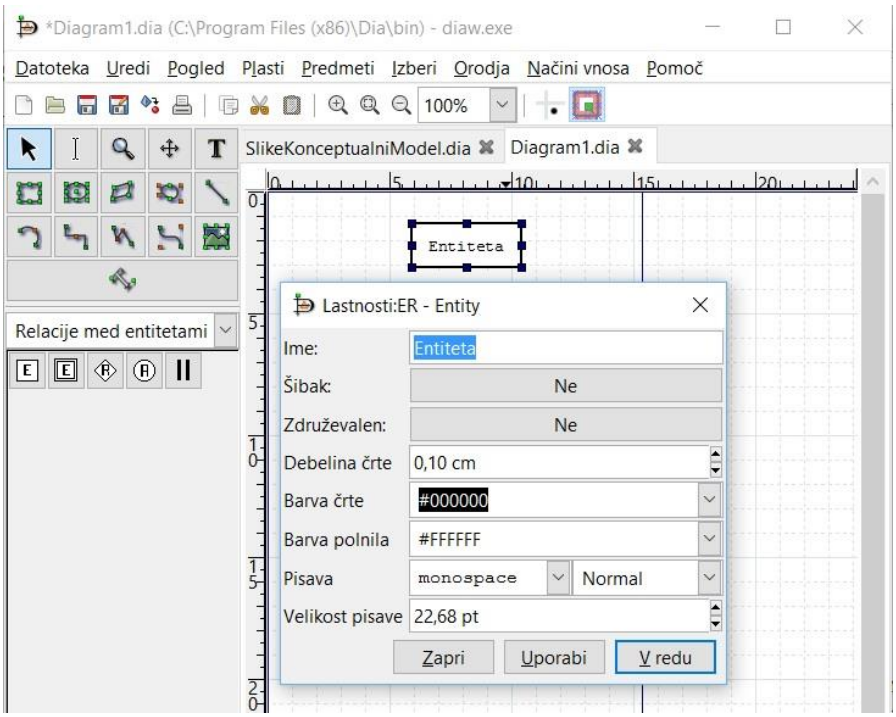
Slika 47: Orodje DIA

V orodju DIA gradnike oziroma oblike, ki sestavljajo diagram, izrišemo tako, da si najprej izberemo orodje glede na področje diagrama (npr. UML, diagram procesa, razmerje med entitetami …). V našem primeru torej izberemo `Relacije med entitetami`. Po izbiri vrste diagrama se prikažejo ustrezni gradniki, s katerim izdelamo želen diagram. Tako se po izbiri orodja `Relacije med entitetami` prikažejo gradniki: entiteta, šibka entiteta, relacija, atribut in sodelovanje (povezava). S približevanjem miške posameznemu gradniku se nam izpiše njegov pomen, kar prikazuje Slika 48.



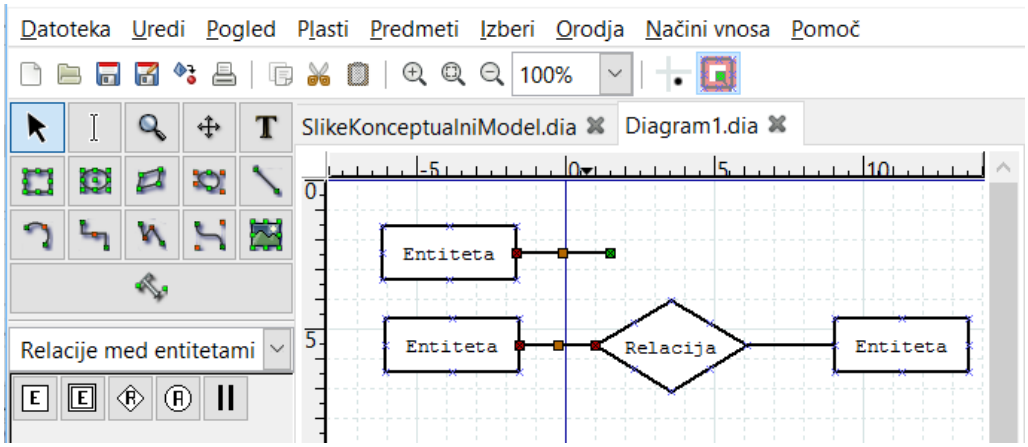
Slika 48: Posamezni predmeti oziroma oblike pri ER-diagramu

Če želimo določen gradnik dodati na risalno površino, preprosto kliknemo nanj in ga dodamo s klikom z miško. Gradnik premaknemo tako, da kliknemo kjerkoli znotraj gradnika in ga nato z miško povlečemo na želeno mesto. Če pa želimo gradnik oblikovati (npr. spremeniti ime, pisavo, velikost pisave, barvo črte, barvo polnila …) ga dvokliknemo ali pa uporabimo desni klik z miško in izberemo možnost `Lastnosti`, kot prikazuje Slika 49.



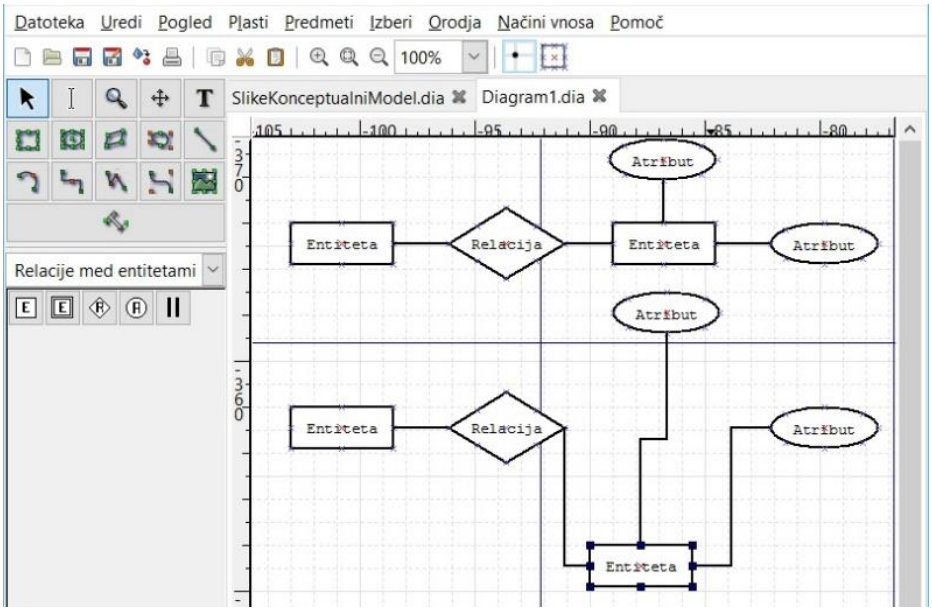
Slika 49: Prikaz lastnosti entitetnega tipa

Gradnike na risalni površini nato povežemo med seboj. Črte najdemo v levem delu, kjer so zapisana programska orodja, ali v menijski vrstici v zavihku `Orodja`. Gradniki in črte so opremljeni s točkami. Te točke nam omogočajo povezavo z drugimi gradniki. Ko je točka na črti povezana z nekim gradnikom, postane rdeča (rdeč kvadratek), v nasprotnem primeru pa je zelena (zelen kvadratek).
V našem primeru bomo med seboj povezovali entitetne tipe. Nato bomo vsak entitetni tip povezali še s pripadajočimi atributi. Entitetne tipe med seboj povežemo z relacijo, kot prikazuje Slika 50.



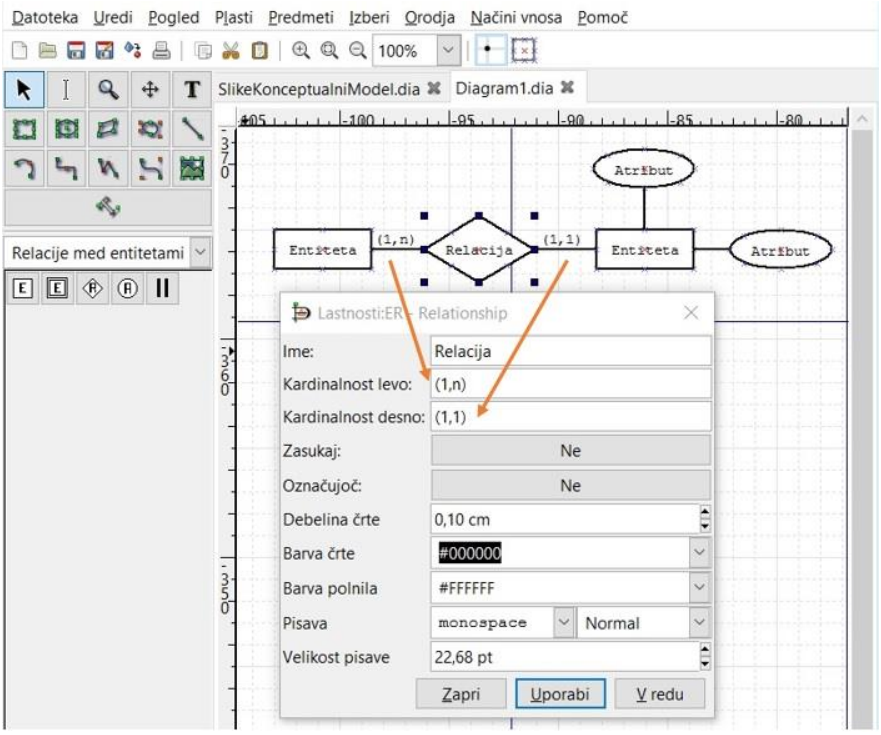
Slika 50: Povezovanje gradnikov

V primeru, da želimo prestaviti celoten entitetni tip, ki smo ga predhodno že povezali z razmerjem z drugim tipom, lahko to storimo, saj nam povezave, ki smo jih naredili, »ostanejo«. To lahko vidimo na sliki Slika 51.



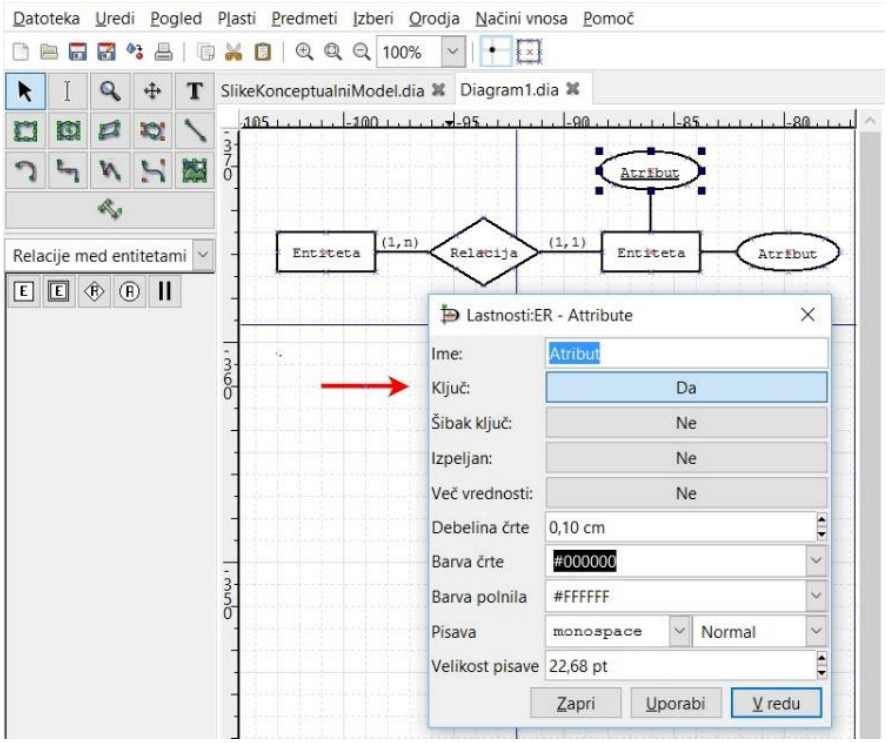
Slika 51: Prikaz premika entitetnega tipa

Številnost oziroma kardinalnost razmerja v orodju DIA določimo tako, da dvokliknemo na relacijo oziroma uporabimo desni klik z miško in izberemo `Lastnosti`. Odpre se nam okno, kjer v polje `Kardinalnost levo` in polje `Kardinalnost desno` vpišemo ustrezno številnost razmerja. Na sliki Slika 52 je prikazan primer opredelitve številnosti z notacijo (*min*, *max*).



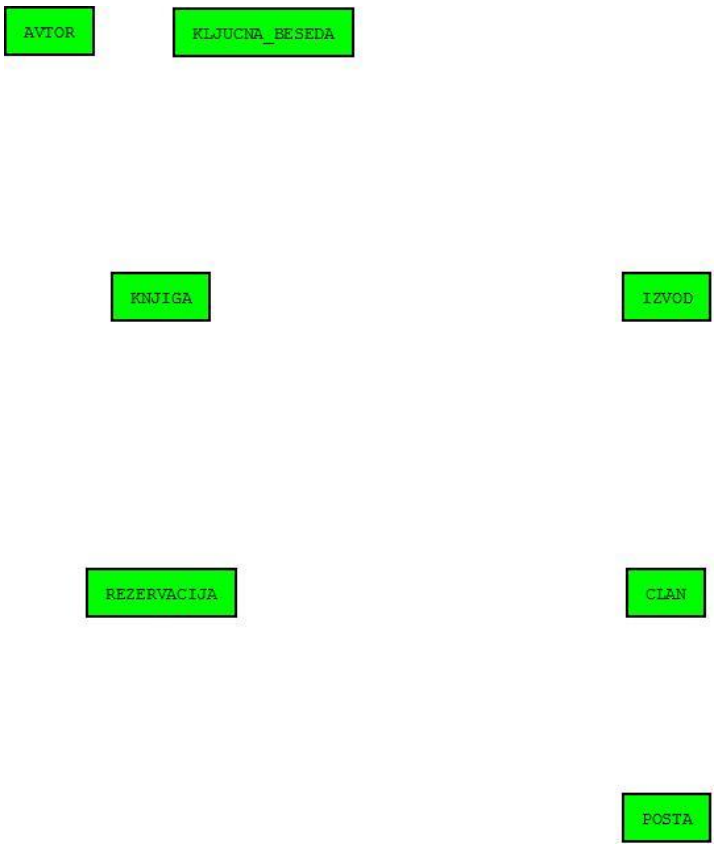
Slika 52: Prikaz določitve številnosti razmerja

Atribute, ki sestavljajo primarni ključ, določimo tako, da odpremo lastnosti atributa (bodisi z dvoklikom na atribut ali z desnim klikom na atribut in izbiro `Lastnosti`) in pri ključu označimo »Da«. Atribut, ki predstavlja primarni ključ, postane podčrtan, kot prikazuje Slika 53.



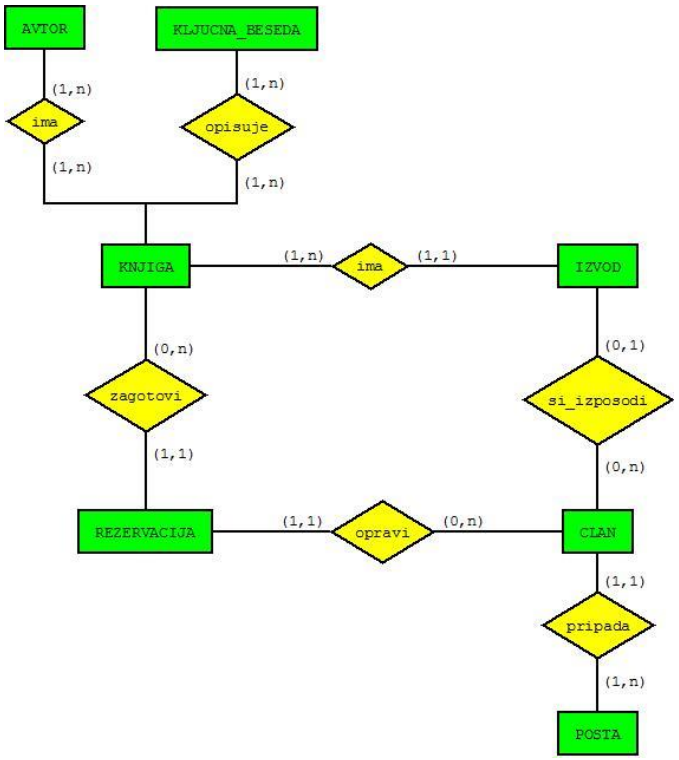
Slika 53: Prikaz določitve primarnega ključa

Oglejmo si sedaj izris ER-diagrama v orodju DIA za primer poenostavljene podatkovne baze knjižnice. Najprej v diagram vnesemo vse entitetne tipe. Kot vidimo (Slika 54), smo jih v DIA obarvali z zeleno.



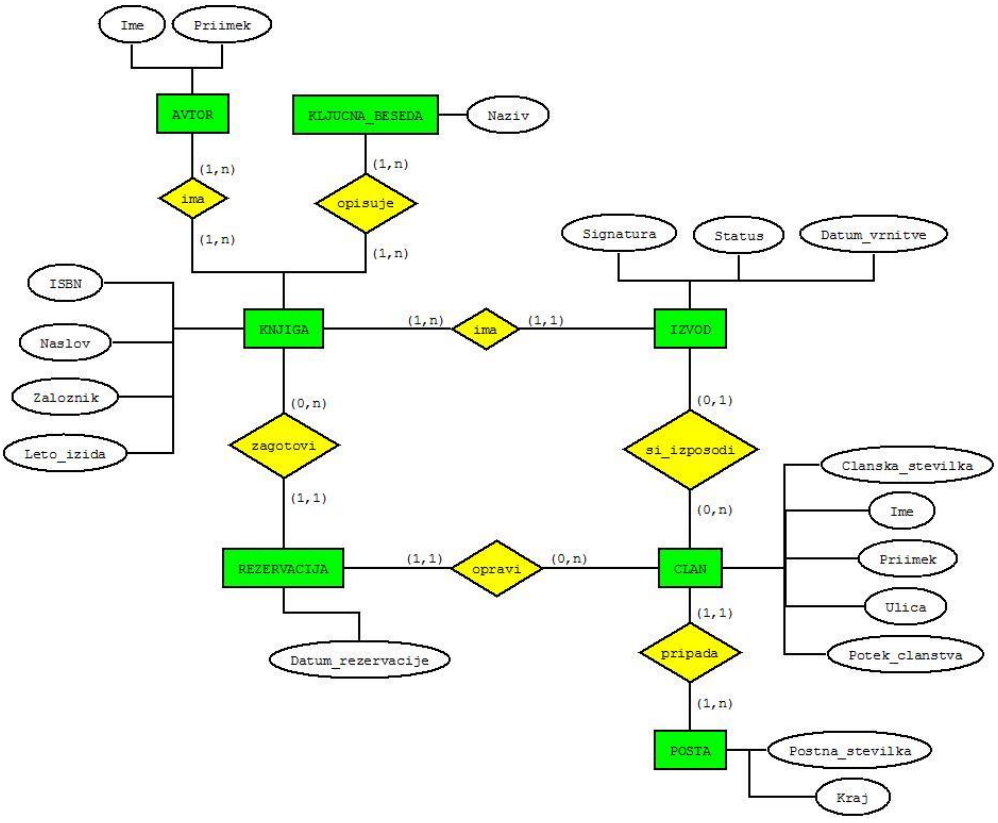
Slika 54: Diagram z dodanimi entitetni tipi

Nato dodamo ustrezna razmerja med entitetnimi tipi, ki jim po opisanem postopku določimo številnost (Slika 55).



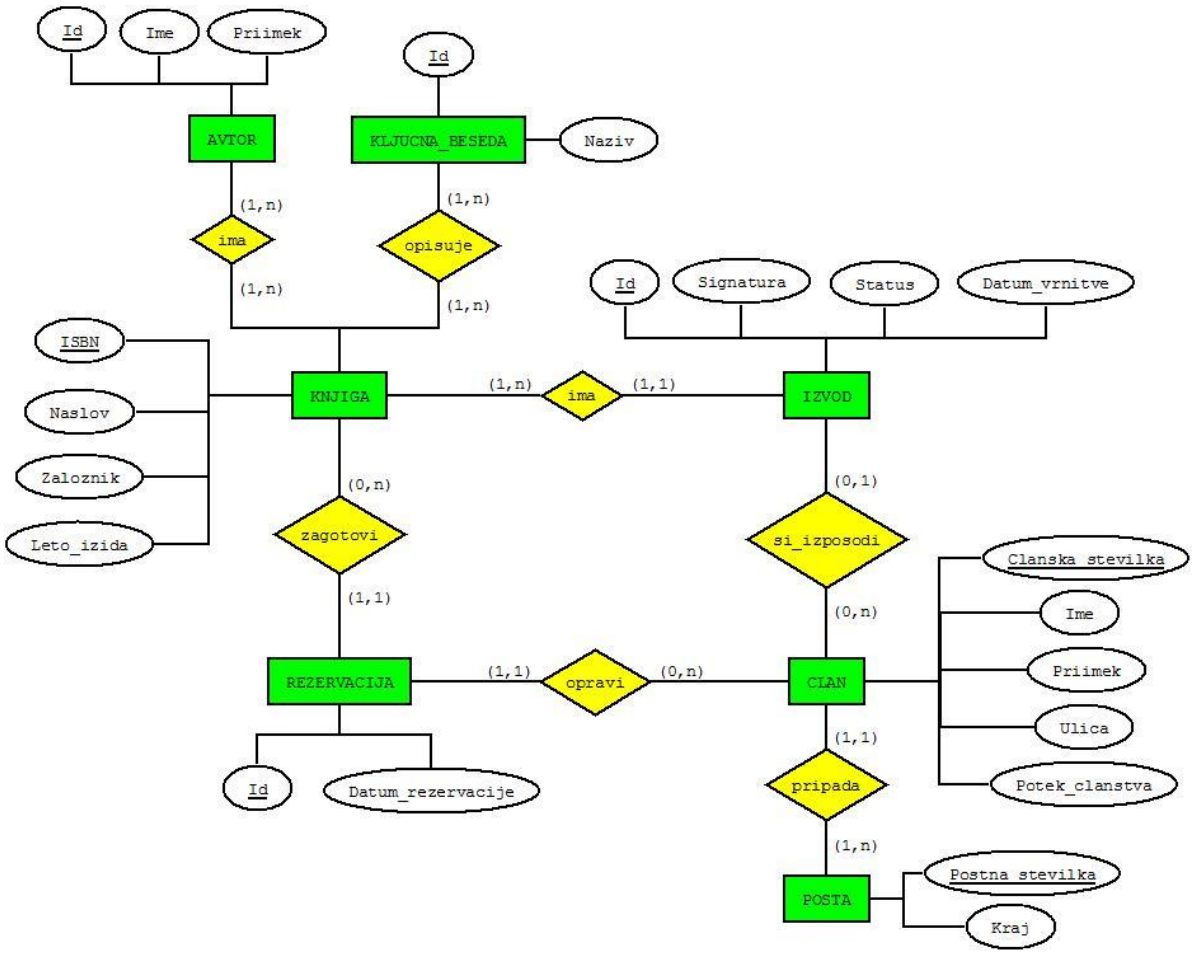
Slika 55: Diagram z dodanimi razmerji in številnost razmerij

Entitetnim tipom dodamo še pripadajoče atribute (Slika 56).



Slika 56: Diagram z dodanimi atributi

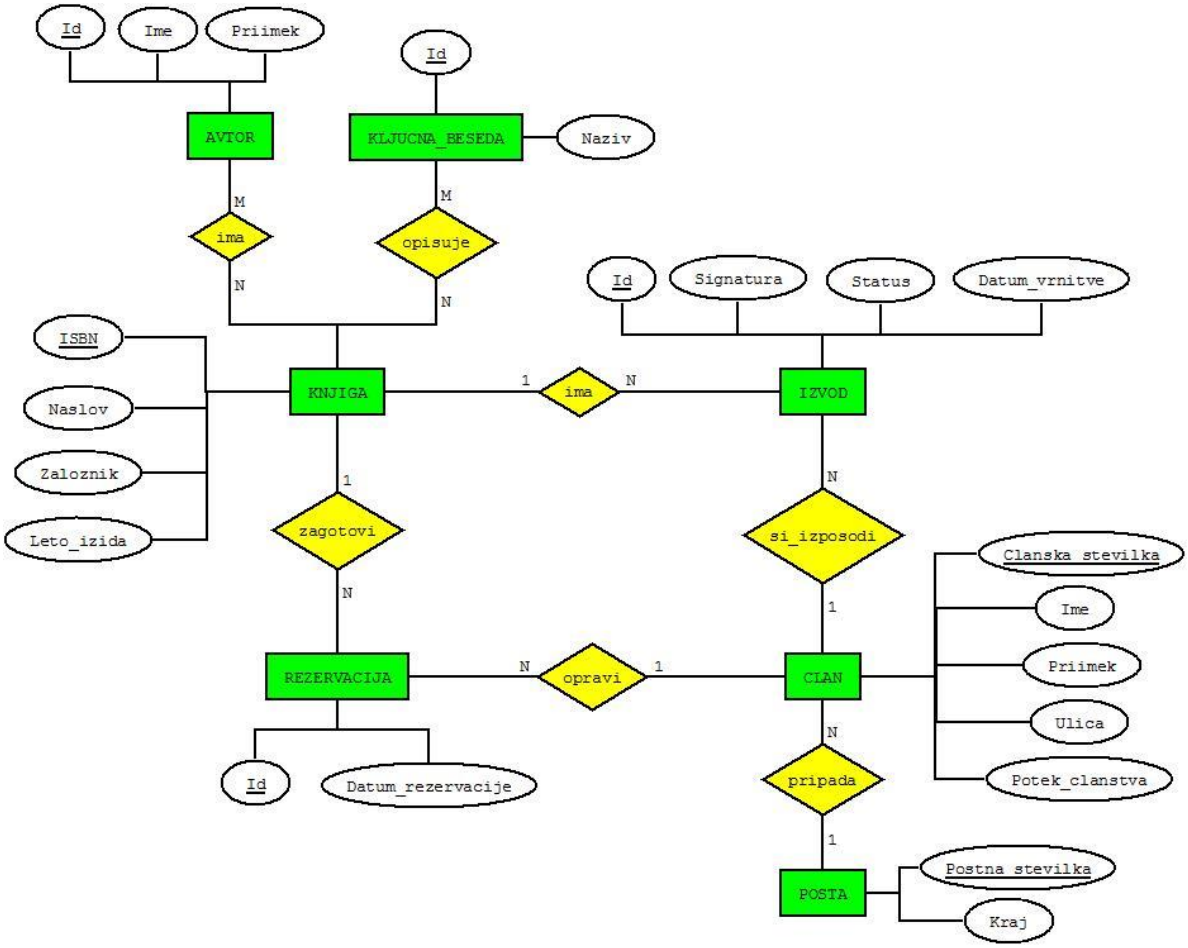
Potem, ko določimo še primarne ključe posameznega entitetnega tipa, smo dobili ustrezen ER-diagram (Slika 57).



Slika 57: Diagram z dodanimi primarnimi ključi

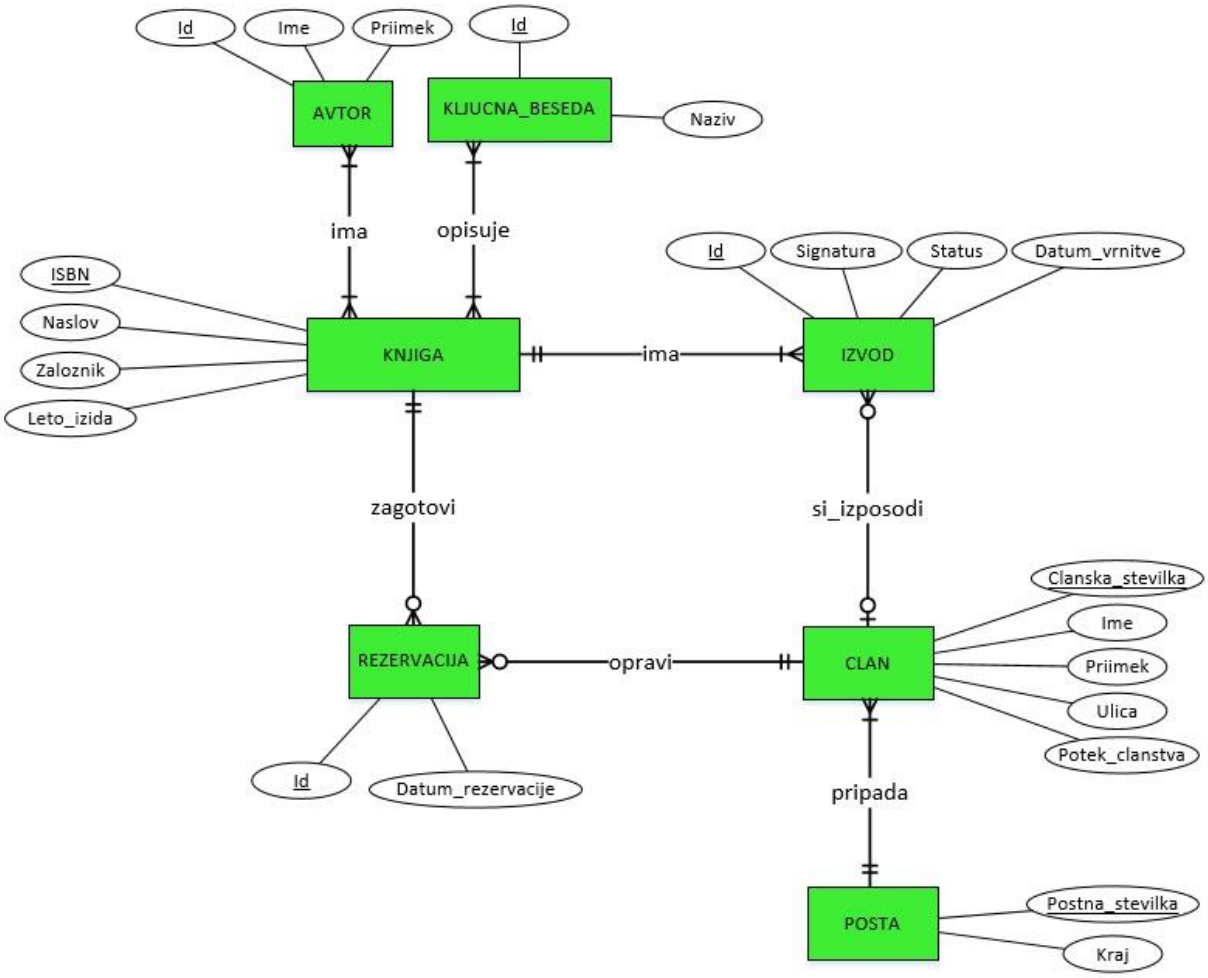
Vidimo, da lahko iz njega razberemo praktično vse ključne zahteve, tudi če ne poznamo uporabniških zahtev. Iz opredeljene številnosti za razmerje »si izposodi« npr. vidimo, da si vsak član lahko izposodi več izvodov. Pri razmerju »ima«, ki povezuje entitetna tipa »KNJIGA« in »AVTOR«, lahko iz minimalne vrednosti številnosti razberemo, da ima vsaka knjiga vsaj enega avtorja. Če pogledamo relacijo »ima« v nasprotni smeri, pa lahko iz maksimalne vrednosti številnosti razberemo, da je nekdo lahko avtor več knjig. Na podlagi atributov pri entitetnem tipu »ČLAN« pa lahko razberemo, katere podatke o članih shranjuje knjižnica.

Če bi želeli diagram narisati s pomočjo Chenove notacije, bi bil diagram videti tako, kot prikazuje Slika 58.



Slika 58: ER-diagram knjižnice s Chenovo notacijo

Orodje DIA ne omogoča izrisa ER-diagrama v notaciji »vranjih nog«, zato smo diagram na sliki Slika 59 narisali v orodju Microsoft Visio.



Slika 59: ER-diagram knjižnice z notacijo "vranjih nog"

S končnim diagramom (predstavljen na sliki Slika 57) lahko izvedemo nov razgovor z uporabniki. Pogovori z uporabniki so takrat lažji, saj na podlagi diagrama oziroma slike lahko hitro ugotovijo morebitne napake in pomanjkljivosti. Na primer, naročnik lahko ugotovi, da so pri knjigah pozabili na ilustratorja. Ugotovijo lahko tudi, da so pozabili na pravilo, da si član ne sme izposoditi več kot 10 izvodov sočasno in ne sme imeti sočasno rezerviranih več kot 3 knjige. Ali pa naročnik doda zahtevo, da je potreben še podatek o tem, koliko časa je nekdo že član. Tukaj se potem pojavijo številna vprašanja, npr. kako hraniti te podatke? Ali je bolje, da imamo zabeležen datum ali vsaj leto vpisa? Ali je bolje, da te podatke hranimo kot število let? Uporabniki imajo lahko tudi zahtevo, da želijo za vsako knjigo vedeti, kdo jo je imel izposojeno v določenem obdobju. V tem primeru bi morali diagram v precejšnjem delu popraviti in ga ponovno izrisati, saj bi morali za vsako knjigo hraniti tudi zgodovino izposoje.

Primer kaže, da določene spremembe, ki jih zahteva naročnik, lahko opravimo oziroma dodamo enostavno, spet druge pa dejansko zahtevajo ponovno izdelavo precejšnjega dela modela (kot v primeru potrebe po informaciji, kdaj je bila kakšna knjiga izposojena in komu).

4 LOGIČNI MODEL

Logično načrtovanje podatkovne baze nastopi po konceptualnem modeliranju. Cilj logičnega načrtovanja je izdelava logičnega modela. Logični model je model podatkov, ki upošteva vrsto podatkovne baze. Poznamo naslednje vrste podatkovnih baz: hierarhične, mrežne, relacijske in objektne. Če si za ciljno podatkovno bazo izberemo relacijsko podatkovno bazo, potem govorimo o relacijskem podatkovnem modelu, o katerem bo govora tudi v tem poglavju. Pred samim načrtovanjem logičnega modela je torej treba vedeti, kateri tip podatkovne baze in sistem za upravljanje podatkovnih baz ustreza organizaciji oziroma uporabnikom podatkovne baze. Relacijski podatkovni model je grafično predstavljen v obliki relacijskega podatkovnega diagrama, ki je izdelan na podlagi izbranega SUPB.

Do logičnega modela relacijske podatkovne baze lahko pridemo na dva načina. Prvi način izhaja direktno iz uporabniških zahtev. Kot smo že omenili, so uporabniške zahteve najpogosteje podane v obliki opisa problema v naravnem jeziku (v pisni ali ustni obliki), z obrazci (dokumentacije) in preko datotečne strukture datotek, v katerih uporabniki hranijo podatke. Drugi način pa izhaja iz konceptualnega modela, v našem primeru ER-modela, ki smo ga izdelali v fazi konceptualnega načrtovanja. V primeru, da izhajamo iz konceptualnega modela, je pomembno, da skupaj z uporabniki podatkovne baze konceptualni model v fazi konceptualnega načrtovanja kar se da natančno dodelamo. Najboljše izhodišče za izdelavo logičnega modela je konceptualni model v obliki ER-diagrama. Izdelava logičnega modela direktno iz opisa problema ni priporočljiva, saj lahko naredimo veliko več napak kot pa v primeru, če izhajamo samo iz ER-diagrama.
Ker smo se v diplomski nalogi omejili na relacijske podatkovne baze in pri konceptualnem načrtovanju na entitetno-relacijski model (ER-model), bomo v nadaljevanju obravnavali le pretvorbo oziroma preslikavo ER-modela v relacijski podatkovni model.

Velikokrat se srečamo z vprašanjem, zakaj potrebujemo logični model, ko že imamo konceptualni model? Bistvena razlika je v tem, da SUPB ne podpirajo neposredno konceptualnega modela (ER-modela), ampak podpirajo logični model (relacijski podatkovni model).
Logični podatkovni model določa logično strukturo baze podatkov. Opisuje podatke na način, ki ustreza ciljnemu sistemu za upravljanje podatkovnih baz. Naš ciljni sistem za upravljanje podatkovnih baz je relacijski, zato bomo v nadaljevanju opisali logični model, ki ustreza relacijskemu podatkovnemu modelu.

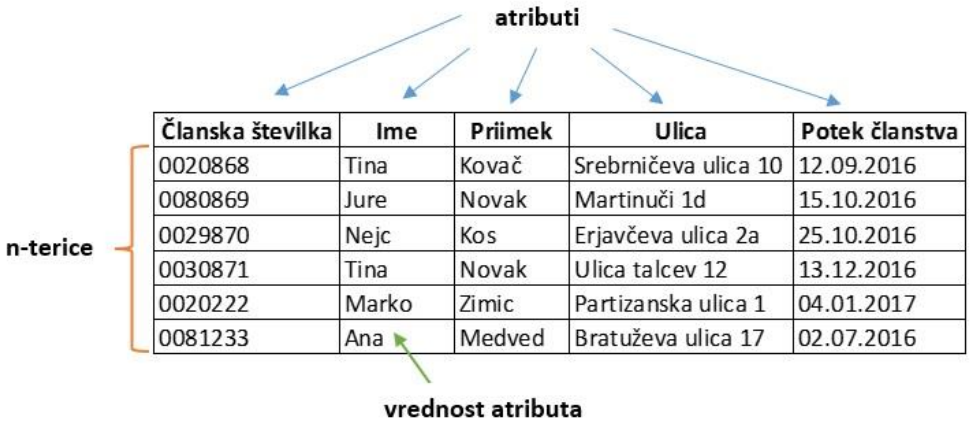
4.1 Relacijski podatkovni model

Relacijski podatkovni model je zgrajen na matematičnem konceptu relacij, ki jih fizično predstavljamo s tabelami. Vsaka tabela je sestavljena iz vrstic in stolpcev. Med tabelami obstajajo logične povezave, ki jih izvedemo s pomočjo uporabe enakih vrednosti v stolpcih tabel. Relacijski podatkovni model je ravno tako kot ER-model enostaven za razumevanje. Razlika med modeloma je v tem, da SUPB neposredno podpira relacijski podatkovni model, ne pa tudi ER-modela. Hkrati pa relacijski podatkovni diagram prikazuje več podrobnosti kot ER-diagram.
Vsebina tabel je uporabnikom dostopna s pomočjo povpraševalnih jezikov, ki so osnovani na relacijski algebri ali pa na relacijskem računu. Najpogosteje uporabljen povpraševalni jezik je jezik SQL. SQL (Structured Query Language) je strukturiran povpraševalni jezik, ki poleg dostopa do vsebine tabel omogoča tudi definiranje podatkovnih struktur, specifikacijo omejitev podatkov, podeljevanje dostopnih pravic in zaščito celovitosti podatkovne baze. Podrobneje bomo SQL predstavili v nadaljevanju.

Pri relacijskem podatkovnem modelu se bomo srečali z nekaterimi pojmi in koncepti, kot so: relacija, atribut, domena atributa, relacijska shema, ključ relacije in integritetne omejitve.

4.1.1 Relacija in atribut

Relacija je predstavljena z dvodimenzionalno tabelo, ki jo sestavlja niz poimenovanih stolpcev in neimenovanih vrstic. Vsak stolpec v tabeli predstavlja atribut relacije, vsaka vrstica pa predstavlja zapis, ki vsebuje podatke o nekem primerku entitete. Vrsticam v tabeli pravimo tudi n-terice. Vrednosti atributov za določeno n-terico so zapisane v presečišču vrstic in stolpcev, tako imenovanih celicah tabele. Primer relacije in njenih atributov prikazuje Slika 60, na kateri je prikazana relacija »ČLAN«. Relacija »ČLAN« prikazuje člane knjižnice, o katerih hranimo naslednje podatke: člansko številko, ime, priimek, ulica in potek članstva. Število atributov (stolpcev) predstavlja stopnjo relacije. Število n-teric (vrstic) relacije predstavlja številnost oziroma kardinalnost relacije. Na primer stopnja relacije »ČLAN«, predstavljene na sliki Slika 60,**Error! Reference source not found.** je 5, številnost relacije »ČLAN«, pa je 6.



Članska številka	Ime	Priimek	Ulica	Potek članstva
0020868	Tina	Kovač	Srebrničeva ulica 10	12.09.2016
0080869	Jure	Novak	Martinuči 1d	15.10.2016
0029870	Nejc	Kos	Erjavčeva ulica 2a	25.10.2016
0030871	Tina	Novak	Ulica talcev 12	13.12.2016
0020222	Marko	Zimic	Partizanska ulica 1	04.01.2017
0081233	Ana	Medved	Bratuževa ulica 17	02.07.2016

Slika 60: Shematski prikaz relacije "ČLAN"

Res je, da je relacija zaradi lažje razumljivosti predstavljena z dvodimenzionalno tabelo, vendar pa niso vse tabele tudi relacije. Če povzamemo po knjigi (Connolly, in drugi, 2005), imajo relacije naslednje lastnosti, ki jih razlikujejo od ostalih tabel:

- vsaka relacija ima svoje ime;
- vsaka »celica« relacije ima natanko eno atomarno (nedeljivo) vrednost (lahko je tudi nedefinirana – Null);
- vsak atribut ima svoje ime, ki je različno od imen drugih atributov relacije;
- vrednosti atributov imajo isto domeno (zalogo vrednosti);
- vrstni red atributov (stolpcev) ni pomemben;
- vrstni red n-teric (vrstic) ni pomemben;
- vsaka vrstica je različna, ni podvajanja njihovih vrednosti.

4.1.2 Domena atributa

Vnosi v tabelo (podatki) so podatkovne vrednosti iz neke določene zaloge vrednosti, ki je opredeljena za vsak atribut posebej (Projekt Colos). To pomeni, da moramo za vsak atribut v relaciji določiti domeno. Domena je množica dovoljenih vrednosti atributa. Vrednost atributa je podatkovni element nekega podatkovnega tipa, npr. znakovni, številski, datumski itd. Atributi pa lahko imajo tudi določene omejitve, kot so: le tri vrednosti, točno 10 znakov, le pozitivna števila itd. Zato bi lahko rekli, da domena predstavlja zalogo vrednosti, ki jih lahko zavzame atribut, za katerega pa lahko opredelimo tudi

dodatne omejitve. Tem omejitvam pravimo omejitve domen in podrobneje jih bomo predstavili v nadaljevanju. Izbor možnih zalog vrednosti določa sistem za upravljanje podatkovnih baz (SUPB). Različni SUPB-i ponujajo različne tipe podatkov. Osnovni tipi podatkov, ki jih ponujajo SUPB-i, so nizi znakov, števila (cela, realna), datum, čas itd. Poglejmo si primer relacije »ČLAN« na sliki Slika 60 in določimo domeno za atribute »Članska številka«, »Ime« in »Potek članstva«. Članska številka je množica sedemmestnih znakov, ki pa morajo biti sestavljeni izključno iz števk, saj le na ta način predstavljajo veljavno člansko številko. Ime je množica znakov, ki predstavlja imena članov knjižnice. Potek članstva je množica datumov, ki povedo, kdaj članom knjižnice poteče članstvo.

4.1.3 Relacijska shema

Vse sestavine relacijske podatkovne baze morajo biti opisane. Opis relacijske podatkovne baze je shema podatkovne baze. Shema podatkovne baze je sestavljena iz shem posameznih relaciji (Projekt Colos). Vsaki relaciji tako pripada relacijska shema. Kot smo že omenili, relacijsko podatkovno bazo prikazujemo kot množico tabel. Vsaka tabela sestoji iz: imena, čelne vrstice in podatkovnih vrstic. Ime predstavlja naslov tabele. Čelna vrstica pojasnjuje pomen posameznih stolpcev (atributov) v tabeli. Podatkovne vrstice predstavljajo množico zapisov (vrstic) v tabeli. Če relacijo predstavimo kot tabelo, prevzeme relacijska shema vlogo naslova in čelne vrstice v tabeli. Posamezna relacijska shema tako opredeljuje ime tabele, imena atributov in njihove domene. Poglejmo si relacijsko shemo za primer relacije »ČLAN«, predstavljene na sliki Slika 60.

Primer relacijske sheme ČLAN:

Ime tabele: ČLAN
Atributi tabele ČLAN: Članska številka, Ime, Priimek, Ulica, Potek članstva
Domena in omejitve atributov tabele ČLAN:

- *Članska številka:* znakovni podatkovni tip; zahtevan vnos 7 števk, posamezna vrednost se v tabeli ne ponovi
- *Ime:* znakovni podatkovni tip
- *Priimek:* znakovni podatkovni tip
- *Ulica:* znakovni podatkovni tip; zahtevan vnos v obliki »Ime_ulice hišna_številka« (Primer: Srebrničeva ulica 10)
- *Potek članstva:* datumski podatkovni tip; zahtevan vnos v obliki »dd.mm.llll« (Primer: 13.12.2016)

Vrstice tabele so opredeljene z množico zapisov. Relacija »ČLAN« na sliki Slika 60 ima 6 zapisov oziroma vrstic. Posamezna vrstica v tabeli lahko predstavlja dovoljen (legalen) zapis, lahko pa nedovoljen (nelegalen) zapis. Tako lahko zapise preprosto razdelimo v dve množici, v tiste, ki predstavljajo resnična dejstva, in tiste, ki predstavljajo neresnična dejstva. Če so vrednosti vseh atributov skladne z vsemi omejitvami, ki so navedene v relacijski shemi, potem vrstica predstavlja resnična dejstva (dovoljen zapis). V nasprotnem primeru predstavlja neresnična dejstva (nedovoljen zapis). Na primer, če za atribut »Ime« vnesemo vrednost »Maja«, le-ta predstavlja resnično dejstvo, saj je podatek res znakovnega tipa. Če pa vnesemo npr. vrednost 12345 ta predstavlja neresnično dejstvo, saj smo v relacijskih shemi »ČLAN« navedli, da je ime znakovni tip podatka in zato ne moremo namesto znakovnega vnesti številski tip. Za to, ali so zapisi v posamezni relaciji (tabeli) dovoljeni oziroma nedovoljeni, v praksi skrbi sistem za upravljanje podatkovnih baz. Zavedati se moramo, da SUPB lahko skrbi le za pravilnost tipa podatka in za tisto pravilnost, ki jo lahko izrazimo v obliki pravil oziroma omejitev, ne pa tudi za »vsebinske« napake. Na primer SUPB ne more vedeti, da če namesto imena 'Maja' vnesemo ime 'Maaj' ali pa 'mcjxh' to pomeni nepravilno vrednost. Čeprav smo v obeh primerih

vnesli res znakovni tip podatka, to ne pomeni, da smo vnesli tudi pravilno ime. To so »vsebinske« napake, za katere mora skrbeti tako upravitelj kot tudi uporabniki podatkovne baze.

4.1.4 Ključ relacije

Vsaka relacija je definirana kot množica enoličnih zapisov. Vsaka vrstica relacije je torej različna. To pomeni, da ne moreta imeti dva zapisa v relaciji hkrati enake kombinacije vrednosti za vse podmnožice atributov. Na primer v relaciji »ČLAN«, predstavljene na sliki Slika 61, nimamo vrstice, v kateri bi vse vrednosti atributov bile enake kot v kakršnikoli drugi vrstici. Da lahko identificiramo posamezno vrstico relacije, moramo vsaki relaciji določiti ključ. O ključih smo že govorili, ko smo opisovali in gradili ER-model. Tam smo definirali pojme kandidat za ključ, primarni ključ in sestavljen ključ. Tukaj bomo uvedli še nekaj dodatnih pojmov (superključ, nadomestni ključ, tuji ključ), ker so tudi ti pomembni del pri gradnji relacijskega podatkovnega modela. Tako kot pri definiranju entitetnega ključa bomo tudi tu sledili definicijam, ki so navedene v knjigi (Connolly, in drugi, 2005).
Pri ER-modelu smo omenili, da je kandidat za ključ minimalna množica atributov, ki so potrebni za enolično identifikacijo. Kako pridemo do te minimalne množice, bomo razložili sedaj. Najprej je treba definirati pojem superključ. Atribut ali množica atributov, ki enolično določajo vrstico relacije, se imenuje **superključ**. Superključ relacije torej enolično identificira vsako vrstico relacije. Vsaka relacija ima vsaj en privzeti superključ, in sicer množico vseh atributov. Pri predstavitvi superključa se bomo navezali na primer, ki smo ga obravnavali pri definiranju ključev v ER-modelu. Primer superključa bomo torej prikazali na relaciji »ČLAN«, ki je predstavljena na sliki Slika 61. Relacijo »ČLAN« sestavljajo atributi »Članska številka«, »Ime«, »Priimek«, »Ulica« in »Potek članstva«. Superključ relacije »ČLAN« je lahko množica atributov »Članska številka«, »Ime« in »Priimek«. Na podlagi vseh treh atributov skupaj namreč lahko identificiramo posameznega člana knjižnice. Ravno tako je kombinacija atributov »Članska številka« + »Ime« in kombinacija »Članska številka« + »Priimek« superključ relacije »ČLAN«. Superključ pa je lahko tudi atribut »Članska številka« ali kar množica vseh atributov: »Članska številka«, »Ime«, »Priimek«, »Ulica«, »Potek članstva«, saj lahko tudi na podlagi atributa »Članska številka« ali na podlagi množice vseh atributov identificiramo posameznega člana. Superključ relacije »ČLAN« pa ne more biti atribut »Ime«, saj imamo lahko člane z enakim imenom. Zato samo na podlagi imena ne moremo enolično opredeliti posameznega člana. Enako velja za atribute »Priimek«, »Ulica« in »Potek članstva«.

Članska številka	Ime	Priimek	Ulica	Potek članstva
0020868	Tina	Kovač	Srebrničeva ulica 10	12.09.2016
0080869	Jure	Novak	Martinuči 1d	15.10.2016
0029870	Nejc	Kos	Erjavčeva ulica 2a	25.10.2016
0030871	Tina	Novak	Ulica talcev 12	13.12.2016
0020222	Marko	Zimic	Partizanska ulica 1	04.01.2017

Slika 61: Relacija "ČLAN"

Superključ lahko vsebuje tudi atribute, ki niso nujno potrebni za enolično identifikacijo. Nas pa zanimajo samo tisti superključi, ki vsebujejo le minimalno množico atributov, ki so potrebni za enolično identifikacijo. Denimo, da je superključ v relaciji »ČLAN« sestavljen iz atributov »Članska številka«, »Ime« in »Priimek«. Iz te množice atributov bi lahko odstranili atribut »Ime«, saj lahko posamezno vrstico v relaciji »ČLAN« identificiramo samo na podlagi atributa »Članska številka« in atributa »Priimek«. Na enak način lahko odstranimo tudi atribut »Priimek«. Tako smo iz superključa, ki je bil sestavljen iz množice treh atributov (»Članska številka« + »Ime« + »Priimek«), dobili minimalno

množico, ki jo sestavlja atribut »Članska številka«. Kot smo omenili, tej minimalni množici pravimo kandidat za ključ. Atribut »Članska številka« je torej superključ, hkrati pa tudi kandidat za ključ.
Zavedati se moramo, da trenutnega primera relacije ne moremo uporabiti kot »dokaz«, da je atribut ali kombinacija atributov kandidat za ključ, saj dejstvo, da v primeru, ki ga proučujemo, ni podvojenih vrednosti atributov, ne garantira, da se te vrednosti ne morejo podvojiti.

Lahko rečemo, da je kandidat za ključ superključ, ki mu ne moremo odstraniti nobenega atributa. Povedali smo, da imamo lahko več kandidatov za ključ in da izmed vseh kandidatov za ključ izberemo enega in ga označimo kot primarni ključ. Kot smo omenili, imamo v primeru relacije »ČLAN« dva kandidata za ključ (»Članska številka« in »Ime« + »Priimek«). Za primarni ključ smo izbrali atribut »Članska številka«. Tiste kandidate za ključ, ki jih nismo izbrali za primarni ključ, imenujemo **nadomestni ključi**. V primeru relacije »ČLAN« je torej kandidat za ključ, ki ga tvorita atributa »Ime« in »Priimek«, nadomestni ključ.
Podatki, shranjeni v eni relaciji, so včasih lahko le taki, kot so že shranjeni v neki drugi relaciji. Temu rečemo, da je podatek v neki relaciji **tuji ključ** v drugi relaciji. Tuji ključ torej povezuje dve relaciji med seboj. Slika 62 prikazuje relacijo "ČLAN" in relacijo "POŠTA". Vidimo, da so vrednosti atributa "Poštna številka" v relaciji »ČLAN« take, kot so že shranjene v relaciji »POŠTA«. Zato pravimo, da je atribut »Poštna številka« v relaciji »ČLAN« tuji ključ. V relaciji »POŠTA« pa primarni ključ predstavlja atribut »Poštna številka«. Preko atributa »Poštna številka« torej povezujemo relaciji »ČLAN« in »POŠTA«.

Članska številka	Ime	Priimek	Ulica	Poštna številka	Potek članstva
0020868	Tina	Kovač	Srebrničeva ulica 10	5210	12.09.2016
0080869	Jure	Novak	Martinuči 1d	5292	15.10.2016
0029870	Nejc	Kos	Erjavčeva ulica 2a	5000	25.10.2016
0030871	Tina	Novak	Ulica talcev 12	5210	13.12.2016
0020222	Marko	Zimic	Partizanska ulica 1	5000	04.01.2017

Poštna številka	Kraj
5210	Deskle
5292	Renče
5000	Nova Gorica

Slika 62: Povezava relacije "ČLAN" in relacije "POŠTA"

4.1.5 Integritetne omejitve

Integritetne omejitve so omejitve oziroma pogoji, ki jih morajo izpolnjevati podatki, ki so zapisani v podatkovni bazi. Zagotavljajo, da so zapisani podatki v podatkovni bazi točni in skladni. Poznamo več tipov integritetnih omejitev. V nadaljevanju bomo predstavili nekatere, ki jih upoštevamo pri relacijskem podatkovnem modelu.
Kot smo omenili pri domeni atributa, domena vključuje tudi omejitve, kot so: le pozitivna števila, točno 10 števk, vnos v obliki »dd.mm.llll«, le dve vrednosti itd. Te omejitve predstavljajo množico dovoljenih vrednosti za atribute relacije in jih imenujemo **omejitve domen**. Omejitve domene atributov za relacijo »ČLAN« smo predstavili že pri relacijskih shemi. Poglejmo si zdaj primer omejitve domen za atribute relacije »IZVOD«. Relacija »IZVOD«, predstavljena na sliki Slika 63, je sestavljena iz atributov: »Id«, »Signatura«, »Status« in »Datum vrnitve«. Atribut »Id« predstavlja zaporedno številko izvoda. Torej gre za številski tip podatka. Dejansko tu nimamo nobene konkretne omejitve domene. Lahko pa bi recimo določili, da izvode začnemo številčiti pri 1000. Torej bo vsaka zaporedna številka izvoda večja kot 1000. Atribut »Signatura« je število, predstavlja mesto oziroma lokacijo, kjer je posamezen izvod.

Je znakovni tip podatka, ki je sestavljen izključno iz števk. Atribut »Status« predstavlja prost oziroma izposojen izvod knjige. Gre za znakovni tip podatka. Kot vidimo na sliki Slika 63, ima lahko le dve privzeti vrednosti, in sicer: `prost` ali `izposojen`. Torej atribut »Status« omejimo le na dve vrednosti. Atribut »Datum vrnitve« predstavlja datum vrnitve izvoda v primeru, da je izvod izposojen. Kot vidimo na sliki Slika 63, v primeru, da je vrednost atributa »Status« `prost`, datuma vrnitve ni. V tem primeru vrednost datuma ne obstaja oziroma je ne moremo določiti. To označimo z vrednostjo Null. Nekateri SUPB nam omogočajo, da nastavimo omejitev, ki določa, da so možne le vrstice, kjer je v stolpcu »Status« vrednost `prost` in v stolpcu »Datum vrnitve« vrednost Null oziroma vrstice s stolpcem »Status« z vrednostjo `izposojen` in v stolpcu »Datum vrnitve« nek smiseln datum. Atribut »Datum vrnitve« je datumski tip podatka. Določimo lahko, v kakšni obliki ga bomo zapisovali. Na primer, če ga želimo zapisati kot dan, mesec in leto (npr. »10.05.2016«), bomo kot omejitev navedli, da mora biti vnos v obliki »dd.mm.llll«.

Id	Signatura	Status	Datum vrnitve
1	004	prost	Null
2	821	izposojen	12.06.2016
3	004	izposojen	27.05.2016
4	821	prost	Null
5	011	prost	Null

Slika 63: Relacija "IZVOD"

Med integritetne omejitve uvrščamo tudi **omejitev, ki določa obvezen vnos podatkov**. To je pogoj, s katerim določimo, kateri atributi v podatkovni bazi ne smejo imeti nedoločene vrednosti oziroma vrednost Null. Vedeti moramo, da vrednost Null ni enaka številčni vrednosti nič ali znakovnem nizu, napolnjenemu s presledki. To je posebna vrednost, ki ima lahko v danem zapisu več pomenov, in sicer: atribut v zapisu ne nastopa, vrednost atributa v zapisu ni znana ali vrednost atributa v zapisu še ni zabeležena. Na sliki Slika 63 vidimo, da ima atribut »Datum vrnitve« pri določenih zapisih vrednost Null. V tem primeru Null pomeni, da atribut v zapisu ne nastopa. Kajti atribut »Datum vrnitve« nastopa samo v primeru, ko je izvod izposojen. Zato v tem primeru ne moremo reči, da atribut »Datum vrnitve« zahteva obvezen vnos podatka. V primeru atributa »Status« pa vidimo, da dejansko za vsak izvod vodimo evidenco, ali je le-ta izposojen ali prost. Zato lahko rečemo, da atribut »Status« zahteva obvezen vnos podatka. To pa pomeni, da za atribut »Status« v podatkovni bazi ne smemo hraniti vrednost Null.

Naslednji dve integritetni omejitvi sta eni izmed najpomembnejših. Z njima zagotavljamo celovitost podatkov. Prva omejitev je povezana s prej omenjeno omejitvijo obvezen vnos podatkov. Imenujemo jo **entitetna omejitev**. Entitetna omejitev določa, da primarni ključ ne sme imeti vrednost Null. Torej mora biti primarni ključ relacije obvezen podatek. Po definiciji je primarni ključ atribut (ali množica atributov), ki ga uporabljamo za enolično identifikacijo vrstice relacije. Na primer, primarni ključ relacije »ČLAN« je atribut »Članska številka«. Zato pri vnosu zapisa v relacijo »ČLAN« ne smemo dopustiti, da bi za atribut »Članska številka« vnesli vrednost Null. Enako velja tudi za primarni ključ (atribut »Id«) pri relaciji »IZVOD«. Če pa predpostavimo, da pri relaciji »ČLAN« namesto članske številke za primarni ključ izberemo atributa »Ime« in »Priimek«, potem bi le-ta bil sestavljen primarni ključ. V tem primeru pri vnosu zapisa v relacijo »ČLAN« ne smemo dopustiti, da bi oba atributa imela vrednost Null.
Druga omejitev se nanaša na tuje ključe. Kot smo povedali prej, so podatki, shranjeni v eni relaciji, včasih lahko le taki, kot so že shranjeni v neki drugi relaciji. Temu rečemo, da je podatek v neki relaciji tuj ključ v drugi relaciji.
Če torej imamo tak primer, vpeljemo omejitev med tema dvema relacijama. Tej omejitvi pravimo **referenčna omejitev**. Referenčna omejitev povezuje dve relaciji in zagotavlja skladnost med zapisi v

teh dveh relacijah. To pomeni, da če spremenimo eno relacijo, potem moramo preveriti tudi drugo relacijo in jo po potrebi spremeniti, da ohranimo skladnost podatkov. Če v relaciji obstajajo tuji ključi, potem se vrednosti le-teh morajo ujemati z vrednostmi primarnega ključa v osnovni relaciji ali pa morajo biti enake Null. Vzemimo za zgled relaciji »ČLAN« in »POŠTA« predstavljeni na sliki Slika 62. Kot vidimo na sliki, se vrednosti atributa »Poštna številka« v relaciji »ČLAN« ujemajo z vrednostmi primarnega ključa relacije »POŠTA«. Če želimo spremeniti relacijo »POŠTA«, potem moramo preveriti tudi relacijo »ČLAN« in jo po potrebi spremeniti. Namreč atribut "Poštna številka" se v relaciji "ČLAN" sklicuje na primarni ključ »Poštna številka« v relaciji "POŠTA". Torej je atribut »Poštna številka« v relaciji »ČLAN« tuji ključ. V relaciji »ČLAN« ne moremo ustvariti novega zapisa, ki bi imel poštno številko npr. 5290, vse dokler ni ta poštna številka zapisana v relaciji »POŠTA«. Lahko pa ustvarimo nov zapis v relaciji »ČLAN«, ki bo imel za poštno številko vrednost Null. Na ta način poskrbimo za primer, ko se knjižnici pridruži nov član, ki pa še ni bil dodeljen v območje (kraj), iz katerega prihaja.

4.2 Orodja za izdelavo logičnega modela

Pri predstavitvi logičnih modelov, še posebej relacijskih podatkovnih diagramov, si pomagamo s CASE-orodji. Kot smo omenili, večina CASE-orodij vsebuje poseben algoritem, s pomočjo katerega lahko konceptualni model, dobljen v fazi konceptualnega načrtovanja, preslika v logični podatkovni model v nekaterih specifičnih SUPB. CASE-orodja vključujejo tudi orodja, ki načrtovalcu omogočajo izris logičnega modela. Nekaj orodij, ki omogočajo izris logičnega modela, smo omenili že pri orodjih za izdelavo konceptualnega modela (ER-modela). Med orodja, ki omogočajo izris logičnega modela, uvrščamo orodja, kot so:

- DBDesigner (http://fabforce.net/dbdesigner4/),
- Case Studio 2 (demo verzija dostopna na: http://download.cnet.com/Case-Studio-2/3001-10254_4-10517119.html?hlndr=1),
- Open Model Sphere (http://www.modelsphere.com/org/),
- TOAD Data Modeler (http://www.casestudio.com/enu/products.aspx),
- PowerDesigner (http://go.sap.com/product/data-mgmt/powerdesigner-data-modeling-tools.html),
- OracleDesigner (http://www.oracle.com/technetwork/developer-tools/designer/overview/index-082236.html),
- Erwin Data Modeler (http://erwin.com/products/data-modeler),
- DeZign for Databases (http://www.datanamic.com/dezign/).

V diplomski nalogi bomo za izdelavo logičnega modela uporabili CASE-orodje DBDesigner 4. Izbrali smo ga zato, ker je preprost in pregleden. Njegova prednost je tudi v tem, da je prostodostopen in hkrati dovolj zmogljiv. Izbrano orodje DBDesigner 4 ponuja več različnih zapisov notacij. V nadaljevanju bomo uporabljali notacijo »vranjih nog«.

4.3 Preslikava ER-modela v relacijski podatkovni model

Povedali smo, da v fazi logičnega načrtovanja izvedemo preslikavo ER-modela v relacijski podatkovni model. Omenili smo tudi, da obstajata dva razloga za preslikavo ER-modela v relacijski podatkovni model. Prvi razlog je ta, da SUPB ne podpira neposredno ER-modela. Drugi pa je ta, da SUPB podpira relacijski ali objektno-relacijski podatkovni model. Poglejmo si sedaj postopek preslikave ER-modela v relacijski podatkovni model.

Postopek preslikave lahko opravimo na dva načina:

1. **avtomatizirano** – orodja CASE načeloma podpirajo avtomatizacijo procesa ustvarjanja podatkovne baze in s tem avtomatično preslikajo ER model v relacijski podatkovni model ali
2. **ročno** – sledimo pravilom za preslikavo in naredimo sheme relacij.

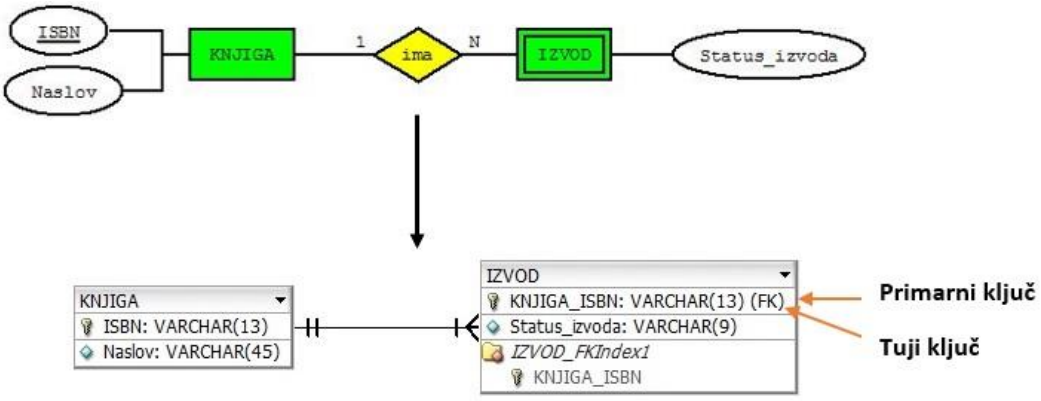
Orodje DBDesigner ne podpira avtomatske pretvorbe diagrama (Slika 57), ki smo ga izdelali v orodju DIA. Omogoča pa nam, da diagram, ki ga narišemo, lahko prikažemo kot »konceptualni« model, se pravi kot model, ki vsebuje entitetne tipe, njihove atribute in primarne ključe.
V nadaljevanju bomo predstavili postopek za ročno preslikavo ER-modela v relacijski podatkovni model. Za ponazoritev bomo uporabili konceptualni model knjižnice, ki je prikazan kot ER-diagram na sliki Slika 57.
Pri preslikavi v relacijski podatkovni model je treba preslikati naslednje elemente ER-modela: močne entitetne tipe, šibke entitetne tipe, razmerja med entitetnimi tipi in hierarhije entitetnih tipov.
Pri pretvorbi ER-modela v relacijski podatkovni model običajno sledimo naslednjim korakom:

1. **Preslikava močnih entitetnih tipov**
   Vsakemu močnemu entitetnemu tipu v ER-modelu priredimo relacijo. Priporočljivo je, da je ime relacije enako imenu entitetnega tipa. Vsak enostavni atribut entitetnega tipa postane atribut relacije. V primeru, da entitetni tip vsebuje tudi sestavljene atribute, v relacijo vključimo zgolj njegove enostavne komponente. Primarni ključ entitetnega tipa postane primarni ključ pripadajoče relacije. Na primer v ER-diagramu knjižnice entitetni tip »KNJIGA« preslikamo v relacijo z imenom »KNJIGA«, kot prikazuje Slika 64. Atributi »ISBN«, »Naslov«, »Založnik« in »Leto izida« entitetnega tipa »KNJIGA« postanejo atributi relacije. Primarni ključ (»ISBN«) entitetnega tipa »KNJIGA« postane primarni ključ relacije »KNJIGA«.

Slika 64: Preslikava entitetnega tipa "KNJIGA" v relacijo "KNJIGA"

2. **Preslikava šibkih entitetnih tipov**
   Za vsak šibki entitetni tip v ER-modelu priredimo relacijo. Vanjo vstavimo vse enostavne atribute oziroma enostavne komponente sestavljenih atributov šibkega entitetnega tipa. Kot smo omenili, je primarni ključ šibkega entitetnega tipa delno ali v celoti odvisen od močnega oziroma starševskega entitetnega tipa, s katerim je v razmerju. Zato da lahko identificiramo primarni ključ šibkega entitetnega tipa, moramo najprej preslikati vse močne entitetne tipe. Primarni ključ relacije določimo kot kombinacijo primarnih ključev starševskega entitetnega tipa in morebitnih delnih ključev šibkega entitetnega tipa. Primarni ključ, ki je določen kot kombinacija primarnih ključev starševskega entitetnega tipa, hkrati predstavlja tudi tuji ključ relacije. Tuje ključe sicer določimo šele pri preslikavi razmerij, vendar je na sliki Slika 65 tuji ključ že označen. Primer preslikave šibkega entitetnega tipa bomo prikazali na entitetnemu tipu »IZVOD« (Slika 30). Šibki entitetni tip »IZVOD« preslikamo v relacijo »IZVOD«, kot prikazuje Slika 65. Najprej preslikamo močni entitetni tip »KNJIGA« v relacijo »KNJIGA«. Nato

preslikamo še šibki entitetni tip »IZVOD« v relacijo »IZVOD«. V relacijo »IZVOD« vstavimo atribut »Status izvoda«. Ključ relacije »IZVOD« je atribut »ISBN«, ki je določen preko primarnega ključa relacije »KNJIGA«. Atribut »ISBN« je torej primarni ključ in hkrati tuji ključ relacije »IZVOD«.



Slika 65: Preslikava šibkega entitetnega tipa "IZVOD" v relacijo "IZVOD"

Na sliki je prikazano tudi razmerje med obema relacijama, ki jo načeloma izvedemo šele na naslednjem koraku. Prav tako je že označen tudi tuji ključ, ki nastane na naslednjem koraku.

3. Preslikava razmerij

Razmerje 1 : N

Najprej si bomo ogledali razmerja, ki nimajo dodatnih oziroma svojih atributov. Taka so vsa razmerja na ER-modelu (Slika 57), ki smo ga razvili pri konceptualnem načrtovanju. V tem primeru za vsako razmerje s številnostjo 1 : N v ER-modelu poiščemo relaciji, ki predstavljata entitetna tipa, ki ju razmerje povezuje. Razmerje 1 : N določa, da je vsaka entiteta na N-strani povezana z največ eno entiteto na 1-strani razmerja. Zato relaciji, ki je na N-strani razmerja, kot tuji ključ vstavimo primarni ključ relacije, ki je na drugi strani razmerja.

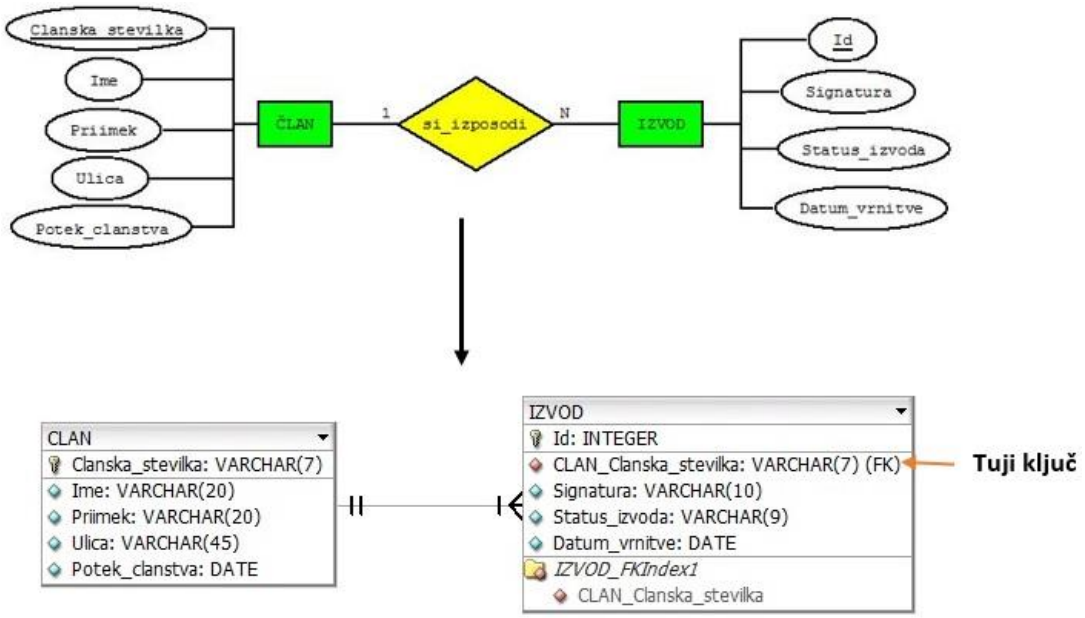
V primeru, če gre za identifikacijsko razmerje (razmerje med močnim in šibkim entitetnim tipom), je tuji ključ sestavni del primarnega ključa šibkega entitetnega tipa.

Pri primeru na sliki Slika 65 je prikazano razmerje med močnim entitetnim tipom »KNJIGA« in šibkim entitetnim tipom »IZVOD«. V tem primeru v relacijo, ki je na N-strani razmerja (v relacijo »IZVOD«) kot tuji ključ vstavimo primarni ključ relacije, ki je na drugi strani razmerja (relacija »KNJIGA«). Ker je primarni ključ relacije »IZVOD« v celoti odvisen od primarnega ključa relacije »KNJIGA«, je tuji ključ relacije »IZVOD« enak primarnemu ključu relacije »IZVOD«. Atribut »ISBN« v relaciji »IZVOD« tako predstavlja hkrati primarni in tuji ključ relacije.

Če pa gre za neidentifikacijsko razmerje (razmerje med močnima entitetnima tipoma), ostaneta primarna ključa entitetnih tipov nespremenjena. Razmerje »si izposodi« na sliki Slika 66 povezuje entitetna tipa »ČLAN« in »IZVOD«. Razmerje 1 : N med omenjenima entitetnima tipoma preslikamo tako, da v prirejeno relacijo »IZVOD« vključimo tuji ključ, s katerim se sklicujemo na primarni ključ prirejene relacije »ČLAN«. Tuji ključ je na sliki označen z oznako »Foreign Key« (FK). Primarna ključa entitetnih tipov »ČLAN« in »IZVOD« ostaneta enaka. Glede na omenjeno razmerje »si izposodi« bi kot rešitev dobili sledeči relaciji:

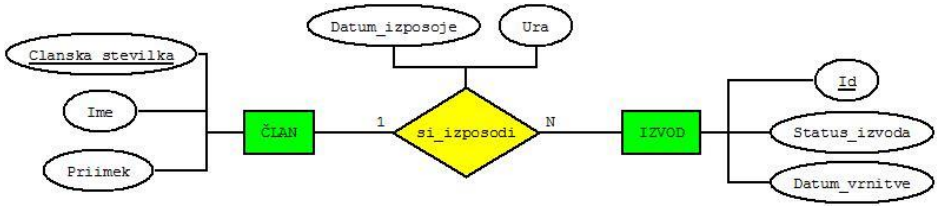
ČLAN (Članska številka, Ime, Priimek, Ulica, Potek članstva)
IZVOD (Id, Signatura, Status, Datum vrnitve, #Članska številka)

Znak # predstavlja tuji ključ.



Slika 66: Preslikava razmerja s številostjo 1 : N

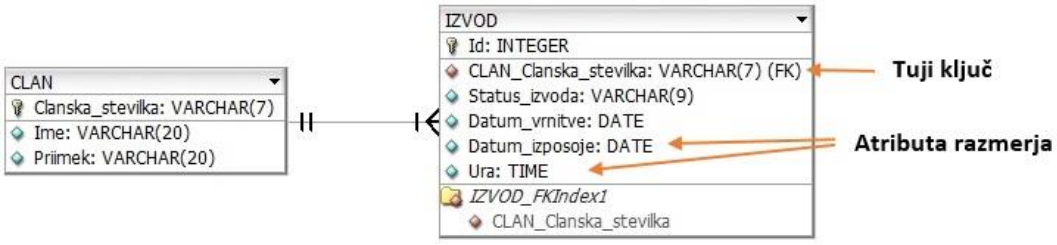
Povedati moramo še, kaj storimo, če ima določeno razmerje svoje atribute (glej razmerje s slike Slika 67).



Slika 67: Razmerje 1 : N z atributi

V tem primeru moramo v relacijo, ki je na N-strani razmerja, vstaviti še enostavne atribute, ki jih ima razmerje. Če razmerje vsebuje sestavljene atribute, pa v relacijo vključimo zgolj njegove enostavne komponente. Razmerje »si_izposodi« na sliki Slika 67 povezuje entitetna tipa »ČLAN« in »IZVOD«. Razmerje ima tudi dva atributa »Datum izposoje« in »Ura«. Razmerje s številostjo 1 : N med entitetnima tipoma »ČLAN« in »IZVOD« preslikamo tako, da v prirejeno relacijo »IZVOD« vključimo tuji ključ, s katerim se sklicujemo na primarni ključ prirejene relacije »ČLAN«. V relacijo »IZVOD« pa vključimo tudi atributa »Datum izposoje« in »Ura«. Primarna ključa entitetnih tipov »ČLAN« in »IZVOD« ostaneta enaka. Glede na omenjeno razmerje »si_izposodi« bi kot rešitev dobili sledeči relaciji:

ČLAN (<u>Članska številka</u>, Ime, Priimek, Ulica, Potek članstva)
IZVOD (<u>Id</u>, Signatura, Status, Datum vrnitve, Datum izposoje, Ura, #Članska številka)



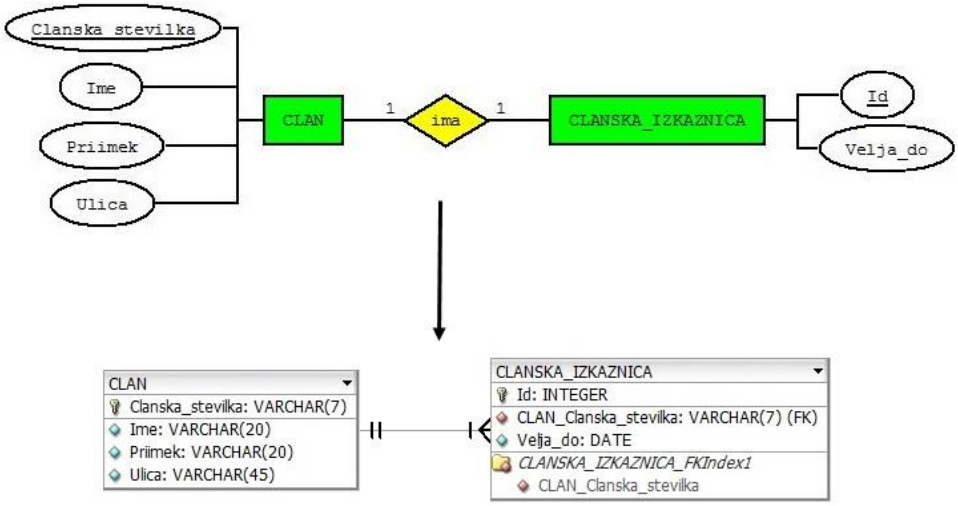
Slika 68: Preslikava razmerja 1 : N z atributi

Razmerje 1 : 1

Za vsako razmerje s številnostjo 1 : 1 v ER-modelu poiščemo relaciji, ki predstavljata entitetna tipa, ki ju razmerje povezuje. Izberemo eno izmed relacij in vanjo kot tuji ključ vstavimo primarni ključ druge relacije (relacije, ki je nismo izbrali). Nikakor pa ne smemo vstaviti tujega ključa pri obeh relacijah, ker bi to pripeljalo do nepotrebnega podvajanja podatkov.

V ER-diagramu knjižnice nimamo razmerja 1 : 1. Zato bomo preslikavo prikazali za razmerje »ima« med entitetnima tipoma »ČLAN« in »ČLANSKA IZKAZNICA«. Vsak član knjižnice ima natanko eno člansko izkaznico, vsaka članska izkaznica pripada le enemu članu. Razmerja »ima« s številnostjo 1 : 1 preslikamo tako, da v prirejeno relacijo »ČLANSKA IZKAZNICA« kot tuji ključ vstavimo primarni ključ (atribut »Članska številka«) prirejene relacije »ČLAN«, kot prikazuje spodnji opis:

ČLAN (Članska številka, Ime, Priimek, Ulica)
ČLANSKA IZKAZNICA (Id, Velja do, #Članska številka)



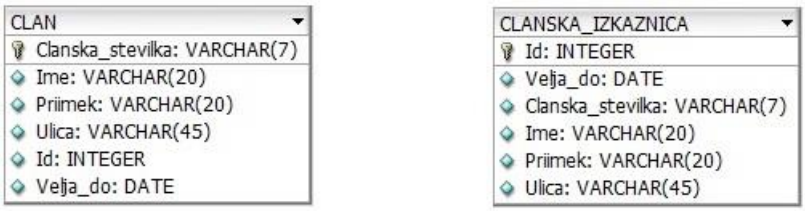
Slika 69: Preslikava razmerja s številnostjo 1 : 1

Kot vidimo na sliki Slika 69 smo iz razmerja s številnostjo 1 : 1 dobili razmerje s številnostjo 1 : N. To se zgodi zato, ker nam pri preslikavi razmerja s številnostjo 1 : 1 orodje DBDesigner samodejno pretvori razmerje 1 : 1 v razmerje 1 : N. Relacija, v katero vstavimo tuji ključ, je tako vedno na strani mnogo (N).

V primeru, da ima razmerje 1 : 1 svoje atribute, le-te vključimo v relacijo s tujim ključem (relacijo, ki smo jo izbrali in vanjo vstavili tuji ključ).

Preslikavo razmerja s številnostjo 1 : 1 lahko rešimo tudi tako, da za oba entitetna tipa ustvarimo eno skupno relacijo, v katero združimo atribute obeh entitetnih tipov. Novonastala relacija ima tako dva kandidata za ključ. To sta ključa entitetnih tipov, ki sodelujeta v tem razmerju. Za ključ novonastale relacije izberemo le enega od kandidatov. Glede na zgoraj omenjeno razmerje »ima« bi kot rešitev lahko naredili eno od naslednjih dveh relacij:

ČLAN (Članska številka, Ime, Priimek, Ulica, Id, Velja do) ali
ČLANSKA IZKAZNICA (Id, Velja do, Članska številka, Ime, Priimek, Ulica)



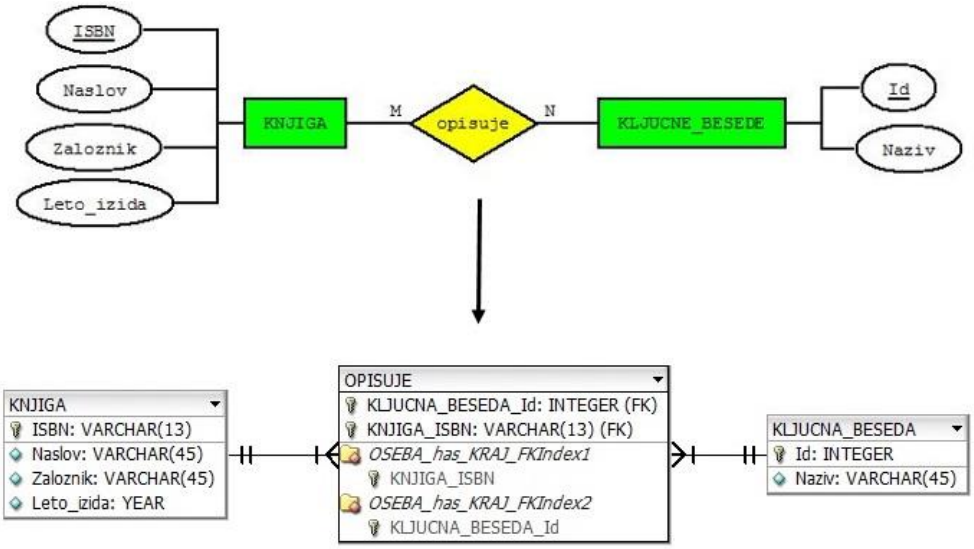
Slika 70: Možni rešitvi drugega pristopa preslikave razmerja s številnostjo 1 : 1

V primeru, da ima razmerje 1 : 1 svoje atribute tudi te vključimo v novonastalo relacijo.
Ta način reševanja preslikave razmerja s številnostjo 1 : 1 ni ravno najboljša. V primeru brisanja podatkov o članu bi namreč hkrati izgubili tudi podatke o članski izkaznici in obratno. Če nimamo na obeh straneh entitetnih tipov popolne udeležbe (na obeh straneh razmerja številnost (1,1)), pa bomo poleg tega v relaciji imeli zapise z vrednostjo Null.

Razmerje M : N

Za vsako razmerje v ER-modelu s številnostjo M : N v logičnem modelu uvedemo novo relacijo. Poiščemo relaciji, ki predstavljata entitetna tipa, ki ju razmerje povezuje. Nato primarna ključa obeh relacij vstavimo kot tuja ključa v novonastalo relacijo. Hkrati pa primarna ključa obeh relacij tvorita tudi primarni ključ novonastale relacije. Ključ relacije, ki nastane kot posledica razmerja M : N, je torej vedno sestavljen.
Pri preslikavi ER-diagrama knjižnice razmerje »opisuje« med entitetnima tipoma »KNJIGA« in »KLJUČNA BESEDA« preslikamo tako, da ustvarimo novo relacijo »OPISUJE«. V relacijo »OPISUJE« kot tuja ključa vstavimo primarna ključa (»ISBN« in »Id«), s katerima se sklicujemo na relaciji »KNJIGA« in »KLJUČNA BESEDA«, kar prikazuje Slika 71. Atributa »ISBN« in »Id«, ki smo ju vstavili v novonastalo relacijo, hkrati predstavljata tudi ključ novonastale relacije »OPISUJE«.

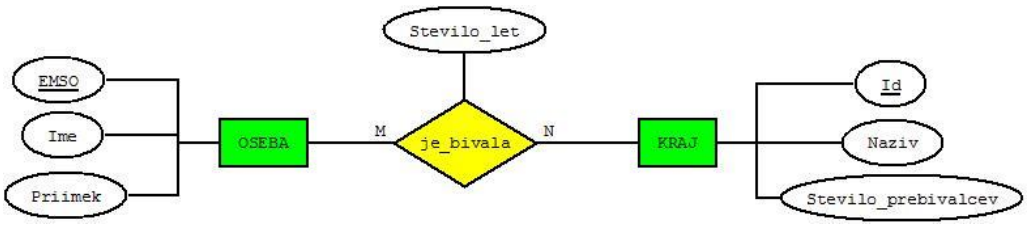
KNJIGA (<u>ISBN</u>, Naslov, Založnik, Leto izida)
KLJUČNA BESEDA (<u>Id</u>, naziv)
OPISUJE (#<u>ISBN</u>, #<u>Id</u>)



Slika 71: Preslikava razmerja s številnostjo M : N

V primeru, da ima razmerje M : N svoje atribute (glej sliko Slika 72), le-te vključimo kot atribute v novonastalo relacijo. Kot smo že omenili, v našem primeru nimamo takšnih razmerij. Zato

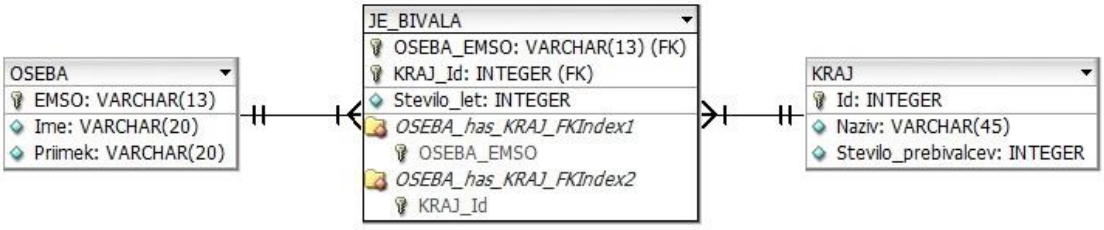
bomo primer preslikave razmerja M : N s svojimi atributi prikazali na primeru, ki je predstavljen na sliki Slika 72.



Slika 72: Primer razmerja M : N s svojimi atributi

Razmerje »je bivala« povezuje entitetna tipa »OSEBA« in »KRAJ«. Razmerje vsebuje tudi atribut »Število let«. Razmerje »je bivala« s številnostjo M : N preslikamo tako, da ustvarimo novo relacijo »JE BIVALA«. V novonastalo relacijo kot tuja ključa vstavimo primarna ključa (»EMŠO« in »Id«), s katerima se sklicujemo na relaciji »OSEBA« in »KRAJ«. Atributa »EMŠO« in »Id« hkrati predstavljata tudi ključ novonastale relacije. Atribut razmerja (»Število let«) vstavimo kot atribut v novonastalo relacijo »JE BIVALA«, kot prikazuje Slika 73.

OSEBA (EMŠO, Ime, Priimek)
KRAJ (Id, Naziv, Število prebivalcev)
JE BIVALA (#EMŠO, #Id, Število let)



Slika 73: Preslikava razmerja M : N s svojimi atributi

4. Preslikava večvrednostnih atributov

Tudi za vsak večvrednosti atribut entitetnega tipa ER-modela v logičnem modelu ustvarimo novo relacijo. Primarni ključ entitetnega tipa, ki vsebuje večvrednostni atribut, vstavimo kot tuji ključ v novonastalo relacijo. V novonastalo relacijo vključimo tudi atribut, ki predstavlja večvrednostni atribut. V primeru, da je večvrednostni atribut sestavljen, v novo relacijo vključimo zgolj njegove enostavne komponente. Primarni ključ novonastale relacije je kombinacija večvrednostnega atributa in primarnega ključa entitetnega tipa.

Primer večvrednostnega atributa je denimo atribut »Avtor« entitetnega tipa »KNJIGA«. Kot smo omenili, ima knjiga lahko več avtorjev. V ER-diagramu knjižnice smo sicer večvrednostnim atributom (»Avtor« in »Ključne besede«) priredili nov entitetni tip že v fazi konceptualnega načrtovanja. Omenjena večvrednostna atributa bi lahko v fazi konceptualnega načrtovanja pustili znotraj entitetnega tipa »KNJIGA«, kot prikazuje sledeč opis:

KNJIGA (ISBN, Naslov, Avtor, Ključne besede, Založnik, Leto izida)

Oglejmo si, kako bi bila sedaj videti pretvorba v logični model. Za večvrednosti atribut »Avtor« ustvarimo novo relacijo »AVTOR«. Kot tuji ključ nove relacije bi vstavili primarni ključ (ISBN) entitetnega tipa »KNJIGA«. Avtorja knjige beležimo z imenom in priimkom. Zato v prirejeno relacijo vstavimo še atribut »Ime« in atribut »Priimek«, ki predstavljata večvrednostni atribut »Avtor«, kot prikazuje spodnji opis:

KNJIGA (ISBN, Naslov, Ključne besede, Založnik, Leto izida)
AVTOR (Id, Ime, Priimek, #ISBN)

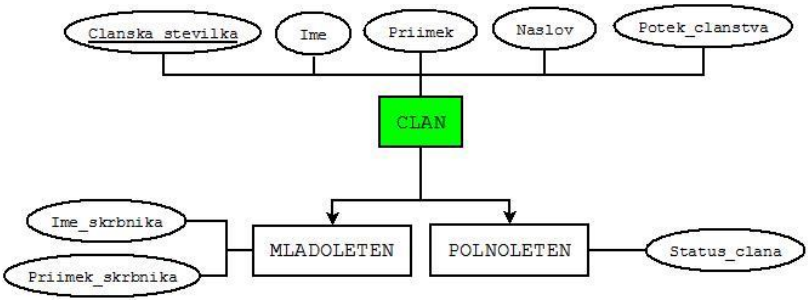
Enako storimo tudi za večvrednostni atribut »Ključne besede«. Dobljeno rešitev prikazuje spodnji opis relacij:

KNJIGA (ISBN, Naslov, Založnik, Leto izida)
AVTOR (Id, Ime, Priimek, #ISBN)
KLJUČNA BESEDA (Id, Naziv, #ISBN)

5. Preslikava hierarhije entitetnih tipov

Generalizacije oziroma specializacije v ER-modelu ne moremo direktno preslikati v relacijski podatkovni model. Obstajata dve možnosti preslikave hierarhije entitetnih tipov. Oglejmo si zgled.
Denimo, da v modelu knjižnice entitetnemu tipu »ČLAN« glede na datum rojstva priredimo podtipa »MLADOLETEN« in »POLNOLETEN«. Predstavitev v ER-modelu je sledeča:

ČLAN (Članska številka, Ime, Priimek, Naslov, Potek članstva)
Podtip MLADOLETEN (Ime skrbnika, Priimek skrbnika)
Podtip POLNOLETEN (Status člana)



Slika 74: Hierarhija entitetnega tipa "ČLAN"

Prva možnost preslikave je ta, da nadtip in vse njegove podtipe združimo v eno relacijo. To storimo tako, da nadtipu priredimo atribute podtipov. Glede na zgornji zgled je rešitev prve možnosti relacija:

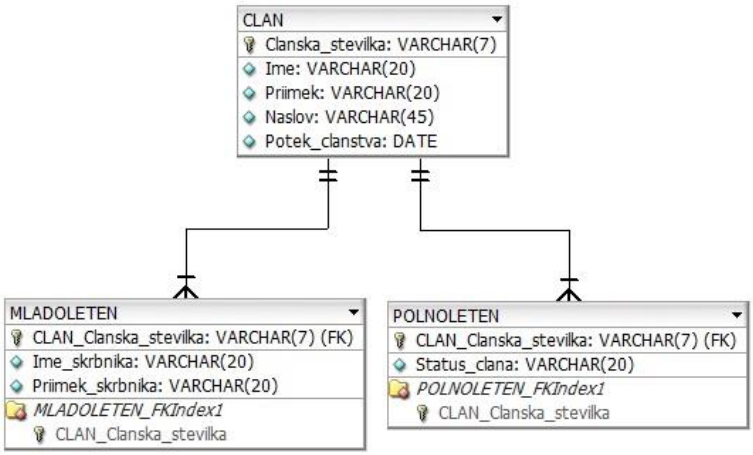
ČLAN (Članska številka, Ime, Priimek, Datum rojstva, Naslov, Potek članstva, Ime skrbnika, Priimek skrbnika, Status člana)

Slika 75: Prva možnost preslikave entitetnega tipa "ČLAN"

Zavedati se moramo, da bomo v tem primeru imeli v relaciji veliko zapisov z vrednostjo Null. Ker atributa »Ime skrbnika« in »Priimek skrbnika« pripadata samo mladoletnim članom, bodo polnoletni člani imeli za omenjena atributa v bazi vneseno vrednost Null. Enako bo imel atribut »Status« vrednost Null za vse mladoletne člane.

Druga možnost je ta, da nadtipu in vsakemu podtipu priredimo novo relacijo. Novonastale relacije podtipov podedujejo primarni ključ nadtipa. V našem zgledu dobimo:

ČLAN (Članska številka, Ime, Priimek, Naslov, Potek članstva)
MLADOLETEN (#Članska številka, Ime skrbnika, Priimek skrbnika)
POLNOLETEN (#Članska številka, Status člana)



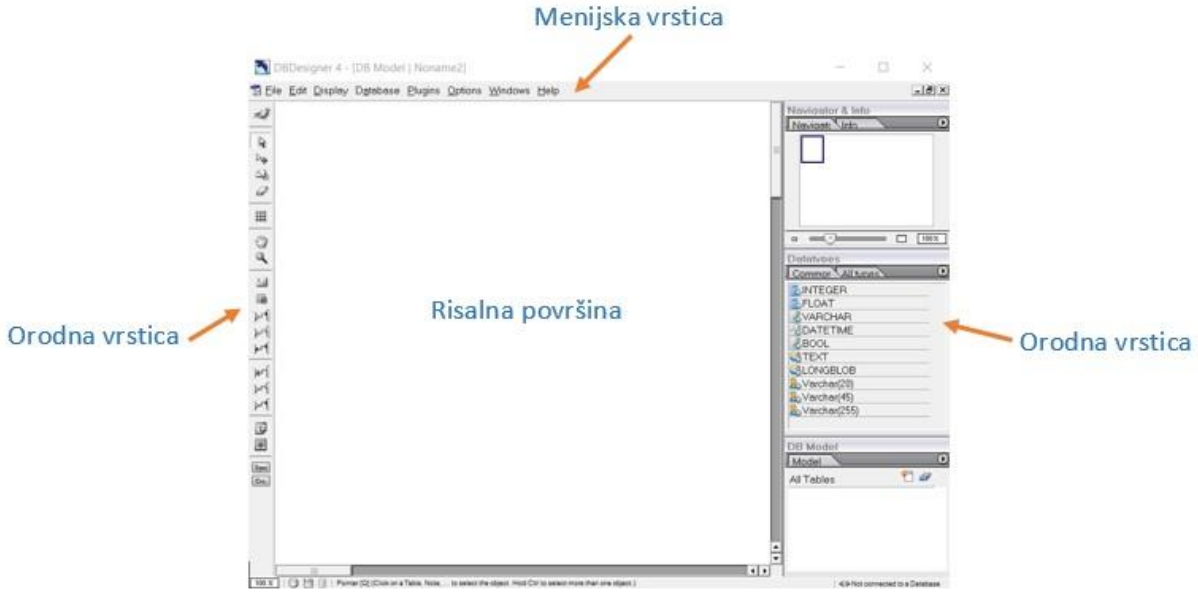
Slika 76: Druga možnost preslikave entitetnega tipa "ČLAN"

S tem načinom preslikave se izognemo nepotrebnim vrednostim Null, ki se pojavijo pri uporabi prvega pristopa.

4.4 Izdelava relacijskega podatkovnega modela z DBDesigner 4

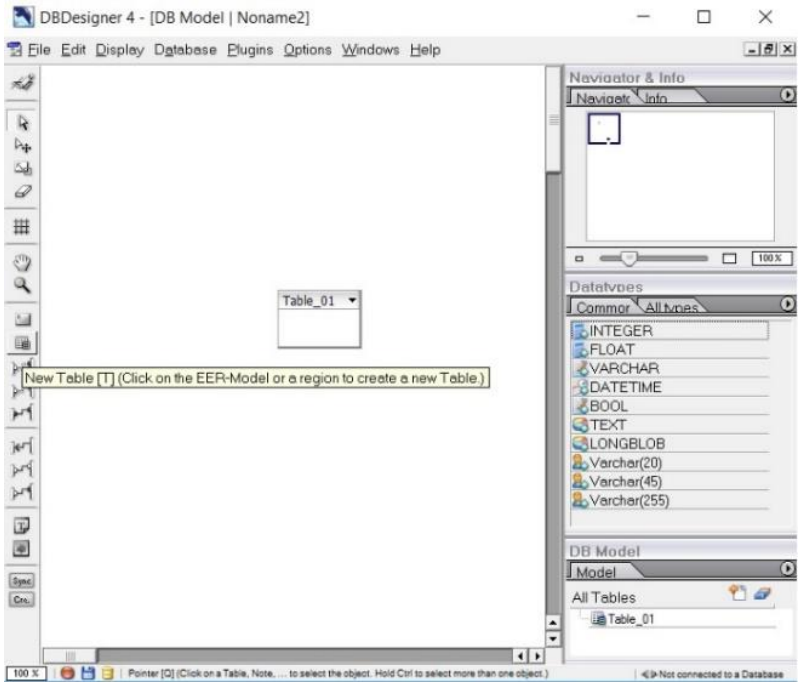
DBDesigner 4 (v nadaljevanju DBDesigner) je odprtokodno in brezplačno orodje za modeliranje podatkovnih modelov. Združuje oblikovanje, modeliranje, ustvarjanje in vzdrževanje podatkovnih baz. Razvil ga je Michael G. Zinner leta 2002/2003. DBDesigner je razvit in optimiziran za podporo podatkovnih baz, ki jih upravljamo s sistemom MySQL. Orodje nam ponuja ogromno funkcionalnosti, od različnih tehnik modeliranja podatkovnih baz do njihovega grafičnega urejanja. Med možnostmi, ki jih orodje ponuja, so grafično modeliranje logičnega modela in urejanje logičnega modela, način oblikovanja in način poizvedovanja (Design Mode/Query Mode), generiranje SQL-skripte za ustvarjanje podatkovne baze, obratni inženiring podatkovnih baz, avtomatsko določanje tujega ključa, ustvarjanje novega podatkovnega tipa, izvoz diagrama kot slike, tiskanje diagrama itd.
Pri izdelavi relacijskega podatkovnega diagrama v orodju DBDesigner bomo sledili korakom, ki smo jih opisali v prejšnjem razdelku. Glede na ER-diagram knjižnice, ki smo ga izdelali pri konceptualnem modelu, bomo najprej preslikali entitetne tipe in nato še razmerja med entitetnimi tipi. V programskih orodjih se pri risanju diagramov praviloma izogibamo šumnikom in presledkom (presledek nadomestimo s podčrtajem).

Uporabniški vmesnik orodja DBDesigner je sestavljen iz več delov. Zgoraj je glavni meni oziroma menijska vrstica, na levi in desni strani je orodna vrstica, na sredini pa je risalna površina. Orodna vrstica na levi strani nam omogoča ustvarjanje objektov, kot so: regije, relacije oziroma tabele in povezave med tabelami. Nudi pa nam tudi funkcije za brisanje objektov, njihovo povečavo, premikanje itd. Orodna vrstica na desni strani vsebuje tri dele. Prvi del predstavlja celotno risalno površino, kjer se lahko s preprostim klikom hitreje gibljemo (navigiramo) po njeni površini. Drugi del našteva možne podatkovne tipe. Tretji del prikazuje objekte, ki smo jih ustvarili na risalni površini. Uporabniški vmesnik orodja DBDesigner je prikazan na sliki Slika 77.



Slika 77: Orodje DBDesigner

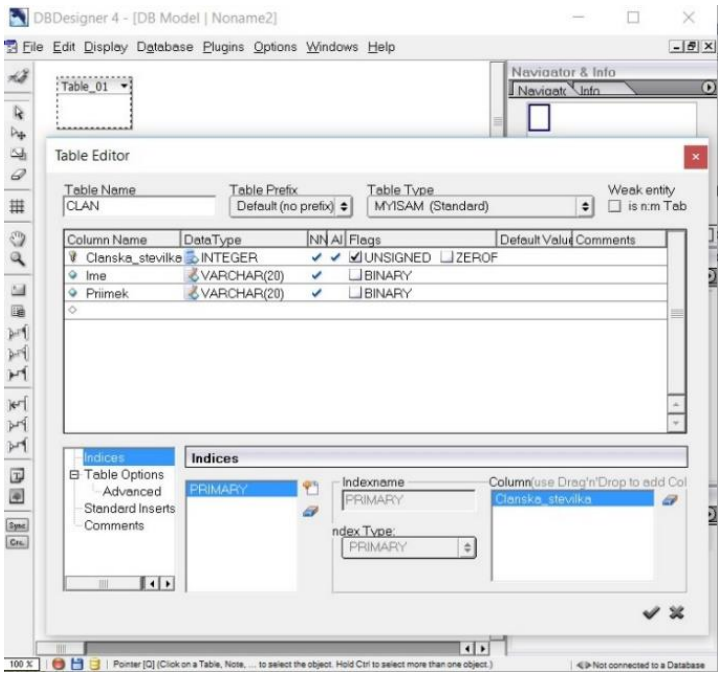
DBDesigner omogoča ustvarjanje in vzdrževanje modelov, ki vsebujejo veliko število objektov. S približevanjem miške posameznemu objektu v levi orodni vrstici se nam izpiše njegov pomen, kar prikazuje Slika 78. Hkrati pa nam izpiše tudi, kaj naj naredimo, da ta izbrani objekt izrišemo na risalni površini. Na primer na sliki Slika 78 vidimo, da nam pri objektu tabela izpiše »`Click on EER model or a region to create a new Table`«.



Slika 78: Posamezni objekti in njihov pomen

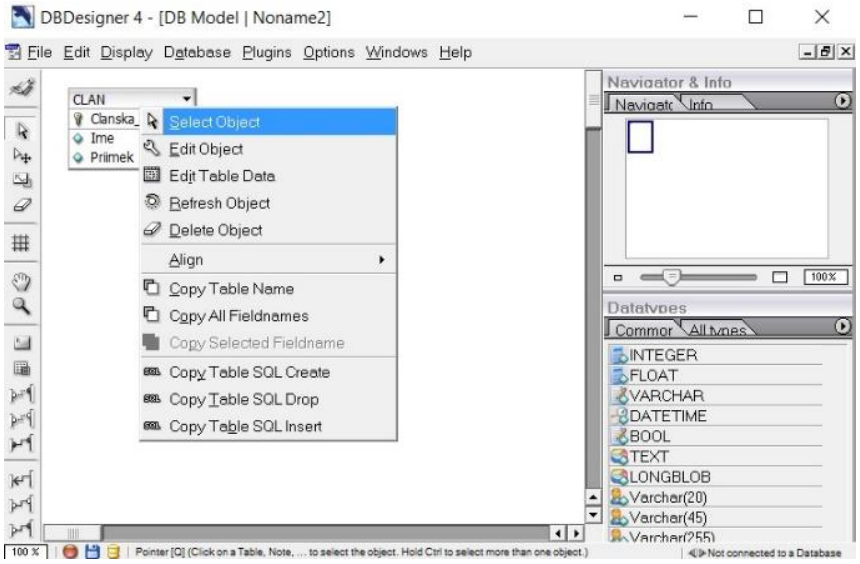
Kot smo že omenili, relacijo v logičnem modelu predstavimo s tabelo. V nadaljevanju bomo za relacijo večinoma uporabljali izraz tabela. Če želimo tabelo dodati na risalno površino, preprosto kliknemo nanjo in jo nato dodamo s klikom z miško na risalno površino. Tabelo premikamo po risalni površini tako, da kliknemo kjerkoli znotraj tabele in jo nato z miško povlečemo na želeno mesto. Če želimo tabeli spremeniti ime, vstaviti pripadajoče atribute ali določiti primarne ključe in ostale omejitve, ki se

nanašajo na podatke (npr. tip podatka, obveznost podatka, prevzeto vrednost …), jo dvokliknemo oziroma uporabimo desni klik z miško in izberemo možnost `uredi objekt` (»`Edit Object`«). Odpre se urejevalno okno tabele, kot prikazuje Slika 79. Prvi vnos v tabelo prevzeto predstavlja atribut, ki je primarni ključ relacije. Na spodnji sliki vidimo, da imamo poleg atributa »Članska številka« ključek, kar pomeni, da je ta atribut primarni ključ relacije »ČLAN«.



Slika 79: Urejevalno okno tabele

Poleg možnosti urejevanja objekta imamo na voljo tudi ostale možnosti, kot so izbira objekta, urejevanje podatkov tabele, osvežitev objekta, izbris objekta, poravnava objekta, kopiranje imena tabele itd. Do omenjenih možnosti pridemo tako, da uporabimo desni klik z miško na posamezen objekt, ki smo ga ustvarili na risalni površini. Meni vseh možnosti prikazuje Slika 80.

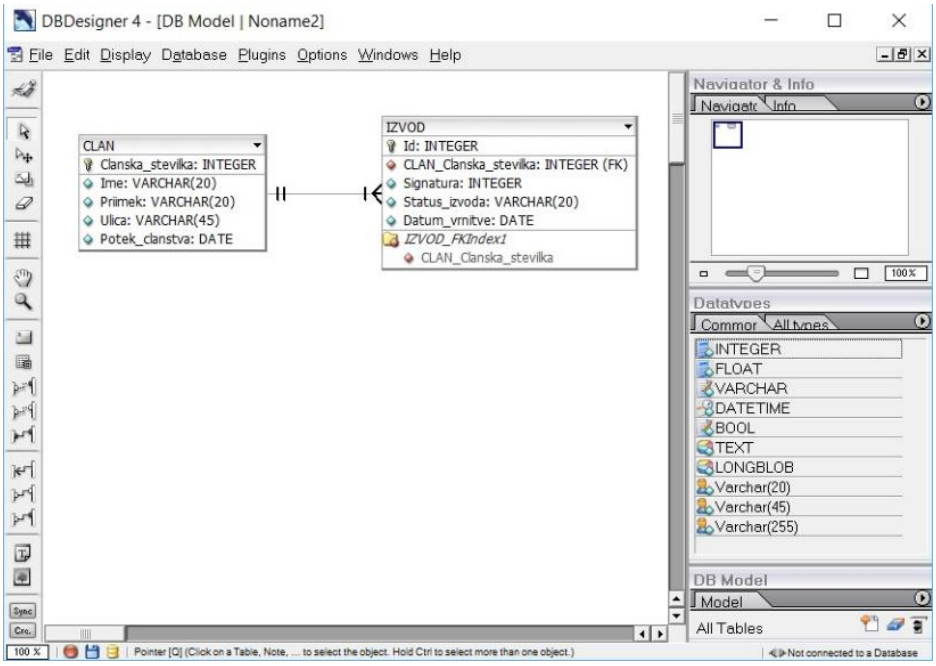


Slika 80: Meni možnosti

Tabele med seboj povežemo z ustreznimi relacijami, ki jih najdemo v levi orodni vrstici. Na voljo imamo več možnih relacij. Z neidentifikacijsko relacijo (»Non-Identifying Relation«) med seboj povezujemo

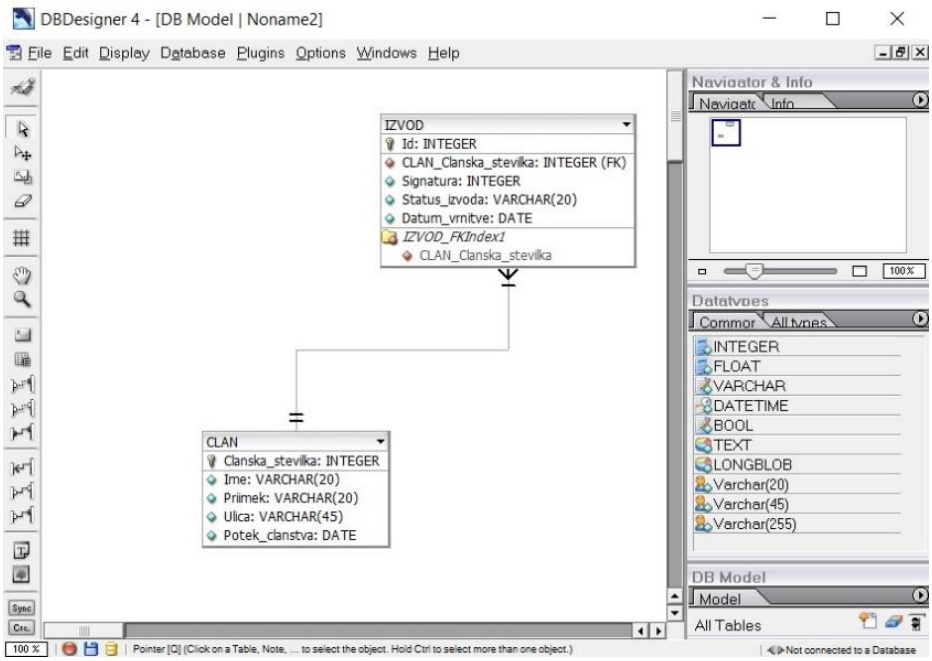
močne entitetne tipe. Šibke in močne entitetne tipe pa med seboj povežemo z identifikacijsko relacijo, ki je v orodju DBDesigner označena kot relacija (»Relation«). Tabeli med seboj povežemo tako, da najprej izberemo povezavo z ustrezno številnostjo. Nato kliknemo z miško na tabeli, ki ju želimo povezati. Orodje nam avtomatsko ustvari povezavo med želenima tabelama, hkrati pa nam orodje avtomatsko generira tudi tuji ključ.

Orodje DBDesigner ponuja več različnih načinov prikaza številnosti povezav. Če želimo spremeniti način prikaza številnosti oziroma notacijo, v menijski vrstici izberemo zavihek `Zaslon` (»`Display`«) in nato pri možnosti `Notacija` (»`Notation`«) izberemo želeni tip notacije. Številnost povezav med tabelami bomo tako kot do sedaj v tem poglavju prikazovali z notacijo »vranjih nog« (»Crow's Foot«). Na sliki Slika 81 vidimo primer povezave s številnostjo 1 : N med tabelama »ČLAN« in »IZVOD«. Pri povezavi 1 : N najprej kliknemo na tabelo, ki je na 1-strani razmerja (v našem primeru je to tabela »ČLAN«), nato pa še na tabelo, ki je na nasprotni, N-strani razmerja (v našem primeru je to tabela »IZVOD«).



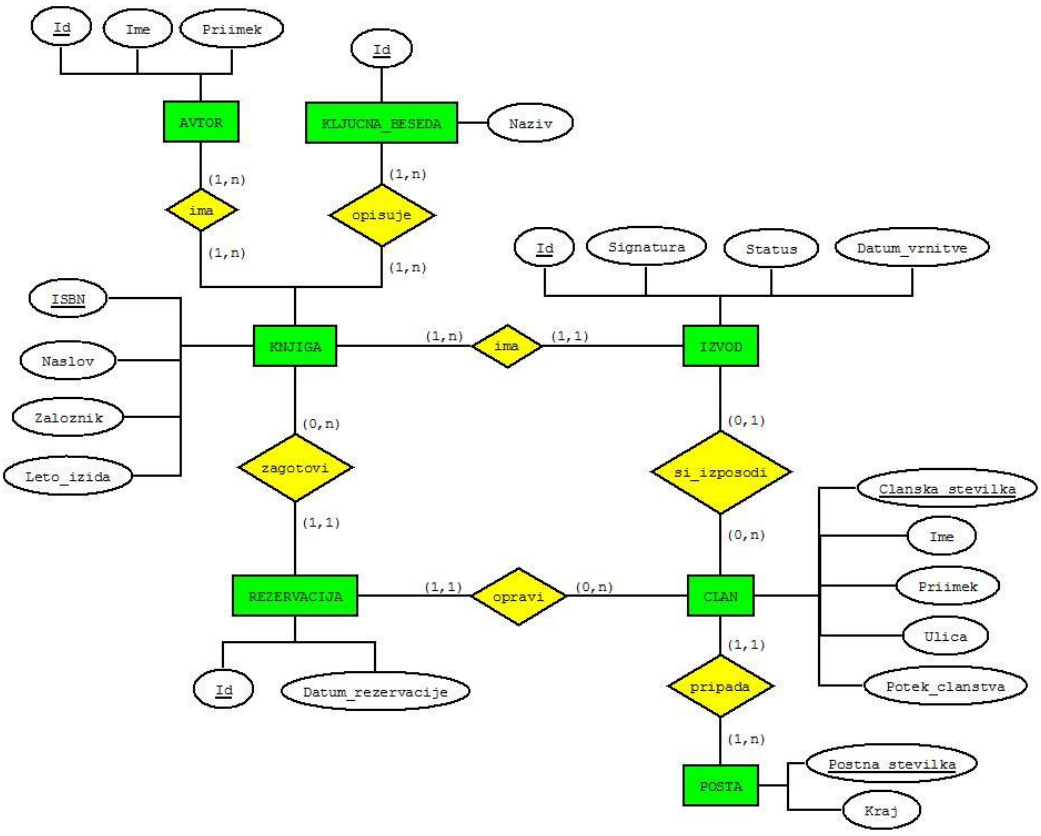
Slika 81: Povezava s številnostjo 1: N med tabelama

V primeru, da želimo prestaviti tabelo, ki smo jo predhodno že povezali z drugo tabelo, lahko to storimo, saj povezave, ki smo jih naredili, »ostanejo«. Kot vidimo na sliki Slika 82, je po premiku tabele »ČLAN« ta še vedno povezana s tabelo »IZVOD«.



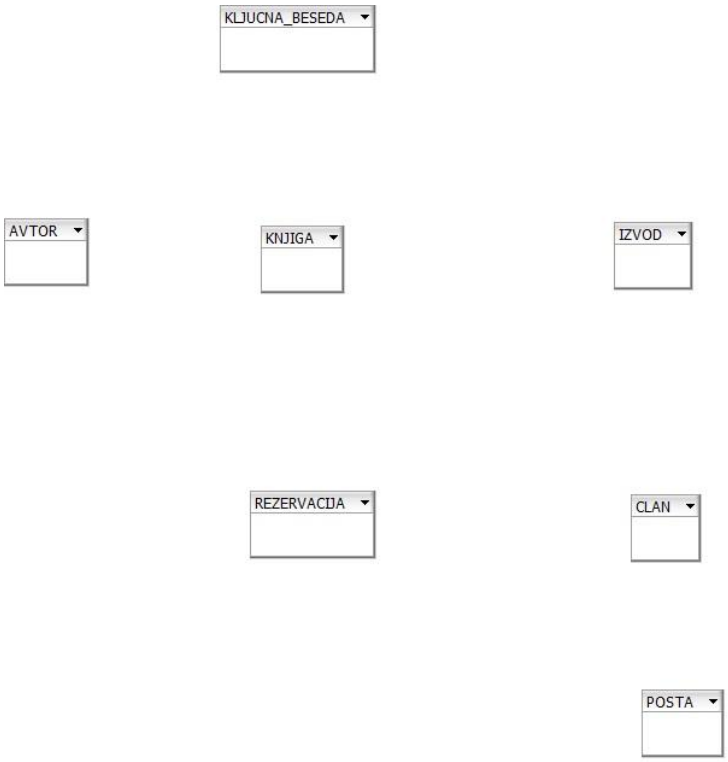
Slika 82: Prikaz premika tabele "ČLAN"

Oglejmo si, kako bi v orodju DBDesigner naredili relacijski logični model za primer podatkovne baze knjižnice. Izhajamo torej iz sledečega ER-modela



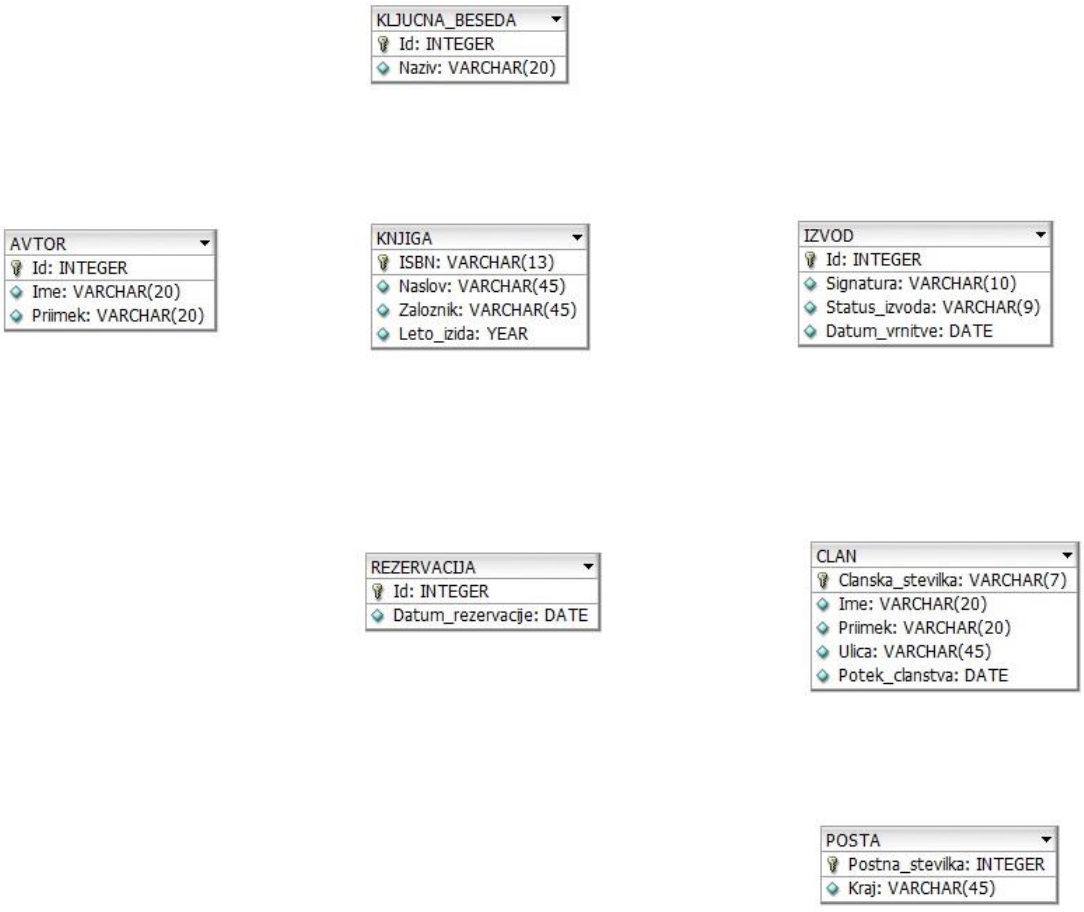
Slika 83: ER-diagram knjižnice

Najprej v diagram preslikamo vse entitete tipe v relacije, ki jih v orodju DBDesigner prikažemo s tabelami, kot prikazuje Slika 84.



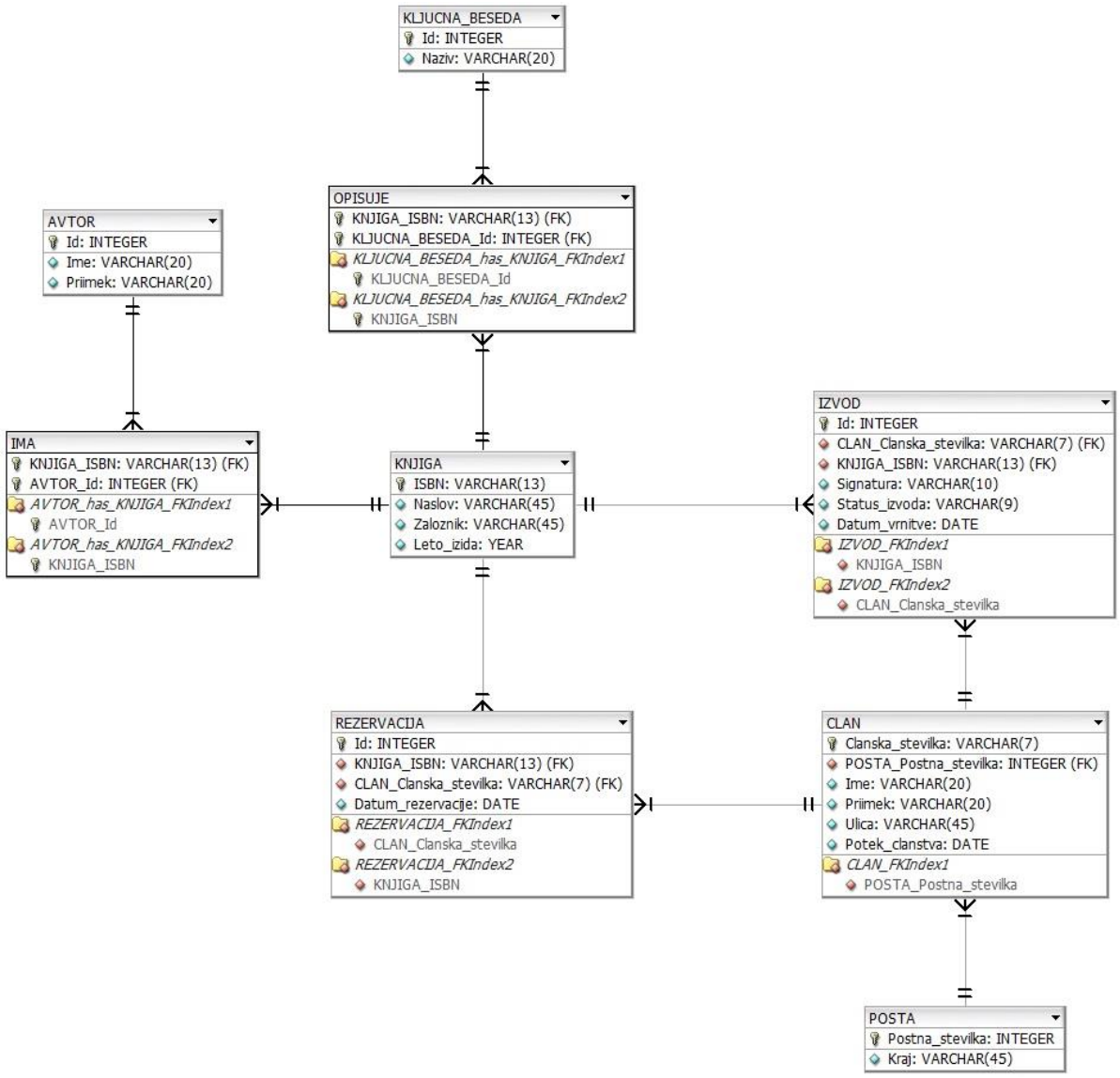
Slika 84: Diagram z dodanimi relacijami

Nato relacijam dodamo ustrezne atribute in določimo primarne ključe (Slika 85).



Slika 85: Diagram z dodanimi atributi in primarnimi ključi relacij

Sedaj relacije med seboj povežemo z ustreznimi povezavami. Ob tem nam orodje DBDesigner avtomatsko ustvari tudi tuje ključe (Slika 86).



Slika 86: Diagram z dodanimi povezavami med relacijami

Iz zgornje slike vidimo, da iz logičnega modela lahko razberemo praktično vse zahteve. V primerjavi s konceptualnim modelom pa logični model vsebuje več podrobnosti. Na primer, iz slike lahko razberemo, kateremu tipu podatkov pripadajo posamezni atributi relacij, medtem ko pri konceptualnem diagramu tega podatka nimamo. Preko tujih ključev (v grafu označeni z oznako FK) lahko razberemo, katere relacije so med seboj povezane in preko katerih atributov se sklicujemo na določeno relacijo.
S končnim diagramom (predstavljen je na sliki Slika 86) lahko izvedemo nov razgovor z uporabniki.

5 FIZIČNI MODEL

Logičnemu načrtovanju sledi faza fizičnega načrtovanja podatkovne baze. Logični model podatkovne baze, ki obsega množico relacij, moramo spraviti v »življenje«. Cilj fizičnega načrtovanja je izdelava fizičnega modela v ciljnem sistemu za upravljanje podatkovnih baz (SUPB-u). Fizični model vsebuje vse relacije (tabele) logičnega modela. Te opremimo z indeksi in določimo integritetne omejitve. V nadaljevanju bomo za relacije uporabljali izraz tabele. Fizični model je prikaz dejanske podatkovne baze. Je rezultat preslikave logičnega modela (v našem primeru relacijskega podatkovnega modela) in je prirejen za specifični SUPB. Opisuje fizične lastnosti, kot so tabele, pogledi, indeksi in ostali gradniki podatkovne baze.

Za izdelavo fizičnega modela je potrebno dobro poznavanje ciljnega SUPB-a, saj le na ta način lahko zagotovimo dobro in zmogljivo podatkovno bazo. Načrtovalec podatkovne baze mora pri izdelavi fizičnega modela vedeti, kako ciljni SUPB deluje in mora biti v celoti seznanjen z njegovimi funkcionalnostmi. Tako mora vedeti, kako izdelati osnovne relacije, ali ciljni SUPB podpira primarne, tuje in nadomestne ključe, ali podpira obveznost podatkov, ali podpira domene, ali podpira integritetne omejitve, ali podpira ustvarjanje indeksov, pregledov itd. Funkcionalnosti SUPB-a se med seboj razlikujejo, zato mora biti fizični model prilagojen za specifičen SUPB.

Fizični model je preslikava logičnega modela, vendar kljub temu lahko obstajajo razlike med logičnim in fizičnim modelom. Tako je logični model normaliziran vsaj do tretje normalne oblike (3. NO) in ne vključuje podvojenih podatkov. Včasih pa se izkaže, da je bolje, da je fizični model denormaliziran (to je nasprotni postopek od normalizacije ali drugače povedano je postopek, ki združuje dve ali več tabel med seboj) in lahko vključuje podvojene podatke. S tem lahko pri določenih SUPB-ih zagotovimo boljšo učinkovitost.

Ko izdelamo fizični model, je treba še preveriti njegovo ustreznost in zmogljivost glede na potrebe in zahteve uporabnikov. Določiti moramo različne skupine uporabnikov in za vsako uporabniško skupino določiti, do katerih delov podatkovne baze lahko dostopajo in katere operacije nad posamezno tabelo lahko izvajajo.

Kot smo omenili, fizični model izdelamo v ciljnem SUPB-u. Sistemi za delo z relacijskimi podatkovnimi bazami ponujajo poseben povpraševalni jezik, tako imenovan SQL. Povpraševalni jezik SQL omogoča tako ustvarjanje fizičnega modela oziroma podatkovne baze kot tudi poizvedovanje po podatkih v podatkovni bazi. Jezik SQL bomo podrobneje predstavili v naslednjem razdelku.

5.1 SQL

S pojavom relacijskega podatkovnega modela v začetku 70. let so se začela tudi prizadevanja za razvoj ustreznega relacijskega povpraševalnega jezika. Leta 1974 je bil definiran in tudi eksperimentalno implementiran Structured English Query Language (SEQUEL), iz katerega se je razvil današnji SQL (Mohorič, 2002). SQL (Structured Query Language) ali strukturirani povpraševalni jezik je danes najbolj razširjen povpraševalni jezik za delo s podatkovnimi bazami. Omogoča nam ustvarjanje podatkovne baze, spreminjanje, branje in upravljanje s podatki, ki so shranjeni v relacijski podatkovni bazi. Razvili so ga v podjetju IBM (Wikipedia). Zaradi izredne učinkovitosti in preproste uporabe v sistemih za upravljanje relacijskih podatkovnih baz se je hitro razširil po svetu. Postal je »standardni« jezik, saj ga je sprejela tako organizacija ANSI (American National Standards Institute) kakor tudi ISO (International Standards Organization). SQL-standard se je razvijal od leta 1986 in danes obstaja več različic. Zadnja različica standarda je bila izdana leta 2011 in je označena z oznako SQL:2011 (Wikipedia).

SQL uporabljamo praktično v vseh relacijskih SUPB-ih. Uporabljamo ga na primer v sistemih MySQL, Oracle, PostgreSQL, MS Access, Microsoft SQL Server, DB2, Informix itd. Noben SUPB pa ne ponuja točne implementacije SQL-standarda. Kljub standardom torej vsak SUPB uporablja različno implementacijo SQL-standarda. Večina SUPB-ov osnovnemu jeziku SQL doda svoje razširitve. To pomeni, da vsak SUPB uporablja svoj način zapisa SQL-stavkov. Tako se lahko zapisi SQL-ukazov, podatkovnih tipov in omejitev razlikujejo glede na posamezen SUPB. Zato je zelo pomembno, da pri fizičnem modelu vemo, kateri ciljni SUPB bomo uporabili za izdelavo modela.
SQL je preprost in podoben naravnem jeziku (programski stavki, ki posnemajo ukaze v naravnem jeziku), zato ga uporabljajo tako načrtovalci, upravitelji podatkovne baze, programerji kot tudi splošni uporabniki.

V jeziku SQL izvajamo več različnih nalog. Od teh je najznačilnejša poizvedovanje po podatkih v podatkovni bazi. Iz podatkovne baze lahko podatke glede na dana merila beremo, zapisujemo, spreminjamo ali brišemo. Druga pomembna naloga, ki jo jezik SQL omogoča, je ustvarjanje podatkovne baze in definiranje njenih sestavnih delov oziroma objektov, kot so: tabele, omejitve, pogledi in indeksi. Jezik SQL obsega več delov. Dva najpomembnejša dela sta:

1. **DDL** (Data Definition Language) – jezik za definiranje podatkov in
2. **DML** (Data Manipulation Language) – jezik za manipulacijo s podatki.

Jezik DDL je podmnožica jezika SQL, ki omogoča ustvarjanje, brisanje in spreminjanje objektov v podatkovni bazi. Osnovni objekt za shranjevanje podatkov je tabela. Poznamo pa tudi druge objekte, ki nam omogočajo lažje, hitrejše in učinkovitejše delo s podatki, kot so npr. indeksi, pogledi na zapise, procedure in podobno. Jezik DML je podmnožica jezika SQL, ki uporabnikom omogoča vnašanje, spreminjanje in brisanje zapisov v podatkovni bazi ter izvajanje poizvedb nad podatki, ki so v podatkovni bazi.

Ker namen diplomske naloge ni spoznavanje jezika SQL, bomo iz množice ukazov, ki jih pozna SQL, uporabili le ukaz **`CREATE`**, s katerim bomo ustvarili tabele podatkovne baze knjižnice, in ukaz **`INSERT`**, s katerim bomo tabele napolnili s podatki.

Oglejmo si osnovni obliki obeh ukazov.

Osnovna oblika ukaza **`CREATE`**:

```
CREATE TABLE ime_tabele
(
ime_stolpca1 podatkovni tip omejitve,
ime_stolpca2 podatkovni tip omejitve,
ime_stolpca3 podatkovni tip omejitve,
....
);
```

V splošnem, ko ustvarimo novo tabelo, navedemo imena in podatkovne tipe njenih stolpcev. Prav tako lahko določimo tudi omejitve, kot so: primarni ključ, tuji ključ, obvezen podatek itd. Osnovna oblika ukaza SQL je sicer preprosta, vendar je njena »polna« oblika lahko veliko bolj zapletena.

Poglejmo si primer.

Tabela poštnih številk in krajev:

```
CREATE TABLE posta
(
postna_stevilka INTEGER PRIMARY KEY,
kraj VARCHAR2(45)
);
```

Tabela članov:

```
CREATE TABLE clan
(
clanska_stevilka VARCHAR2(7) PRIMARY KEY,
ime VARCHAR2(20) NOT NULL,
priimek VARCHAR2(20) NOT NULL,
ulica VARCHAR2(45) NOT NULL,
potek_clanstva DATE NOT NULL,
postna_stevilka INTEGER,
CONSTRAINT clan_fk1 FOREIGN KEY (postna_stevilka) REFERENCES posta(postna_stevilka)
);
```

Kot vidimo iz zgleda, ima vsak stolpec svoje ime in svoj podatkovni tip (`INTEGER`, `VARCHAR2`, `DATE` …). Več o podatkovnih tipih bomo povedali v naslednjem razdelku. Omejitev `PRIMARY KEY` označuje stolpec, ki je primarni ključ. Omejitev `NOT NULL` določa, da stolpec ne sme vsebovati nedefinirane vrednosti oziroma vrednosti Null. Omejitev `clan_fk1` v tabeli »ČLAN« pa nam pove, da je stolpec poštna številka referenca na stolpec postna številka v tabeli »POŠTA«. Predstavlja torej referenčno omejitev (tuji ključ). Več o omejitvah in kako jih določimo, si bomo pogledali v nadaljevanju.

Osnovna oblika ukaza **`INSERT`**:

```
INSERT INTO ime_tabele (ime_stolpca1, ime_stolpca2, ime_stolpca3, ...)
VALUES (vrednost_stolpca1, vrednost_stolpca2, vrednost_stolpca3, ...);
```

V splošnem, ko vstavimo nov zapis (vrstico) v tabelo, navedemo imena stolpcev in nato še njihove pripadajoče vrednosti. Če kakšen stolpec in pripadajoča vrednost nista omenjena, se uporabi privzeta vrednost ali vrednost Null.

Poglejmo si primer.

Vnos podatkov v tabelo poštnih številk in krajev:

```
INSERT INTO posta (postna_stevilka, kraj) VALUES (5000, 'Nova Gorica');
```

V tabelo »POŠTA« smo vstavili novo vrstico z vrednostjo `5000` v stolpec `poštna številka` in vrednostjo `'Nova Gorica'` v stolpec `kraj`.

Vnos podatkov v tabelo članov:

```
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
VALUES ('0020868', 'Tina', 'Kovač', 'Srebrničeva ulica 10', '12.09.2016', 5000);
```

V tabelo »POŠTA« smo vstavili novo vrstico z vrednostmi `'0020868'`, `'Tina'`, `'Kovač'`, `'Srebrničeva ulica 10'`, `'12.09.2016'` in `5000` v stolpcih »Članska številka«, »Ime«, »Priimek«, »Ulica«, »Potek članstva« in »Poštna številka«.

Podrobneje si bomo primere ogledali v nadaljevanju, kjer bomo ukaza `CREATE` in `INSERT` bolje spoznali.

5.2 Osnovni podatkovni tipi v SQL standardu

Kot smo omenili pri logičnem modelu, morajo biti podatki, ki jih vnašamo v tabele podatkovne baze iz neke določene zaloge vrednosti, ki je opredeljena za vsak atribut oziroma stolpec tabele posebej. Vsak stolpec tabele ima svoj podatkovni tip. S tem SUPB točno ve, kakšne podatke lahko pričakuje v določenih stolpcih tabele in katerih ne.
V nadaljevanju bomo predstavili nekatere podatkovne tipe implementirane v SQL-standardu, ki jih lahko uporabljamo pri ustvarjanju tabel. Kot smo omenili, se implementacije SQL-standarda razlikujejo glede na posamezni SUPB. Zato lahko nekateri SUPB-i implementirajo podatkovne tipe podobno, a na svoj način. Vrste podatkovnih tipov in njihova imena so tako odvisna od posameznega SUPB-a, zato je pri izdelavi fizičnega modela pomembno dobro poznavanje ciljnega SUPB-a. Podatkovne tipe, ki jih lahko uporabljamo pri ustvarjanju tabel, bomo povzeli po knjigi (Brumen, 2013).

V tabeli Tabela 2 so navedeni najpogosteje uporabljeni znakovni podatkovni tipi po SQL-standardu.

Podatkovni tip	**Opis**
CHAR(*dolžina)*	Vsebuje znakovne nize fiksne dolžine. Obseg dolžine znakov določimo med oklepaji, npr. CHAR(10). Če je niz znakov krajši od navedene dolžine med oklepaji, se preostali del napolni s presledki do polne dolžine. Na primer, če v polje, definirano kot CHAR(10), vnesemo vrednost 'Maja', bo dejansko shranjenih 10 znakov, zadnjih 6 znakov bodo presledki ('Maja ').
VARCHAR(*dolžina),*	Vsebuje znakovne nize spremenljive dolžine z omejitvijo števila znakov. Največje možno število znakov določimo med oklepaji, npr. VARCHAR(20). Če je niz znakov krajši od velikosti polja, se shrani le vneseno število znakov. Je torej prostorsko učinkovitejši kot tip CHAR.
TEXT	Vsebuje znakovne nize variabilne dolžine brez omejitve števila znakov. Shranimo lahko neomejeno število znakov.

Tabela 2: Znakovni podatkovni tipi

Kot smo omenili, podatkovni tip `CHAR` uporabimo za shranjevanje nizov, za katere vemo njihovo fiksno dolžino. Če podatkovni tip `CHAR` uporabimo za shranjevanje nizov spremenljive dolžine, bo le-ta zasedel veliko nepotrebnega prostora na disku. V našem primeru knjižnice bi podatkovni tip `CHAR` lahko uporabili npr. za atribut »Članska številka«, saj vemo, da je obseg njegove dolžine vedno 7 znakov (števk). Za atribute, kot so »Ime«, »Priimek«, »Kraj« in podobno, je pa primernejši podatkovni tip `VARCHAR`, saj so dolžine omenjenih atributov spremenljive. Na ta način bomo prihranili prostor. Kot bomo videli v nadaljevanju, bomo fizični model za primer knjižnice izdelali v SUPB-u Oracle. Oracle sicer podpira uporabo podatkovni tip `VARCHAR`, ampak raje uporablja podatkovni tip `VARCHAR2`. Podatkovni tip `VARCHAR2` je sinonim za podatkovni tip `VARCHAR`, kar pomeni, da imata oba podatkovna tipa trenutno enak pomen. Tako bomo v nadaljevanju za znakovne podatkovne tipe uporabljali podatkovni tip `VARCHAR2`.

V tabeli Tabela 3 so navedeni najpogosteje uporabljeni številski podatkovni tipi po SQL-standardu.

Podatkovni tip	**Opis**
INT ali INTEGER	Predstavlja celoštevilčne vrednosti v mejah od 0 do 4.294.967.295 oziroma od – 2.147.438.648 do 2.147.438.647.
NUMERIC*(n, o)*	Predstavlja število, ki ima natančnost *n* (število, ki označuje maksimalno število števk, ki jih bo število vsebovalo) in obseg *o* (število, ki označuje, koliko od teh števk, ki jih število vsebuje, zastopa decimalna mesta). Obseg ne sme biti negativen ali večji od natančnosti. Privzeti obseg je 0. Na primer zapis NUMERIC(3,2) predstavlja število z natančnostjo 3 in obsegom 2. To pomeni, da dopušča števila do vključno 9.99.
DECIMAL*(n, o)*	Je podoben podatkovnemu tipu NUMERIC. Ravno tako lahko določimo natančnost *n* in obseg *o*. Tako zapis DECIMAL (5,2) predstavlja število z natančnostjo 5 in obsegom 2. To pomeni, da dopušča števila do vključno 999.99.
REAL	Predstavlja realna števila oziroma vrednosti s plavajočo vejico enojne natančnosti. Natančnost je običajno odvisna od strojne opreme.
DOUBLE	Predstavlja realna števila oziroma vrednosti s plavajočo vejico dvojne natančnosti. Natančnost je odvisna od sistema (operacijskega, na katerem teče SUPB), vendar pa mora biti večja od tipa REAL.
FLOAT*(n)*	Predstavlja realna števila oziroma vrednosti s plavajočo vejico. Določimo lahko tudi natančnost *n* (maksimalno število števk, ki jih bo število vsebovalo).
NUMBER*(n, o)*	Predstavlja števila s plavajočo vejico. Določimo lahko tudi natančnost *n* (maksimalno število števk, ki jih bo število vsebovalo) in obseg *o (*število, ki označuje, koliko od teh števk, ki jih število vsebuje, zastopa decimalna mesta*)*. Če obsega ne določimo, je privzet obseg 0. Nekateri SUPB-i uporabljajo ta podatkovni tip tudi za shranjevanje celoštevilskih vrednosti.

Tabela 3: Številski podatkovni tipi

Kot smo omenili, podatkovni tip `INTEGER` uporabljamo za shranjevanje celih števil. Lahko ga uporabimo za shranjevanje npr. številnih števil, kot so število oseb, število avtomobilov, število knjig in podobno. V primeru knjižnice atributi z oznako »Id« predstavljajo zaporedje celih števil (npr. 1, 2, 3, 4, 5 …), zato bomo za te atribute uporabili podatkovni tip `INTEGER`. V Oracle se za shranjevanje celih števil pogosto uporablja tudi podatkovni tip `NUMBER`. Podatkovni tip `INTEGER` nam ne dopušča določitev obsega števila števk, zato bomo v primeru knjižnice uporabili za atribut »Leto izida« podatkovni tip `NUMBER`. Leto izida knjige bomo shranjevati s štirimi številkami, kar bomo zapisali kot `NUMBER(4)`. Zapis `NUMBER(4)` nam torej dopušča vnos števila z največ štirimi številkami. Ostali podatkovni tipi, `NUMERIC`, `DECIMAL`, `REAL`, `DOUBLE` in `FLOAT`, se uporabljajo za shranjevanje realnih števil, kot je npr. cena, davčna stopnja, količina, povprečje in podobno. Razlikujejo se predvsem v obsegu natančnosti, ki je odvisna od SUPB-a in operacijskega sistema, na katerem teče SUPB. V

primeru knjižnice bomo od številskih podatkovnih tipov uporabili podatkovna tipa `INTEGER` in `NUMBER`.

V tabeli Tabela 4 so navedeni najpogosteje uporabljeni datumski podatkovni tipi po SQL-standardu.

Podatkovni tip	**Opis**
DATE	Predstavlja datum. Shranjuje torej leto, mesec in dan v letu. Oblike shranjevanja zapisov datumov so odvisne od posameznega SUPB-a. Kasneje lahko osnovno obliko spreminjamo z raznimi funkcijami, ki jih omogoča izbrani SUPB.
TIME	Predstavlja uro v obliki HH:MM:SS ali HH:MM:SS.sF. Shranjuje torej podatke o času, ki ga predstavljajo ure, minute in sekunde (npr. 10:25:12). Včasih je treba uradni čas uskladiti z astronomskim časom in je tako treba dodati še delce sekunde, kar predstavlja oznaka *sF*.
TIMESTAMP	Predstavlja datum in uro. Shranjuje torej tako podatke o letu, mesecu in dnevu določenega datuma, kot tudi ure, minute in sekunde določenega časa (npr. 2016-05-16 11:13:34).
YEAR	Predstavlja leto datuma. Oblike in dovoljena vrednost shranjevanja zapisov leta so odvisne od posameznega SUPB-a.
INTERVAL	Omogoča shranjevanje časovnega obdobja. Shranjujemo lahko tako datumske intervale (leto/mesec) kot tudi časovne intervale (dan/sekunda), odvisno od posameznega SUPB-a. S temi podatkovni tipi lahko tudi računamo.

Tabela 4: Datumski podatkovni tipi

V primeru knjižnice bomo od datumskih podatkovnih tipov uporabili podatkovni tip `DATE`. Podatkovni tip `DATE` bomo uporabili za shranjevanje vseh atributov, ki so datumskega tipa. To so atributi »Potek članstva«, »Datum vrnitve« in »Datum rezervacije«. Za prej omenjen atribut »Leto izida« pa bi lahko uporabili tudi podatkovni tip `YEAR`, seveda, če ga ciljni SUPB podpira. Kot smo omenili, bomo v nadaljevanju fizični model izdelali v SUPB Oracle, ki pa ne podpira podatkovnega tipa `YEAR`. Podpira pa ga npr. MySQL. Način zapisa in shranjevanja leta je odvisen predvsem od posameznega SUPB-a in obsega dovoljenih vrednosti, ki jih ponuja SUPB za podatkovni tip `YEAR`. Zato je res pomembno, da se pred izdelavo fizičnega modela dobro seznanimo s SUPB-om (kaj podpira, kaj ne, kakšne zapise dovoljuje, kakšne ne itd.), na katerem bomo model izdelovali.

V podatkovni bazi knjižnice sicer nimamo atributov, ki bi zahtevali npr. logični tip podatka ali slikovni tip podatka. Ampak določeni primeri, ki jih modeliramo, so lahko taki, da bomo te tipe podatkov potrebovali. Zato bomo v nadaljevanju navedli še najpogosteje uporabljen logični in slikovni podatkovni tip.

V tabeli Tabela 5 je naveden najpogosteje uporabljen logični podatkovni tip po SQL-standardu.

Podatkovni tip	Opis
BOOLEAN	Predstavlja logični podatkovni tip z logičnimi vrednostmi *pravilno* (TRUE) in *napačno* (FALSE).

Tabela 5: Logični podatkovni tipi

V tabeli Tabela 6 je naveden najpogosteje uporabljen slikovni podatkovni tip po SQL-standardu.

Podatkovni tip	Opis
BLOB (Binary large object)	Podpira shranjevanje zelo velikih dvojiško kodiranih podatkov, kot so slike, avdio in video posnetki. Omogoča velikosti med 32 kilobajti in 4 gigabajti ter več.

Tabela 6: Slikovni podatkovni tipi

5.3 Orodja za izdelavo fizičnega modela

Izdelava fizičnega modela podatkovne baze poteka v ciljnem SUPB-u. Naj še enkrat omenimo nekatere najbolj uporabljene sisteme za upravljanje relacijskih podatkovnih baz:

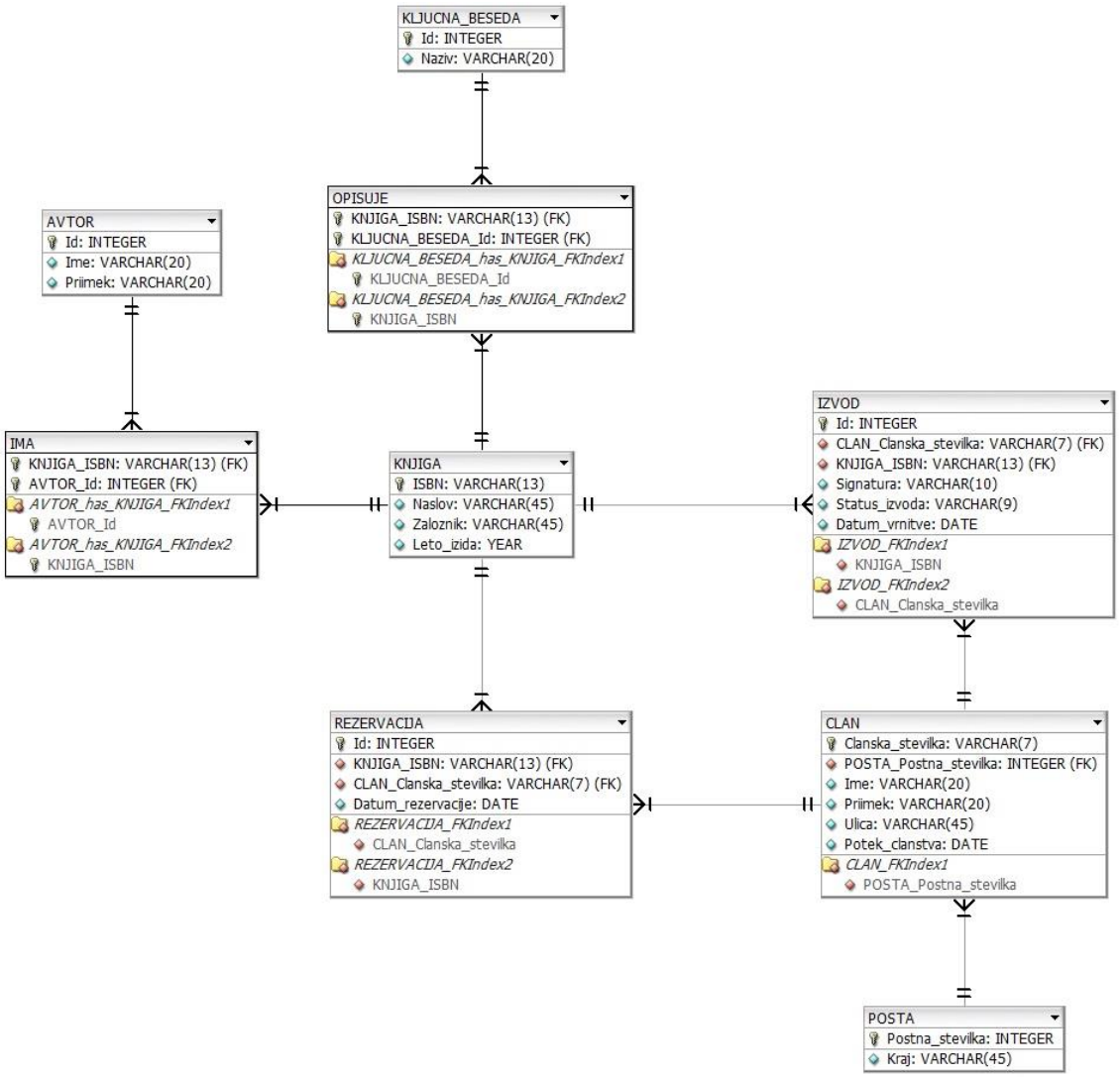
- Oracle (http://www.oracle.com/index.html),
- MySQL (https://www.mysql.com/),
- Microsoft SQL Server (https://www.microsoft.com/sqlserver),
- PostgreSQL (https://www.postgresql.org/),
- IBM DB2 (https://www.ibm.com/analytics/us/en/technology/db2/).

Kot smo omenili, vsak SUPB vsebuje poseben jezik (SQL), ki nam omogoča izdelavo fizičnega modela. S pomočjo programskih stavkov in ukazov, ki jih jezik SQL ponuja, lahko ustvarimo fizični model. Programske stavke oziroma SQL-kodo, s katero ustvarimo fizični model, lahko izdelamo tudi na osnovi grafične predstavitve logičnega modela v izbranem CASE-orodju. Kot že rečeno, CASE-orodja pospešujejo razvoj podatkovne baze tako, da v čim večji meri avtomatizirajo razvojne faze. Večina CASE-orodij vsebuje funkcijo, ki omogoča samodejno generiranje SQL-kode za ciljni SUPB. Na podlagi generirane SQL-kode v CASE-orodju lahko v ciljnem SUPB-u ustvarimo fizični model podatkovne baze. Logični model poenostavljene podatkovne baze knjižnice smo izdelali v CASE-orodju DBDesigner. V nadaljevanju (razdelek 5.5) bomo prikazali tudi SQL-kodo, ki nam jo omenjeno orodje samodejno generira na podlagi logičnega diagrama, predstavljenega na sliki Slika 87.

V diplomski nalogi bomo torej izdelavo fizičnega modela prikazali na dva načina, s pisanjem ustreznih SQL-stavkov in s pomočjo orodja DBDesigner. Najprej si bomo pogledali, kako izdelamo fizični model s pisanjem ustreznih ukazov v ciljnem SUPB-u. Ciljni SUPB, ki smo si ga izbrali, je Oracle Database. Oracle Database (v nadaljevanju Oracle) je sistem za upravljanje relacijskih podatkovnih baz, ki ga proizvaja in trži podjetje Oracle. SUPB Oracle ponuja ogromno funkcij, s katerimi bazo ustvarimo in jo nato upravljamo. Izbrali smo ga zato, ker je preprost za namestitev, uporabo in upravljanje ter je priljubljen med uporabniki. Ponuja tudi brezplačno različico (potrebna pa je predhodna registracija uporabnika) Oracle Database Express Edition 11g (Oracle Database XE), ki jo bomo tudi uporabili za izgradnjo našega fizičnega modela. Podatkovna baza Oracle Database XE je dosegljiva na spletnem naslovu podjetja

Oracle (http://www.oracle.com/index.html). Brezplačna verzija Oraclove baze je sicer okrnjena, vendar za naš primer zadostuje. Na voljo je za operacijske sisteme Windows in Linux in omogoča nam shranjevanje do 11 GB podatkov.

Pri izdelavi fizičnega modela poenostavljene baze knjižnice bomo izhajali iz naslednjega logičnega podatkovnega modela (Slika 87).



Slika 87: Logični podatkovni model knjižnice

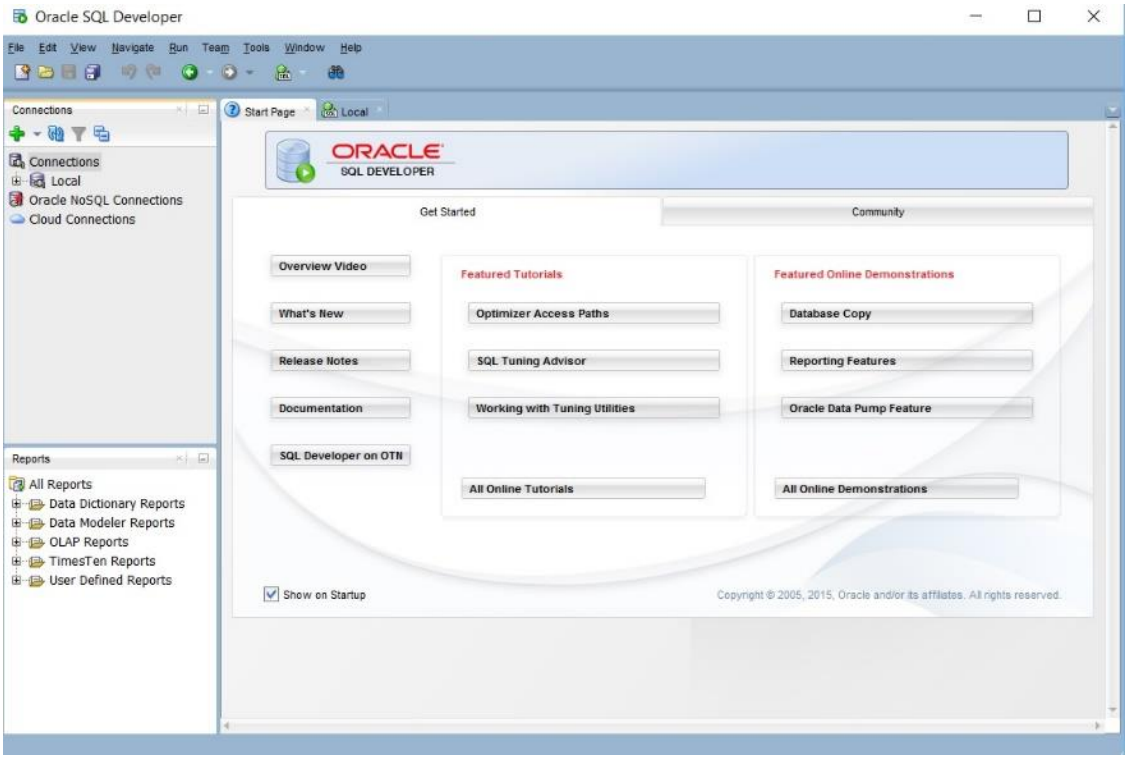
Za izgradnjo poenostavljene podatkovne baze knjižnice (fizičnega modela) bomo uporabili razvojno okolje Oracle SQL Developer (v nadaljevanju SQL Developer). V SQL Developerju bomo definirali strukturo podatkovne baze. Z ukazom `CREATE` bomo najprej ustvarili tabele, ki nimajo tujega ključa (»KLJUČNA_BESEDA«, »AVTOR«, »POŠTA« in »KNJIGA«), nato pa še tabele s tujim ključem (»ČLAN«, »REZERVACIJA«, »IZVOD«, »IMA« in »OPISUJE«). Narejene tabele bomo nato napolnili še z nekaj testnimi podatki.

5.4 Razvojno okolje SQL Developer

SQL Developer je brezplačno integrirano razvojno okolje, ki poenostavlja razvoj in upravljanje Oraclovih podatkovnih baz. Ponuja vrsto funkcij, kot so: ustvarjanje tabel, pogledov, indeksov in drugih objektov podatkovne baze, omogoča pregled podatkov v tabelah, uvoz in izvoz podatkov v želeni obliki (XML,

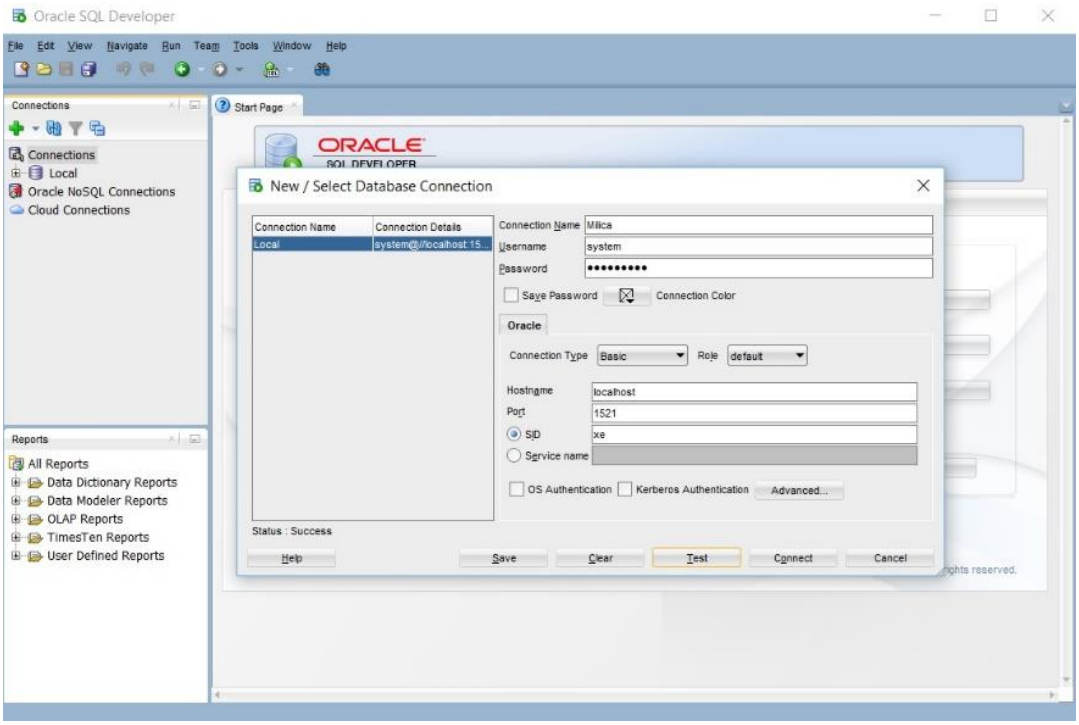
Excel, HTML, PDF itd.), izvajanje poizvedbe in SQL-skript, pregledovanje in ustvarjanje poročil in še veliko drugih funkcij. SQL Developer ima tudi funkcije sodobnega integriranega razvojnega okolja, kot so razhroščevalniki, samodejno dopolnjevanje ukazov, formatiranje kode itd. Omogoča zunanje razširitve, tako tiste, ki jih ponuja Oracle, kot tudi od drugih ponudnikov.

Kot smo omenili, je SQL Developer brezplačno razvojno okolje in je dostopno na spletni strani podjetja Oracle. Za uspešno namestitev in delovanje SQL Developerja potrebujemo tudi razvojni paket Java Development Kit (JDK). Na sliki Slika 88 je predstavljeno razvojno okolje SQL Developer.



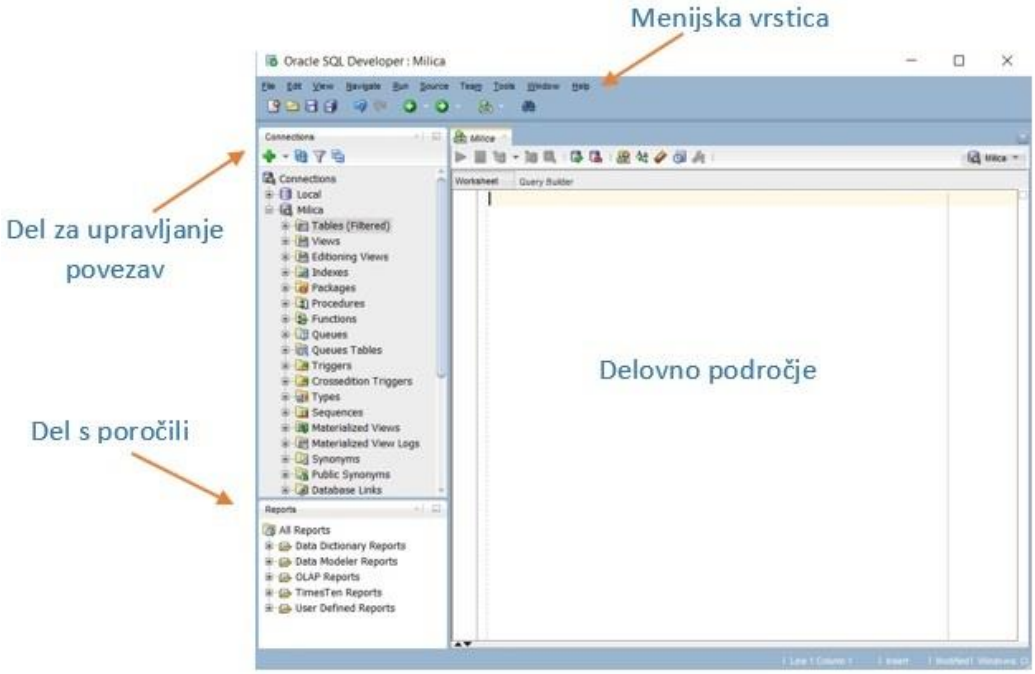
Slika 88: Razvojno okolje SQL Developer

SQL Developer je vmesnik, s pomočjo katerega se povežemo in upravljamo Oraclove podatkovne baze. Predpostavimo, da smo na računalnik namestili strežnik Oracle Database XE.
Na začetku je treba ustvariti novo povezavo na Oraclovo bazo. Povezavo ustvarimo s klikom na zeleno ikono plus () in nato kliknemo na `Nova povezava` (»`New connection`«). Odpre se nam okno, v katerem, kot prikazuje Slika 89, dopolnimo polja z manjkajočimi podatki. Nato kliknemo na gumb `Test` (»`Test`«). Če so vsi vneseni podatki točni, se po izvedbi testa izpiše status uspešen (»Success«). Po uspešnem testu kliknemo na gumb `Poveži` (»`Connect`«), s katerim se povežemo na bazo (v našem primeru na Oracle Database XE).



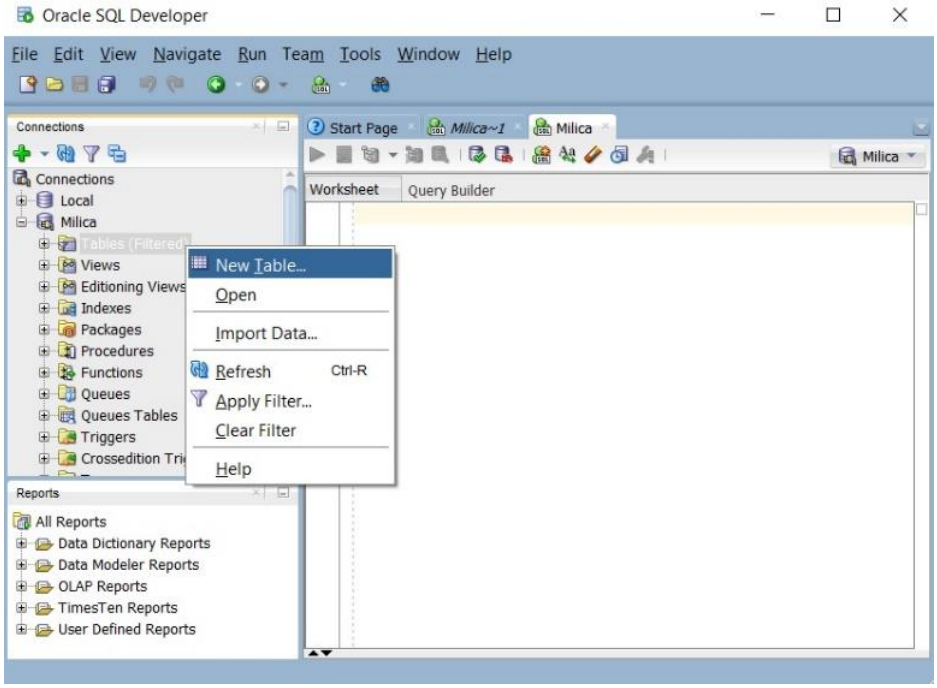
Slika 89: Okno povezave na bazo

Po uspešni povezavi s podatkovno bazo se odpre okno, ki je sestavljeno iz dveh delov. V zgodnjem prvem delu je upravljalnik povezav z navigatorjem med objekti, kot so: tabele, indeksi, pregledi, funkcije, procedure, paketi itd. V spodnjem prvem delu pa je zbirka vnaprej pripravljenih sistemskih in uporabniških poročil. Drugi del pa predstavlja delovno področje oziroma delovni list (Worksheet), ki je namenjen izvajanju SQL-ukazov in skript, pregledovanju rezultatov itd. Zgoraj je še glavni meni oziroma menijska vrstica. Sestavne dele razvojnega okolja Oracle SQL Developer prikazuje Slika 90.



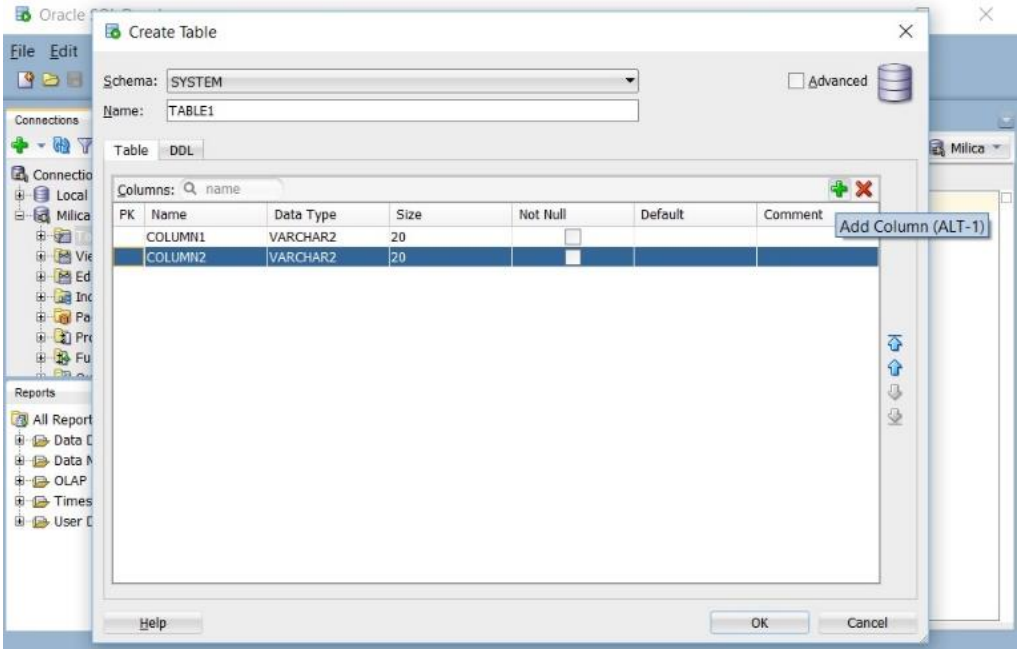
Slika 90: Sestavni deli razvojnega okolja SQL Developer

V delovnem listu razvojnega okolja SQL Developer bomo izdelali fizični model podatkovne baze knjižnice. Ustvarili bomo torej tabele, določili integritetne omejitve in nato tabele napolnili s podatki. Objekte fizičnega modela (tabele, poglede, indekse itd.) lahko v SQL Developer ustvarimo na dva načina. Prvi način je ustvarjanje objektov s pomočjo jezika SQL. Torej v delovnem listu preprosto pišemo ustrezne SQL-ukaze, s katerimi objekte ustvarimo, brišemo, spreminjamo, napolnimo s podatki itd. Drugi način pa omogoča, da objekte ustvarimo, brišemo, spreminjamo s preprostim klikanjem. Na primer, če želimo ustvariti novo tabelo, preprosto uporabimo desni klik z miško na objekt `Tabele` (»`Tables`«) in izberemo možnost `Ustvari novo tabelo` (»`New Table`«), kot prikazuje Slika 91.



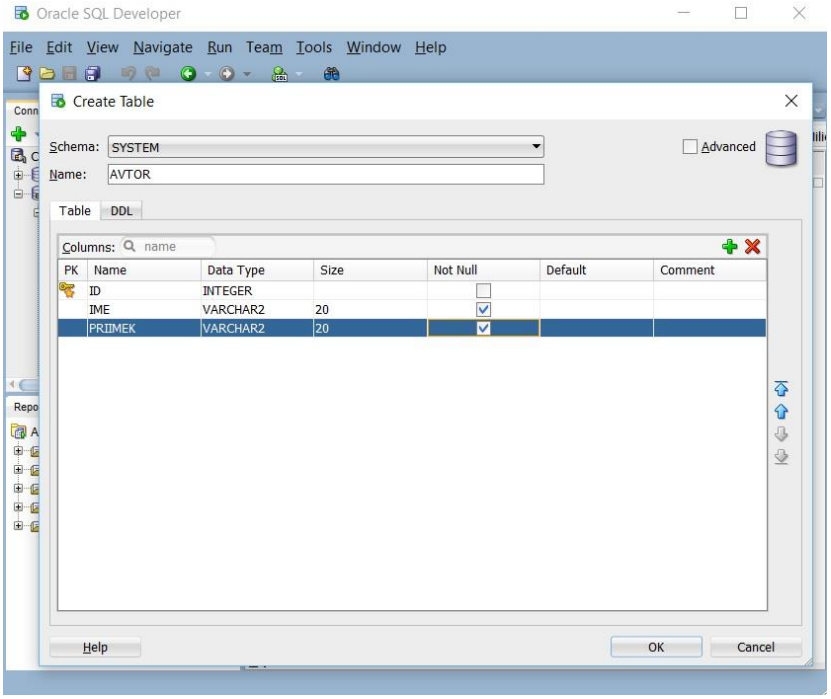
Slika 91: Prikaz ustvarjanja nove tabele

Odpre se pojavno okno, kjer v posamezna polja vpišemo ustrezne podatke, kar prikazuje Slika 92.

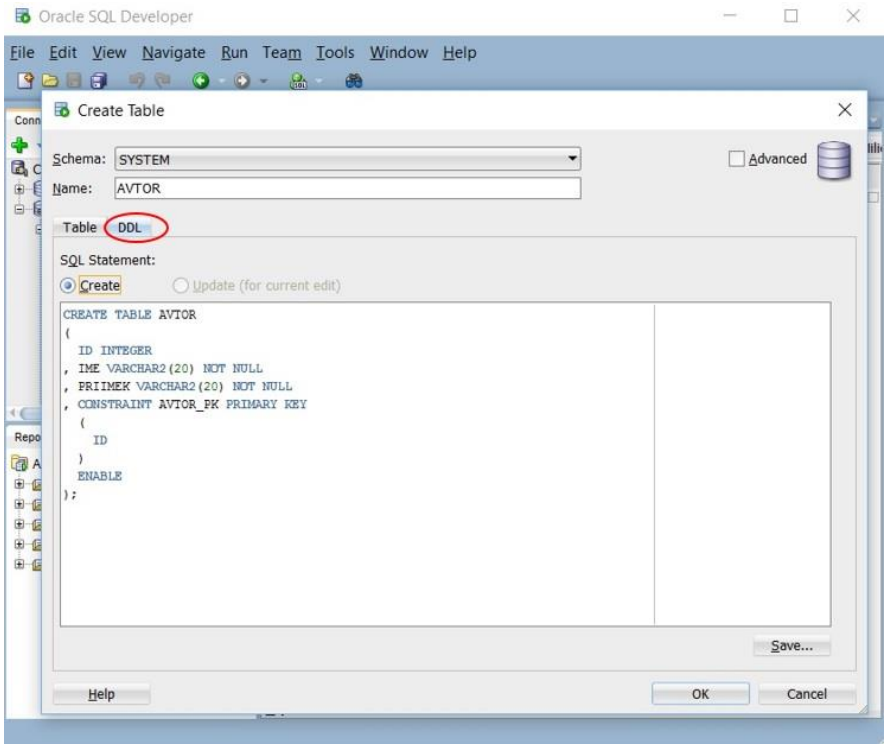


Slika 92: Pojavno okno za ustvarjanje tabele

Pri tem lahko stolpcem določimo tudi integritetne omejitve. O teh sintaksah bomo podrobneje spregovorili v naslednjem razdelku (podpoglavje 5.4), ker pa bomo v našem primeru tabele ustvarili in jih napolnili s pomočjo ukazov jezika SQL, se pri tem ne bomo ustavljali. Naj samo omenimo, da lahko s klikom na zavihek »DDL« pogledamo, kakšen je ustrezen SQL ukaz za tabelo, ki smo jo ustvarili s klikanjem. Slika 93 prikazuje tabelo »AVTOR«, ki smo jo ustvarili s klikanjem. Slika 94 pa prikazuje njen SQL-ukaz, ki ga dobimo s klikom na zavihek »DDL«. Ta način ustvarjanja tabel je lahko tudi priročen način učenja SQL-ukaza `CREATE TABLE`.

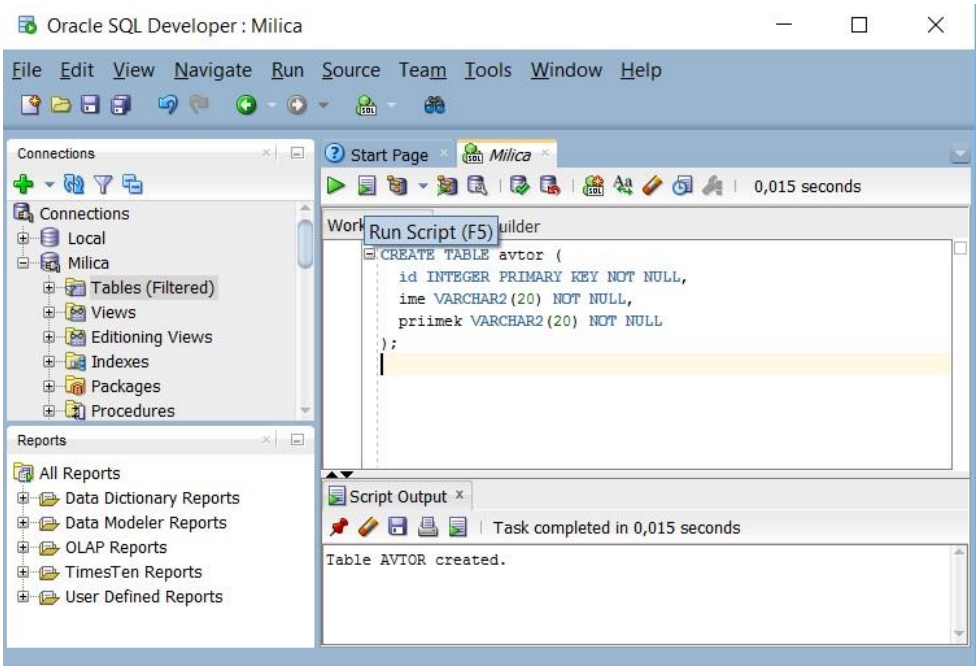


Slika 93: Ustvarjanje tabele "AVTOR" v pojavnem oknu



Slika 94: Pogled SQL-ukaza za tabelo "AVTOR"

Poglejmo, kako s pisanjem SQL-ukazov ustvarimo tabelo. Kot bomo videli v nadaljevanju, tabelo ustvarimo s pomočjo SQL-ukaza `CREATE`. Slika 95 prikazuje primer ustvarjanja tabele »AVTOR« v delovnem listu SQL Developer. Za izvedbo ukaza je potreben klik na gumb »`Run Statement`« ( ) ali »`Run Script`« ( ). Posamezen ukaz pa lahko poženemo tudi s tipkama »Ctrl+Enter« ali tipko »F5«.



Slika 95: Prikaz pisanja SQL ukaza

V nadaljevanju si bomo podrobneje ogledali, kako v SQL določimo integritetne omejitve, s pomočjo katerih ustvarimo tabele.

5.5 Določitev integritetnih omejitev

Podatki v podatkovni bazi morajo biti natančni, celoviti in razpoložljivi vsem uporabnikom podatkovne baze. Če podatki niso natančni in celoviti, bodo potem tudi delo in odločitve, ki jih sprejemamo na podlagi teh podatkov, napačne ali pa vsaj ne najboljše. Za celovitost podatkov v podatkovni bazi poskrbimo s pomočjo omejitev oziroma pravil, ki jih določimo nad podatki. V nadaljevanju si bomo pogledali, kako z uporabo jezika SQL določimo omejitve, ki določajo obveznost podatkov, omejitve domen, entitetno omejitev (primarni ključ) in referenčno omejitev (tuji ključ).

5.5.1 Omejitev, ki določa obvezen vnos podatka

Kaj je omejitev, ki določa obvezen vnos podatka, smo povedali že v 4. poglavju, in sicer v razdelku 4.1. Naj ponovno omenimo, da je to pogoj, s katerim določimo, kateri stolpci (atributi) v podatkovni bazi ne smejo imeti nedoločene vrednosti oziroma vrednosti Null. Če želimo preprečiti vnos vrednost Null v posamezni stolpec tabele, v jeziku SQL pri definiciji stolpca uporabimo omejitev **`NOT NULL`**.
Poglejmo si ponovno tabelo »IZVOD«, ki je prikazana na sliki Slika 96.

Id	Signatura	Status	Datum vrnitve
1	004	prost	Null
2	821	izposojen	12.06.2016
3	004	izposojen	27.05.2016
4	821	prost	Null
5	011	prost	Null

Slika 96: Tabela "IZVOD"

Tabelo bi ustvarili tako:

```
CREATE TABLE izvod (
id INTEGER,
signatura VARCHAR2(10),
status VARCHAR2(9),
datum_vrnitve DATE
);
```

Vendar s tem nismo določili nobenih omejitev. Omejitev, ki določa obvezen vnos podatka, bomo definirali le za stolpec »Status«:

```
CREATE TABLE izvod (
...
status VARCHAR2(9) NOT NULL,
datum_vrnitve DATE
);
```

V primeru knjižnice bomo omejitev `NOT NULL` uporabili pri vseh atributih razen pri atributu »Kraj« v tabeli »POŠTA« in atributu »Datum vrnitve« v tabeli »IZVOD«. S tem bomo dosegli, da se bodo vrednosti Null v bazi pojavile čim manjkrat oziroma le takrat, ko je to potrebno. Vrednosti Null nam namreč lahko povzročijo veliko težav, zato je pomembno, da za podatke, ki so v situacijah, ki jih modeliramo, obvezni, postavimo omejitev `NOT NULL`.

5.5.2 Omejitve domene

V razdelku 3.1 smo omenili, da moramo za vsak stolpec v tabeli določiti domeno. Del posla smo opravili s tem, da smo določili podatkovni tip. Vendar to včasih ni dovolj. Določeni stolpci v tabeli imajo lahko podatke, ki so le del podatkov, ki jih omogoča nek podatkovni tip. Na primer v določenem stolpcu imamo lahko le pozitivna števila, le dve vrednosti itd. Te omejitve v jeziku SQL izvedemo z uporabo določila **`CHECK`**. Določilo `CHECK` je zelo koristno in uporabno, saj omogoča raznovrstne omejitve glede vrednosti polj. Omejitve domen najpogosteje nastavimo že pri samem ustvarjanju tabele, lahko pa tudi kasneje. Sintaksa določila `CHECK` je sledeča:

```
CHECK (iskalni pogoj)
```

V iskalnem pogoju se lahko neposredno sklicujemo na stolpec tabele, v kateri ustvarimo omejitev. Na ta način ustvarimo omejitev stolpca. Recimo, da želimo v tabeli »IZVOD« v stolpcu »Signatura« preprečiti, da bi tam kot podatek imeli negativno število. V tem primeru bi nastavili omejitev, da signatura nikoli ne more biti manjša od 0. To prikazuje spodnji primer:

```
CREATE TABLE izvod (
...
signatura INTEGER NOT NULL CHECK (signatura>0),
...
);
```

Omejitev pa lahko definiramo tudi kot omejitev nad tabelo in jo zapišemo kasneje v tabeli, kot prikazuje naslednji primer:

```
CREATE TABLE izvod (
...
signatura VARCHAR2(10) NOT NULL,
...
CHECK (signatura>0),
...
);
```

Pogoje v določilu `CHECK` lahko poljubno kombiniramo, npr. z logičnimi operatorji in (`AND`), ali (`OR`), ne (`NOT`). Lahko pa v pogoju uporabimo tudi operatorje, kot sta `IN`, `BETWEEN` itd. Spomnimo se, da smo med uporabniškimi zahtevami za podatkovno bazo knjižnice navedli, da v stolpcu »Status« tabele »IZVOD« hranimo le dve vrednosti, in sicer `prost` in `izposojen`. To omejitev zapišemo tako, da za podatkovni tip uporabimo `VARCHAR(9)`, nato pa znotraj omejitve `CHECK` uporabimo operator `IN` in naštejemo možne vrednosti, ki jih lahko zavzema stolpec »Status«:

```
CREATE TABLE izvod (
...
status VARCHAR2(9) NOT NULL,
CHECK (status IN ('prost', 'izposojen')),
...
);
```

Poglejmo si še en zgled iz primera knjižnice, kjer bomo potrebovali omejitev `CHECK`. Radi bi, da se v knjižnici hranijo samo knjige, ki so izšle po letu 1800. Torej želimo, da so v stolpcu »Leto izida« v tabeli »KNJIGA« vrednosti, ki jih vnašamo, večje od 1800. To bomo dosegli tako, da za podatkovni tip stolpca »Leto izida« uporabimo `NUMBER(4)`, nato pa znotraj omejitve `CHECK` določimo, da vrednost ne sme biti večja od 1800, kot prikazuje naslednji primer:

```
CREATE TABLE knjiga (
...
leto_izida NUMBER(4) NOT NULL CHECK (leto_izida>1800),
...
);
```

S tem, ko smo stolpec »Leto izida« omejili, da mora biti vneseno število večje od 1800, smo preprečili tudi, da bi uporabniki za leto izida knjige vnesli števila, kot so npr. 2, 12, 123 in podobno, ki jih sicer podatkovni tip `NUMBER(4)` dopušča, vendar pa ne predstavljajo veljavnega leta.

5.5.3 Entitetna omejitev

Kot smo omenili v razdelku 3.1, entitetna omejitev določa, da primarni ključ ne sme imeti vrednosti Null. To pomeni, da stolpec, ki sestavlja primarni ključ, ne sme vsebovati vrednosti Null. Če pa je primarni ključ sestavljen npr. iz dveh stolpcev, ne smemo dopustiti, da bi oba stolpca imela vrednost

Null. Vsaka tabela ima lahko le en primarni ključ. Stolpec (ali množica stolpcev), ki ga izberemo za primarni ključ, dobi v tabeli posebno mesto. Stolpec v tabeli, ki predstavlja primarni ključ, definiramo tako, da mu pripišemo omejitev **`PRIMARY KEY`**.
Za zgled vzemimo stolpec »Id« tabele »IZVOD«, ki smo ga izbrali za primarni ključ. Pri definiranju stolpca »Id« dodamo omejitev `PRIMARY KEY`, kot prikazuje spodnji primer:

```
CREATE TABLE izvod (
id INTEGER PRIMARY KEY,
...
);
```

V primeru, da primarni ključ sestavljata dva ali več stolpcev, pa ne moremo uporabiti omejitve `PRIMARY KEY` pri definiciji več stolpcev, ampak je treba omejitev definirati kot omejitev tabele. Torej za definicijami vseh stolpcev napišemo omejitev `PRIMARY KEY (seznam stolpcev)`.

Seznam stolpcev predstavlja stolpce, ki sestavljajo primarni ključ.

Če bi pri primeru tabele »IZVOD« primarni ključ sestavljala npr. stolpca »Id« in »Signatura«, bi za definicijami vseh stolpcev napisali omejitev, kot prikazuje naslednji primer:

```
CREATE TABLE izvod (
id INTEGER,
signatura VARCHAR2(10),
...
PRIMARY KEY (id, signatura)
);
```

Lahko pa uporabimo še eno obliko zapisa omejitve tabele, kjer samo omejitev poimenujemo. Sintaksa te omejitve je naslednja:

```
CONSTRAINT ime_omejitve PRIMARY KEY(seznam stolpcev).
```

Seznam stolpcev predstavlja stolpce, ki sestavljajo primarni ključ.

Za naš primer bi torej napisali:

```
CREATE TABLE izvod (
id INTEGER,
signatura VARCHAR2(10),
...
CONSTRAINT izvod_pk PRIMARY KEY(id, signatura)
);
```

V primeru knjižnice bomo omejitev `PRIMARY KEY` uporabili pri definiranju vseh primarnih ključih. Z omejitvijo `PRIMARY KEY` poleg stolpca oziroma za definicijami vseh stolpcev bomo tudi preprečili vnos vrednosti Null v te stolpce. Pri tabelah »KLJUČNA BESEDA«, »AVTOR«, »POŠTA«, »ČLAN«, »IZVOD« in »REZERVACIJA« bomo omejitev `PRIMARY KEY` definirali pri samih stolpcih, saj v omenjenih tabelah primarni ključ sestavlja samo en stolpec. Za tabeli »IMA« in »OPISUJE« pa bomo omejitev `PRIMARY KEY` definirali kot omejitev tabele, saj imata omenjeni tabeli primarni ključ sestavljen iz dveh stolpcev.

Poglejmo si še primer definiranja primarnega ključa za tabelo »IMA«:

```
CREATE TABLE ima (
ISBN VARCHAR2(10),
id INTEGER,
CONSTRAINT ima_pk PRIMARY KEY(ISBN, id),
...
);
```

5.5.4 Referenčna omejitev

Omenili smo, da referenčna omejitev povezuje dve tabeli in zagotavlja skladnost med zapisi v teh dveh tabelah. To pomeni, da zagotovimo, da so v določenem stolpcu ene tabele le tisti podatki, ki so v stolpcu druge tabele. Pri tem mora ta stolpec druge tabele biti primarni ključ, stolpec prve tabele pa smo poimenovali tuji ključ. Tuji ključ definiramo lahko na več načinov. Načini definiranja tujega ključa so odvisni od SUPB-a, ki ga uporabljamo. V našem primeru bomo tuji ključ definirali z uporabo določila `CONSTRAINT`. Kot smo omenili pri entitetni omejitvi, imamo v eni tabeli lahko le en primarni ključ, vendar pa imamo lahko več tujih ključev. Sintaksa ukaza za definiranje tujega ključa je:

```
CONSTRAINT ime_omejitve
FOREIGN KEY (ime tujega ključa)
REFERENCES ime_tabele(ime primarnega ključa)
```

V primeru, da sta imeni stolpcev v eni in drugi tabeli (ime primarnega ključa in ime tujega ključa) enaka, lahko izpustimo ime primarnega ključa, na katerega se tuji ključ nanaša.

```
CONSTRAINT ime_omejitve
FOREIGN KEY (ime tujega ključa)
REFERENCES ime_tabele
```

Določilo `REFERENCES` podaja način, kako se referencirana tabela (tabela s tujim ključem) sklicuje na primarni ključ druge tabele.

Vzemimo za zgled tabeli »ČLAN« in »POŠTA«, predstavljeni na sliki Slika 97.

Članska številka	Ime	Priimek	Ulica	Poštna številka	Potek članstva
1020868	Tina	Kovač	Srebrničeva ulica 10	5210	12.09.2016
1080869	Jure	Novak	Martinuči 1d	5292	15.10.2016
1029870	Nejc	Kos	Erjavčeva ulica 2a	5000	25.10.2016
1030871	Tina	Novak	Ulica talcev 12	5210	13.12.2016
1020222	Marko	Zimic	Partizanska ulica 1	5000	04.01.2017

Poštna številka	Kraj
5210	Deskle
5292	Renče
5000	Nova Gorica

Slika 97: Tabela "ČLAN" in "POŠTA"

Referenčna integriteta med omenjenima tabelama mora obstajati med stolpcem »Poštna številka« v tabeli »POŠTA«, ki je primarni ključ, in med stolpcem »Poštna številka«, ki je v tabeli »ČLAN« tuji ključ. Zapis referenčne omejitve med omenjenima tabelama prikazuje naslednji primer:

```
CREATE TABLE posta (
 postna_stevilka INTEGER PRIMARY KEY,
 kraj VARCHAR2(45)
);

CREATE TABLE clan (
clanska_stevilka VARCHAR(7) PRIMARY KEY,
...
postna_stevilka INTEGER,
CONSTRAINT clan_fk1 FOREIGN KEY (postna_stevilka) REFERENCES
posta(postna_stevilka)
);
```

Omejitev `clan_fk1` pove, da je stolpec »Poštna številka« referenca na stolpec »Poštna številka« v tabeli »POŠTA«. To pomeni, da je stolpec »Poštna številka« v tabeli »ČLAN« tuji ključ, ki se sklicuje na primarni ključ tabele »POŠTA«. Vrednosti v stolpcu »Poštna številka« v tabeli »ČLAN« se morajo tako ujemati z vrednostmi primarnega ključa (»Poštna številka«) v tabeli »POŠTA« ali pa morajo biti Null.

Ker je ime tujega ključa v tabeli »ČLAN« enako imenu primarnega ključa v tabeli »POŠTA«, lahko pri definiranju tujega ključa izpustimo ime primarnega ključa, kot kaže naslednji primer:

```
CREATE TABLE posta (
postna_stevilka INTEGER PRIMARY KEY,
kraj VARCHAR2(45)
);

CREATE TABLE clan (
clanska_stevilka VARCHAR(7) PRIMARY KEY,
...
postna_stevilka INTEGER,
CONSTRAINT clan_fk1 FOREIGN KEY (postna_stevilka) REFERENCES posta
);
```

Referenčno omejitev bomo v primeru knjižnice potrebovali za vse tabele, ki imajo tuji ključ, to so tabele »ČLAN«, »REZERVACIJA«, »IZVOD«, »IMA« in »OPISUJE«. Z določitvijo referenčne omejitve bomo tabele med seboj povezali in tako zagotovili celovitost podatkov.

Poglejmo si še en zgled uporabe referenčne omejitve v primeru knjižnice. Radi bi določili referenčno omejitev za tabelo »REZERVACIJA«. Stolpec »ISBN« se v tabeli rezervacija sklicuje na primarni ključ tabele »KNJIGA«, stolpec »Članska številka« pa na primarni ključ tabele »ČLAN«, zato moramo najprej ustvariti tabeli »KNJIGA« in »ČLAN«. Kot smo prej omenili, se tabela »ČLAN« sklicuje na tabelo »POŠTA«, zato moramo pred tabelo »ČLAN« ustvariti še tabelo »POŠTA«. Če želimo določiti referenčno omejitev za tabelo »REZERVACIJA«, moramo torej prej ustvariti tabele »KNJIGA«, »POŠTA« in »ČLAN«, kot prikazuje naslednji primer:

```
CREATE TABLE knjiga(
ISBN VARCHAR2(13) PRIMARY KEY,
...
);

CREATE TABLE posta (
postna_stevilka INTEGER PRIMARY KEY,
kraj VARCHAR2(45)
);

CREATE TABLE clan (
clanska_stevilka VARCHAR2(7) PRIMARY KEY,
...,
postna_stevilka INTEGER,
CONSTRAINT clan_fk1 FOREIGN KEY (postna_stevilka) REFERENCES
  posta(postna_stevilka)
);

CREATE TABLE rezervacija (
id INTEGER PRIMARY KEY,
...,
clanska_stevilka VARCHAR2(7),
ISBN VARCHAR2(13),
CONSTRAINT rezervacija_fk1 FOREIGN KEY (clanska_stevilka) REFERENCES
  clan(clanska_stevilka)
CONSTRAINT rezervacija_fk2 FOREIGN KEY (ISBN) REFERENCES knjiga(ISBN)
);
```

5.5.5 Omejitvi UNIQUE in DEFAULT

V določenih situacijah, ki jih modeliramo, imamo lahko podano zahtevo, da mora določen stolpec ali kombinacija stolpcev vsebovati enolične vrednosti. Torej želimo, da so vse vrednosti v posameznem stolpcu ali vrednosti v kombinaciji stolpcev različne. Tega z do zdaj spoznanimi omejitvami ne moremo doseči. V nadaljevanju si bomo pogledali, kako v SQL določimo, da določen stolpec ali kombinacija stolpcev vsebuje le enolične vrednosti. Zgled bomo prikazali na primeru knjižnice.

Želimo torej, da stolpec ali kombinacija stolpcev vsebuje enolične vrednosti. Vse vrednosti v posameznem stolpcu ali v kombinaciji stolpcev morajo torej biti različne. To sicer že velja za primarni ključ, a vemo, da lahko enoličnost dosežemo tudi pri ostalih stolpcih, ne da bi jih določili za primarni ključ. Samo enoličnost stolpcev brez tega, da bi stolpec določili za primarni ključ, dosežemo z uporabo določila **`UNIQUE`**. Vzemimo za zgled tabelo »KNJIGA«, predstavljeno na sliki Slika 98.

ISBN	Naslov	Založnik	Leto izida
9789610118176	Srečni človek	Mladinska knjiga	2012
9616176722	Uvod v SQL	Flamingo	2002
9616046055	Načrtovanje relacijskih podatkovnih baz	BI-TIM	1997
9789610204954	SQL	DZS	2013
9789610027300	Skoraj popolna	Učila International	2015
9619075501	Informatika	Bons	1999

Slika 98: Tabela "KNJIGA"

Želimo, da so naslovi knjig enolični, saj nočemo v bazi imeti dveh knjig z istim naslovom. Takrat enoličnost za stolpec »Naslov« definiramo z določilom `UNIQUE`:

```
CREATE TABLE knjiga (
...
naslov VARCHAR2(45) NOT NULL UNIQUE,
...
);
```

Morda pa je taka omejitev preveč omejujoča. Želimo le preprečiti, da bi enak naslov imeli dve knjigi, izdani v istem letu. Hočemo torej, da bo vrednost kombinacije stolpcev »Naslov« in »Leto izida« enolična. Uporabimo omejitev tabele in napišemo:

```
CREATE TABLE knjiga (
...
naslov VARCHAR2(45) NOT NULL,
...
leto_izida NUMBER(4) NOT NULL,
UNIQUE (naslov, leto_izida)
);
```

Lahko pa bi omejitev tudi poimenovali in napisali:

```
CREATE TABLE knjiga (
...
naslov VARCHAR2(45) NOT NULL,
...
leto_izida NUMBER(4) NOT NULL,
CONSTRAINT ena_na_leto UNIQUE (naslov, leto_izida)
);
```

V poglavju o omejitvah si oglejmo še eno možnost, ki pravzaprav ni omejitev. Uporabnikom bi radi omogočili, da jim ne bi bilo treba vnašati določenih podatkov, ki se pogosto ponavljajo.
Denimo, da želimo uporabnikom omogočiti, da jim pri vnosu nove knjige v tabelo »IZVOD« ne bi bilo treba vedno, ko je knjiga prosta, vnašati v stolpec »Status« vrednost `'prost'`. V tem primeru določimo stolpcu »Status« privzeto vrednost `'prost'`. Tako npr. vemo, da je vedno, ko v tabelo »IZVOD« dodamo nov zapis (novo knjigo), vrednost stolpca "Status" enaka `'prost'`. Zato nam pri novem vnosu, v primeru, ko je izvod prost, ni treba vnašati vrednosti.

V SQL privzeto vrednost določimo v definiciji stolpca s pomočjo določila **`DEFAULT`**. Privzeta vrednost mora ustrezati vrednosti podatkovnega tipa stolpca in morebitnim dodatnim omejitvam, ki jih ima ta stolpec. Privzeta vrednost pa je lahko tudi vrednost Null.

Za stolpec »Status« v tabeli »IZVOD« želimo torej določiti privzeto vrednost `'prost'`. To storimo tako, da pri definiranju stolpca »Status« dodamo določilo `DEFAULT`, kot prikazuje spodnji primer:

```
CREATE TABLE izvod (
...
status VARCHAR2(9) DEFAULT 'prost',
...
);
```

5.6 Ustvarjanje tabel

Preden začnemo podatke shranjevati v podatkovno bazo, moramo najprej ustvariti njeno strukturo. Torej ustvariti moramo tabele, kjer bomo hranili podatke. Tabele v SQL ustvarimo z ukazom **`CREATE TABLE`**. Osnovno obliko smo si že ogledali. V splošnem ta ukaz zahteva še ime tabele, seznam stolpcev in njihovih podatkovnih tipov ter po potrebi še omejitve, s katerim določimo različne omejitve stolpcev. Osnovna sintaksa ukaza je:

```
CREATE TABLE ime_tabele (
ime_stolpca1 podatkovni_tip1 omejitve,
ime_stolpca2 podatkovni_tip2 omejitve,
ime_stolpca3 podatkovni_tip3 omejitve,
...
);
```

Kot smo omenili, so tabele sestavljene iz stolpcev (atributov) in vrstic (zapisov). Vsaka tabela pa mora imeti tudi svoje ime. Ime stolpca je niz, ki dejansko predstavlja posamezen atribut tabele. Podatkovni tip predstavlja enega od osnovnih SQL podatkovnih tipov, ki smo jih predstavili v razdelku 4.2. Pri ustvarjanju naše poenostavljene podatkovne baze knjižnice v Oracle bomo uporabili podatkovne tipe `INTEGER`, `NUMBER`, `VARCHAR2` in `DATE`. Omejitve pa uporabljamo za navedbo omejitev stolpcev oziroma specifičnih lastnosti stolpcev (npr. obvezen vnos podatka, le pozitivna števila, primarni ključ, tuji ključ itd.).

Oglejmo si zdaj, kako v Oracle ustvarimo tabele naše baze. Najprej ustvarimo tabele brez tujih ključev. Slika 99 prikazuje tabele brez tujih ključev, te so: »KLJUČNA BESEDA«, »AVTOR«, »POŠTA« in »KNJIGA«.

```
--------------------------------------------------------------------------
--Tabela ključnih besed, ki jih knjige vsebujejo
--------------------------------------------------------------------------
CREATE TABLE kljucna_beseda (
  id INTEGER PRIMARY KEY,
  naziv VARCHAR2(45) NOT NULL
);

--------------------------------------------------------------------------
--Tabela avtorjev knjig
--------------------------------------------------------------------------
CREATE TABLE avtor (
  id INTEGER PRIMARY KEY,
  ime VARCHAR2(20) NOT NULL,
  priimek VARCHAR2(20) NOT NULL
);

--------------------------------------------------------------------------
--Tabela poštnih številk s kraji
--------------------------------------------------------------------------
CREATE TABLE posta (
  postna_stevilka INTEGER PRIMARY KEY,
  kraj VARCHAR2(45)
);

--------------------------------------------------------------------------
--Tabela knjig, ki so v knjižnici
--------------------------------------------------------------------------
CREATE TABLE knjiga (
  ISBN VARCHAR2(13) PRIMARY KEY,
  naslov VARCHAR2(45) NOT NULL,
  zaloznik VARCHAR2(45) NOT NULL,
  leto_izida NUMBER(4) NOT NULL CHECK (leto_izida>1800)
);
```

Slika 99: Tabele brez tujih ključev

Kot vidimo, smo zahtevali, da so vsi podatki, z izjemo kraja, vneseni. Nato ustvarimo še tabele s tujimi ključi. Slika 100 prikazuje tabele s tujimi ključi. Te so: »ČLAN«, »REZERVACIJA«, »IZVOD«, »IMA« in »OPISUJE«.

```
--------------------------------------------------------------------------
--Tabela članov knjižnice
--------------------------------------------------------------------------
CREATE TABLE clan (
  clanska_stevilka VARCHAR2(7) PRIMARY KEY,
  ime VARCHAR2(20) NOT NULL,
  priimek VARCHAR2(20) NOT NULL,
  ulica VARCHAR2(45) NOT NULL,
  potek_clanstva DATE NOT NULL,
  postna_stevilka INTEGER,
  CONSTRAINT clan_fk1 FOREIGN KEY (postna_stevilka) REFERENCES posta(postna_stevilka)
  );

--------------------------------------------------------------------------
--Tabela rezervacij članov, s katero si zagotovijo želeno knjigo
--------------------------------------------------------------------------
CREATE TABLE rezervacija (
  id INTEGER PRIMARY KEY,
  datum_rezervacije DATE NOT NULL,
  ISBN VARCHAR2(13),
  clanska_stevilka VARCHAR2(7),
  CONSTRAINT rezervacija_fk1 FOREIGN KEY (ISBN) REFERENCES knjiga(ISBN),
  CONSTRAINT rezervacija_fk2 FOREIGN KEY (clanska_stevilka) REFERENCES clan(clanska_stevilka)
);

--------------------------------------------------------------------------
--Tabela izvodov knjig
--------------------------------------------------------------------------
CREATE TABLE izvod (
  id INTEGER PRIMARY KEY,
  signatura VARCHAR2(10) NOT NULL,
  status_izvoda VARCHAR2(9) NOT NULL CHECK (status_izvoda IN ('prost', 'izposojen')),
  datum_vrnitve DATE,
  clanska_stevilka VARCHAR2(7),
  ISBN VARCHAR2(13),
  CONSTRAINT izvod_fk1 FOREIGN KEY (clanska_stevilka) REFERENCES clan(clanska_stevilka),
  CONSTRAINT izvod_fk2 FOREIGN KEY (ISBN) REFERENCES knjiga(ISBN)
);

--------------------------------------------------------------------------
--Tabela ima, ki združuje knjige in avtorje
--------------------------------------------------------------------------
CREATE TABLE ima (
  ISBN VARCHAR2(13),
  id INTEGER,
  CONSTRAINT ima_pk PRIMARY KEY(ISBN, id),
  CONSTRAINT ima_fk1 FOREIGN KEY (ISBN) REFERENCES knjiga(ISBN),
  CONSTRAINT ima_fk2 FOREIGN KEY (id) REFERENCES avtor(id)
);

--------------------------------------------------------------------------
--Tabela opisuje, ki združuje knjige in ključne besede
--------------------------------------------------------------------------
CREATE TABLE opisuje (
  ISBN VARCHAR2(13),
  id INTEGER,
  CONSTRAINT opisuje_pk PRIMARY KEY(ISBN, id),
  CONSTRAINT opisuje_fk1 FOREIGN KEY (ISBN) REFERENCES knjiga(ISBN),
  CONSTRAINT opisuje_fk2 FOREIGN KEY (id) REFERENCES kljucna_beseda(id)
);
```

Slika 100: Tabele s tujimi ključi

Tabele na sliki Slika 99 in sliki Slika 100 smo izdelali tako, da smo sami napisali ustrezne SQL-ukaze na osnovi logičnega modela, ki smo ga izdelali v fazi logičnega načrtovanja. Prej pa smo omenili, da nam določena orodja omogočajo tudi generiranje SQL-kode na podlagi izdelanega logičnega modela. Med

tovrstna orodja pa spada tudi orodje DBDesigner, ki smo ga uporabili za izdelavo logičnega modela. Poglejmo si sedaj, kako izgleda SQL-koda, ki nam jo samodejno generira orodje DBDesigner.

Če želimo v DBDesignerju izdelati SQL-kodo na podlagi logičnega diagrama, preprosto kliknemo na zavihek Datoteka (»File«) in nato pri možnosti Izvozi (»Export«) izberemo Ustvari SQL skripto (»SQL Create Script«). Kot smo omenili, je DBDesigner razvit in optimiziran za podporo MySQL, zato se sintaksa SQL-ukazov nekoliko razlikuje od naše, ki smo jo izdelali v Oracle.

```
CREATE TABLE KLJUCNA_BESEDA (
 Id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
 Naziv VARCHAR(20) NOT NULL,
 PRIMARY KEY(Id)
);

CREATE TABLE KNJIGA (
 ISBN VARCHAR(13) NOT NULL,
 Naslov VARCHAR(45) NOT NULL,
 Zaloznik VARCHAR(45) NOT NULL,
 Leto_izida YEAR NOT NULL,
 PRIMARY KEY(ISBN)
);

CREATE TABLE POSTA (
 Postna_stevilka INTEGER UNSIGNED NOT NULL,
 Kraj VARCHAR(45) NOT NULL,
 PRIMARY KEY(Postna_stevilka)
);

CREATE TABLE AVTOR (
 Id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
 Ime VARCHAR(20) NOT NULL,
 Priimek VARCHAR(20) NOT NULL,
 PRIMARY KEY(Id)
);

CREATE TABLE CLAN (
 Clanska_stevilka VARCHAR(7) NOT NULL,
 POSTA_Postna_stevilka INTEGER UNSIGNED NOT NULL,
 Ime VARCHAR(20) NOT NULL,
 Priimek VARCHAR(20) NOT NULL,
 Ulica VARCHAR(45) NOT NULL,
 Potek_clanstva DATE NOT NULL,
 PRIMARY KEY(Clanska_stevilka),
 FOREIGN KEY(POSTA_Postna_stevilka)
 REFERENCES POSTA(Postna_stevilka)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION
);

CREATE TABLE IZVOD (
 Id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
 CLAN_Clanska_stevilka VARCHAR(7) NOT NULL,
 KNJIGA_ISBN VARCHAR(13) NOT NULL,
 Signatura VARCHAR(10) NOT NULL,
 Status_izvoda VARCHAR(9) NOT NULL,
 Datum_vrnitve DATE NULL,
 PRIMARY KEY(Id),
 FOREIGN KEY(KNJIGA_ISBN)
 REFERENCES KNJIGA(ISBN)
 ON DELETE NO ACTION
```

```
 ON UPDATE NO ACTION,
 FOREIGN KEY(CLAN_Clanska_stevilka)
 REFERENCES CLAN(Clanska_stevilka)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION
);

CREATE TABLE REZERVACIJA (
 Id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
 KNJIGA_ISBN VARCHAR(13) NOT NULL,
 CLAN_Clanska_stevilka VARCHAR(7) NOT NULL,
 Datum_rezervacije DATE NOT NULL,
 PRIMARY KEY(Id),
 FOREIGN KEY(CLAN_Clanska_stevilka)
 REFERENCES CLAN(Clanska_stevilka)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION,
 FOREIGN KEY(KNJIGA_ISBN)
 REFERENCES KNJIGA(ISBN)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION
);

CREATE TABLE IMA (
 KNJIGA_ISBN VARCHAR(13) NOT NULL,
 AVTOR_Id INTEGER UNSIGNED NOT NULL,
 PRIMARY KEY(KNJIGA_ISBN, AVTOR_Id),
 FOREIGN KEY(AVTOR_Id)
 REFERENCES AVTOR(Id)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION,
 FOREIGN KEY(KNJIGA_ISBN)
 REFERENCES KNJIGA(ISBN)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION
);

CREATE TABLE OPISUJE (
 KNJIGA_ISBN VARCHAR(13) NOT NULL,
 KLJUCNA_BESEDA_Id INTEGER UNSIGNED NOT NULL,
 PRIMARY KEY(KNJIGA_ISBN, KLJUCNA_BESEDA_Id),
 FOREIGN KEY(KLJUCNA_BESEDA_Id)
 REFERENCES KLJUCNA_BESEDA(Id)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION,
 FOREIGN KEY(KNJIGA_ISBN)
 REFERENCES KNJIGA(ISBN)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION
);
```

Če pogledamo obe kodi, kodo, ki smo jo napisali sami z ustreznimi ukazi, in kodo, ki nam jo je generiralo orodje DBDesigner, opazimo nekaj razlik. Na primer pri kodi, ki smo jo pridobili z orodjem DBDesigner, vidimo, da imamo pri definiranju tujega ključa dodan del `ON DELETE NO ACTION` in `ON UPDATE NO ACTION`. Načeloma lahko te dele pri sami definiciji tujega ključa izpustimo. Če te dele odstranimo, postane koda precej podobna tisti, ki smo jo napisali sami. V kodi, ki nam jo je generiralo orodje DBDesigner, pri podatkovnem tipu `INTEGER` opazimo dodano določilo `UNSIGNED`. To določilo pomeni nepredznačeno obliko števila in ga lahko uporabljamo v MySQL za stolpec s celimi števili. Nepredznačen tip `UNSIGNED` v MySQL uporabimo bodisi ko želimo preprečiti vnos negativnih števil v

stolpec bodisi ko potrebujemo večje zgornje številsko območje za stolpec (podatkovni tip `INTEGER` z določilom `UNSIGNED` predstavlja obseg od 0 do 4.294.967.295). Pri določenih stolpcih (pri stolpcih z oznako »Id«) opazimo tudi določilo `AUTO_INCREMENT`. Določilo `AUTO_INCREMENT` uporablja MySQL in deluje podobno kot objekt zaporedje (`SEQUENCE`) v Oracle, ki ga bomo podrobneje spoznali v nadaljevanju. V stolpcih z določilom `AUTO_INCREMENT` se vrednost samodejno poveča za 1, ko v tabelo dodamo nov zapis.

S tem, ko smo v ciljnem SUPB-u ustvarili tabele in jim določili integritetne omejitve, smo izdelali fizični model relacijske podatkovne baze. Vendar če želimo testirati ustreznost in zmogljivost fizičnega modela, moramo tabele napolniti še s testnimi podatki. Zato si bomo v naslednjem poglavju pogledali, kako v tabele vnašamo podatke.

5.7 Vnos podatkov v tabele

Kot smo omenili, vnos podatkov v tabele ni sestavni del fizičnega modela, ampak je ta del potreben, če želimo preveriti zmogljivost fizičnega modela. Da preverimo zmogljivost, moramo poznati delovno obremenitev, ki jo podpira podatkovna baza. Delavno obremenitev pa sestavljajo večinoma poizvedbe in posodobitve tabel. Če želimo testirati hitrost poizvedovanja po tabelah oziroma posodobitve tabel, moramo imeti tabele napolnjene s podatki. Prav tako je treba tabele napolniti s podatki, če želimo preveriti ustreznost in zmogljivost podatkovne baze glede na potrebe in zahteve uporabnikov.

Podatke vnašamo v tabele s pomočjo SQL-ukaza, ki se prične z besedo **`INSERT`**. Imamo več možnih načinov vnosa podatkov. Pri prvem načinu ukazu `INSERT` sledi seznam vrednosti stolpcev. Vrednosti stolpcev morajo biti v enakem vrstnem redu, ki smo ga uporabili v času ustvarjanja tabel. Osnovna sintaksa prvega načina je:

```
INSERT INTO ime_tabele VALUES (vrednost1, vrednost2, ..., vrednostn)
```

Pri drugem načinu pa ukazu `INSERT` sledi navedba stolpcev in nato še njim pripadajoče vrednosti. Osnovna sintaksa drugega načina je:

```
INSERT INTO ime_tabele (stolpec1, stolpec2, ..., stolpecn)
VALUES (vrednost1, vrednost2, ..., vrednostn)
```

Drugi način vnosa je praviloma boljši. Osnovni razlog je ta, da se pri uporabi te oblike vnosa zmanjšajo morebitne napake. Pri prvem načinu vnosa podatkov se hitro zgodi, da se bodo napačni podatki zapisovali v napačni stolpec. Pri drugem načinu pa se tem napakam izognemo, saj najprej navedemo stolpce in šele nato njihove pripadajoče vrednosti.

Če pogledamo tabele baze, ki jo uporabljamo za zgled, opazimo, da večkrat pričakujemo, da so podatki v nekem stolpcu zaporedne številke. Taki so na primer vsi stolpci z oznako »Id« pri tabelah »KLJUČNA BESEDA«, »AVTOR«, »REZERVACIJA« in »IZVOD«.

Da ne bo potreben ročen vnos zaporednih številk, pri katerem se mimogrede tudi zmotimo in posamezno zaporedno številko preskočimo, lahko v SQL ustvarimo tudi **zaporedje** (**`SEQUENCE`**). Zaporedje je poseben objekt, ki hrani eno samo celo število. Z ukazom `ime_zaporedja.nextval` dobimo trenutno vrednost števca v zaporedju, hkrati pa se števec v tem zaporedju poveča za 1.

Novo zaporedje ustvarimo z:

```
CREATE SEQUENCE ime_zaporedja;
```

Če pa želimo, da se zaporedje začne pri denimo 10000, je sintaksa sledeča:

```
CREATE SEQUENCE ime_zaporedja START WITH 10000;
```

Kot smo omenili, bomo v primeru knjižnice zaporedja uporabili za vnos podatkov v stolpce z oznako »Id« pri tabelah »KLJUČNA BESEDA«, »AVTOR«, »REZERVACIJA« in »IZVOD«.

Primer uporabe sekvence za primarni ključ (»Id«) tabele »KLJUČNA BESEDA«:

```
CREATE SEQUENCE KBstevec;

INSERT INTO kljucna_beseda (id, naziv)
VALUES (KBstevec.nextval, 'podatkovne baze');
```

Sedaj bomo v tabele podatkovne baze knjižnice, ki smo jih ustvarili v prejšnjem razdelku, vnesli še nekaj testnih podatkov. Kot smo omenili, testne podatke vnesemo, da lahko preverimo ustreznost in zmogljivost podatkovne baze. Za vnos podatkov bomo uporabili drugi način vnosa. V ukazu `INSERT` bomo torej najprej navedli stolpce in nato še njihove pripadajoče vrednosti.
Vrstni red vnosa podatkov v tabele je pomemben. Kot smo omenili, se v določenih tabelah sklicujemo na primarni ključ nekatere druge tabele. Zato moramo paziti, da imamo pred vnosom podatkov v tabelo s tujim ključem predhodno že vnesene podatke v tabeli, na katero se tuji ključ nanaša.
V našem primeru bomo najprej napolnili tabele brez tujih ključev, to so tabele »KLJUČNA BESEDA«, »AVTOR«, »POŠTA« in »KNJIGA«.

Najprej vnesemo nekaj podatkov v tabelo »KLJUČNA BESEDA«, kot prikazuje Slika 101.

```
--------------------------------------------------------------------------
--Vnos podatkov v tabelo ključnih besed
--------------------------------------------------------------------------
--Števec ključnih besed se samodejno povečuje sam
CREATE SEQUENCE KBstevec;

INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'SQL');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'baze');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'človek');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'podatki');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'database');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'podeželje');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'družina');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'računalništvo');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'življenje');
INSERT INTO kljucna_beseda (id, naziv) VALUES (KBstevec.nextval, 'relacijske baze podatkov');
```

Slika 101: Vnos podatkov v tabelo "KLJUČNA BESEDA"

Sledi vnos podatkov v tabelo »AVTOR« (Slika 102).

```
--------------------------------------------------------------------------
--Vnos podatkov v tabelo avtorjev
--------------------------------------------------------------------------
--Števec avtorjev se samodejno povečuje sam
CREATE SEQUENCE Astevec;

INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Tomaž', 'Mohorič');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Boštjan', 'Brumen');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Arto', 'Paasilinna');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Mirko', 'Vintar');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Ramez', 'Elmasri');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Shamkant', 'Navathe');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Susan', 'Mallery');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Josip', 'Jurčič ');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Miro', 'Gradišar');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Gortan', 'Resinovič');
INSERT INTO avtor (id, ime, priimek) VALUES (Astevec.nextval, 'Peter', 'Mrhar');
```

Slika 102: Vnos podatkov v tabelo "AVTOR "

Nato sledi vnos podatkov v tabelo »POŠTA« (Slika 103).

```
--------------------------------------------------------------------------
--Vnos podatkov v tabelo pošta
--------------------------------------------------------------------------
INSERT INTO posta (postna_stevilka, kraj) VALUES (5000, 'Nova Gorica');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5292, 'Renče');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5210, 'Deskle');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5213, 'Kanal');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5290, 'Šempeter pri Gorici');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5220, 'Tolmin');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5230, 'Bovec');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5270, 'Ajdovščina');
INSERT INTO posta (postna_stevilka, kraj) VALUES (5250, 'Solkan');
INSERT INTO posta (postna_stevilka, kraj) VALUES (1000, 'Ljubljana');
```

Slika 103: Vnos podatkov v tabelo "POŠTA"

Vnos podatkov v tabelo »KNJIGA« prikazuje Slika 104.

```
-------------------------------------------------------------------------
--Vnos podatkov v tabelo knjig
-------------------------------------------------------------------------
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9616046055', 'Načrtovanje relacijskih podatkovnih baz', 'BI-TIM', '1997');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9789610204954', 'SQL', 'DZS', '2013');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9789610027300', 'Skoraj popolna', 'Učila International', '2015');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9616046128', 'Podatkovne baze', 'BI-TIM', '2002');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9789610118176', 'Srečni človek', 'Mladinska knjiga', '2012');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9616176722', 'Uvod v SQL', 'Flamingo', '2002');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('8681049810', 'Informatika', 'Moderna organizacija', '1994');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9619075501', 'Informatika', 'Bons', '1999');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9781292025605', 'Fundamentals of database systems', 'Pearson education limited', '2014');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9789612750152', 'Deseti brat', 'Karantanija', '2010');
INSERT INTO knjiga (ISBN, naslov, zaloznik, leto_izida)
  VALUES ('9789610130420', 'Na lovu za spomini', 'Mladinska knjiga', '2014');
```

Slika 104: Vnos podatkov v tabelo "KNJIGA"

Ker imamo predhodno vnesene podatke v tabeli »POŠTA« lahko sedaj vnesemo podatke tudi v tabelo »ČLAN«, kot prikazuje Slika 105.

```
-------------------------------------------------------------------------
--Vnos podatkov v tabelo članov
-------------------------------------------------------------------------
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('0020868', 'Tina', 'Kovač', 'Srebrničeva ulica 10', '12.09.2016', 5210);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1080869', 'Jure', 'Novak', 'Martinuči 1d', '15.10.2016', 5292);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1029870', 'Nejc', 'Kos', 'Erjavčeva ulica 2a', '25.10.2016', 5000);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1030871', 'Tina', 'Novak', 'Ulica talcev 12', '13.12.2016', 5210);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1020222', 'Marko', 'Zimic', 'Partizanska ulica 1', '04.01.2017', 5000);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1103301', 'Tjaša', 'Medved', 'Ulica padlih borcev 3', '25.12.2016', 5290);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1029998', 'Tim', 'Konjedic', 'Erjavčeva ulica 10', '12.10.2016', 5000);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1100208', 'Luka', 'Novak', 'Šolska ulica 28', '28.1.2017', 5250);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('1011019', 'Jožica', 'Medvešček', 'Bazoviška ulica 10', '02.02.2017', 5220);
INSERT INTO clan (clanska_stevilka, ime, priimek, ulica, potek_clanstva, postna_stevilka)
  VALUES ('2000168', 'Jože', 'Nanut', 'Grajska cesta 5', '18.09.2016', 5213);
```

Slika 105: Vnos podatkov v tabelo "ČLAN"

Tabela »REZERVACIJA« se sklicuje na tabeli »ČLAN« in »KNJIGA«. Obe tabeli smo predhodno napolnili s podatki, zato lahko sedaj podatke vnesemo tudi v tabelo »REZERVACIJA« (Slika 106).

```
--------------------------------------------------------------------------
--Vnos podatkov v tabelo rezervacij
--------------------------------------------------------------------------
--Števec rezervacij se samodejno povečuje sam
CREATE SEQUENCE Rstevec;

INSERT INTO rezervacija (id, datum_rezervacije, ISBN, clanska_stevilka)
  VALUES (Rstevec.nextval, '28.5.2016', '9789610118176', '1103301');
INSERT INTO rezervacija (id, datum_rezervacije, ISBN, clanska_stevilka)
  VALUES (Rstevec.nextval, '20.5.2016', '9789610130420', '1103301');
INSERT INTO rezervacija (id, datum_rezervacije, ISBN, clanska_stevilka)
  VALUES (Rstevec.nextval, '25.5.2016', '9619075501', '2000168');
INSERT INTO rezervacija (id, datum_rezervacije, ISBN, clanska_stevilka)
  VALUES (Rstevec.nextval, '30.5.2016', '9616046055', '1103301');
INSERT INTO rezervacija (id, datum_rezervacije, ISBN, clanska_stevilka)
  VALUES (Rstevec.nextval, '20.5.2016', '9789610118176', '1029998');
INSERT INTO rezervacija (id, datum_rezervacije, ISBN, clanska_stevilka)
  VALUES (Rstevec.nextval, '18.5.2016', '9789610118176', '1100208');
```

Slika 106: Vnos podatkov v tabelo "REZERVACIJA"

Enako velja tudi za tabelo »IZVOD« (Slika 107).

```
--------------------------------------------------------------------------
--Vnos podatkov v tabelo izvodov
--------------------------------------------------------------------------
--Števec izvodov se samodejno povečuje sam
CREATE SEQUENCE Istevec;

INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '821', 'izposojen', '4.6.2016', '0020868', '9789610118176');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'izposojen', '2.6.2016', '1029998', '9616046055');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'prost', Null, Null, '9616046128');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'prost', Null, Null, '9616046128');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'prost', Null, Null, '9789610204954');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '821', 'izposojen', '8.6.2016', '1011019', '9789610027300');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'prost', Null, Null, '9616176722');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'prost', Null, Null, '8681049810');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '659', 'izposojen', '13.6.2013', '1020222', '9619075501');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '004', 'izposojen', '12.6.2013', '1080869', '9781292025605');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '320', 'prost', Null, Null, '9789612750152');
INSERT INTO izvod (id, signatura, status_izvoda, datum_vrnitve, clanska_stevilka, ISBN)
  VALUES (Istevec.nextval, '821', 'izposojen', '15.6.2016', '1029870', '9789610130420');
```

Slika 107: Vnos podatkov v tabelo "IZVOD"

Sledi vnos podatkov v tabelo »IMA«, ki povezuje knjige z njihovimi avtorji (Slika 108). Tako tabelo »KNJIGA« kot tabelo »AVTOR« smo že napolnili s podatki.

```
-----------------------------------------------------------------------
--Vnos podatkov v tabelo ima
-----------------------------------------------------------------------
INSERT INTO ima (ISBN, id) VALUES ('9616046055', 1);
INSERT INTO ima (ISBN, id) VALUES ('9789610204954', 2);
INSERT INTO ima (ISBN, id) VALUES ('9789610027300', 7);
INSERT INTO ima (ISBN, id) VALUES ('9616046128', 1);
INSERT INTO ima (ISBN, id) VALUES ('9789610118176', 3);
INSERT INTO ima (ISBN, id) VALUES ('9616176722', 11);
INSERT INTO ima (ISBN, id) VALUES ('8681049810', 9);
INSERT INTO ima (ISBN, id) VALUES ('8681049810', 10);
INSERT INTO ima (ISBN, id) VALUES ('9619075501', 4);
INSERT INTO ima (ISBN, id) VALUES ('9781292025605', 5);
INSERT INTO ima (ISBN, id) VALUES ('9781292025605', 6);
INSERT INTO ima (ISBN, id) VALUES ('9789612750152', 8);
INSERT INTO ima (ISBN, id) VALUES ('9789610130420', 3);
```

Slika 108: Vnos podatkov v tabelo "IMA"

Ostane nam še vnos podatkov v tabelo »OPISUJE«, ki povezuje knjige s ključnimi besedami (Slika 109).

```
-----------------------------------------------------------------------
--Vnos podatkov v tabelo opisuje
-----------------------------------------------------------------------
INSERT INTO opisuje (ISBN, id) VALUES ('9616046055', 1);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046055', 2);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046055', 4);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046055', 10);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610204954', 1);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610204954', 2);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610204954', 10);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610027300', 7);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046128', 1);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046128', 2);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046128', 4);
INSERT INTO opisuje (ISBN, id) VALUES ('9616046128', 10);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610118176', 3);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610118176', 6);
INSERT INTO opisuje (ISBN, id) VALUES ('9616176722', 1);
INSERT INTO opisuje (ISBN, id) VALUES ('9616176722', 2);
INSERT INTO opisuje (ISBN, id) VALUES ('9616176722', 4);
INSERT INTO opisuje (ISBN, id) VALUES ('9616176722', 10);
INSERT INTO opisuje (ISBN, id) VALUES ('8681049810', 4);
INSERT INTO opisuje (ISBN, id) VALUES ('8681049810', 7);
INSERT INTO opisuje (ISBN, id) VALUES ('8681049810', 8);
INSERT INTO opisuje (ISBN, id) VALUES ('9619075501', 4);
INSERT INTO opisuje (ISBN, id) VALUES ('9619075501', 7);
INSERT INTO opisuje (ISBN, id) VALUES ('9781292025605', 1);
INSERT INTO opisuje (ISBN, id) VALUES ('9781292025605', 5);
INSERT INTO opisuje (ISBN, id) VALUES ('9789612750152', 3);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610130420', 6);
INSERT INTO opisuje (ISBN, id) VALUES ('9789610130420', 9);
```

Slika 109: Vnos podatkov v tabelo "OPISUJE"

Ko imamo tabele napolnjene s podatki, izvedemo še testiranje podatkovne baze. Testiranje je pomembno opravilo, ki zajema vrsto postopkov. Podrobnejši opis pa bi presegal okvire te diplomske naloge, zato opozorimo le, da na testiranje, ki ga izvedemo potem, ko imamo bazo ustvarjeno, nikakor ne smemo pozabiti. Ne smemo pa pozabiti tudi, da po testiranju podatkovne baze lahko določimo, kateri uporabnik oziroma skupina uporabnikov lahko dostopa do katerih tabel in katere operacije lahko

izvaja nad posamezno tabelo. Torej definiramo tako imenovane uporabniške vloge in za posamezno vlogo določimo pravice uporabnikom, ki imajo to vlogo. Tako bomo zagotovili nadzor nad podatkovno bazo in preprečili nepooblaščene dostope do podatkov v podatkovni bazi.

Ko je podatkovna baza enkrat zgrajena, jo je treba še nadzorovati, vzdrževati in po potrebi spreminjati ter dopolnjevati. Tudi nadzorovanje in vzdrževanje podatkovne baze sta pomembni opravili. Ravno tako kot testiranje pa bi tudi podrobnejši opis nadzorovanja in vzdrževanja podatkovne baze presegel okvire te diplomske naloge. Naj omenimo samo, da ne smemo pozabiti, da se okolje ves čas spreminja, zato moramo tudi podatkovno bazo ves čas prilagajati novim zahtevam in vzdrževati tako po programski kot tudi po strojni plati.

6 ZAKLJUČEK

V diplomski nalogi smo opisali in prikazali postopek razvoja relacijske podatkovne baze. Spoznali smo, da je pri izdelavi podatkovne baze pomembno predvsem sodelovanje načrtovalcev z uporabniki podatkovne baze skozi njen celoten razvoj. Ključnega pomena je tudi, da načrtovalci in uporabniki govorijo isti jezik. V nasprotnem primeru zelo hitro pride do razlik med željami uporabnikov in razumevanjem načrtovalcev. Dobro pa moramo poznati tudi orodja (njihove funkcije), ki jih bomo pri izdelavi podatkovne baze uporabljali.

Temo načrtovanja in modeliranja relacijske podatkovne baze sem izbrala zato, ker sem želela bolje spoznati postopek razvoja podatkovne baze in utrditi znanje na tem področju. Ugotovila sem, da je načrtovanje in modeliranje podatkovne baze zahteven proces, ki od načrtovalca zahteva veliko odločitev v posameznih fazah načrtovanja. Bistveno je predvsem, da v začetni (konceptualni) fazi načrtovanja izdelamo dober model, saj nam bo le tak lahko dobro služil v nadaljnjih fazah načrtovanja podatkovne baze.

Da bi diplomska naloga zajela celoten proces izdelave podatkovne baze, bi jo bilo treba dopolniti še z delom o normalizaciji podatkov, ki predstavlja pomemben del pri načrtovanju podatkovne baze.

7 VIRI IN LITERATURA

Ambler, Scott W. AgileData - Data Modeling. [Online] http://www.agiledata.org/essays/dataModeling101.html (dostop 16. 6. 2016).

Brumen, Boštjan. 2013. *SQL osnove strukturiranega poizvedovalnega jezika.* Ljubljana: DZS, 2013.

Chen, Peter P. 1976. The Entity-Relationship Model - Toward a Unified View of Data. 1976.

Connolly, Thomas M. and Begg, Carolyn E. 2005. *Database Systems (A Practical Approach to Design, Implementation and Management).* London: Addison-Wesley, 2005.

Dolžan, Janez. 2001. Informacijski sistemi. [Online] 2001. http://users.volja.net/bojanperic/informacijski%20sistemi-zapiski.doc (dostop 16. 6. 2016).

Drenovec, Jože. 2006. Baze podatkov - Modeliranje. *Tehniški šolski center Kranj.* [Online] 2006. http://drenovec.tsckr.si/ (dostop 16. 6. 2016).

Dybka, Patrycja. 2014. ERD Notations in Data Modeling. 2014.

Elmasri, Ramez and Navathe, Shamkant B. 2003. *Fundamentals of database systems - Fourth Edition.* s.l.: Addison Wesley, 2003.

Gradišar, Miro and Resinovič, Gortan. 1994. *Informatika.* Kranj: Moderna organizacija v sestavi Fakultete za organizacijske vede Kranj, 1994.

Mohorič, Tomaž. 1997. *Načrtovanje podatkovnih baz.* Ljubljana: BI-TIM, 1997.

—. 2002. *Podatkovne baze.* Ljubljana: BI-TIM, 2002.

Mrhar, Peter. 2002. *Uvod v SQL.* Nova Gorica: Založba Flamingo, 2002.

Oracle. Oracle Database. [Online] http://www.oracle.com/index.html (dostop 16. 6. 2016).

Projekt Colos. E-gradiva za predmet Računalništvo. *colos.* [Online] http://colos.fri.uni-lj.si/eri/RACUNALNISTVO/ (dostop 16. 6. 2016).

Ramakrishnan, Raghu and Gehrke, Johannes. 2003. *Database management systems - third edition.* New York: McGraw-Hill, 2003.

Remžgar, Urša. 2012. Načrtovanje relacijskih podatkovnih baz z entitetno relacijskimi diagrami. [Online] 2012. https://ucilnica.fmf.uni-lj.si/pluginfile.php/6394/mod_resource/content/0/diplomaUr%C5%A1aRem%C5%BEgar.pdf (dostop 16. 6. 2016).

Spletno_mesto_EMRIS. *Spletno mesto EMRIS - strukturni razvoj IS.* [Online] http://www2.gov.si/mju/emris.nsf/Zvezek3?OpenFrameSet (dostop 16. 6. 2016).

Teorey, Toby, Lightstone, Sam and Nadeau, Tom. 2006. *Database Modeling & Design: Logical Design - Fourth Edition.* San Francisco: Morgan Kaufmann Publishers, 2006.

Wikipedia. ANSI-SPARC_Architecture. *wikipedia.* [Online] https://en.wikipedia.org/wiki/ANSI-SPARC_Architecture (dostop 16. 6. 2016).

—. Entity–relationship model. [Online] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model (dostop 16. 6. 2016).

—. SQL. [Online] https://en.wikipedia.org/wiki/SQL (dostop 16. 6. 2016).

Wikipedija. Sistem za upravljanje s podatkovnimi zbirkami. [Online] https://sl.wikipedia.org/wiki/Sistem_za_upravljanje_s_podatkovnimi_zbirkami (dostop 16. 6. 2016).