Alenka Rožanec

BAZE PODATKOV

VISOKOŠOLSKO SREDIŠČE NOVO MESTO
FAKULTETA ZA UPRAVLJANJE, POSLOVANJE IN INFORMATIKO
NOVO MESTO

Alenka Rožanec

BAZE PODATKOV

Novo mesto, 2017

Dr. Alenka Rožanec
BAZE PODATKOV

Izdala in založila © Fakulteta za upravljanje, poslovanje in informatiko Novo mesto

Uredila dr. Jasmina Starc

Recenziral dr. Ivan Gerlič

Za jezikovno neoporečnost je odgovorna avtorica učbenika.

Grafična priprava Bojan Nose, Visokošolsko središče Novo mesto

Kataložni zapis o publikaciji (CIP)
pripravili v Narodni in univerzitetni knjižnici v Ljubljani

COBISS.SI-ID=290575360

ISBN 978-961-6309-42-4 (pdf)

KAZALO

KAZALO SLIK

KAZALO TABEL

1 Uvod v podatkovne baze

Poslovni sistemi so dandanes bolj kot kdajkoli prej odvisni od zmožnosti pridobivanja natančnih in pravočasnih podatkov ter zmožnosti njihovega učinkovitega preoblikovanja v informacije, tako za operativno izvajanje poslovnih procesov, kot tudi upravljanje in odločanje. Kvalitetne in pravočasne informacije za poslovni sistem lahko predstavljajo konkurenčno prednost. Brez zmožnosti za upravljanje z velikimi količinami podatkov in zmožnostmi za hitro iskanje ustreznih podatkov ter njihovo preoblikovanje v informacije za različne vrste uporabnikov, postanejo podatki breme za organizacijo. Odgovor nam dajejo podatkovne baze oziroma sistemi za upravljanje s podatkovnimi bazami kot ena od temeljnih vrst informacijske tehnologije in integralni del praktično vsake aplikacije ali informacijskega sistema.

1.1 Podatek in informacija

Izraza podatek in informacija se pogosto uporabljata kot sinonima, vendar pa je potrebno med njima razlikovati. Poglejmo si najprej nekaj definicij podatkov (ANSI, ISO, Everest, povzeto po Mohorič, 1992, str. 2-3):

- **Definicija 1:** *Podatek je poljubna predstavitev s pomočjo simbolov ali analognih veličin, ki ji je pripisan, ali seji lahko pripiše pomen.*
- **Definicija 2:** *Podatek je predstavitev dejstva, koncepta ali instrukcije na formaliziran način, ki je primeren za komunikacijo, interpretacijo ali obdelavo s strani človeka ali stroja.*
- **Definicija 3:** *Podatki so dejstva predstavljena z vrednostmi (številke, znaki, simboli), ki imajo pomen v določenem kontekstu.*

Podatek je lahko **diskreten**, če se pri predstavitvi uporabljajo simboli (npr. stopinje cezija), ali pa **analogen**, če se za predstavitev uporablja fizikalna veličina (npr. dolžina živosrebrnega stolpca).

Druga definicija pravi, da mora biti predstavitev izvedena na **formaliziran način**, kar pomeni, da mora obstajati nek predpis, po katerem simbole ali vrednosti zapisujemo ali beremo.

Vsem trem definicijam pa je skupno, da se podatku lahko pripiše nek pomen na osnovi predpisa oziroma znotraj nekega konteksta. Podatek je tako le nosilec informacije oziroma njegova fizična predstavitev.

Po ANSI in ISO velja:

- **Definicija 4:** *Informacija je pomen, ki ga človek pripiše podatkom s pomočjo znanih konvencij, ki so uporabljene pri njeni predstavitvi.*

Everest je zapisal naslednjo definicijo:

- **Definicija 5:** *Informacija so ovrednoteni podatki v specifični situaciji.*

Langefors je podal naslednjo definicijo informacije (Langefors 1980 v Mohorič, 1992, str. 3):

- **Definicija 6:** *Informacija je novo spoznanje, ki ga človek doda svojemu poznavanju sveta.* Odnos med podatki in informacijo podaja naslednja formula: I = i(D, S, t), kjer pomeni:

  I – informacija, ki jo posredujejo podatki
  i - Informacijska funkcija
  D – podatki
  S - sprejemna struktura – prejemnikovo znanje
  t - čas, ki je na voljo za interpretacijo podatkov.

Iz navedene definicije izhaja:

- **Podatki niso informacije.**
- Podatki ne vsebujejo informacije.
- **Podatki posredujejo informacijo prejemniku, katerega sprejemna struktura je konsistentna z izbrano predstavitvijo podatkov in modelom sveta, na katerega se nanašajo.**
- Če je količina podatkov tako velika, da se jih v času, ki je na voljo za ukrepanje, ne da interpretirati, se lahko zgodi, da s podatki ni posredovana nobena informacija.

Gradišar in drugi (2012, str. 35) pa informacijo definirajo kot:

- **Definicija 7:** *Informacija je tako zaporedje znakov v nekem jeziku, ki je sintaktično pravilno, razumljivo in ima za prejemnika uporabno vrednost.*

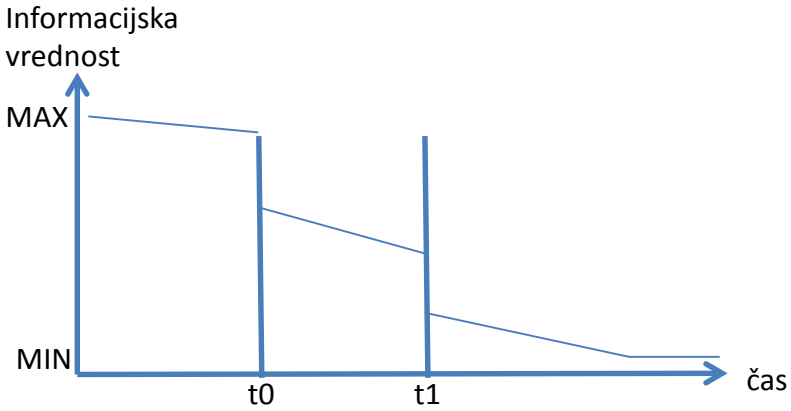
Informacijo navadno opredelimo s tremi dimenzijami (Gradišar in drugi, 2012, str. 36-38):

- vrednostjo,
- količino,
- kakovostjo.

Informacijsko vrednost lahko določimo kot vrednost spremembe v obnašanju prejemnika, zmanjšano za stroške pridobitve informacije. Prispevek določene informacije k boljšemu odločanju pa je včasih težko ali nemogoče ugotoviti. Vrednost informacije se s časom manjša. Če dobimo informacijo dovolj zgodaj, da se lahko na njeni podlagi dobro odločimo, je njena vrednost visoka. Če informacija pride prepozno, pa je njena vrednost lahko zelo nizka celo enaka 0. **Uporabna vrednost informacije z vidika odločanja in upravljanja torej ni stalna, temveč se s časom manjša. Vrednost informacije je odvisna tudi od njene kakovosti (**Gradišar in drugi, 2012, str. 38**).**

Informacija **je merljiva količina** in jo na podlagi dogovora v teoriji informacij merimo oz. izmerimo z **osnovno enoto BIT (***BInary digiT***)**. Informacija **odstrani določeno stopnjo neznanja.** Količina informacije, ki jo dobimo, je tem večja, čim več novega nam pove. Sporočilo z večjo količino informacije nas bolj preseneti. Količina informacije je večja, če izvemo, da se je zgodil malo verjeten dogodek in manjša,če smo dogodek pričakovali (Gradišar, 2012, str. 37).

Slika 1: Informacijska vrednost s časom pada



Vir: Gradišar in Resinovič, 1998, str. 51.

Formula za izračun **količine informacije** (Shannon):

$$I = -\log_2 p(x)\ [bit] \qquad p(x)–\ \text{verjetnost nastopa dogodka } x$$

Če poiščemo vzporednice med Shannonovo in Langeforsovo definicijo informacije, potem so podatki v slednjem primeru sporočilo, da se je pripetil dogodek x, sprejemna struktura pa je poznavanje verjetnosti nastopa posameznih dogodkov $p(x_i)$. Informacija, ki jo izračunamo po Shannonovi formuli, se ujema s trditvijo iz Langeforsove definicije, da je informacija le novo spoznanje. Če se namreč pripeti zelo verjeten dogodek (p(x)=1), potem je to enako sprejetim podatkom, ki nam sporočajo nekaj, kar smo že vedeli (I=0), in nimamo torej ničesar dodati svojemu prejšnjemu znanju (povzeto po Mohorič, 2002, str. 2-3).

Če ima sistem n enako verjetnih stanj pa lahko uporabimo naslednjo formulo (Gradišar in Resinovič, 1998, str. 40):

$$I = \log_2 n$$

Primer 1.1.1: Kovanec se z enako verjetnostjo nahaja v dveh različnih položajih (cifra, grb). Količina informacije, ki je potrebna,da določimo stanje kovanca je:

$I = -\log_2 ½ = 1$ bit

Primer 1.1.2: Imejmo predalčnik s kroglico z osmimi predali. Kroglica je lahko v kateremkoli predalu z enako verjetnostjo. Koliko informacije dobimo, ko izvemo, da je kroglica v tretjem predalu?

$I = -\log_2 8 = 3$ bite

Za uspešno upravljanje in odločanje na podlagi informacij je zelo pomembna njihova kakovost. Kakovost informacije se kaže v tem, kako spodbuja prejemnika k dejanjem oziroma boljšim odločitvam. Sodila merjenja kakovosti informacije so (Gradišar in Resinovič, str. 48-51):

- **Dostopnost:** do informacije je potreben dovolj hiter dostop, saj se s časom njena vrednost zmanjšuje. Dostopnost merimo od časa, ko uporabnik informacijo zahteva, do trenutka, ko jo dobi. Čas je odvisen od metod in sredstev za iskanje informacije.

- **Točnost:** na osnovi netočnih informacij uporabnik sprejema napačne sklepe in odločitve. V informacijski proces je potrebno vgraditi kontrole, ki omogočajo odkrivanje in popravljanje napak in tako zmanjšujejo količino netočnih informacij.

- **Pravočasnost:** za sprejemanje odločitev je pomembna pravočasna informacija, vendar se pravočasnost in točnost pogosto izključujeta. Kontrolni mehanizmi pogosto podaljšujejo čas ustvarjanja informacije. Sistemi morajo biti čim bolj odzivni, da je sprememba stanja in obnašanja v organizaciji in okolju čim prej na voljo uporabniku v obliki informacije.

- **Popolnost:** popolna informacija je tista, ki daje uporabniku vse potrebno za sprejemanje ustreznih odločitev in akcij. Absolutno popolne informacije ni!

- **Zgoščenost:** prevelika popolnost lahko za uporabnika po drugi strani pomeni preveliko zasičenost, zato ne bo mogel pregledati in uporabiti vseh informacij. Informacija mora biti za določen namen ravno prav zgoščena – **kratka in jedrnata.**

- **Ustreznost:** ugotavljamo do kakšne mere je informacija prilagojena informacijskim potrebam uporabnika. Informacijske potrebe različnih uporabnikov so različne, tudi pri istem uporabniku pa se s časom spreminjajo.

- **Razumljivost:** posredovanje v ustrezni obliki in jeziku, da jo uporabnik lahko razume in uporabi.

- **Objektivnost:** predstavitev pojava mora biti stvarna in nepristranska.

1.2 Podatkovna baza

Podatkovna baza je organizirana zbirka podatkov in je danes integralni del vsake poslovne aplikacije ali informacijskega sistema. Lahko jo razumemo kot veliko shrambo najrazličnejših podatkov, ki jo hkrati uporabljajo številni oddelki in uporabniki. Namesto neurejene množice datotek so v primeru uporabe podatkovne baze vsi podatki shranjeni na enem mestu, njihovo podvajanje pa je zmanjšano na minimum. Podatkovna baza navadno ni last enega oddelka, ampak gre za pomemben organizacijski vir. Podatkovna baza polega samih podatkov vsebuje tudi njihove opise. Opise imenujemo sistemski katalog (ang. system catalog), podatkovni slovar (ang. data dictionary) ali metapodatki (podatki o podatkih) (Connolly in Begg, 2010, *str. 15*).

Podajmo nekaj definicij:

- **Definicija 8:** *Podatkovna baza je model okolja, ki služi kot osnova za sprejemanje odločitev in izvajanje akcij (Mohorič, 1992, str. 10).*
- **Definicija 9:** *Podatkovna baza je mehanizirana, večuporabniška, formalno definirana in centralno nadzorovana zbirka podatkov (Mohorič, 1992, str. 12).*
- **Definicija 10:** *Podatkovna baza je deljena zbirka logično povezanih podatkov in njihovih opisov, načrtovanih za zadovoljitev informacijskih potreb poslovnega sistema* (Connolly in Begg, 2010, str. 15).
- **Definicija 11:** *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 s podatkovno bazo – SUPB (Mohorič, 1992, str. 12).*

Podatkovna baza je načrtovana in zgrajena z nekim namenom in skladno s tem odraža določen vidik realnega sveta oziroma hrani le tiste podatke, ki so za določeno domeno pomembni. V podatkovnih bazah hranimo podatke o **entitetah**, ki so lahko osebe, dogodki, predmeti, pa tudi njihove medsebojne **povezave**. V primeru šole je npr. entiteta vsak študent, ki to šolo obiskuje, in prav tako vsak predmet, ki se na šoli poučuje. O entitetah nas zanimajo določene lastnosti, ki jih imenujemo **atributi**. V primeru študenta so te lastnosti navadno: vpisna številka, ime, priimek, datum rojstva, naslov, itd., v primeru premeta pa: številka predmeta, letnik, semester, število kreditnih točk, obveznosti. Poleg tega nas zanimajo tudi povezave med entitetami. Tako ni dovolj, da imamo le seznam vseh študentov in vseh predmetov, ampak moramo vedeti tudi, katere predmete posluša vsak posamezni študent, ali kateri vsi študenti so vpisani na določen predmet. Za podatkovno bazo je tako značilno, da so podatki o entitetah in povezavah v njej strukturirani.

Podatkovna baza je tako zelo pomembno sredstvo za delovanje poslovnega sistema in njegovega informacijskega sistema, zato jo je potrebno učinkovito upravljati. Upravljanje podatkovne baze zajema (Mohorič, 1992, str. 11):

- zagotavljanje **razpoložljivosti** podatkov in
- **nadzor nad uporabo** podatkov. Sem sodi skrb za:
  - **celovitost** (integriteto) podatkov,
  - uporabo podatkov v skladu z njihovim namenom in ustrezna **zaupnost**,
  - uporabnost podatkov tudi v prihodnje.

Razpoložljivost (ang. availability) pomeni učinkovit, sočasen dostop vseh uporabnikov do različnih podatkov, kadarkoli jih pri svojem delu potrebujejo. Med uporabnike podatkovne baze poleg končnih uporabnikov štejemo tudi razvijalce aplikacij, skrbnika podatkovne baze (ang. Database administrator) ter uporabniške programe.

Celovitost ali integriteta (ang. integrity) podatkov pomeni, da so podatki konsistentni navznoter in z zunanjim svetom. Širše lahko pri integriteti govorimo tudi o kvaliteti podatkov (ang. data quality) vsebovanih v podatkovni bazi. Kvalitetni podatki so pravočasni, popolni in izvirajo iz zanesljivih virov. Mehanizmi za zagotavljanje celovitosti podatkov so: preverjanje vhodnih podatkov, obnavljanje PB in nadzor nad sočasnim dostopom.

Za uporabo podatkov v skladu z njihovim namenom morajo vsi uporabniki pravilno razumeti podatke, ki so v PB zapisani, kar zagotovimo z opisi njihovega pomena. **Mehanizmi za upravljanje dostopa** (ang. access control) uporabnikom omogočajo le dostop do podatkov, ki jih glede na svojo vlogo v poslovnem sistemu potrebujejo za opravljanje svojega dela. S tem je zagotovljena ustrezna **zaupnost podatkov**, glede na vrsto podatkov. Pri tem mehanizmi omogočajo tudi določitev vrste dostopa kot so branje, dodajanje, spreminjanje, brisanje, spreminjanje strukture PB. **Upravljanje z dostopnimi pravicami je ena od ključnih nalog skrbnika PB.**

Da bodo podatki **uporabni tudi v prihodnje**, moramo skrbeti za prilagajanje strukture podatkovne baze spreminjajočim se poslovnih zahtevam ter posodabljati informacijsko infrastrukturo za hranjenje podatkov (strojno opremo, sistem za upravljanje s podatkovno bazo) (povzeto po Mohorič, 1992, str. 12-14).

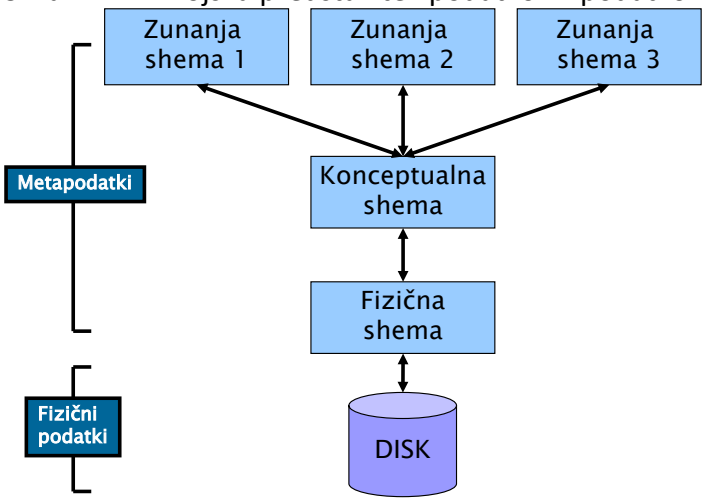
1.3 Tri-nivojska predstavitev podatkov v podatkovni bazi

Zelo pomembna lastnost SUPB je, da omogoča podatkovno neodvisnost - programi so neodvisni od fizičnega načina shranjevanja in strukturiranja podatkov v PB. Da bi dosegli podatkovno neodvisnost podatke v PB opišemo na treh ravneh (Mohorič, 1992, str. 17-20, Connolly in Begg, 2010, str. 36-41, Ramakrishnan in Gehrke, 2003, str. 12-16):

- z zunanjo shemo,
- s konceptualno ali logično shemo in
- s fizično (notranjo) shemo.

Metapodatkovna baza torej vsebuje tri vrste opisov fizičnih podatkov kot prikazuje slika (Slika 2).

Slika 2: Tri-nivojska predstavitev podatkov v podatkovni bazi



Konceptualna ali logična shema opisuje podatke z vidika podatkovnega modela, ki ga PB uporablja. Npr. podatki o entitetnih tipih (profesor, študent, predmet, predavalnica,...), podatki o povezavah (predava, posluša,...). Proces izdelave konceptualne sheme se imenuje konceptualno ter logično načrtovanje. Konceptualno načrtovanje je podrobno predstavljeno v poglavju 4, logično načrtovanje relacijske PB pa v poglavju 5.2.

Fizična shema podaja podrobnosti o shranjevanju podatkov. Predstavi, kako so podatki iz konceptualne sheme dejansko shranjeni na sekundarnem pomnilniku, npr. trdem disku ali magnetnem traku. Proces izdelave fizične sheme se imenuje načrtovanje fizične PB. V tem koraku se je potrebno določiti, kakšno datotečno organizacijo bomo uporabili za shranjevanje podatkov in kreirati indeksne datoteke. Načrtovanje fizične relacijske podatkovne baze je predstavljeno v poglavju 5.4.

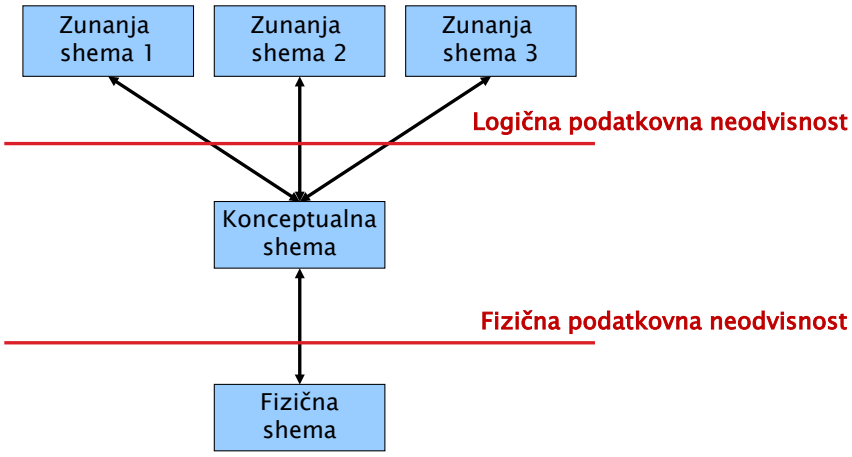
Tudi **zunanje sheme** uporabljajo koncepte podatkovnega modela (gradnike konceptualne sheme). Zunanja shema se uporablja za dostop do podatkov, ki je prilagojen določenemu uporabniku ali skupini uporabnikov. Vsaka zunanja shema se sestoji iz enega ali več pogledov (ang. view) na konceptualno shemo. Pogled je logična tabela, ki ne obstaja v fizični podatkovni bazi.

Opisano tri-nivojsko predstavitev podatkov uporabljamo za doseganje podatkovne neodvisnosti. Poznamo dve vrsti podatkovne neodvisnosti (Mohorič, 1992, str. 20):

- Fizično podatkovno neodvisnost in
- Logično podatkovno neodvisnost.

Konceptualna shema zagotavlja **fizično podatkovno neodvisnost** (Slika 2), saj skrije podrobnosti o tem, kako so podatki dejansko shranjeni na disku, o strukturi datotek in o indeksih. Dokler ostaja konceptualna shema nespremenjena, spremembe na fizičnem nivoju ne vplivajo na programe, ki podatke uporabljajo. Lahko pa spremembe vplivajo na učinkovitost.

Slika 3: Podatkovna neodvisnost



Zunanje sheme pa zagotavljajo **logično podatkovno neodvisnost.** Dodatne logične povezave med podatki, razširitve/izločitev entitetnih tipov in atributov v konceptualni shemi ne vplivajo na uporabnike, ki jih te spremembe ne zadevajo.

1.4 Mesto podatkovne baze v poslovnem sistemu

Podatkovne baze se v poslovnih sistemih uporabljajo predvsem z dvema namenoma: za hranjenje transakcijskih podatkov, ki se uporabljajo pri izvajanju različnih poslovnih procesov in za upravljanje poslovnega sistema.

Hranjenje in uporaba transakcijskih podatkov, ki se zajemajo in obdelujejo pri operativnem izvajanju poslovnih procesov, npr. prodaja ali nabava izdelkov/storitev. V tem primeru gre za podroben zapis vseh podatkov, ki so potrebni za izvedbo celotnega poslovnega procesa (npr. v primeru prodaje podatkov o kupcu, kraju in načinu dostave, ceni, količini itd.). Tovrstne podatkovne baze so lahko del funkcionalnih aplikacij ali celovitih informacijskih rešitev, prav tako so lahko tudi del različnih spletnih aplikacij.

Uporaba podatkov za upravljanje poslovnega sistema: pokrivanje informacijskih potreb vodstvenih delavcev na taktični in strateški ravni organizacije. Pogosto se tukaj kažejo potrebe po agregiranih podatkih, npr. prodaja po izdelkih, mesecih, regijah. Uporabo podatkov za upravljanje in odločanje delimo na standardna poročila in ad hoc poizvedbe, za katere se uporabljajo specializirana orodja. Pri pripravi poročil ali ad hoc poizvedovanju lahko dostopamo do transakcijskih podatkovnih baz, lahko pa podjetje uvede specializirano podatkovno bazo, imenovano podatkovno skladišče, kamor s posebnimi transformacijski postopki prepiše transakcijske podatke. Del podatkovnega skladišča pogosto predstavljajo tudi podatki iz okolja poslovnega sistema, kar pomeni, da v njem integriramo zunanje in notranje podatke, kar je za upravljanje in odločanje zelo pomembno.

Vprašanja za ponavljanje

1. Kaj je podatek?
2. Kaj je informacija?
3. V kakšnem medsebojnem odnosu sta po Lengeforsovi definiciji podatek in informacija?
4. S katerimi dimenzijami opredelimo informacijo?
5. Podajte Shannonovo formulo za izračun količine informacije.
6. Zakaj so za poslovni sistem pomembne kakovostne informacije?
7. Katera sodila kakovosti informacije poznate?
8. Kaj je podatkovna baza?
9. Kaj vsebuje podatkovna baza?
10. Kaj je katalog oz. podatkovni slovar?
11. Kaj zajema upravljanje podatkovne baze?
12. Kaj je razpoložljivost podatkov?
13. Kaj je celovitost podatkov?
14. Zakaj je pomembno upravljanje dostopa do podatkov in kaj omogoča?
15. Kdo so ključni uporabniki podatkovne baze?
16. Katere vrste shem obsega tri-nivojska predstavitev podatkov? Zakaj je pomembna?
17. Kaj je podatkovna neodvisnost?
18. Kateri vrsti podatkovne neodvisnosti poznate? Kaj je njun namen?
19. Zakaj je podatkovna baza pomemben vir poslovnega sistema?
20. Za katere namene in v kakšnih vrstah aplikacij oz. informacijskih rešitev se uporablja podatkovna baza?

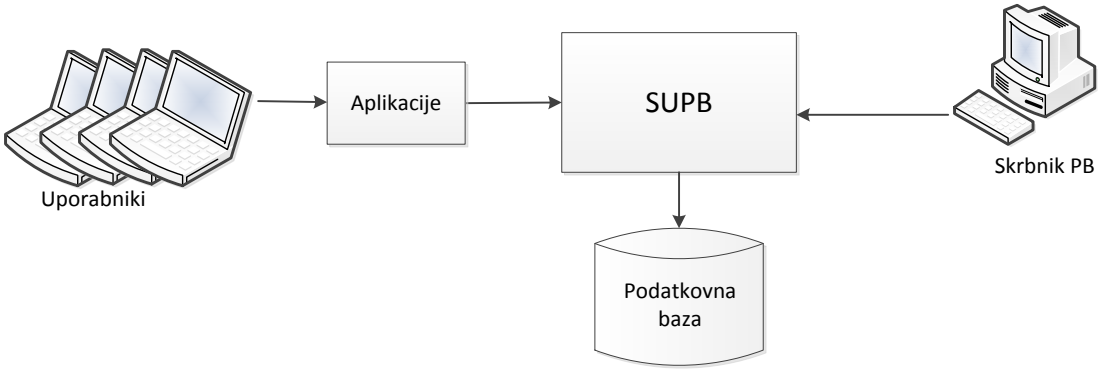
2 Sistem za upravljanje s podatkovno bazo

Sistem za upravljanje s podatkovno bazo – SUPB (ang. Database Management System) je programska oprema za obvladovanje velikih količin podatkov. Danes na tem področju prevladujejo predvsem relacijski SUPB najbolj znanih podjetij s področja IT, to so: Oracle, IBM (DB2) in Microsoft (MS SQL). Vsako od njih ponuja svoj SUPB, uveljavili pa so se tudi nekateri odprtokodni SUPB, najbolj popularen med njimi je MySQL.

- **Definicija 12:** Sistem za upravljanje s podatkovno bazo *(SUPB) je skupek programske opreme, ki omogoča definiranje, kreiranje, vzdrževanje in nadzor nad dostopom do podatkov v podatkovni bazi (Connolly in Begg, 2010, str. 16).*

Gre za programsko opremo, ki predstavlja vmesni člen med uporabniško aplikacijo in podatkovno bazo. Navadno le skrbnik podatkovne baze neposredno dostopa do SUPB (Slika 4).

Slika 4: SUPB



2.1 Prednosti uporabe SUPB

Shranjevanje podatkov v datoteke direktno na disk ima številne slabosti, ki jih rešuje uporaba SUPB kot vmesnega člena. Ko so aplikacije še dostopale direktno do podatkov na disku, je bilo potrebnega veliko dodatnega programiranja (navadno v jezikih 3. generacije kot je bil na primer Cobol), saj je vsaka aplikacija morala vsebovati tudi funkcije, ki jih danes opravlja SUPB. Vzdrževanje takšnih aplikacij je bilo zahtevno, vsebovale so veliko vrstic kode, spreminjanje podatkov struktur je bilo zahtevno in zamudno, pogosto aplikacije niso imele implementiranih ustreznih varnostnih funkcij. Težava je bila tudi podatkovna odvisnost (tesna povezanost podatkov in aplikacije) in zato slaba prenosljivost med različnimi sistemi.

V primeru uporabe vmesnega člena, programske opreme, imenovane SUPB, le-ta implementira številne pomembne funkcije, ki so jih prej morali programerji implementirati v vsaki aplikaciji posebej. S tem se bistveno zmanjša količina kode, potrebna za delo s podatki, zelo pa se poveča tudi kakovost in hitrost dela s podatki. Zagotovljena je podatkovna neodvisnost, ki ima številne prednosti. Zelo pomembni za to področje so seveda različni standardi, ki definirajo enoten način

upravljanja podatkov, ki omogoča uporabo istih oz. zelo podobnih ukazov ne glede na konkretni SUPB.

Najpomembnejši standard na tem področju je zagotovo SQL (Structured Query Language), ki predstavlja standard za upravljanje s podatki v relacijskih SUPB. SQL je postal standard organizacije ANSI (American National Standards Institute) v letu 1986 in organizacije ISO (International Organization for Standardization) leta 1987. Kasneje je bil standard večkrat dopolnjen (zadnja verzija ISO/IEC 9075). Kljub obstoju standarda na tem področju pa SQL koda večinoma brez popravkov ni prenosljiva med različnimi SUPB-ji, lahko bi rekli, da le-ti govorijo različna narečja tega jezika.

2.2 Funkcije SUPB

Že sam Codd je leta 1982 opredelil osem funkcij in storitev, ki jih mora nuditi vsak SUPB. V nadaljevanju jih podajamo deset (povzeto po Connolly in Begg, 2010, str. 49-54, Mohorič, 1992, str. 22-29).

SUPB po eni strani omogoča dostop do podatkov, po drugi strani pa izvaja zaščito in nadzor do uporabe podatkov. Tako se v grobem njegove funkcije dele na **dostopne** in **kontrolne**. Med dostopne funkcije štejemo tiste, ki so namenjene splošnim uporabnikom (npr. vnos podatkov, ažuriranje podatkov, poizvedbe) in omogočajo uporabo podatkovne baze ter tiste, ki skrbniku PB omogočajo izgradnjo PB (npr. definiranje shem, kreiranje PB). Prva skupina kontrolnih funkcij, imenovana zaščitne funkcije, je prikrita uporabnikom in se samodejno aktivira ob uporabi dostopnih funkcij (npr. preverjanje vrednosti vhodnih podatkov, preverjanje dostopnih pravic, zagotavljanje celovitosti pri sočasni uporabi PB s strani več uporabnikov). Druga skupina, imenovana nadzorne funkcije, pa omogoča skrbniku PB zbiranje podatkov o uporabi PB in aktivnostih SUPB. Ti podatki nadalje skrbniku omogočajo reorganizacijo (optimizacijo) fizične podatkovne baze pa tudi odkrivanje zlorab pri uporabi PB.

Ključne funkcije tipičnega SUPB tako so (Connolly in Begg, 2010, str. 16):

- **Upravljanje s podatki (SQL):**
  - **Kreiranje podatkovnih struktur** (ang. Create) je omogočeno z jezikom DDL - Data Definition Language.
  - **Vzdrževanje podatkov** (ang. Insert, Update, Delete) izvajamo z uporabo jezika DML - Data Manipulation Language.
  - **Izvajanje povpraševanja**: povpraševalni jeziki (ang. Query Language).
- **Mehanizmi nadzora nad dostopom do podatkov zagotavljajo:**
  - dostop do podatkov v skladu z avtorizacijo,
  - skladnost podatkov,
  - sočasni dostop do podatkov z uporabo mehanizmov zaklepanja ter
  - obnovo podatkov po transakcijskih in sistemskih nesrečah (razveljavljanje transakcij, ponavljanje transakcij…).

2.2.1 Shranjevanje, pridobivanje in spreminjanje podatkov

SUPB mora omogočati shranjevanje, pridobivanje in posodabljanje baze podatkov. Gre za temeljno funkcijo vsakega SUPB. Pri tem je pomembno, da SUPB pred uporabniki skrije interno fizično implementacijo PB (kot je organizacija datotek) in zagotovi višje nivojske mehanizme (npr. jezika SQL in QBE), ki omogočajo shranjevanje, pridobivanje in posodabljanje podatkov na uporabniško prijazen način.

2.2.2 Dostopnost kataloga PB

Sistemski katalog (ang. system catalog) je po arhitekturi temeljna komponenta SUPB. Sistemski katalog, imenovan tudi podatkovni slovar (ang. data dictionary) ali metapodatki (podatki o podatkih) opisuje podatke, vsebovane v podatkovni bazi. Količina informacij in način njihove uporabe se med SUPB-ji razlikujejo, a tipično sistemski katalog vsebuje:

- Imena, tipe in velikosti podatkovnih elementov,
- Imena povezav,
- Integritetne omejitve in
- Imena uporabnikov z njihovimi dostopnimi pravicami,
- Zunanje, konceptualne in notranje sheme s preslikavami med njimi
- Statistike uporabe PB (npr. frekvence transakcij).

2.2.3 Podpora transakcijam

SUPB mora vsebovati mehanizem, ki zagotavlja, da se pri ažuriranju vedno v celoti izvedejo vsa ažuriranja, ki jih obsega določena transakcija ali pa nobeno izmed njih, saj bi v nasprotnem primeru PB postala nekonsistentna.

- **Definicija 13:** *Transakcija je sklop akcij, izvajan s strani uporabnika ali aplikacije, ki dostopa ali spreminja vsebino podatkovne baze, ki mora biti izveden v celoti, ali pa ne sme biti izveden noben del (*Connolly in Begg, 2010, str. 51*).*
- **Definicija 14:** *Transakcija je zaporedje ažuriranj, ki povzroče prehod podatkovne baze iz enega veljavnega stanja v novo veljavno stanje, torej ohranja konsistentnost podatkovne baze (Mohorič, 1992, str. 183).*

V primeru, da pride med izvajanjem transakcije, ki vsebuje več ažuriranj, do napake na računalniškem sistemu, mora SUPB izvesti razveljavitev že izvedenih ažuriranj, da zagotovi ponovno konsistentno stanje PB.

Primeri enostavnih transakcij so na primer dodajanje novega študenta, predmeta ali učitelja v podatkovno bazo študentskega informacijskega sistema. Tudi brisanja študentov, predmetov ali učiteljev so primeri transakcij.

Kompleksnejša transakcija pa bi bila na primer prenos nosilstva predmetov iz več predhodnih učiteljev na novega učitelja, ki se zaposli na šoli. Ker je tukaj potrebno ažurirati podatke na več mestih v podatkovni bazi, se lahko zgodi, da pride do začasne odpovedi sistema med izvajanjem te transakcije, kar bi pomenilo, da so določeni predmeti bili prepisani na novega nosilca, določeni pa

so ostali pisani na stare nosilce. V tem primeru je potrebno transakcijo najprej razveljaviti, da PB zavzame staro konsistentno stanje (pred začetkom izvajanja te transakcije) ter nato celotno transakcijo ponoviti.

Transakcija v svojem življenjskem ciklu prehaja med stanji: aktivna, uspešna, ponesrečena, uspešno zaključena, neuspešno zaključena. V primeru ponesrečene transakcije, npr. podatkovne nesreče, mora SUPB poskrbeti za razveljavitev vseh njenih ažuriranj v PB, nato postane neuspešno zaključena. Takšne transakcije, ki se niso uspešno zaključile zaradi napake na sistemu, SUPB potem vrne v ponovno izvajanje.

Transakcija se začne z operacijo **Začetek transakcije**, zaključi pa z ukazom **Pomni** (ang. Commit) ali **Pozabi** (ang. Cancel, Rollback). Ukaz Pomni je sporočilo SUPB, da so se vsa ažuriranja v okviru transakcije uspešno končala in se naj zato vse spremembe v PB ohranijo. Ukaz Pozabi pa je navodilo, naj se vsa ažuriranja v okviru te transakcije razveljavijo. SUPB implementirajo različne načine za razveljavitev transakcij in njihovo ponovno izvajanje (npr. obnavljanje s senčnimi stranmi, z dnevnikom in kopijo).

2.2.4 Sočasni dostop do podatkovne baze

SUPB mora vsebovati mehanizem za sočasni dostop (ang. concurrency control), ki zagotavlja, da so v primeru ažuriranj podatkovne baze s strani več uporabnikov, le ta izvedena pravilno.

SUPB mora zagotavljati sočasni dostop do podatkovne baze velikemu številu uporabnikov. V primeru, da uporabniki podatke le berejo, je to enostavno. Težava nastane, ko več uporabnikov dostopa do podatkov in vsaj eden od njih podatke tudi ažurira. Takrat hitro lahko nastopijo nekonsistentnosti v podatkovni bazi.

Navedeni različni načini zaseganja vplivajo na:
- obseg sočasnosti pri izvajanju transakcij,
- obseg podatkov o odobrenih in zahtevanih zaseženjih,
- stopnjo dodatne obremenitve SUPB z izvajanjem nadzora nad zaseženji.

Slika 5: Zaporedje izvajanja transakcij T1 in T2

Time	T_1	T_2	bal_x
t_1		read(bal_x)	100
t_2	read(bal_x)	$bal_x = bal_x + 100$	100
t_3	$bal_x = bal_x - 10$	write(bal_x)	200
t_4	write(bal_x)		90
t_5			90

Vir: Connolly in Begg, 2010, str. 52.

Poglejmo primer izvajanja dveh transakcij T1 in T2 s slike (Slika 5). Transakcija T1 dvigne z računa 10 €, T2 pa nanj položi 100 €. Stanje na računu je pred začetkom izvajanja T1 in T2 enako 100 €. Če bi se transakciji izvedli zaporedoma (brez časovnega prekrivanja), bi končno stanje bilo 190 €.

Vendar pa se transakciji v našem primeru začneta izvajata skoraj hkrati, obe tako prebereta začetno stanje 100 €. T2 nato prva spremeni stanje računa s pologom 100 € na 200 €. Stanje shrani in s tem ažurira podatkovno bazo. Medtem, T1 zmanjša svojo kopijo stanja računa z dvigom 10 € (stanje te kopije je zdaj 90 €). Stanje 90 € T1 nato zapiše v podatkovno bazo in s tem prepiše stanje, ki ga je vzpostavila T2 pred njo. S tem se polog 100 € s strani T2 izgubi.

Da ne bi prihajalo do takšnih nepravilnosti SUPB-ji implementirajo različne mehanizme sočasnega dostopa do podatkovne baze, od zaseganja zapisov podatkovne baze (ang. locking) do časovnega označevanja (ang. timestamping).

Zaseganje zapisov se lahko izvaja na različnih nivojih podatkovne baze in sicer:

- Na logičnem nivoju lahko zasegamo
  - relacije
  - n-terice v relacijah,...
- Na fizičnem nivoju lahko zasegamo:
  - celotno fizično podatkovno bazo,
  - tabele,
  - fizične bloke oz. strani,
  - fizične zapise v tabeli.

Naloga nadzora nad sočasno uporabo PB je:

- ohraniti podatkovno bazo v konsistentnem stanju ter hkrati
- dopustiti čim večjo sočasnost izvajanja transakcij.

2.2.5 Obnavljanje podatkovne baze po nesrečah

SUPB mora vsebovati mehanizme, ki zagotavljajo obnovo podatkovne baze v primeru kakršnih koli podatkovnih nesreč. Nesreče se med seboj razlikujejo po vzrokih in posledicah, od vrste nesreče pa so odvisni postopki obnavljanja. Ločimo naslednje vrste podatkovnih nesreč (Mohorič, 1992, str. 182-200):

- transakcijske nesreče, ki jih odkrijejo uporabniški programi,
- transakcijske nesreče, ki jih odkrijeta SUPB ali operacijski sistem,
- sistemske nesreče,
- diskovne nesreče.

Transakcijske nesreče pogosto nastanejo zaradi napačnih vhodnih podatkov. Podatki lahko kršijo pravila (ki jih imenujemo omejitve), ki povedo kakšen tip, dolžino in obliko podatkov lahko vnesemo v posamezno polje podatkovne baze. Če imamo na primer polje EMŠO, pravilo lahko pravi, da je vanj možno vnesti le številke, in da je dolžina točno 13. V nasprotnem primeru gre za podatek, ki ga ne bo možno vnesti. Vhodni podatki so lahko tudi v protislovju z obstoječimi podatki v bazi. V obeh primerih transakcije ni mogoče uspešno izvesti.

Pri izvajanju transakcije lahko pride tudi do prekinitve izvajanja programa (npr. deljenje z 0). Tovrstne napake detektira operacijski sistem in jih sporoči SUPB, ki do sedaj izveden ažuriranja v okviru transakcije razveljavi.

Za **sistemsko nesrečo** štejemo izgubo podatkov v notranjem pomnilniku, zaradi prekinitve delovanja le-tega. Napaka je lahko povzročena s prekinitvijo napajanja ali napako pri branju ukaza ali podatka iz notranjega pomnilnika. Sistemska nesreča povzroči prekinitev izvajanja trenutno aktivnih transakcij. Pred nadaljnjo uporabo PB po sistemski nesreči, je potrebno obnoviti PB v zadnje veljavno stanje pred nesrečo in ponoviti izvajanje prekinjenih transakcij.

Diskovne nesreče pomenijo izgubo podatkov, shranjenih na disku. Razlog je lahko okvara diskovne površine, okvara bralno pisalnih glav, okvara krmilnika diska itd. Tudi pri teh nesrečah so lahko nekatere transakcije med svojim izvajanjem prekinjene. Ko PB spet začne delovati, je potrebno obnoviti PB v zadnje veljavno stanje pred nesrečo in ponoviti izvajanje prekinjenih transakcij.

Da bi zagotovili možnost obnove PB po vseh navedenih vrstah nesreč, moramo implementirati različne vrste **podvajanja shranjenih podatkov** ter **podvajanja komponent računalniškega sistema**. Tako na primer lahko podvojimo podatkovno bazo in sicer na različnih nivojih: diskovni krmilnik zapisuje sočasno podatke na dva fizično ločena diska. Če se ena PB pokvari, se uporablja njena dvojnica. Za večjo zanesljivost se lahko podvoji tudi diskovni krmilnik ali še druge komponente računalniškega sistema. Navedena rešitev omogoča hitro obnavljanje PB predvsem po diskovnih in delno sistemskih podatkovnih nesrečah, vendar je relativno draga. Delna ali inkrementalna kopija se lahko izvaja tudi v času, ko je PB v uporabi.

Zanesljivost delovanja se poveča tudi z RAID redundanco diskov (ang. redundant array of independent disks), ki povečuje zanesljivost diska in s tem omogoča večjo varnost shranjenih podatkov. Poznamo različne vrste RAID redundance (RAID 0, RAID 1…). Kot zanesljivo varovanje pred diskovnimi nesrečami se navadno uporablja periodično **kopiranje PB na magnetni trak**.

Pri transakcijskih nesrečah, ko je potrebno razveljaviti že izvedena ažuriranja, pride v poštev **obnavljanje s senčnimi stranmi**. Gre za zapisovanje ažuriranj na proste bloke na disku in ne direktno v podatkovno bazo. V primeru uspešno zaključene transakcije, se ti bloki vključijo v PB namesto starih - neažuriranih.

Obnavljanje z dnevnikom in kopijo temelji na občasni izdelavi kopije PB in izdelavi dnevnika. S kopijo PB lahko obnovimo bazo v veljavno stanje pred diskovno nesrečo. V dnevnik se zapisujejo podatki, s katerimi je možno s kopijo obnovljeno PB obnoviti v zadnje veljavno stanje tik pred nesrečo (vsebuje tudi podatke, s katerimi je možno ponovno izvesti tudi transakcije, ki so bile zaradi nesreče prekinjene).

2.2.6 Avtorizacijske storitve

Avtorizacijske storitve (ang. authorization services) zagotavljajo, da lahko do podatkov podatkovne baze dostopajo samo avtorizirani uporabniki. Podatkovno bazo želimo namreč zaščiti pred neavtoriziranimi dostopi, namernimi ali nenamernimi. Podatkovna baza navadno vsebuje podatke različnih stopenj zaupnosti. Medtem ko različne šifrante lahko vidijo vsi uporabniki, pa na primer dovolimo vpogled v plače zaposlenih samo vodstvenim delavcem (vsak vodja npr. lahko vidi plače le svojih podrejenih). V bazi študentskega informacijskega sistema je smiselno, da vse podatke o ocenah študentov vidi le referat, posamezni učitelji pa le ocene pri predmetih, katerih nosilci so. Poleg vpogledov je potrebno natančno določiti tudi, katere podatke lahko kdo vpisuje ali spreminja.

SUPB mora tako vsebovati orodje, s katerim skrbnik PB nastavlja dostopne pravice skupinam uporabnikov, skladno z njihovo vlogo v poslovnem sistemu.

2.2.7 Integritetne storitve

Integriteta podatkovne baze se nanaša na pravilnost in konsistentnost v njej vsebovanih podatkov. Lahko jo razumemo kot enega od varnostnih mehanizmov. Integriteto PB zagotavljamo z različnimi omejitvami (ang. constraints). Gre za pravila, ki jih podatkovna baza oz. v njej vsebovani podatki ne smejo kršiti. Poznamo različne vrste integritetnih omejitev, ki so podrobneje opisane v poglavju 5.2.2.

Podajmo primer: polje za vnos imena študenta lahko vsebuje le črke in je dolgo npr. največ 20 znakov. Če bo uporabnik pomotoma pri vnosu vtipkal številko ali katerega od posebnih znakov, bo kršeno navedeno integritetno pravilo in vnos ne bo možen.

Pogosta integritetna omejitev je tudi NOT NULL, ki pomeni, da je določen podatek obvezen. Primer: pri vnosu novega študenta mu moramo določiti njegovo vpisno številko ter vnesti ime in priimek.

2.2.8 Storitev podatkovne neodvisnosti

SUPB mora zagotavljati neodvisnost programov od fizične realizacije podatkovne baze. Podatkovno neodvisnost navadno zagotavljamo z uporabo različnih vrst shem (konceptualna, zunanja, notranja). Ločimo fizično podatkovno neodvisnost in logično podatkovno neodvisnost. **Konceptualna shema** zagotavlja **fizično podatkovno neodvisnost**, saj skrije podrobnosti o tem, kako so podatki dejansko shranjeni na disku, o strukturi datotek in o indeksih. **Zunanje sheme** pa zagotavljajo **logično podatkovno neodvisnost** (glej tudi poglavje 1.3).

2.2.9 Administratorska orodja

Gre za orodja, ki skrbniku podatkovne baze omogočajo izvajanje administrativnih opravil nad podatkovno bazo kot so (Connolly in Begg, 2010, str. 54):

- **Orodja za spremljanje**: spremljanje delovanja in uporabe PB,
- **Statistično analitična orodja**: omogočajo izdelavo statistik uporabe in performans PB.
- **Indeksirna orodja**: omogočajo spremembe načinov indeksiranja PB za zagotovitev boljših performans njenega delovanja.
- **Orodja za realokacijo in brisanje**: omogočajo fizično odstranitev zbrisanih zapisov z naprav za shranjevanje ter realokacijo PB, kadar je to potrebno.

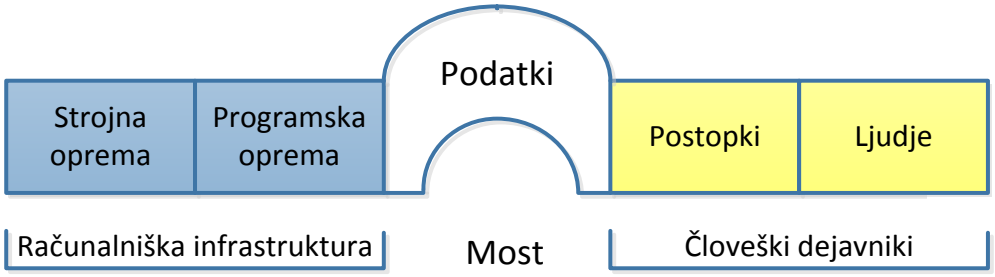
2.2.10 Podpora komuniciranju

SUPB mora omogočati dostop do podatkovne baze preko omrežja, saj navadno želimo imeti eno samo centralno podatkovno bazo, nameščeno na tako imenovanem podatkovnem strežniku, do katere dostopajo uporabniki s svojimi računalniki preko omrežja. Programska oprema, ki omogoča komunikacijo s SUPB, se imenuje upravljavec podatkovnih komunikacij (ang. Data Communication Manager) in ni del SUPB. SUPB mora biti sposoben komunikacije z različnimi vrstami upravljavcev komunikacij.

2.3 Komponente okolja SUPB

Okolje SUPB sestavljajo naslednje komponente (Connolly in Begg, 2010, str. 18-21): strojna oprema, programska oprema, podatki, postopki in ljudje (Slika 6).

Slika 6: Okolje SUPB



Vir: Connolly in Begg, 2010, str. 18.

Za delovanje SUPB potrebujemo **strojno opremo** (ang. hardware), na katero DBMS namestimo. Kakšno vrsto strojne opreme potrebujemo, je odvisno od samih potreb poslovnega sistema in zmogljivosti, ki jih mora zagotavljati (npr. količine podatkov, števila uporabnikov, števila transakacij, stopnje varnosti). Včasih za namestitev SUPB zadostuje že osebni računalnik, najpogosteje postavimo zmogljivejši računalnik, ki deluje kot strežnik, v primeru velikih sistemov (npr. banke) pa lahko SUPB teče tudi na superračunalniku. SUPB se med seboj razlikujejo po minimalnih zahtevah strojne opreme, notranjega in zunanjega pomnilniškega prostora. Za računalnik, kjer bo nameščen SUPB je potrebno zagotoviti dovolj hitrega pomnilnika in diska. SUPB imajo tudi različne zahteve glede operacijskega sistema, na katere jih je možno namestiti.

Programsko opremo (ang. software) sestavlja sam SUPB in različna druga programska orodja, ki omogočajo poizvedovanje po podatkovni bazi z uporabo jezikov QBE ali SQL, generatorje obrazcev, generatorje poročil in druga orodja za hitro izdelavo aplikacij (tudi programski jezik). Pomembna je tudi omrežna programska oprema, če bo SUPB deloval v mrežnem načinu.

Podatki (ang. data) so s stališča uporabnika najpomembnejše komponenta okolja SUPB, saj predstavljajo most med tehnološkimi komponentami in človeškimi komponentami. Podatke sestavljajo operativni podatki in metapodatki (podatki, ki opisujejo strukturo shranjenih podatkov).

Postopki (ang. procedures) obsegajo navodila za načrtovanje in uporabo podatkovne baze oz. SUPB in sicer:

- načini prijave v SUPB,
- načini uporabe posameznih orodij,
- postopki za zagon in zaustavitev SUPB
- postopki izdelave varnostnih kopij PB
- postopki v primeru okvar strojne ali programske opreme, na katerih teče SUPB, ali samega SUPB. Postopki za restavriranje podatkovne baze po takšnih nesrečah (npr. odpovedi diska, na katerem je bila shranjena PB).

- postopki za spreminjanje strukture PB (tabel, povezav,…), reorganizacija PB čez več diskov ali računalnikov, nastavitve performančnih parametrov, nastavitve parametrov izdelave varnostnih kopij itd.

Zadnjo komponento okolja predstavljajo **ljudje** (ang. human), uporabniki podatkovne baze. Sem spadajo:

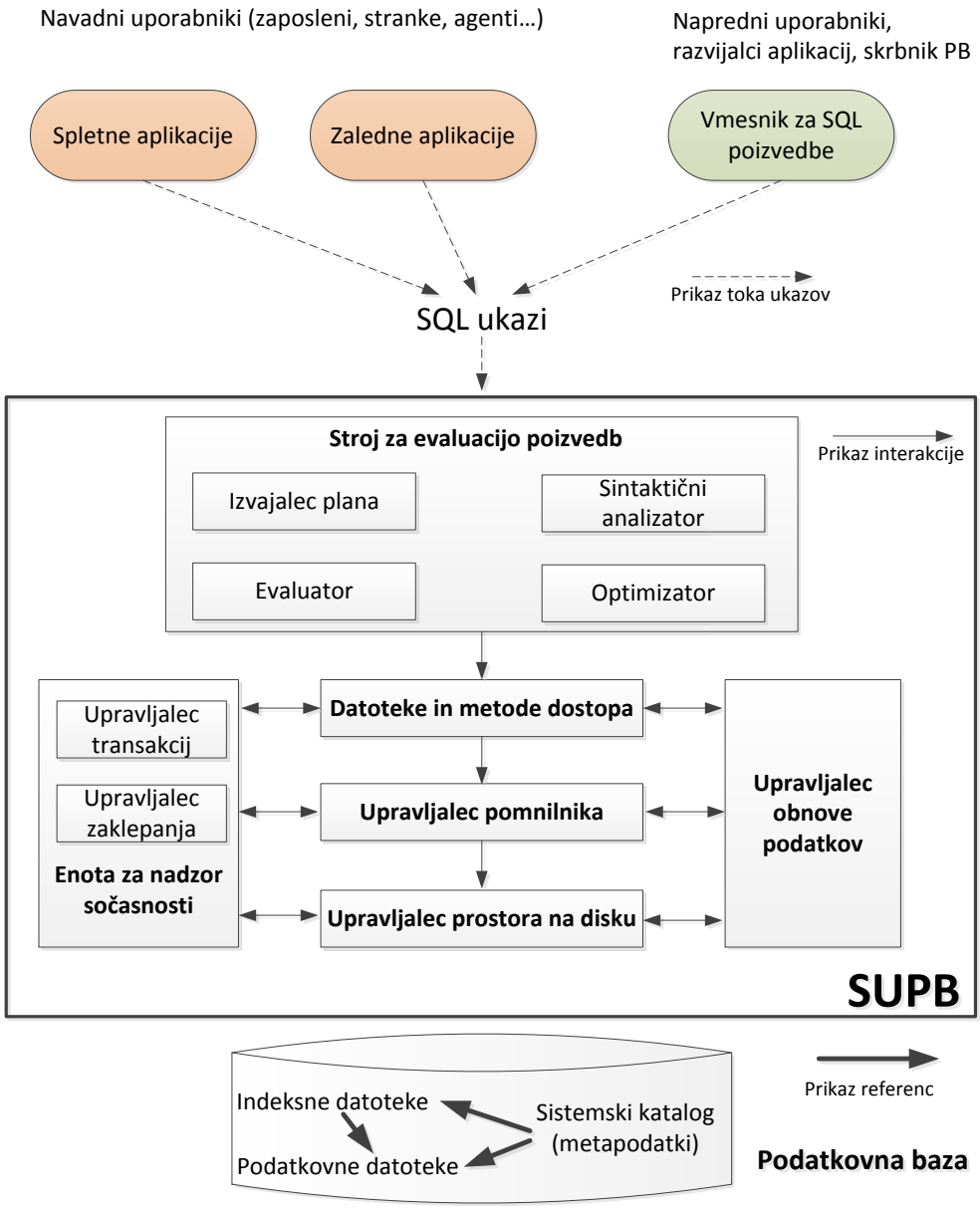
- skrbnik podatkovne baze (ang. DBA – Database Administrator),
- analitiki in načrtovalci PB,
- razvijalci aplikacij in seveda
- končni uporabniki.

2.4 Delovanje SUPB

Slika 7 prikazuje zgradbo SUPB. SUPB sestavljajo naslednje komponente (Ramakrishnan in Gehrke, 2003, str. 19-21):

- **Stroj za evaluacijo poizvedb** (ang. Query Evaluation Engine):
  - Sintaktični analizator (ang. Parser): Sintaktično analizira poizvedbo, ki jo SUPB-ju posreduje aplikacija.
  - Optimizator (ang. Optimizer): Na podlagi informacij o tem, kako so podatki shranjeni, izdela učinkovit plan za izvajanje poizvedbe. Plan izvajanja predstavlja načrt za izvedbo poizvedbe in je navadno predstavljen kot drevo relacijskih operatorjev.
  - Evaluator operatorjev (ang. Operator Evaluator): Na osnovi plana izvajanja analizira poizvedbo.
  - Izvajalec plana (ang. Plan Executor): Izvede poizvedbo po navodilih plana poizvedbe.
- **Datoteke in metode dostopa** (ang. Files and Access Methods): enota, ki omogoča delo z datotekami.
- **Upravljalec pomnilnika** (ang. Buffer Manager): Prenaša strani iz diska v pomnilnik glede na bralne potrebe.
- **Upravljalec prostora na disku** (ang. Disk Space Manager): Najnižji nivo SUPB je zadolžen za upravljanje z diskom. Vse operacije višjih plasti se tukaj prevedejo v nizko-nivojske ukaze za delo z diskom.
- **Enota za nadzor sočasnosti** (ang. Concurrency Control) sestavljata dve komponenti:
  - Upravljalec transakcij (ang. Transaction Manager): Zagotavlja zaseganje podatkov z uporabo določenih protokolov in skrbi za razporejanje izvajanja transakcij.
  - Upravljalec zaklepanja (ang. Lock Manager): Vzdržuje informacije o zahtevanih in odobrenih zaseženjih podatkov.
- **Upravljalec obnove podatkov** (ang. Recovery Manager): Vzdržuje dnevnik in skrbi za obnavljanje sistema v zadnje skladno stanje pred nesrečo.

Slika 7: Zgradba SUPB



Vir: Ramakrishnan in Gehrke, 2003, str. 20.

2.5 Naloge skrbnika podatkovne baze

Skrbnik podatkovne baze je zelo pomembna vloga v poslovnem sistemu, saj je od kakovosti njegovega dela odvisna celovitost, razpoložljivost in zaupnost podatkov v podatkovni bazi. Njegove naloge so (Ramakrishnan in Gehrke, 2003, str. 22):

- Kreiranje fizičnih objektov podatkovne baze;
- Zagon in zaustavitev delovanja SUPB;
- Uglaševanje podatkovne baze:
  - Upravljanje s konfiguracijskimi parametri za zagon instance podatkovne baze,
  - Upravljanje s parametri delovanja podatkovne baze.

- Nadzor nad dnevniki in ostalimi sistemskimi datotekami podatkovne baze;
- Skrb za varnost dostopa do podatkovne baze;
- Skrb za obnovitev podatkovne baze v primeru podatkovnih nesreč. Skrbi tudi za vse postopke, ki obnovitev omogočajo;
- Sodeluje z ostalimi akterji v okviru IS z namenom spoznavanja trenutnih in bodočih potreb za potrebe zagotavljanja optimalnega delovanja podatkovnih baz izbranega SUPB.

Vprašanja za ponavljanje

1. Kaj je sistem za upravljanje s podatkovno bazo (SUPB)?
2. Katere mehanizme za upravljanje s podatki vsebuje SUPB?
3. Katere mehanizme za nadzor nad dostopom do podatkov vsebuje SUPB?
4. Naštej in opiši komponente okolja SUPB.
5. Katere komponente sestavljajo SUPB? Kako delujejo?
6. Naštej ključne uporabnike SUPB in njihova opravila?
7. Kaj so naloge skrbnika podatkovne baze?

3 Podatkovni modeli in vrste SUPB

Model, s katerim opišemo, kaj bi želeli hraniti v podatkovni bazi ter kakšne povezave obstajajo med elementi, ki jih želimo hraniti, se imenuje **podatkovni model**. Podatkovni model je način, kako na visoki ravni abstrakcije opišemo podatke, ki jih želimo hraniti ter skrijemo nepomembne podrobnosti. Podatkovni model odraža uporabnikovo percepcijo realnega sveta. V resnici izraža uporabnikovo predstavo o tem, kako naj bodo podatki shranjeni.

Grad in Jaklič (1996, str. 43) definirata podatkovni model kot:

- **Definicija 15:** *posplošeno predstavitev (ponazoritev) podatkov o objektih, dogodkih, aktivnostih in njihovih povezavah znotraj obravnavanega sistema.*

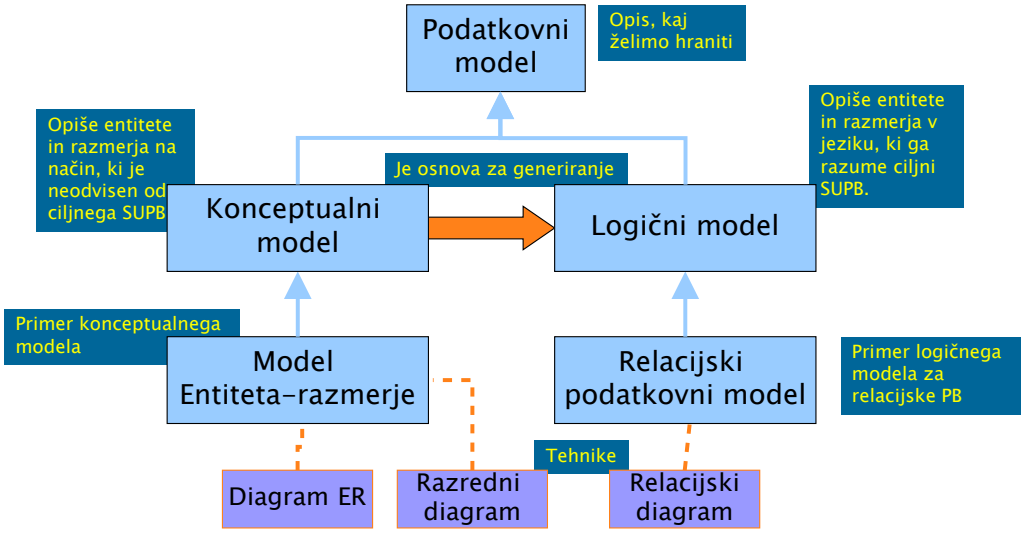
V splošnem gre pri podatkovnih modelih za opisne mehanizme, s pomočjo katerih so predstavljene vse tri ravni podatkovne baze. Zunanje in konceptualne sheme predstavljajo logičen opis podatkov (logični podatkovni modeli), medtem ko notranja shema predstavlja njihov fizični opis (fizični podatkovni modeli).

Logični podatkovni modeli se delijo na (Mohorič, 1992, str. 104):

- **Površinske podatkovne modele:** Temeljijo na abstraktni podatkovni strukturi, za katero sta definirani množici operacij in integritetnih omejitev. Podatkovna neodvisnost ni popolna, obstaja tudi problem semantičnega opisa podatkov.
- **Globinske (semantične) podatkovne modele:** Za razliko od površinskih podatkovnih modelov so tehnološko povsem neodvisni, omogočajo pa tudi boljši opis pomena shranjenih podatkov.

V naslednjem poglavju so podrobneje predstavljeni štirje najpogosteje uporabljani površinski (logični) podatkovni modeli.

Slika 8: Podatkovni model



Konceptualni podatkovni model opiše entitete in razmerja med njimi na način, ki je neodvisen od nadaljnje izbire ciljnega SUPB. Najpogosteje uporabljana vrsta konceptualnega modela je model **entiteta-razmerje** ali **ER model**. **Logični podatkovni model** pa je oblika podatkovnega modela, ki ga razume cilji SUPB. Pri transformaciji iz konceptualnega v logični model je tako potrebno izbrati ciljni SUPB (npr. Oracle, MS SQL …). Najpogosteje uporabljana vrsta logičnega modela je relacijski model, ki se uporablja, kadar želimo izdelati relacijsko podatkovno bazo. **Fizični podatkovni model** predstavlja v primeru relacijskega model kar skripta v jeziku SQL, ki je izdelana za točno določen SUPB, v katerem jo je moč zagnati, kar povzroči kreiranje PB in njenih gradnikov.

Načrtovanje podatkovne baze obsega tri glavne faze (Connolly in Begg, 2010, str. 271-275):

- **Konceptualno načrtovanje:** izdelava konceptualnega podatkovnega modela, ki je neodvisen od kasnejše fizične implementacije podatkovne baze (ciljnega SUPB, uporabljenega programskega jezika ali strojne platforme). Model mora zajeti znanje o poslovni domeni, ki izhaja iz uporabniških zahtev ter je podlaga za nadaljnje faze načrtovanja, zato je njegova pravilnost zelo pomembna. Konceptualno načrtovanje je podrobneje predstavljeno v poglavju 4.
- **Logično načrtovanje:** izdelava logičnega podatkovnega modela, ki temelji na znanju zajetem s konceptualnim modelom ter posebnostih izbrane vrste logičnega modela (npr. mrežni, relacijski, objektni model). Logični model mora v teoriji še vedno biti neodvisen od posebnosti konkretnega SUPB (npr. ORACLE relacijske baze, MYSQL relacijske baze), vendar v praksi modelirna orodja tega ne upoštevajo vedno. Logično načrtovanje relacijske podatkovne baze je podrobneje predstavljeno v poglavju 5.2. Logični model je osnova za naslednjo fazo, fizično načrtovanje PB.
- **Fizično načrtovanje:** izdelava fizičnega podatkovnega modela, ki upošteva vse specifike ciljnega SUPB (npr. ORACLE 11g relacijske baze). Prvi korak je torej izbira konkretnega SUPB, v katerega bomo preslikali v predhodni fazi izdelan logični model. Če izberemo relacijski SUPB, to pomeni tri ključna opravila. Kreiranje PB (tabel in vseh omejitev, indeksov), določitev najprimernejših struktur za shranjevanje podatkov in indeksnih tabel ter načrtovanje varnostne sheme.

Podroben opis izdelave konceptualnega podatkovnega modela se nahaja v poglavju 4, v poglavjih 5.2 in 5.4 pa sta opisana postopka logičnega in fizičnega načrtovanja.

Za načrtovanje podatkovne baze je smiselno uporabiti tako imenovana CASE (Computer Aided Software Engineering) orodja, npr. Oracle SQL Developer Data Modeler, SAP Sybase PowerDesigner. Tovrstna orodja omogočajo načrtovanje na vseh treh nivojih ter avtomatsko pretvorbo med modeli na različnih nivojih. V poglavju 8.1 na primerih predstavljamo orodje SAP Sybase PowerDesigner in v poglavju 8.2 Oracle SQL Developer Data Modeler.

Poznamo več vrst logičnih modelov (Mohorič, 1992, str. 109-180, Grad in Jaklič, 1996, str. 45-75, Johnson, 1997, str. 9, Rob in Coronel, 2004, str. 33-55):

- **Hierarhični podatkovni model**
  - Hierarhični SUPB: IBM-ov IMS.
- **Mrežni podatkovni model**
  - Mrežni SUPB: IDS in IDMS.
- **Relacijski podatkovni model**
  - Relacijski SUPB: DB2, Oracle, MS SQL Server, MySQL, MS Access.
- **Objektni podatkovni model**
  - Objektni SUPB: Objectstore, Versant.
- **Objektno-relacijski podatkovni model**
  - Objektno-relacijski SUPB: Oracle, Informix, PostgreSQL.

Najbolj popularen logični podatkovni model je relacijski model. Objektni podatkovni model predstavlja novejšo tehnologijo, medtem ko sta hierarhični in mrežni podatkovni model predstavnika starejših modelov. V nadaljevanju prestavljamo značilnosti navedenih vrst modelov ter značilne predstavnike SUPB.

3.1 Hierarhični podatkovni model

Hierarhični podatkovni model je nastal konec petdesetih let in temelji na predpostavki, da so podatki organizirani pretežno v hierarhični drevesni strukturi. Ta predpostavka se je kasneje izkazala za napačno, posledica česar so zahteve po spremembah hierarhičnega modela oziroma po razvoju novih vrst modelov.

Osnovni gradnik hierarhične podatkovne strukture (gozda) je drevo, ki je sestavljeno iz med seboj povezanih zapisov. Poglavitna slabost hierarhičnega modela je velika redundanca podatkov v primeru, ko nimamo opravka s povsem hierarhičnim problemom. Omejitev so rešili z uporabo koncepta povezanih dreves, pri čemer pa pravzaprav ne gre več za drevesa, temveč za vrsto grafa, kar pa je že zelo blizu značilnostim mrežnega podatkovnega modela.

3.2 Mrežni podatkovni model

Pri mrežnem podatkovnem modelu je hierarhična podatkovna struktura zamenjana s splošnim grafom, kar omogoča bolj splošne povezave med vozlišči (entitetami). Razvit je bil konec šestdesetih

let s strani organizacije CODASYL (Conference On Data Systems Languages) oziroma njene delovne skupine DBTG (Data Base Task Group) z namenom reševanja nehierarhičnih podatkovnih problemov. Podatkovna struktura je predstavljena z mrežo, njena osnovna gradnika pa sta set (povezava zapisov dveh logičnih datotek) in zapis. Množica zapisov predstavlja entitete določenega tipa shranjene v samostojni logični datoteki. Temeljni problem mrežnega podatkovnega modela je zapletenost uporabe in slaba preglednost, kar je moteče predvsem pri izdelavi kompleksnih modelov.

3.3 Relacijski podatkovni model

Relacijski podatkovni model temelji na relacijski teoriji, katere utemeljitelj je E. F. Codd. Razvit je bil leta 1970. Na relacijskem modelu temeljijo skoraj vse današnje relacijske podatkovne baze oziroma relacijski SUPB. Gre za tržno najbolj razširjene SUBP, ki se dandanes uporabljajo za shranjevanje podatkov v večini poslovnih informacijskih rešitev. Največji tržni delež na tem področju tako imajo IBM (SUPB DB2), Oracle (SUPB Oracle 11g) in v zadnjem času tudi Microsoft (SUPB SQL Server). Zelo razširjen in priljubljen je tudi odprtokodni relacijski SUPB z imenom MySQL.

Podatkovna baza je v relacijskem modelu predstavljena kot množica med seboj povezanih tabel. Operacije nad podatki v tabelah se izvajajo s pomočjo povpraševalnih jezikov temelječih na relacijski algebri (postopkovni jeziki) ali relacijskem računu (nepostopkovni jeziki). Njegovi glavni prednosti v primerjavi s predhodnima modeloma sta formalna definiranost in osnovanost na matematičnih formulah - relacijah ter zagotovljena podatkovna neodvisnost, saj ne vsebuje elementov fizičnega shranjevanja podatkov. Relacijski podatkovni model oziroma relacijsko podatkovno bazo podrobno obravnavamo v poglavju 5.

3.4 Objektni podatkovni model

Objektno modeliranje podatkov je rezultat evolucije objektnega načina razmišljanja v informacijski tehnologiji. Objektni pristop se je sprva uveljavil na področju programskih jezikov, šele v devetdesetih letih pa je dosegel razmah tudi na področju razvoja informacijskih sistemov in v tem okviru podatkovnega modeliranja. Od drugih pristopov se ločuje po tem, da medsebojno povezuje modeliranje podatkov in procesov. **Temeljni koncept modela je objekt, ki v sebi združuje tako podatke kot razpoložljive metode nad njimi.** Splošno mnenje je, da je objektno modeliranje zelo blizu človekovega načina mišljenja, zato so mu napovedovali zelo svetlo prihodnost. Kljub temu dandanes večina komercialnih sistemov za upravljanje podatkovnih baz še vedno temelji na relacijskem podatkovnem modelu, čeprav že nekaj časa obstajajo tudi SUPB, ki temeljijo na **objektnem modelu** (npr. Objectstore in Versant), ki pa imajo še vedno določene slabosti (npr. manj prijazen poizvedovalni jezik), zaradi katerih se še niso uspeli docela uveljaviti.

3.5 Objektno-relacijski podatkovni model

Objektno-relacijski model se je v raziskovalnih krogih razvil že v zgodnjih devetdesetih letih prejšnjega stoletja po uveljavitvi objektno usmerjenih programskih jezikov. Najznačilnejša predstavnika sta bila **Illustra** in **PostgreSQL**, razvita na univerzi Barkeley. Sredi devetdesetih pa so

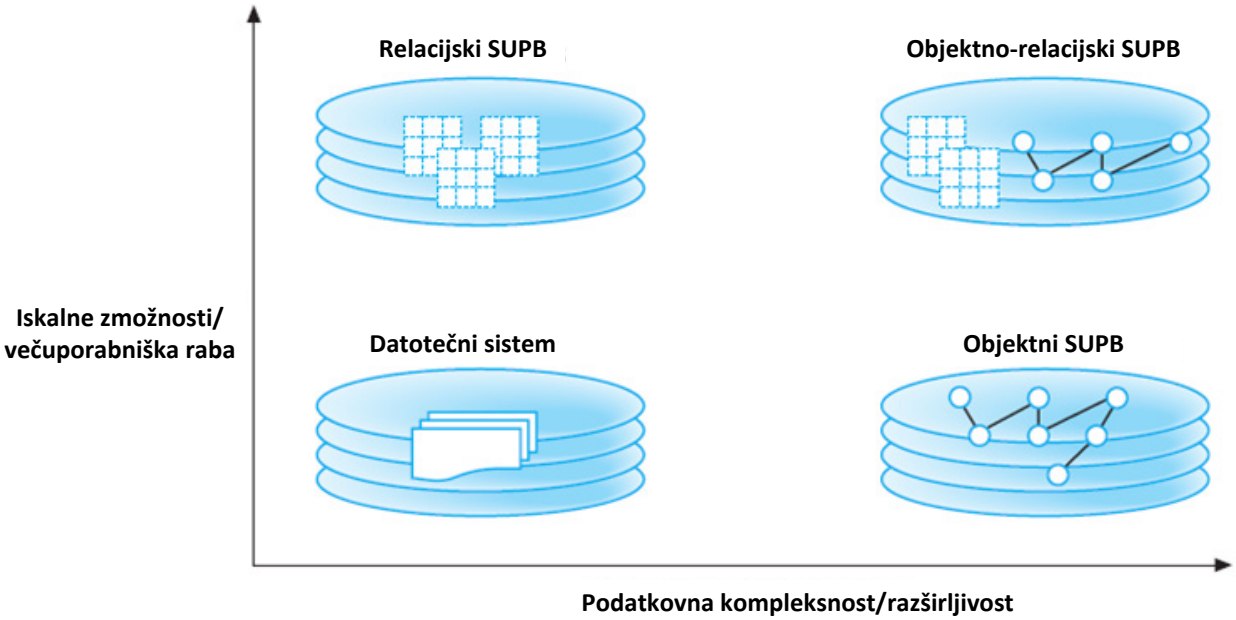
se pojavili tudi prvi komercialni objektno-relacijski SUBP (http://en.wikipedia.org/wiki/Object-relational_database). Vsi trije največji ponudniki relacijskih SUPB – Oracle, Microsoft in IBM – so tako svoje SUPB razširili tudi v objektno-relacijske SUPB.

Objektno relacijski model oziroma objektno-relacijska baza je podobna relacijski bazi, le da dodatno podpira tudi objektno orientiranost. Omogoča torej razširitev relacijskega podatkovnega modela z lastnimi podatkovnimi tipi in metodami. S tem skuša po eni strani odpraviti pomanjkljivosti klasičnega relacijskega SUPB, po drugi strani pa se ne izpostaviti pomanjkljivostim objektnega. Številni ponudniki relacijskih SUPB se zavedajo, da zgolj relacijski model pogosto ne zadošča zahtevnejšim aplikacijam in so zato potrebne določene razširitve. Sodobne aplikacije uporabljajo objektno usmerjenost: uporabniško razširljive tipe, enkapsulacijo, dedovanje, polimorfizem, objektno identiteto. Proizvajalci relacijskih SUPB tako stremijo k vključitvi navedenih funkcij, vendar so realizacije od proizvajalca do proizvajalca različne. Tako ne obstaja enoten relacijsko-objektni model. Lahko bi rekli, da obstaja množica različic tega modela, katerih značilnosti so odvisne od načina in stopnje realiziranosti objektnih razširitev. Vsi objektno-relacijski modeli tako uporabljajo klasične relacijske tabele in poizvedovalni jezik, vključujejo pa tudi možnost kreiranja objektov. Nekateri imajo možnost shranjevanja metod (ali procedur, ali prožilcev) kot tudi podatkov v podatkovno bazo (Connolly in Begg, 2010, str. 922-923).

3.6 Primerjava različnih vrst podatkovnih modelov oziroma SUPB

Stonebraker (1996) je skušal prikazati prednosti in slabosti posameznih vrst SUPB, ki temeljijo na obravnavanih vrstah podatkovnih modelov (Slika 9). V spodnjem levem kvadrantu se nahajajo preproste aplikacije, ki procesirajo enostavne podatke in nimajo velikih potreb po poizvedbah. Sem sodijo na primer urejevalniki besedil, ki za shranjevanje uporabljajo kar datotečni sistem.

Slika 9: Primerjava različnih vrst SUPB ter datotečnega sistema



Vir: Connolly in Begg, 2010, str. 924.

V spodnjem desnem kvadrantu so tiste aplikacije, ki obdelujejo kompleksne podatke, a nimajo večjih zahtev po poizvedovanju. Za te vrste uporabe, na primer programe za računalniško podprto načrtovanje, so objektni SUPB lahko primerna izbira. V zgornjem levem kvadrantu so tiste aplikacije, ki obdelujejo preproste podatke, a imajo kompleksne zahteve za poizvedovanju. V to skupino sodi večina tradicionalnih poslovnih aplikacij. V zgornjem desnem kvadrantu pa so tiste napredne aplikacije, ki obdelujejo kompleksne podatke in imajo tudi zahteve po kompleksnih poizvedbah (povzeto po Connolly in Begg, 2010, str. 923-924).

Vprašanja za ponavljanje

1. Kaj je podatkovni model?
2. Katera vrsta konceptualnega podatkovnega modela je najbolj razširjena?
3. Katere tri faze sestavljajo postopek načrtovanja podatkovne baze?
4. Kateri dve glavni vrsti podatkovnih modelov poznate?
5. Katere vrste logičnih podatkovnih modelov poznate?
6. Naštejte predstavnike različnih vrst SUPB glede na vrste podatkovnih modelov, na katerih temeljijo.
7. Katera vrsta logičnih podatkovnih modelov oz. SUPB je v praksi danes najbolj uveljavljena?
8. Kaj združuje relacijsko-objektni podatkovni model oz. SUPB? Zakaj se je uveljavil?
9. Katere vrste SUPB so najprimernejši za različne kombinacije podatkovne kompleksnosti (visoka, nizka) ter potreb po iskalnih zmožnostih (visoka, nizka)?

4 Konceptualno načrtovanje podatkovne baze

Konceptualno načrtovanje je opredelitev podatkovnih potreb oz. zahtev poslovne domene s pomočjo konceptualnega modela. Konceptualno načrtovanje preko konceptualnega modela poskrbi za opis pomena podatkov, potrebnih za poslovno domeno. Konceptualnega načrtovanja ne moremo avtomatizirati, za njegovo izvedbo je odgovoren analitik. Gre za prenos semantike v model. Zelo pomembno je sodelovanje uporabnikov in interakcija z uporabniki, saj so uporabniki nosilci znanja o poslovni domeni oziroma poznavalci semantike. Konceptualno načrtovanje mora upoštevati tudi poslovna pravila, ki v domeni veljajo. Z vidika semantične pravilnosti podatkovne baze je konceptualno načrtovanje najbolj kritično, saj se napake narejene pri konceptualnem načrtovanju prenašajo naprej na naslednje modele. **Konceptualni model je neodvisen od vrste ciljnega SUPB**, vendar pa v primeru, da že v začetku vemo, kateri SUPB bo uporabljen, lahko že v tej fazi upoštevamo tudi nekatere njegove omejitve (npr. v relacijski PB ne moremo imeti večvrednostnih atributov).

Konceptualni model mora biti preprost, enostaven za uporabo ter lahko in nedvoumno razumljiv, saj služi tudi za komunikacijo med uporabnikom in analitikom. Vsak koncept modela mora imeti svoj jasno definiran pomen. Model naj bo predstavljen tudi v grafični obliki, saj ta bistveno poveča njegovo informativnost. Pri tem se morajo grafični simboli za posamezne koncepte jasno razlikovati med seboj.

4.1 Tehnike konceptualnega načrtovanja

Konceptualne modele lahko predstavimo z različnimi diagramskimi tehnikami. Najpogosteje uporabljani tehniki za predstavitev konceptualnih podatkovnih modelov sta **diagram entiteta-razmerje** (ang. entity-relationship diagram) ter **razredni diagram** (ang. class diagram). V nadaljevanju poglavja obravnavamo **diagram entiteta-razmerje, imenovan tudi entitetni diagram**. Razredni diagrami pa so predstavljeni v poglavju 7, kjer predstavljamo značilnosti objektne podatkovne baze in njenega načrtovanja.

Pri diagramski tehniki entiteta-razmerje je znanih več notacij, od katerih se najbolj pogosto uporabljata Martinova in Chenova notacija. Chen je svojo notacijo predstavil že leta 1976, uporablja pa se v metodologiji Merise. Martinova notacija pa je opisana v več gradivih, ki jih je Martin s sodelavci predstavil v drugi polovici osemdesetih. **V nadaljevanju predstavljeni primeri so izdelani z uporabo Martinovo notacije.** Z uvajanjem novih konceptov je bila razvita **tehnika razširjenih entitetnih diagramov** (ang. extended Entity-Relationship Diagram - eERD), ki je semantično bogatejša (Krisper in drugi, 2004).

4.2 Gradniki konceptualnega modela

Osnovni gradniki konceptualnega modela entiteta razmerje (modela ER) so (Mohorič, 1997, str. 58, Rob in Coronel, 2004,str. 124-125):

- entitetni tip,
- atribut,
- razmerje,
- enolični identifikator (entitetni identifikator),
- ureditev tipov vključno z generalizacijo in specializacijo.

Podatki, ki jih zbiramo o entiteti, se nanašajo na njihove lastnosti - atribute, ki so pomembni za izvajanje poslovnih procesov. Entiteta je realni ali abstraktni predmet obravnave (npr. oseba, predmet, dogodek), o katerem zbiramo podatke in je značilen ali pomemben za poslovni sistem.

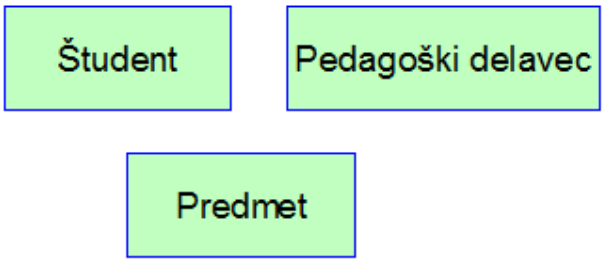
4.2.1 Entitetni tip

Z abstrakcijo ugotovimo, da obstajajo po lastnostih sorodni tipi entitet. V domeni visoke šole so to npr. študenti, predmeti, pedagoški delavci. Predstavimo jih z entitetnimi tipi ŠTUDENT, PREDMET, PEDAGOŠKI DELAVEC (Slika 10).

- **Definicija 16:** *Entitetni tip je skupina entitet (objektov) z enakimi lastnostmi, pri čemer lahko gre za objekte iz realnega sveta ali abstraktne objekte (Connolly in Begg, 2010, str. 322).*

Na entitetnih diagramih se **entitetni tip predstavi s pravokotnikom**, v katerega je vpisan njegov naziv. Običajno je to samostalnik v ednini, ki je kratek, hkrati pa dovolj dobro in nedvoumno opisuje oz. predstavlja vlogo in pomen entitetnega tipa.

Slika 10: Primeri entitetnih tipov pri načrtovanju podatkovne baze visoke šole

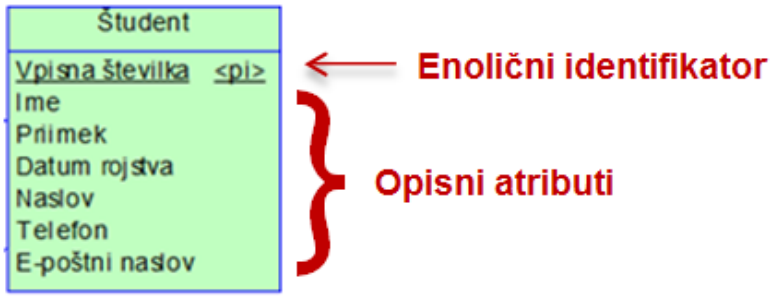


4.2.2 Atribut

Entiteta ima lahko veliko lastnosti, le del teh lastnosti je zanimiv oz. pomemben za opazovano poslovno domeno. Lastnosti, ki so pomembne za opazovano poslovno domeno, vključimo v konceptualni model tako, da jih kot **atribute** dodamo entitetnemu tipu. Za entitetni tip študent tako ugotovimo, da so zanj pomembne lastnosti - atributi VPISNA ŠTEVILKA, IME, PRIIMEK, DATUM ROJSTVA, NASLOV in drugi.

- **Definicija 17:** *Atributi predstavljajo lastnosti entitet, tako da identificirajo, tipizirajo, poimenujejo, opisujejo, kvalificirajo entitete. Atributi se v grobem delijo na **identifikacijske**, **opisne** in **izvedene**.*

Slika 11: Primeri atributov entitetnega tipa ŠTUDENT



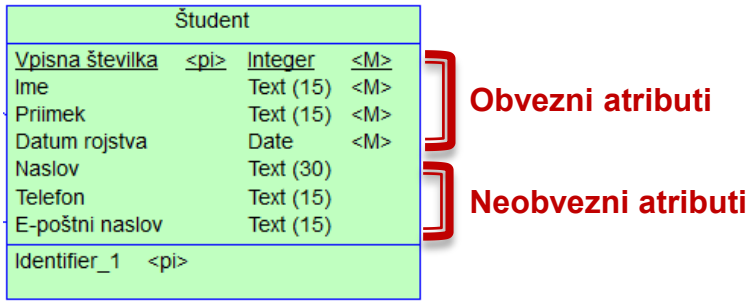
Z **enoličnim identifikatorjem entitetnega tipa** (ang. primary indentifier, oznaka <pi>) se vsaka entiteta enolično in nedvoumno identificira. Enolični identifikator entitete predstavlja tisto podmnožico lastnosti, ki posamezno entiteto enolično določa. Enolični identifikator entitete je lahko sestavljen iz enega ali več atributov ter ene ali več povezav. Identifikacija je lastnost, ki je entiteti trajno pripisana, ne glede na spremembo njene strukture ali stanja, omogoča pa tudi spremljanje zgodovine njene pojave v IS. Vrednosti identifikacijskega atributa ni dovoljeno spreminjati. Z ozirom na to, ali tvorijo enolični identifikator le atributi znotraj entitetnega tipa ali pa je v enoličnem identifikatorju tudi kakšno razmerje, ločimo med močnim entitetnim tipom in šibkim entitetnim tipom. V primeru entitetnega tipa ŠTUDENT (), enolični identifikator predstavlja VPISNA ŠTEVILKA, ki je atribut znotraj tega tipa, zato gre v tem primeru za močni entitetni tip. Enolični identifikator označimo z oznako <pi> in ga podčrtamo, da se loči od ostalih atributov.

Opisni atributi opisujejo količinske in kakovostne lastnosti entitet. Njihove vrednosti se lahko spreminjajo glede na spreminjanje stanja in lastnosti entitet. V primeru entitetnega tipa ŠTUDENT (Slika 11) so vsi ostali atributi, razen VPISNE ŠTEVILKE, opisni atributi.

Vrednosti izvedenih atributov se izračunajo iz definiranih vrednosti drugih atributov. Formule, algoritmi in logični izrazi za izračun vrednosti teh atributov so tudi del specifikacije podatkovnega modela. Izvedeni atributi niso v skladu s 3. normalno obliko, vendar se dopuščajo, če so pod nadzorom. V primeru entitetnega tipa ŠTUDENT (Slika 11) izvedenih atributov nimamo. Izvedeni atribut bi v tem primeru lahko bil STAROST, ki bi se izračunal iz atributa DATUM ROJSTVA.

Kadar je vrednost pri nekem atributu obvezna, pravimo, da je to **obvezni atribut** (ang. Mandatory). Če vrednost ni obvezna, je to **neobvezni atribut**. Atributi, ki so del enoličnega identifikatorja, so vedno tudi obvezni atributi. Vsak atribut pripada **določenemu podatkovnemu** tipu (ang. data type). Podatkovni tip atributu določimo v skladu s pomenom atributa. Najbolj pogosto uporabljani podatkovni tipi so znakovni, numerični in datumski. Pri nekaterih podatkovnih tipih je potrebno določiti še dolžino atributa, na primer pri znakovnem in numeričnem. Pri datumskem to ni potrebno, saj je datum vedno enake dolžine. Slika 12 prikazuje obvezne in neobvezne atribute entitetnega tipa ŠTUDENT. Obvezni atributi so označeni z oznako <M>, ki pomeni njihovo obveznost (ang. Mandatory). Obveznost pomeni, da bo pri vnosu novega študenta v podatkovno bazo potrebno vnesti vrednosti vpisne številke, imena, priimka in datuma rojstva, medtem ko je dopuščeno, da se podatki o naslovu telefonu in e-poštnem naslovu ne vnesejo. Definirani so tudi podatkovni tipi z dolžinami. Tako na primer atributa ime in priimek omogočata vnos do 15 znakov (Text(15)).

Slika 12: Določitev obveznosti atributov ter podatkovnih tipov entitetnega tipa ŠTUDENT



4.2.3 Razmerje

Entitete nastopajo v medsebojnih povezavah. Entitete istega tipa nastopajo v istovrstnih povezavah. Vrste povezav med entitetami se v modelu ER obravnavajo kot razmerja med entitetnimi tipi. Entitetni diagram tako prikazuje tudi **razmerja** (ang. relationship).

- **Definicija 18:** *Razmerje je množica smiselnih povezav med entitetnimi tipi (Connolly in Begg, 2010, str. 324).*

Vsako razmerje ima naziv, ki je običajno glagol ali glagolski samostalnik in opisuje vlogo entitet v njem. Najpogostejša so razmerja med dvema entitetnima tipoma, čeprav je možna tudi povezava entitetetnega tipa samega s sabo in pa povezava več entitetnih tipov med seboj. Med parom entitetnih tipov je na diagramu lahko prikazanih tudi več razmerij: npr. ŠTUDENT, KRAJ – ima stalno prebivališče, začasno prebiva (Slika 13). Razmerje ima **atributiven značaj**. To pomeni, da z razmerji med entitetnimi tipi ravno tako opisujemo lastnosti entitet.

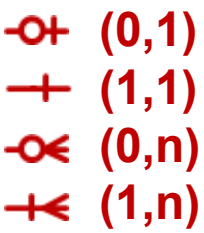
Slika 13: Razmerja in številnosti

Pomembna lastnost razmerja je **številnost,** ki prikazuje, s koliko entitetami je določena entiteta v razmerju in obratno. Poznamo naslednje možne vrste številnosti (Krisper in drugi, 2004):

- **ena proti ena:** vsaka primerek entitetnega tipa A je povezan z natančno enim primerkom entitetnega tipa B in obratno,
- **ena proti mnogo:** vsak primerek entitetnega tipa A je povezan z nič, enim ali več primerki entitetnega tipa B, vsak primerek entitetnega tipa B pa je povezan z natančno enim primerkom entitetnega tipa A ali

- **mnogo proti mnogo:** pri povezavi med primerki entitetnih tipov A in B ni omejitev, kar pomeni, da je vsak primerek entitetnega tipa A povezan z nič, enim ali več primerki entitetnega tipa B, in obratno.

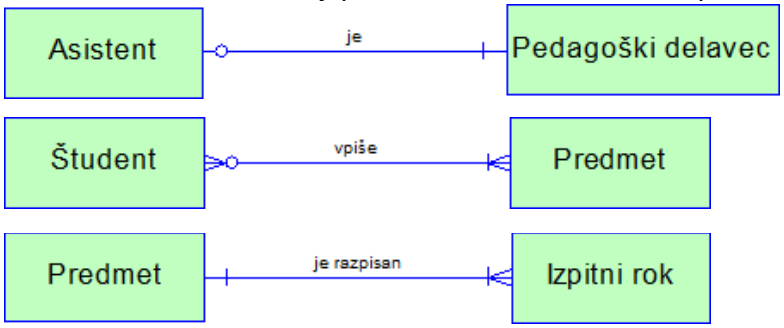
Slika 14: Grafični prikaz številnosti in obveznosti razmerij/povezav



Slika 14 podaja grafični prikaz različnih vrst številnosti in obveznosti. Pri tem je potrebno povedati, da se grafični prikaz v različnih modelirnih orodjih lahko nekoliko razlikuje.

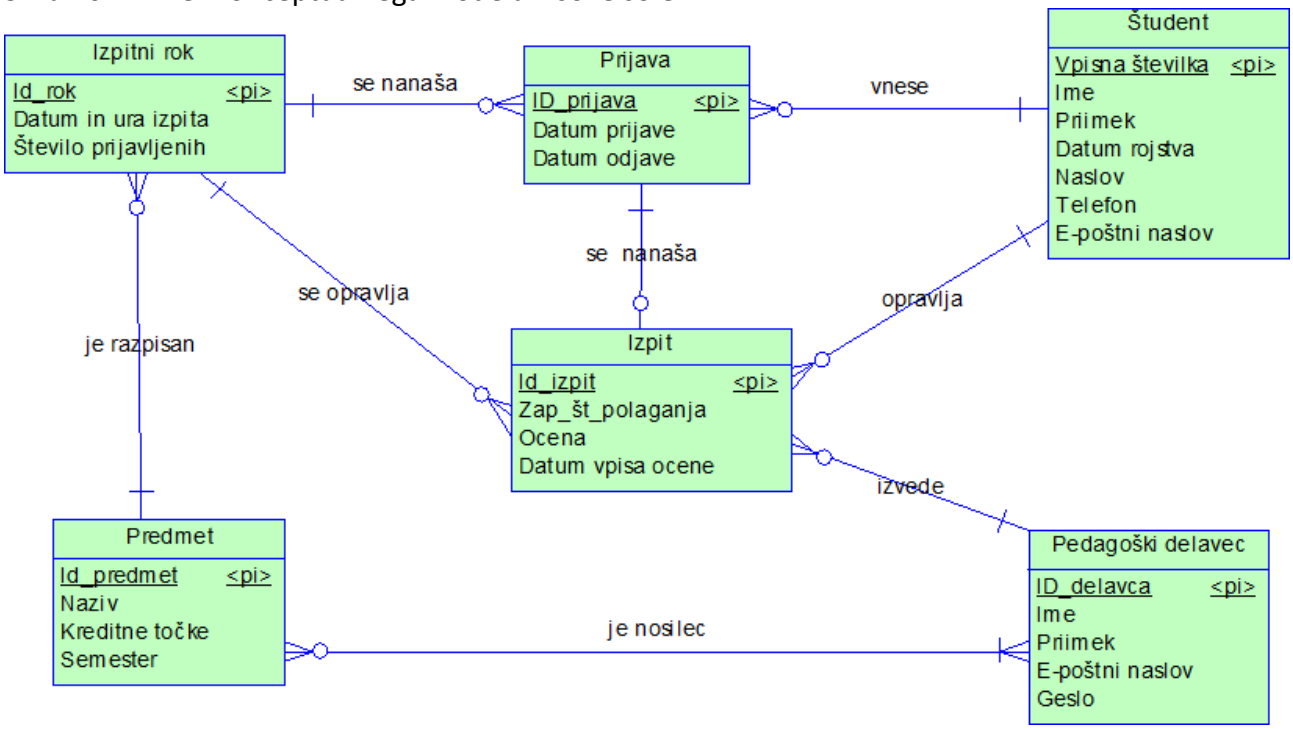
Razmerje »ima stalno prebivališče« med ŠTUDETOM in KRAJEM (Slika 13) pove, da ima vsak študent stalno prebivališče v natančno enem kraju. V obratno smer pa, da ima v določenem kraju (iz šifranta vseh krajev) stalno prebivališče nič, eden ali več študentov. Razmerje »začasno prebiva« med ŠTUDETOM in KRAJEM pove, da študent lahko nima stalnega prebivališča, lahko pa ga ima v natanko enem kraju. V obratno smer pa, da v določenem kraju (iz šifranta vseh krajev) začasno prebiva nič, eden ali več študentov. Tukaj je pod okrilje številnosti dodana tudi obveznost razmerja.

Slika 15: Primeri razmerij/povezav med entitetnimi tipi



Slika 15 prikazuje primere povezav med entitetnimi tipi v domeni visoke šole. Prvo razmerje »je« pove, da je vsak asistent pedagoški delavec. V drugo stran pa, da pedagoški delavec ni nujno asistent (lahko je učitelj). Drugo razmerje »vpiše« pove, da vsak študent vpiše enega ali več predmetov. V drugo smer pa, da je na nek predmet lahko vpisanih nič ali več študentov. Tretje razmerje »je razpisan« pove, da je za vsak predmet razpisan en ali več izpitnih rokov. V drugo smer pa, da se določen izpitni rok razpiše za natanko en predmet.

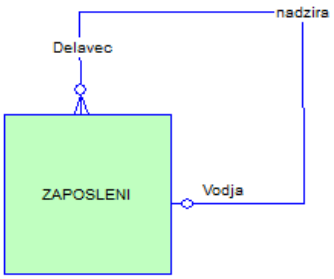
Slika 16: Primer konceptualnega modela visoke šole



Slika 16 prikazuje primer konceptualnega modela visoke šole, ki vključuje entitetne tipe ŠTUDENT, PEDAGOŠKI DELAVEC, PREDMET, IZPIT, PRIJAVA in IZPITNI ROK ter njihova medsebojna razmerja. V konceptualnem modelu lahko nastopajo tudi **večvrednostni** atributi (npr. posamezni študent ima lahko več telefonov ali E-poštnih naslovov). Prav tako lahko nastopajo **razmerja s številnostjo mnogo proti mnogo** kot je primer razmerja »je nosilec« med PREDMETOM in PEDAGOŠKIM DELAVCEM, ki pove, da je pedagoški delavec lahko nosilec nič (če gre za asistenta) ali več predmetov. V drugo stran pa, da ima predmet lahko enega ali več nosilcev. Večvrednostne atribute in razmerja mnogo proti mnogo moramo v primeru izbire relacijske podatkovne baze odpraviti v logičnem modelu.

Rekurzivno razmerje je vrsta razmerja, kjer isti entitetni tip nastopa večkrat v različnih vlogah. Pri navadnih razmerjih navadno ne označujemo vlog na vsaki strani razmerja, ampak razmerja le poimenujemo. Pri rekurzivnih razmerjih je smiselno poimenovati tudi vloge sodelujočih entitet na vsaki strani razmerja. Slika 17prikazuje primer rekurzivnega razmerja.

Slika 17: Primer rekurzivnega razmerja



Z rekurzivnim razmerjem »nadzira« modeliramo hierarhično strukturo poslovnega sistema. Vsakega zaposlenega nadzira nič ali en vodja, njegov nadrejeni (najvišji direktor nima nadrejenega, zato tudi

številnost 0). V drugo smer pa zaposleni nadzira nič ali več delavcev. Če gre za vodstvenega delavca ta nadzira svoje podrejene (teh je lahko več), v primeru da ne gre za vodstvenega delavca pa zaposleni ne nadzira nikogar (številnost 0).

Generalizacija in specializacija sta postopka urejanja entitetnih tipov v hierarhijo oziroma v odnos nadtip-podtip (Rob in Coronel, 2004,str. 150-153).

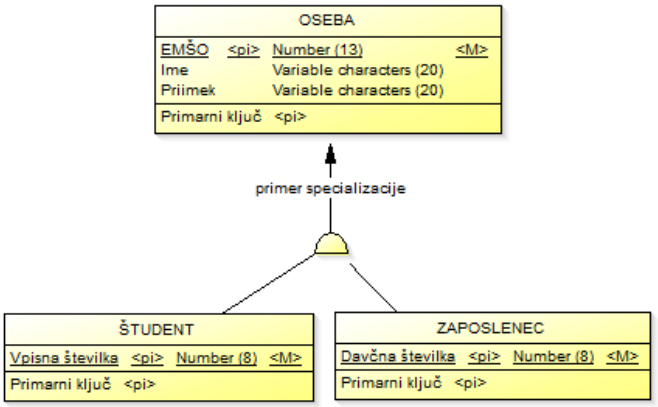
Specializacijo navadno uporabimo pri načrtovanju z vrha navzdol. Za specializacijo entitetnega tipa se navadno odločimo če:

- obstajajo entitete, o katerih želimo poleg skupnih lastnosti, hraniti tudi nekatere specifične lastnosti (specifične atribute),
- nekatere entitete nastopajo v določenih specifičnih razmerjih z drugimi entitetnimi tipi.

Primer specializacije: v enitetnem tipu OSEBA imamo atribute EMŠO, Ime, Priimek, Vpisna številka in Davčna številka. Ker ugotovimo, da Vpisne številke za zaposlene na šoli ne potrebujemo, in prav tako ne potrebujemo davčne številke študentov, se odločimo za specializacijo entitetnega tipa OSEBA. Uvedemo podtipa ŠTUDENT in ZAPOSLENEC. Skupni atributi ostanejo v nadtipu OSEBA, specifične atribute pa premaknemo v podtipa ŠTUDENT in ZAPOSLENEC kot kaže slika (Slika 18).

Če uporabljamo pristop od spodaj navzgor pa lahko najprej identificiramo podtipa ŠTUDENT (z atributi EMŠO, Ime, Priimek, Vpisna številka) in ZAPOSLENEC (EMŠO, Ime, Priimek, Davčna številka). Zatem ugotovimo, da ta dva entitetna tipa vsebujeta skupne atribute in zato izvedemo generalizacijo. To pomeni, da uvedemo nadtip OSEBA, v katerega premaknemo skupne atribute (EMŠO, Ime, Priimek).

Slika 18: Primer specializacije entitetnega tipa OSEBA



Slika 18 prikazuje primer specializacije entitetnega tipa OSEBA na podtipa ŠTUDENT in ZAPOSLENEC. V tem primeru bomo za študente dodatno hranili vpisno številko, za zaposlence pa davčno številko. Seveda za vsakega študenta kot zaposlenega hranimo podatke o EMŠO, imenu in priimku (pravimo, da podtipi dedujejo vse atribute nadtipa, v tem primeru torej entitetnega tipa OSEBA). Če gledamo isto sliko od spodaj navzgor pa rečemo, da tipa ŠTUDENT in ZAPOSLENEC generaliziramo v nadtip OSEBA.

4.3 Konceptualno načrtovanje podatkovne baze na primeru skladišča

4.3.1 Opis domene

Podjetje Hramba d.d. ima v lasti več skladišč na različnih naslovih in krajih. Skladišča so razdeljena na posamezne oštevilčene prostore. Številke prostorov so unikatne glede na posamezno skladišče, sicer se lahko ponovijo.

V podjetju so se odločili, da bodo izgradili nov informacijski sistem, ki bo omogočal izposojo prostorov v skladiščih različnim najemnikom. Najemnik najame enega (najame lahko le celoten prostor) ali več prostorov od nekega datuma naprej za določeno število dni. Isti prostor seveda lahko sposodimo ponovno, vendar moramo preveriti, da le ta ni zaseden oz., da se je zadnji najem že iztekel. Vsak prostor ima določeno kapaciteto in ceno najema na dan.

Najemnik je lahko le pravna oseba z določeno davčno številko, nazivom, naslovom in krajem. Kraje imamo zapisane v šifrantu.

4.3.2 Izdelava konceptualnega podatkovnega modela

V okviru izdelave konceptualnega podatkovnega modela:

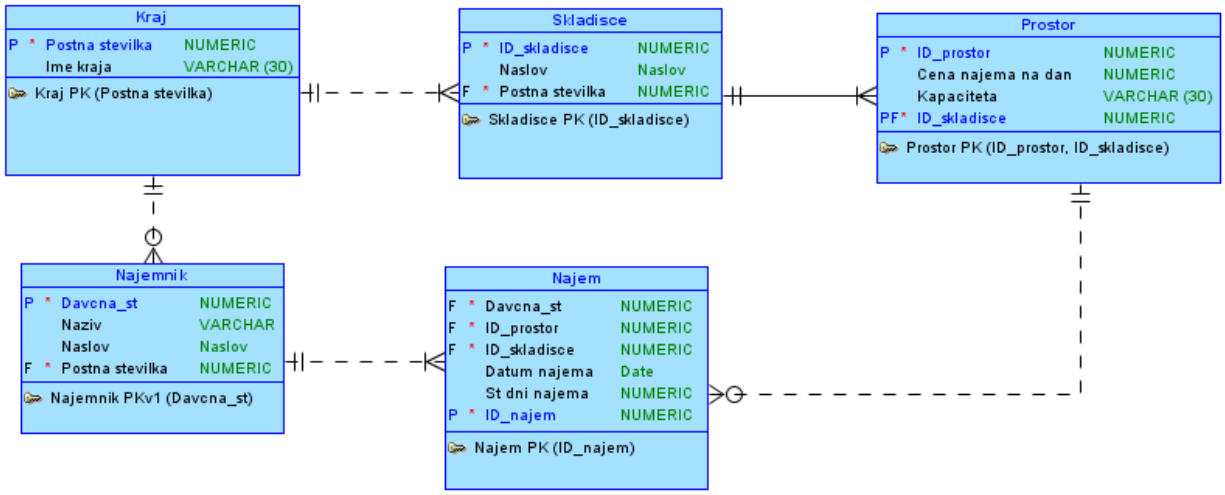
- *Identificiramo entitetne tipe,*
- *Identificiramo atribute,*
- *Atributom določimo podatkovne tipe oziroma domene,*
- *Vsakemu entitetnemu tipu določimo enolični identifikator,*
- *Entitetne tipe medsebojno povežemo z razmerji,*
- *Razmerja ustrezno poimenujemo, določimo njihove številnosti, obveznosti in odvisnosti.*
- *določimo morebitne obvezne atribute.*

Za izdelavo konceptualnega modela uporabimo orodje Oracle SQL Developer Data Modeler.

4.3.3 Konceptualni podatkovni model skladišča

Slika 19 prikazuje konceptualni model PB podjetja Hramba glede na opisano domeno oz. poslovne zahteve podjetja.

Slika 19: Konceptualni podatkovni model podjetja Hramba



4.4 Pristopi k načrtovanju podatkovne baze

Izdelava ER modela poteka v korakih, saj se s problemom, ki ga modeliramo spoznavamo postopoma. Tako najprej izdelamo grobe orise modela ter se kasneje vedno bolj spuščamo v podrobnosti. Postopek izdelave modela lahko do določene mere formaliziramo z uporabo standardnih korakov. Obstajata dva glavna pristopa k načrtovanju podatkovne baze (Mohorič, 1992, str. 67-88):

- pristop od spodaj navzgor in
- pristop z vrha navzdol.

Pri **pristopu od spodaj navzgor** začnemo z identifikacijo lastnosti oz. atributi, poiščemo njihove funkcionalne odvisnosti ter jih nato združujemo v skupine (entitetne tipe oz. relacije). Tak pristop predstavlja na primer normalizacija, ki bo opisana v poglavju 5.3. Pristop od spodaj navzgor je primeren za enostavne podatkovne baze z majhnim številom atributov.

Pri **pristopu z vrha navzdol** začnemo podatkovni model graditi tako, da najprej identificiramo le nekaj osnovnih entitetnih tipov in razmerij. Kasneje dodajamo atribute ter povezave med entitetnimi tipi. Entitetne tipe lahko nadalje razgrajujemo na podtipe, kar imenujemo specializacija. Tak pristop predstavlja izdelava diagrama Entiteta – Razmerje (ER). Pristop z vrha navzdol je primernejši za modeliranje večjih, bolj kompleksnih podatkovnih baz.

Poleg navedenih dveh pristopov poznamo še pristopa **od znotraj navzven** ter **pristop po delih**. Slednji predstavlja v praksi najbolj uporabljan pristop, saj načrtovanje razdelimo na več lažje obvladljivih delov in sicer:

- Najprej kreiramo okvirno shemo z najpomembnejšimi entitetnimi tipi in razmerji med njimi.
- Shemo razdelimo na področja (jedra so identificirani entitetni tipi).
- Za vsako področje izdelamo podmodel.
- Posamezne podmodele na koncu združimo v en podatkovni model.

Prednost pristopa po delih je vzporedno načrtovanje podmodelov posameznih področij, slabost pa, da je potrebno podmodele na koncu združiti, pri čemer lahko naletimo na določene nekonsistentnosti ali nasprotja, ki jih je potrebno razrešiti.

Vprašanja za ponavljanje

1. Kaj je konceptualni podatkovni model?
2. Katere tehnike predstavitve konceptualnega modela poznate?
3. Kateri gradniki nastopajo pri entitetnem modelu oz. diagramu?
4. Kaj je entiteta in kaj entitetni tip?
5. Kaj je atribut?
6. Kakšne vrste atributov poznate?
7. Kaj je enolični identifikator?
8. Kaj je potrebno določiti atributu?
9. Kaj pomeni obveznost atributa? Kateri atribut je vedno obvezen?
10. Kaj je razmerje?
11. Kakšne številnosti razmerij poznate? Kaj pomenijo?
12. Katere vrste pristopov načrtovanja PB poznate?
13. Kaj je značilno za pristop od spodaj navzgor? Za kakšne PB je primeren?
14. Kaj je značilno za pristop z vrha navzdol? Za kakšne PB je primeren?
15. Kaj je značilno za pristop po delih?
16. Kateri od pristopov je v praksi najbolj uporaben? Kaj je njegova prednost in kaj slabost?

Naloge

4.1 Na podlagi opisa domene smučarskih skokov izdelajte konceptualni podatkovni model. Pri tem entitetni tip SKOK modelirajte kot šibki entitetni tip. Ne pozabite:

- določiti podatkovnih tipov oziroma domen atributom,
- določiti enoličnih identifikatorjev,
- določiti morebitnih obveznih atributov,
- določiti in poimenovati razmerij, določiti njihove številnosti, obveznosti in odvisnosti.

Uporabite orodje Oracle SQL Developer Data Modeler.

Opis domene

V Planici vsako leto prirejajo tekmovanje v smučarskih skokih in poletih. O tekmovanjih zbiramo več podatkov: datum in čas začetka tekmovanja, ime tekmovanja (npr. 20. tekmovanje za

Svetovni pokal), predviden čas trajanja in na kateri skakalnici poteka (npr. 90 metrska skakalnica). Neko tekmovanje lahko poteka le na eni izmed skakalnic.

Na tekmovanje se lahko prijavijo tekmovalci, za katere moramo poznati: ime, priimek in ime države iz katere prihaja (npr. Norveška). Za vsakega tekmovalca vodimo tudi podatke o skokih, ki jih je izvedel na posameznem tekmovanju. Za vsak skok poznamo dolžino skoka v metrih (npr. 158,6), skupno oceno za slog v točkah (npr. 8,98), status skoka (npr. uspešen, razveljavljen, padec ipd.) in zaporedno številko skoka (vsak tekmovalec lahko izvede več skokov).

4.2 Na podlagi opisa domene prodajalne avtomobilov izdelajte konceptualni podatkovni model. Ne pozabite:

- določiti podatkovnih tipov oziroma domen atributom,
- določiti enoličnih identifikatorjev,
- določiti morebitnih obveznih atributov,
- določiti in poimenovati razmerij, določiti njihove številnosti, obveznosti in odvisnosti.

Uporabite orodje Oracle SQL Developer Data Modeler.

Opis domene

Podjetje TineCars d.o.o. se ukvarja s prodajo vozil različnih znamk. Za podporo svojemu poslovanju potrebuje informacijski sistem.

V prodajalni avtomobilov o avtomobilu želijo hraniti še naslednje podatke: model, letnik, št. motorja, št.šasije. Dodatno hranijo opise vse dodatne opreme, ki je na voljo.

Cena avtomobila je sestavljena iz cene osnovnega modela ter cene izbrane dodatne opreme. Dodatno na končno ceno lahko vplivajo tudi različne akcije. Za akcijo zabeležimo njeno ime, opis, trajanje in popust, ki ga prinaša.

Pri nakupu si stranka izbere avto določene znamke in modela, nato pa še vso želeno dodatno opremo. Cena brez DDV je tako vsota cene osnovnega vozila in cen vseh izbranih artiklov dodatne opreme. Znesek popusta je odvisen od morebitne akcije za določen model avtomobila. Zatem se izračunata končna cena brez DDV in končna cena z DDV (komentirajte ali boste/ne boste hranili ta dva atributa).

Zabeležimo tudi prodajalca, ki je avto prodal, da mu bo delodajalec lahko izplačal morebitni dodatek za delovno uspešnost.

5 Relacijska podatkovna baza

Za konceptualnim načrtovanjem nastopi logično načrtovanje podatkovne baze. **Osnova logičnega modela je jezik, ki je razumljiv ciljnemu SUPB.** Če izberemo relacijski SUPB, potem govorimo o **relacijskem modelu**.

5.1 Relacijska teorija

Temelje relacijske teorije, na kateri slonijo vse današnje relacijske podatkovne baze, je podal E.F. Codd leta 1970 v članku z naslovom "A Relational Model of Data for Large Shared Data Banks". V njem je podal množico pravil in principov za upravljanje podatkov in jih strnil v **relacijski model**. Ideja se je hitro širila v informacijskih krogih in kmalu je postala predmet proučevanj strokovnjakov po univerzah in v industriji. Pomenil je revolucijo na področju podatkovnih baz in je hitro nadomesti starejše modele (hierarhičnega in mrežnega). Je zelo enostaven za razumevanje in tako tudi neizkušeni uporabniki lahko hitro razumejo vsebino podatkovne baze. Njegova prednost so tudi enostavni, vendar močni jeziki za poizvedovanje po vsebini podatkovne baze (QBE in SQL).

Relacijska teorija podaja temelje danes najbolj razširjenemu relacijskemu podatkovnemu modelu. Osnovna koncepta, ki ju je moč srečati v relacijski teoriji sta **relacija** in **domena**.

5.1.1 Relacija

Po definiciji je relacija r podmnožica kartezijskega produkta domen:

$$r \subset D1 \times D2 \times \ldots \times Dn$$

oziroma množica urejenih n-teric

$$r \equiv \{t1, t2, \ldots tm\},$$

pri čemer je vsaka n-terica sestavljena iz komponent

$$ti \equiv (ki1, ki2, \ldots kin), \quad 1 <= i <= m,$$

ki so elementi domen: $k_{ij} \in D_j$, $1 <= i <= m$, $1 <= j <= n$.

Končno število m določa moč ali številnost relacije, število domen n pa stopnjo relacije (Mohorič, 1992, str. 110).

Relacije istega tipa so v sistemih za upravljanje podatkovnih baz prikazane v obliki dvodimenzionalnih tabel. Vrstica tabele predstavlja posamezno relacijo, medtem ko predstavlja stolpec domeno oziroma atribut. Atribut A_i je v relacijski teoriji definiran kot preslikava množice objektov O v domeno D_i: A_i: $O \rightarrow D_i$. V primeru, ko je relacija funkcija (kar je ena od zahtev v postopku

normalizacije) se lahko relacijo definira tudi s pomočjo preslikav kot množico n-teric (Mohorič, 1992, str. 112).:

$$\mathbf{r \equiv \{(A1(o),A2(o), ...An(o)): o \in O\}}$$

5.1.2 Relacijska shema

Vsaki relaciji pripada natanko ena **relacijska shema**. Relacijska shema predstavlja semantiko oziroma pomen relacije. Relacijsko shemo sestavlja oznaka sheme R ter lista atributov Ai s pripadajočimi oznakami domen Di.

$$\mathbf{R\ (A1: D1,\ A2: D2,\ ...,\ An: Dn)}$$

Primer relacijske sheme:
Oseba(Ime: I, Starost: C, Teža: C) pri čemer domeni I in C obsegata

Domena, ki obsega imena: I ≡ {Tine, Meta, Jure, Ana}

Domena, ki obsega interval celih števil: C ≡ 1, 2,... 200

5.1.3 Funkcionalne odvisnosti

Naslednji pomemben element relacijske teorije so odvisnosti, med njimi najbolj **funkcionalna odvisnost**, s pomočjo katere so definirani tudi ključi relacije.

V relacijski shemi R velja $X \rightarrow Y$ (podmnožica atributov X funkcionalno določa podmnožico atributov Y), če v nobeni relaciji, ki pripada shemi R, ne moreta obstajati dve n-terici, ki bi se ujemali v vrednosti atributov X in se ne bi ujemali v vrednosti atributov Y.

Podmnožica atributov X pa je ključ relacijske sheme R v primeru, ko $X \rightarrow A_1A_2..A_n$ in ne obstaja X', ki bi bil prava podmnožica X in bi prav tako funkcionalno določal vse atribute relacijske sheme.

Relacijska shema lahko vsebuje več ključev (kandidatov za glavni ključ), med katerimi se izbere glavni ključ, ostalim pa preostane vloga nadomestnih ključev. V relacijski teoriji ima pomembno mesto tudi zunanji ali povezovalni ključ, ki predstavlja temelj za vzpostavitev povezav med relacijami (Mohorič, 1992, str. 116).

Vprašanja za ponavljanje

1. Kdo in kdaj je postavil temelje relacijske teorije?
2. Kaj je relacija?
3. Kaj predstavlja relacijska shema? Kako je sestavljena?
4. Kaj pomeni, da v relacijski shemi R X funkcionalno določa Y ($X \rightarrow Y$)?
5. Kdaj je podmnožica atributov X ključ relacijske sheme?
6. Ali relacijska shema lahko vsebuje več ključev? Kaj storimo v tem primeru?

5.2 Logično načrtovanje

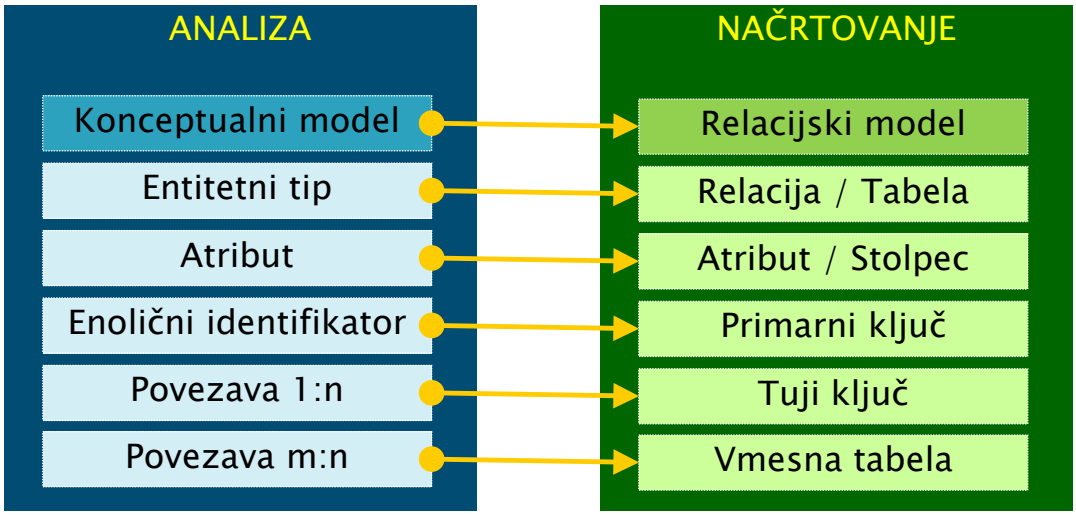
Logično načrtovanje je proces izdelave logičnega podatkovnega modela za podatke specifične domene, ki sledi konceptualnemu načrtovanju. Logični model temelji na specifikah izbrane vrste logičnega modela (npr. mrežni, relacijski, objektni), vendar je neodvisen od specifičnih značilnosti posameznega SUPB (npr. relacijske baze ORACLE) in drugih tehničnih karakteristik računalniškega sistema. **V poglavju obravnavamo logično načrtovanje relacijskega podatkovnega modela.**

Podatkovna baza, ki temelji na relacijskem modelu, je predstavljena z množico relacij, kjer je vsaka relacija tabela z vrsticami in stolpci. Prehod iz konceptualnega v logični model je navadno avtomatiziran s strani CASE orodij. Načelno obstaja tudi metodologija načrtovanja direktno logične podatkovne baze, vendar je zelo priporočljivo začeti z načrtovanjem na konceptualnem nivoju.

5.2.1 Transformacija konceptualnega modela v relacijski model

Slika 20 prikazuje, kako se pri prehodu iz konceptualnega na logični nivo transformirajo posamezni gradniki konceptualnega modela. Sam konceptualni model se v primeru izbire relacijskega modela sedaj imenuje **relacijski model**. Namesto o entitetnih tipih, govorimo o **relacijah** ali kar **tabelah** podatkovne baze. Vsako relacijo oz. tabelo sestavljajo atributi, ki jih v tabeli imenujemo stolpci. Namesto o enoličnem identifikatorju, govorimo o **primarnem ključu** relacije oz. tabele. Vse povezave konceptualnega modela s številnostjo ena proti mnogo pomenijo kreiranje **tujega ključa** v eni izmed povezanih tabel. V primeru povezav s številnostjo mnogo proti mnogo pa se kreira **vmesna tabela**.

Slika 20: Transformacije pri prehodu s konceptualnega na relacijski logični model

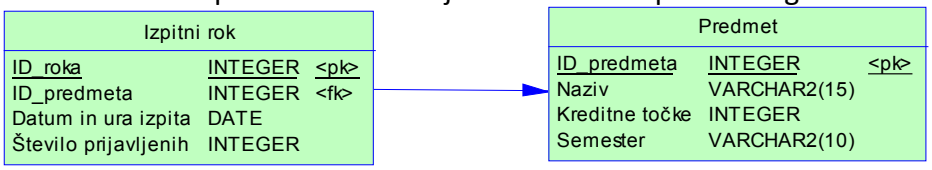


Relacijo si lahko predstavljamo kot dvodimenzionalno tabelo s stolpci in vrsticami (velja za logično strukturo podatkovne baze in ne za fizično). Atribut je poimenovani stolpec relacije. Domena je množica dovoljenih vrednosti enega ali več atributov, ki so vključeni v to domeno.

Lastnosti relacije:

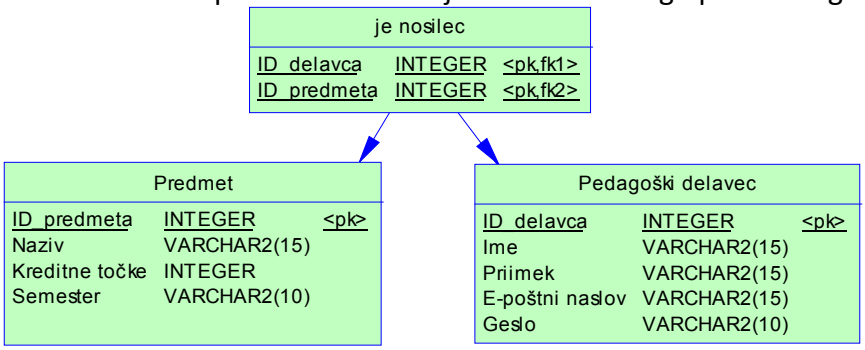
- Ime relacije je enolično. V relacijski shemi podatkovne baze ni dveh relacij z enakim imenom.
- Vsaka celica tabele, ki predstavlja relacijo, vsebuje natančno eno atomarno vrednost.
- Vsak atribut relacije ima enolično ime. V isti relaciji ni dveh atributov, ki bi imela isto ime.
- Vrednosti nekega atributa so vse iz iste domene.
- Vsaka n-terica relacije je enolična → v relaciji ni dveh enakih n-teric.
- Vrstni red atributov v relaciji je nepomemben.
- Vrstni red n-teric v relaciji je nepomemben.

Slika 21: Primer preslikave razmerja številnosti ena proti mnogo



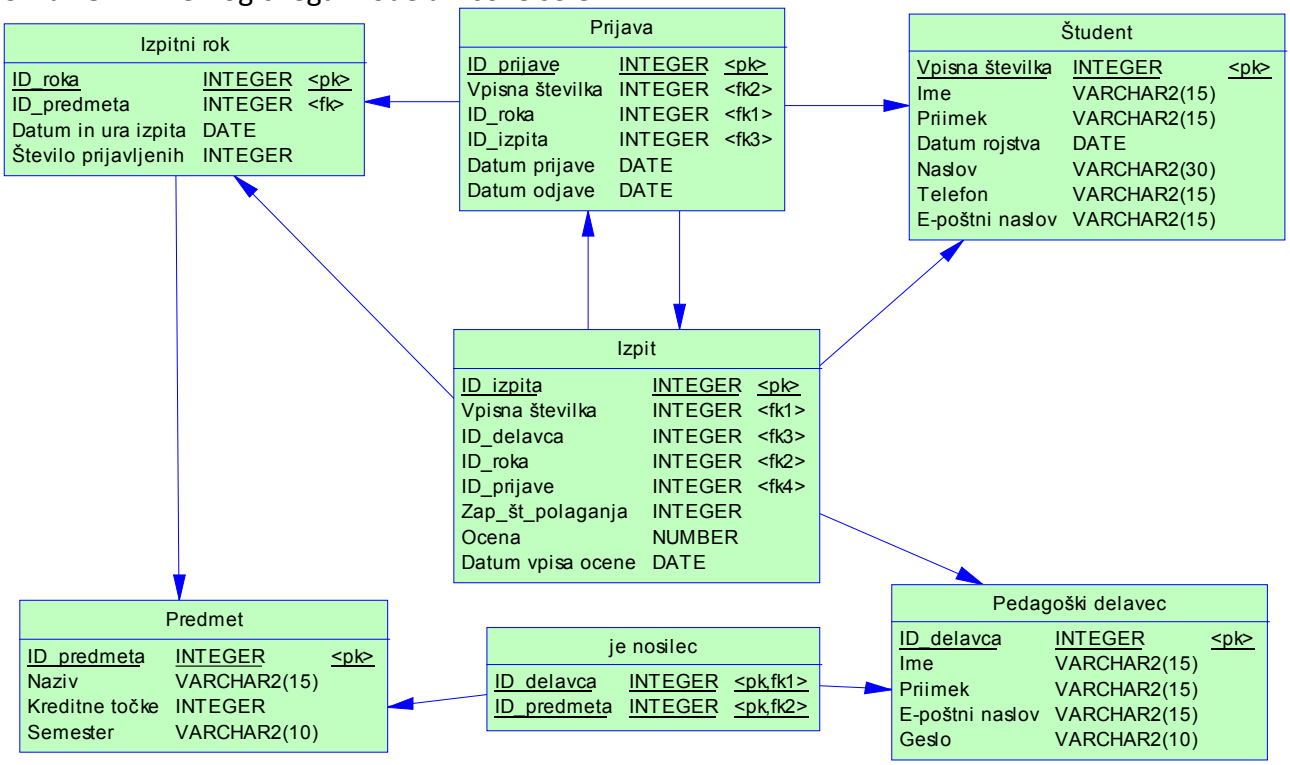
Slika 21 prikazuje izsek preslikave konceptualnega modela s slike (Slika 16) in sicer je bila povezava »je razpisan« med PREDMETOM in IZPITNIM ROKOM številnosti ena proti mnogo. Dejstvo, da je izpitni rok razpisan za natanko določen predmet, zabeležimo s pomočjo tujega ključa ID_predmeta (oznaka <fk> pomeni »forigen key«) v tabeli IZPITNI ROK, ki se pri preslikavi na logični nivo generira v tej tabeli. Atributi, ki so tvorili primarni identifikator na konceptualnem nivoju, sedaj postanejo primarni ključi in so označeni z oznako <pk>, ki pomeni »primary key«. Takšna sta atributa ID_predmeta v tabeli PREDMET in ID_roka v tabeli IZPITNI ROK.

Slika 22: Primer preslikave razmerja številnosti mnogo proti mnogo



Slika 22 prikazuje izsek preslikave konceptualnega modela s slike (Slika 16) in sicer je bila povezava »je nosilec« med PREDMETOM in PEDAGOŠKIM DELAVCEM številnosti mnogo proti mnogo. Zato je v skladu s pravili za preslikave, predstavljenimi s sliko (Slika 20), nastala vmesna tabela za imenom JE NOSILEC. Tabela ima dva tuja ključa ID_delavca (ki je primarni ključ tabele PEDAGOŠKI DELAVEC) in ID_predmeta (ki je primarni ključ tabele PREDMET). Tuja ključa sta označena z oznako <fk>, ki pomeni »foreign key«. S tem povezuje obe tabeli med seboj. Tuja ključa ID_delavca in ID_predmeta pa skupaj tvorita primarni ključ novo nastale tabele JE NOSILEC. Atributi, ki tvorijo primarni ključ, so označeni z oznako <pk>, ki pomeni »primary key«.

Slika 23: Primer logičnega modela visoke šole



Slika 23 prikazuje logični podatkovni model visoke šole, ki je nastal z avtomatsko preslikavo konceptualnega modela s slike (Slika 16) z uporabo CASE orodja. Pri preslikavi je bil izbran relacijski SUPB Oracle 10g. Tukaj velja opozoriti, da CASE orodja pogosto popolnoma ne sledijo teoretični delitvi na konceptualni, logični in fizični model. Tako gre v primeru modela s slike teoretično že za mešanico logičnega in fizičnega modela, saj je potrebno pred transformacijo izbrati konkreten SUPB (v našem primeru Oracle 10g) in ne le vrsto modela, relacijski model. Izdelavo pravega, od SUPB neodvisnega logičnega modela, pa orodje ne omogoča.

5.2.2 Omejitve nad podatkovno bazo

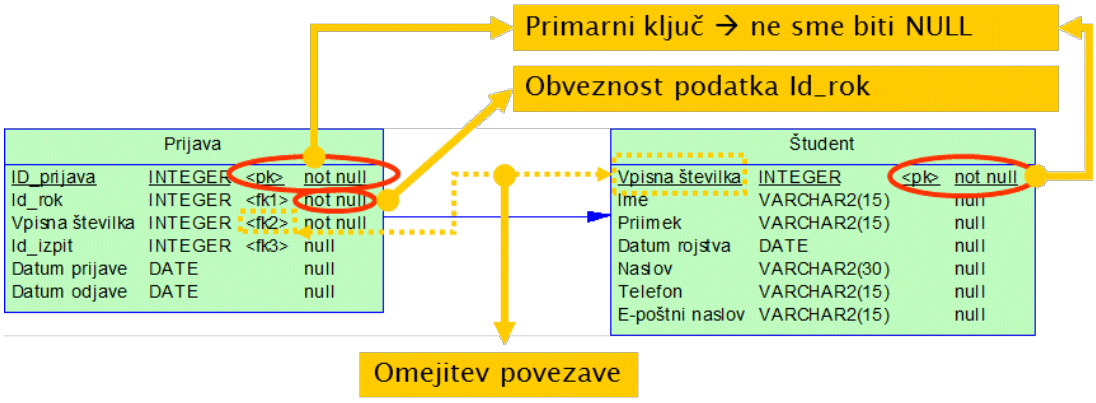
Za celovitost ter skladnost podatkov v podatkovni bazi skrbimo s pomočjo omejitev (ang. constraints). Omejitve v splošnem zagotavljajo smiselno vsebino podatkov – njihovo integriteto. Poznamo več vrst omejitev (Connolly in Begg, 2010, str. 452-453):

- **Obvezni podatki** (ang. required data): za določene atribute predpišemo, da ne morejo biti brez vrednosti (NOT NULL). Brez vrednosti nikoli ne morejo biti atributi, ki so del ključa. Tudi za katerikoli drug atribut lahko predpišemo, da ne sme biti Null.
- **Omejitve domene** (ang. domain constraints): povedo kakšne vrednosti so pri posameznem atributu dovoljene. To pomeni, da v PB ne bo možno vnesti vrednosti izven zaloge vrednosti domene.
- **Števnost** (ang. multiplicity): predstavlja število entitet entitetnega tipa, ki so v razmerju z entitetami drugega tipa, glede na pomen razmerja. Tako omejitev števnosti (Slika 23) med tipom IZPITNI ROK in PREDMET določa, da se izpitni rok razpiše za natanko en predmet (v tipu IZPITNI ROK imamo tuj ključ ID_predmeta, ki pove, za kateri predmet je posamezen rok razpisan).
- **Pravila za celovitost podatkov** (ang. integrity constraints) **delimo v dve skupini:**

- **Celovitost entitet** (ang. entity integrity): v osnovni relaciji ne sme biti noben atribut, ki je del primarnega ključa, enak Null (brez vrednosti) (Slika 24).
- **Celovitost povezav** (ang. referential integrity): če v relaciji obstajajo tuji ključi, potem morajo njihove vrednosti ustrezati tistim, ki so v obliki primarnega ključa zapisane v eni izmed n-teric neke druge ali iste relacije (Slika 24) ali pa mora biti tuji ključ v celoti enak Null.

- **Splošne omejitve** (ang. general constraints): dodatna pravila, ki jih določi uporabnik ali skrbnik podatkovne baze, ki definirajo ali omejujejo nek vidik področja, za katerega je narejena podatkovna baza.

Slika 24: Primeri integritetnih omejitev

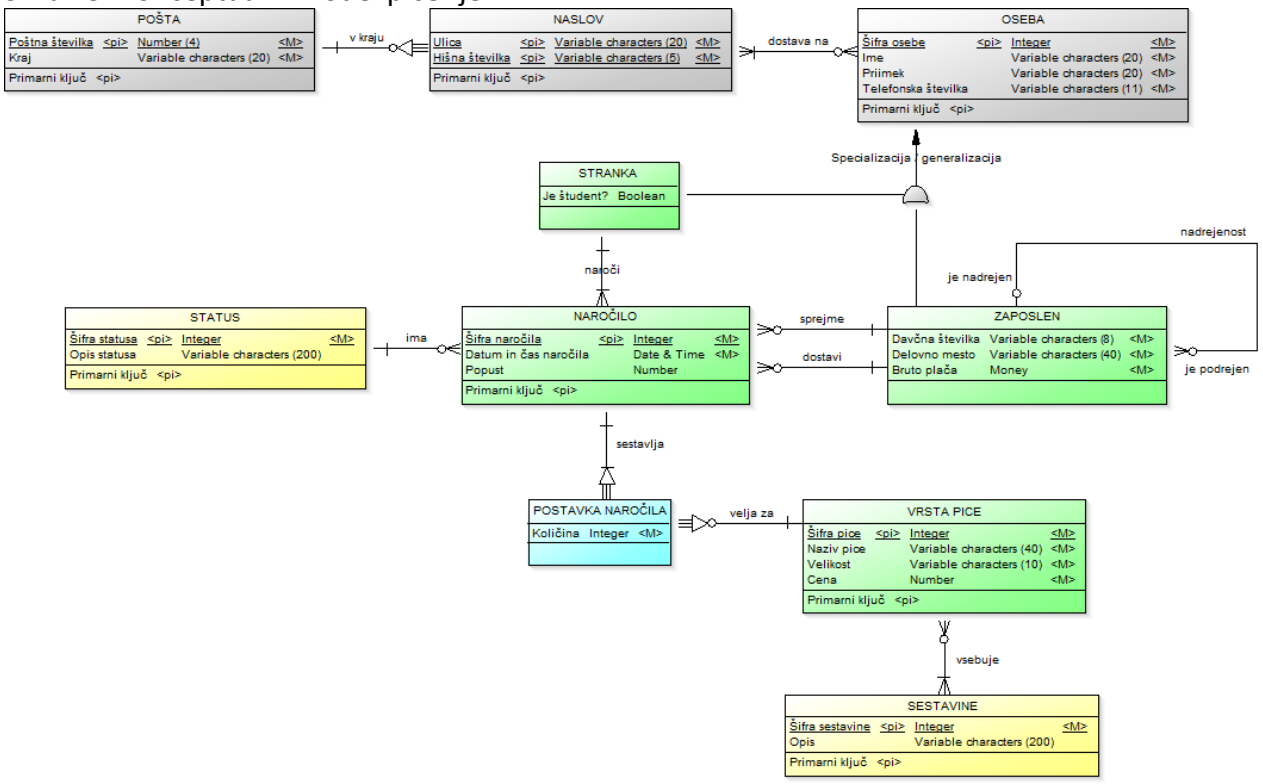


Slika 24 prikazuje primere integritetnih omejitev in sicer omejitev primarnega ključa na atributu ID-prijava, omejitev tujega ključa na atributu Vpisna številka v tipu PRIJAVA in obveznost podatkov (pri atributu ID_prijava tipa PRIJAVA in atributu Vpisna številka tipa ŠTUDENT, ki predstavljata primarna ključa ter pri ID_rok in Vpisna številka v tipu PRIJAVA, ki predstavljata tuja ključa).

5.2.3 Logično načrtovanje podatkovne baze na primeru picerije

Želimo izdelati načrt podatkovne baze, ki bo omogočal beleženje poslovanja picerije. Hraniti želimo podatke o strankah in zaposlenih v piceriji. Nadalje potrebujemo podatke o vrstah pic in vseh njihovih sestavinah. Zabeležiti si želimo vsa naročila. Za vsako naročilo želimo zabeležiti zaposlenega, ki naročilo sprejel in tistega, ki je naročilo dostavil. Za vsako naročilo je potrebno zabeležiti tudi, katere vse pice so bile naročene in v kakšnih količinah. Najprej izdelamo konceptualni podatkovni model, ko prikazuje slika (Slika 25).

Slika 25: Konceptualni model picerije



Zatem konceptualni model preslikamo v logični model. Pri tem uporabimo pravila za preslikave, podana na sliki (Slika 20). **Pazimo na:**

- **preslikavo razmerja entitetnega tipa ZAPOSLEN samega s seboj**. To razmerje modelira podrejenost/nadrejenost zaposlenih.
- **na dvojno povezavo med tipoma ZAPOSLEN in NAROCILO**. Z razmerjem »sprejme« modeliramo, kdo je naročilo sprejel, z razmerjem »dostavi«, pa kateri zaposleni naročilo dostavi.
- entitetna tipa NASLOV in POSTAVKA_NAROČILA sta modelirana kot **šibka entitetna tipa**.
- **povezave številnosti m:n** med tipoma NASLOV in OSEBA (oseba ima lahko več naslovov, na istem naslovu je lahko več oseb) ter VRSTA_PICE in SESTAVINE (pica ima več sestavin, ista vrsta sestavine npr. šunka se lahko uporabi pri izdelavi več različnih vrst pic).

Ob upoštevanju vseh pravil transformacije konceptualnega v logični nivo dobimo relacijski logični podatkovni model picerije, z naslednjimi relacijami. Pri tem so primarni ključi posameznih relacij podčrtani, z oznako # pa so označeni tuji ključi.

Relacijske sheme modela picerije

POSTA (Poštna_številka, Kraj)
NASLOV(Ulica, Hišna_številka, #Poštna_številka)
OSEBA (Šifra_osebe, Ime, Priimek, Telefonska_številka)
DOSTAVA (#Ulica,#Hišna_številka, #Poštna_številkat, #Šifra osebe) // nova tabela med NASLOV in OSEBA zaradi povezave m:n
STRANKA (#Šifra_osebe, Student)

STATUS (Šifra_statusa, Opis)
ZAPOSLEN (Davčna_številka, Delovno_mesto, Bruto_plača, #Šifra_osebe, #Šifra_šefa)
NAROCILO (Šifra_naročila, Datum_in_čas_naročila, Popust, #Šifra_stranke, #Šifra_statusa, #Šifra_sprejme, #Šifra_dostavi)
POSTAVKA_NAROCILA (Količina, #Šifra_naročila, #Šifra_pice)
VRSTA_PICE (Šifra_pice, Naziv_pice, Velikost, Cena)
SESTAVINE (Šifra_sestavine, Opis)
VSEBUJE (#Šifra_pice, #Šifra_sestavine)

Vprašanja za ponavljanje

1. Kaj je logično načrtovanje in kaj je njegov rezultat?
2. Kaj je pri prehodu iz konceptualnega na logični nivo potrebno določiti?
3. Ali je logični model odvisen ali neodvisen od specifik konkretnega SUPB?
4. Kako se pri prehodu iz konceptualnega na logični nivo transformirajo: entitetni tipi, atributi, enolični identifikator, povezave 1:n, povezave m:n?
5. Kaj zagotavljajo omejitve nad podatkovno bazo?
6. Katere vrste omejitev poznaš?
7. Kateri atributi morajo obvezno imeti integritetno omejitev NOT NULL in kaj ta pomeni?
8. Katerim atributom še dodatno lahko pripišemo integritetno omejitev NOT NULL?
9. Kaj pomeni celovitost povezav oz. referenčna integriteta povezav?
10. Kaj zagotavlja omejitev domene?

Naloge

5.2.1 Konceptualni model iz naloge 4.1 (domena smučarskih skokov) preslikajte v logični model.

- Zapišite relacijske sheme.
- Model preslikajte s pomočjo orodja Oracle SQL Developer Data Modeler.

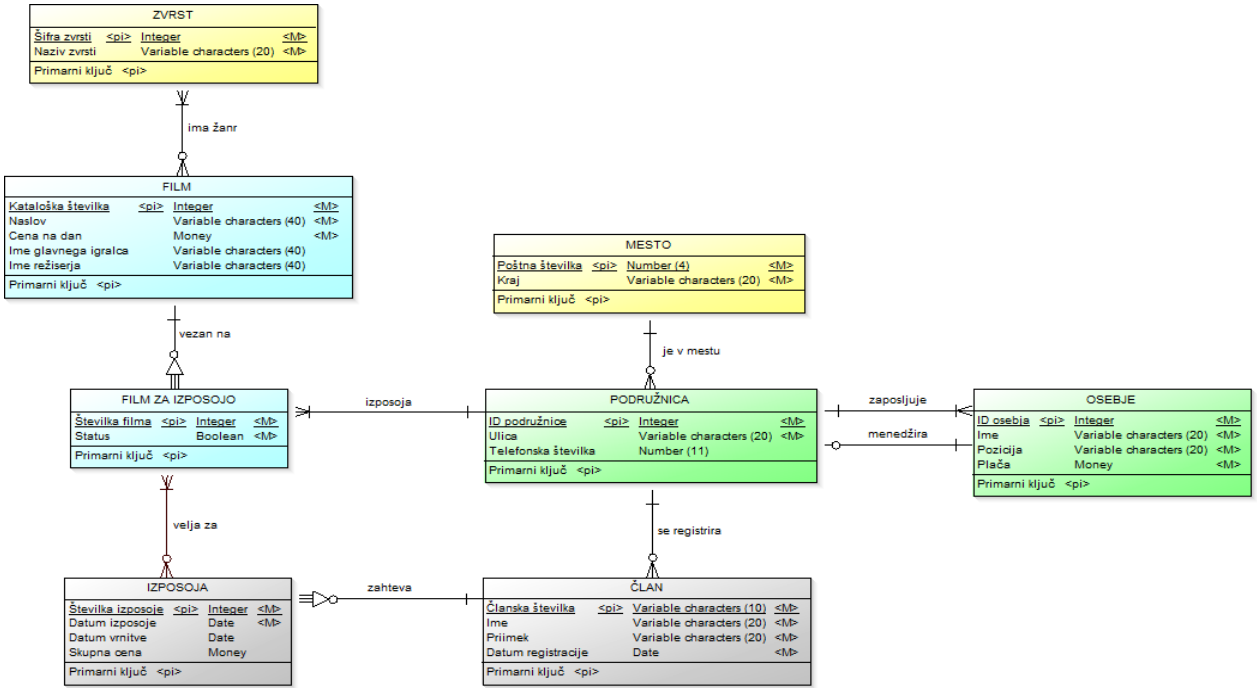
5.2.2 Konceptualni model iz naloge 4.2 (domena prodajalna avtomobilov) preslikajte v logični model.

- Zapišite relacijske sheme.
- Model preslikajte s pomočjo orodja Oracle SQL Developer Data Modeler.

5.2.3 Podani konceptualni model videoteke preslikajte v logični model.

- Zapišite relacijske sheme. Pazite pri entitetnih tipih FILM ZA IZPOSOJO in IZPOSOJA, ki sta modelirana kot šibka entitetna tipa. Podčrtajte primarne ključe, z oznako # pa označite tuje ključe. Pazite na dvojno povezavo med entitetnima tipoma PODRUŽNICA in OSEBJE.
- Model preslikajte tudi s pomočjo orodja Oracle SQL Developer Data Modeler.

Slika 26: Konceptualni model videoteke



5.3 Normalizacija

Grad in Jaklič (1996, str. 101) definirata normalizacijo kot analizo funkcionalnih odvisnosti med svojstvi (atributi, podatkovnimi elementi). Gre za postopek sestavljen iz več korakov, v katerem se kompleksen pogled uporabnikov prevede v množico preprostih in stabilnih podatkovnih struktur.

Rezultat začetne analize podatkovnih potreb je množica nenormaliziranih relacij, za katero je značilna nekonsistentnost in podvajanje podatkov. Vse te pomanjkljivosti je potrebno odpraviti pred kreiranjem fizične podatkovne baze, saj v nasprotnem primeru ni mogoče pričakovati optimalnega delovanja na podatkovni bazi temelječega informacijskega sistema. Osnovni cilj načrtovanja relacijske podatkovne baze je grupirati atribute v relacije tako, da bo čim manj redundance med podatki, saj relacije, ki vsebujejo odvečne podatke, lahko povzročajo ažurne anomalije. Poznamo več vrst ažurnih anomalij:

- anomalije pri dodajanju n-teric v relacijo,
- anomalije pri brisanju n-teric iz relacije in
- anomalije pri spreminjanju n-teric.

Normalizacija je postopek pregleda in preoblikovanja relacij oz. tabel v obliko, pri kateri bo podvajanja podatkov čim manj in ne bo prihajalo do ažurnih anomalij (Rob in Coronel, 2004,str. 184).

Potencialne koristi pravilnega načrtovanja so:

- Spremembe podatkov v podatkovni bazi dosežemo z minimalnim številom operacij → večja učinkovitost; manj možnosti za podatkovne nekonsistentnosti.
- Manjše potrebe po diskovnih kapacitetah za shranjevanje osnovnih relacij → manjši stroški.

Relacija se lahko nahaja v eni izmed naslednjih normalnih oblik (Grad in Jaklič, 1996, str. 101-113, Finkelstein, 1992, str. 61-73, Mohorič, 1992, str. 25-46, Rob in Coronel, 2004,str. 184-221):

- V **prvi normalni obliki**, če ne vsebuje ponavljajočih se vrednosti podatkov (relacija je v tem primeru predstavljena z dvodimenzionalno tabelo).
- V **drugi normalni obliki**, če je v prvi in ne vsebuje nobene delne odvisnosti med glavnim ključem in atributi, ki niso del primarnega ključa.
- V **tretji normalni obliki**, če je v drugi in ne vsebuje tranzitivnih odvisnosti med atributi, ki niso del primarnega ključa.
- V **četrti normalni obliki**, če je v tretji normalni obliki in njeni atributi niso odvisni zgolj od primarnega ključa, temveč tudi od njegovih vrednosti. Z drugimi besedami: iz relacije se odpravijo vsi atributi, ki v njej nastopajo zgolj pogojno.
- V **peti normalni obliki**, če je v četrti in ne vsebuje povratnih odvisnosti. Tovrstni problemi se rešujejo z vpeljavo posebne relacije z nazivom "struktura".

Cilj postopka normalizacije je vzpostavitev takega relacijskega podatkovnega modela, kjer bodo vse relacije vsaj v tretji normalni obliki. V nadaljevanju na primerih obravnavamo prve štiri normalne oblike. Višja normalna oblika pomeni boljšo strukturiranost podatkov v podatkovni bazi in njihovo manjšo redundanco. Po drugi strani pa to pomeni, da je potrebno pri dostopu do podatkov dostopati do večjega števila tabel, ki jih je potrebno povezati z operacijo stika, kar posledično pomeni daljše odzivne čase na poizvedbe uporabnikov. Zato se na račun višje učinkovitosti delovanja podatkovne baze včasih odrečemo višjim normalnim oblikam, npr. 3NO in 4NO . V poglavju 5.4.7 zato spoznamo tudi postopek **denormalizacije**, ki pomeni preoblikovanje relacij iz višje normalne oblike v nižjo (npr. relacije, ki so v 3NO preoblikujemo v 2NO).

5.3.1 Vrste ažurnih anomalij

Do ažurnih anomalij lahko pride pri dodajanju novih vrstic tabelo, pri brisanju vrstic iz tabele in pri spreminjanju podatkov v primerih, ko tabela ni normalizirana. Oglejmo si primere ažurnih anomalij na primeru domene picerije. Imejmo nenormalizirano tabelo Pica, ki hrane podatke o posamezni vrsti pice in njenih sestavinah.

Pica (<u>sifra_pice</u>, naziv_pice, velikost_pice, cena_pice, <u>sifra_sestavine</u>, opis_sestavine.

Vidimo, da se podatki o nazivu pice, šifrah sestavin in opis sestavin ponavlja. Oglejmo si, do kakšnih težav prihaja pri ažuriranju podatkov takšne nenormalizirane tabele.

Tabela 1: Nenormalizirana tabela Pica

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice	Sifra_ sestavine	Opis_ sestavine
15	Vražja	velika	8 €	12	tabasko
15	Vražja	velika	8 €	37	feferoni
17	Vražja	majhna	6 €	15	šunka
18	Morska	majhna	7 €	3	tuna
19	Morska	velika	9 €	12	tabasko

5.3.1.1 Dodajanje zapisov

Oglejmo si možne primere težav v primeru, da želimo v tabelo dodati novo pico. Recimo, da manjka vražja pica srednje velikosti. Njena cena je 7 €. Sestavine te pice so: šunka, tabasko in feferoni, kot vsake vražje pice. Zato, da bi to pico dodali, je potrebno dodati tri nove vrstice, čeprav so naziv pice, in vse sestavine že v podatkovni bazi. Pri tem je seveda potrebno paziti, da se vneseni podatki skladajo s podatki v že obstoječih vrsticah tabele, saj v nasprotnem primeru pride do nekonsistentnosti podatkovne baze. Primer ustrezno ažurirane tabele Pica prikazuje Tabela 2. **Vidimo tudi, da se število podatkov v podatkovni bazi iz 30 poveča na 48. Vpisali smo namreč 18 novih podatkov, od katerih se jih velika večina ponavlja.**

Tabela 2: Ažurirana tabela Pica

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice	Sifra_ sestavine	Opis_ sestavine
15	Vražja	velika	8 €	12	tabasko
15	Vražja	velika	8 €	37	feferoni
17	Vražja	majhna	6 €	15	šunka
18	Morska	majhna	7 €	3	tuna
19	Morska	velika	9 €	12	Tabasko
16	Vražja	srednja	7 €	12	tabasko
16	Vražja	srednja	7 €	37	feferoni
16	Vražja	srednja	7 €	15	šunka

Relacijo Pica normaliziramo tako, da relacijo Pica razbijemo na več relacij. Ker se ista sestavina lahko pojavi na različnih picah, najprej uvedemo tabelo Sestavina, ki predstavlja šifrant vseh sestavin, ki jih picerija uporablja. V tej tabeli je vsaka sestavina zapisana natanko enkrat, kar je bistvena prednost pred prejšnjim primerom, ko je bila zapisana tolikokrat, na kolikor picah je nastopala. Ker ista sestavina lahko nastopa na več različnih vrstah pic (npr. tabasko je sestavina vražje in morske pice) ter ima vsaka vrsta pice tudi več različnih sestavin (npr. vražjo pico sestavljajo šunka, tabasko, feferoni) uvedemo vmesno tabelo Sestavina_na_pici. V osnovni relaciji Pica pa ostanejo preostali atributi.

Dobimo tri relacije:

Sestavina (sifra_sestavine, opis_sestavine)
Sestavina_na_pici (#*sifra_pice, #sifra_sestavine*)
Pica (sifra_pice, naziv_pice, velikost_pice, cena_pice)

Tabela 3: Podatki picerije iz tabele 1 v normalizirani bazi

Sifra_ sestavine	Opis_ sestavine
12	tabasko
37	feferoni
15	šunka
3	tuna

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice
15	Vražja	velika	8 €
17	Vražja	majhna	6 €
18	Morska	majhna	7 €
19	Morska	velika	9 €

Sifra_pice	Sifra_sestavine
15	12
15	37
17	15
18	3
19	12

Sedaj v normalizirano bazo dodamo vražjo pico srednje velikosti (sifra_pice=16), katere cena je 7 € in ima sestavine tabasko, feferoni, šunka. V tem primeru dodajamo v tabelo Pica in sicer dodamo le eno novo vrstico: novo šifro, naziv, velikost in ceno. Sestavine vražje pice so že vnesene in jih ni potrebno ponovno vnašati. Vnesemo še povezave med siframi sestavin in sifro pice v tabelo Sestavina_na_pici. Potrebno je dodati toliko vrstic, koliko sestavin ima pica (v našem primeru torej 3). **Vidimo da smo sedaj vpisali 10 novih podatkov, v primerjavi s predhodnim primerom, kjer jih je bilo 18. Tako vidimo, da je normalizacija pomembna tudi z vidika manjše porabe prostora za hranjenje.**

Tabela 4: Dodajanje vražje srednje pice v normalizirano bazo

Sifra_ sestavine	Opis_ sestavine
12	tabasko
37	feferoni
15	šunka
3	tuna

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice
15	Vražja	velika	8 €
17	Vražja	majhna	6 €
18	Morska	majhna	7 €
19	Morska	velika	9 €
16	Vražja	srednja	7€

Sifra_pice	Sifra_sestavine
15	12
15	37
17	15
18	3
19	12
16	12
16	37
16	15

5.3.1.2 Brisanje zapisov

V primeru, da iz začetne nenormalizirane tabele Pica zbrišemo majhno vražjo pico izgubimo podatke o sestavini 15, šunki, saj ta sestavina nastopa samo pri tej pici. Ko bomo vnašali novo pico s to sestavino, bo sestavino potrebno ponovno vnesti, kar z vidika učinkovitosti uporabnika ni dobro.

Tabela 5: Ažurne anomalije pri brisanju podatkov tabele Pica

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice	Sifra_ sestavine	Opis_ sestavine
15	Vražja	velika	8 €	12	tabasko
15	Vražja	velika	8 €	37	feferoni
17	Vražja	majhna	6 €	15	šunka
18	Morska	majhna	7 €	3	tuna
19	Morska	velika	9 €	12	tabasko

V primeru, da iz začetne normalizirane tabele Pica zbrišemo majhno vražjo pico, podatki o sestavini šunka v tabeli Sestavina ostane. Brišemo še vrstice v tabeli Sestavina_na_pici, in sicer sestavine male vražje pice (ta ima v tabeli le sestavino šunka s šifro 15).

Tabela 6: Brisanje male vražje pice iz normalizirane baze

Sifra_ sestavine	Opis_ sestavine
12	tabasko
37	feferoni
15	šunka
3	tuna

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice
15	Vražja	velika	8 €
17	Vražja	majhna	6 €
18	Morska	majhna	7 €
19	Morska	velika	9 €

Sifra_pice	Sifra_sestavine
15	12
15	37
17	15
18	3
19	12

5.3.1.3 Spreminjanje zapisov

Če želimo v nenormalizirani tabeli spremeniti opis sestavine, npr. tabasko v tabasco, moramo to storiti tolikokrat kot imamo različnih pic s to sestavino v tabeli Pica (v konkretnem primeru moramo narediti dve spremembi). V nasprotnem primeru podatkovna baza postane nekonsistentna.

Tabela 7: Ažurne anomalije pri spreminjanju podatkov tabele Pica

Sifra_ pice	Naziv_ pice	Velikost_ pice	Cena_ pice	Sifra_ sestavine	Opis_ sestavine
15	Vražja	velika	8 €	12	tabasko
15	Vražja	velika	8 €	37	feferoni
17	Vražja	majhna	6 €	15	šunka
18	Morska	majhna	7 €	3	tuna
19	Morska	velika	9 €	12	tabasko

Če želimo isto spremembo narediti v normalizirani bazi, je potrebno popraviti **natanko en podatek v tabeli Sestavina.**

Tabela 8: Spreminjanje opisa sestavine v normalizirani bazi

Sifra_ sestavine	Opis_ sestavine
12	tabasko
37	feferoni
15	šunka
3	tuna

5.3.2 Prva normalna oblika

Pravilo prve normalne oblike zahteva identifikacijo in odstranitev ponavljajočih skupin. Ključ tako dobljene relacije sestavlja ključ osnovne relacije (katere del je bila prvotno ponavljajoča skupina) in ključ ponavljajoče skupine. Poglejmo primer nenormalizirane relacije Voznik. Relacijo Voznik bomo normalizirali do 3. normalne oblike.

Podana je nenormalizirana relacija:

***Prekrsek** (Ime, Priimek, St_dovoljenja, Posta, Kraj,(Datum_in_ura, Znesek, St_tock))*

Relacija Voznik hrani podatke o prekrških voznikov. O vsakem vozniku želimo hraniti njegove osebne podatke, o prekršku pa datum in uro prekrška, znesek kazni in število kazenskih točk. Ker vsak voznik lahko naredi več prekrškov, moramo imeti možnosti za vsakega zabeležiti vse njegove prekrške, kar je v relaciji podano s ponavljajočo skupino atributov o prekršku znotraj dodatnih oklepajev.

Da je relacija v prvi normalni obliki morajo biti izpolnjeni naslednji pogoji:

- **Nima ponavljajočih skupin** (atributi niso večvrednostni).
- Ima opredeljene **funkcionalne odvisnosti** in **primarni ključ**.

Koraki normalizacije v 1. NO:

1. Odpravimo ponavljajoče skupine.
2. Določimo funkcionalne odvisnosti.
3. Določimo primarni ključ.

1. **Odpravimo ponavljajoče skupine:**
   Prekrsek (Ime, Priimek, St_dovoljenja, Posta, Kraj, Datum_in_ura, Znesek, St_tock)

2. **Določimo funkcionalne odvisnosti:**
   St_dovoljenja->(Ime, Priimek, Posta, Kraj)
   Posta-> Kraj
   (St_dovoljenja, Datum_in_ura)->(Znesek, St_tock)

3. **Določimo primarni ključ:**
   St_dovoljenja, Datum_in_ura

Imamo relacijo v 1. NO z določenim primarnim ključem:
Prekrsek (Ime, Priimek, St_dovoljenja, Posta, Kraj, Datum_in_ura, Znesek, St_tock)

5.3.3 Druga normalna oblika

Da je relacija v drugi normalni obliki morajo biti izpolnjeni naslednji pogoji:

- **Relacija je v 1. NO.**
- **Ne vsebuje parcialnih odvisnosti (odvisnosti le od dela primarnega ključa).** To pomeni, da noben atribut, ki ni del ključa, ni funkcionalno odvisen le od dela primarnega ključa, temveč od celotnega ključa.

Če ima relacija n atributov in je **primarni ključ** sestavljen iz **1**, **n-1** ali **n atributov**, je relacija v 2.NO.

1. **Atribute, le delno odvisne od primarnega ključa, prenesemo v novo relacijo. Na podlagi funk. odvisnosti** *St_dovoljenja* $\rightarrow$ *(Ime, Priimek, Posta, Kraj)* **kreiramo novo relacijo Voznik.**
   Voznik (St_dovoljenja, Ime, Priimek, Posta, Kraj)

2. **V relaciji Prekrsek tako ostanejo le atributi, odvisni od celotnega ključa:**
   Prekrsek (#St_dovoljenja, Datum_in_ura, Znesek, St_tock)

Sedaj tako imamo dve relaciji, ki sta obe v 2. NO.

5.3.4 Tretja normalna oblika

Da je relacija v tretji normalni obliki morajo biti izpolnjeni naslednji pogoji:

- **Relacija je v 2. NO.**
- **Ni tranzitivnih funkcionalnih odvisnosti.** To pomeni, da med atributi, ki niso del primarnega ključa ni funkcionalnih odvisnosti.

Če ima relacija n atributov in je **primarni ključ** sestavljen iz **n-1** ali **n atributov**.

1. **Ugotovimo, da znotraj relacije Voznik obstaja tranzitivna odvisnost med atributoma, ki nista del primarnega ključa. Torej upoštevamo funk. odvisnost** *Posta-> Kraj* ***in dobimo naslednji relaciji:***
   Voznik (St_dovoljenja, Ime, Priimek, #Posta)
   Kraj (Posta, Kraj)

2. **Imamo še relacijo, ki je že v 3.NO, saj ne vsebuje tranzitivnih odvisnosti:**
   Prekrsek (#St_dovoljenja, Datum_in_ura, Znesek, St_tock)

S tem je postopek normalizacije tega primera končan. Iz začetne relacije smo dobili tri relacije (Voznik, Kraj, Prekrsek), ki so vse v 3. NO.

5.3.5 Četrta poslovna normalna oblika

Obravnavali bomo **četrto poslovno normalno obliko.** Da je relacija v četrti poslovni normalni obliki, morajo biti izpolnjeni naslednji pogoji:

- **Relacija je v 3. NO.**
- **Atributi so odvisni od primarnega ključa in od vrednosti ključa.**
- **Neobvezen prenesen atribut iz druge relacije, ki je v celoti odvisen od ključa je obvezen.**

Razširimo primer hranjenja podatkov o voznikih tako, da želimo za poklicne voznike hraniti tudi naziv podjetja, kjer so zaposleni, za preostale voznike pa npr. obstoječe kazenske točke.

Imejmo relacijo Voznik z naslednjimi atributi:

Voznik *(St_dovoljenja, Ime, Priimek, #Posta, Podjetje, Obstojece_tocke)*

1. **Ugotovimo, da sta atributa Podjetje in Obstojece_tocke odvisna le od ključa in od vrednosti, zato ju izločimo v dve novi relaciji:**
   Poklicni_voznik (#St_dovoljenja, Podjetje)
   Zasebni_voznik (#St_dovoljenja, Obstojece_tocke)

2. **V začetni relaciji nam tako ostanejo naslednji atributi:**
   Voznik (St_dovoljenja, Ime, Priimek, #Posta)

Navedene tri relacije (Voznik, Poklicni_voznik, Zasebni_voznik) so tako v 4.NO.

Včasih zavestno uporabljamo relacije, ki ne ustrezajo najvišjim normalnim oblikam. Prve in druge normalne oblike nikoli ne kršimo. Višjim normalnim oblikam se včasih odrečemo na račun doseganja boljše učinkovitosti dela s podatkovno bazo.

5.3.6 Normalizacija relacije z uporabo podatkov nenormalizirane tabele

Tabela prikazuje, kdaj so pacienti naročeni pri zobozdravniku. Pacient je naročen na določen dan ob določenem času pri zobozdravniku, ki se nahaja v določenem oddelku. Številka zaposlenega (Št. Zap) enolično določa podatke o zobozdravniku, številka pacienta (Št. Pac.) pa podatke o pacientih (ime, priimek). Vsak dan je zobozdravnik dodeljen enemu oddelku za cel dan.

Tabela 9: Zapisi v nenormalizirani relaciji

Št. Zap.	Ime zobozdravnika	Št. Pac.	Ime in priimek pac.	Datum obiska	Čas obiska	Številka oddelka
S1011	Tone Kovač	P100	Janez Novak	12.3.2010	10.00	S15
S1011	Tone Kovač	P105	Ivan Horvat	12.3.2010	12.00	S15
S1024	Helena Hribar	P108	Tomaž Senica	12.3.2010	10.00	S10
S1024	Helena Hribar	P108	Tomaž Senica	14.3.2010	14.00	S10
S1032	Robert Plevnik	P105	Ivan Horvat	14.3.2010	16.30	S15
S1032	Robert Plevnik	P110	Matej Lorgar	15.3.2010	18.00	S13

Na podlagi opisa domene in podatkov prikazanih v tabeli:

- Identificiraj funkcionalne odvisnosti.
- Identificiraj primarne ključe relacij in tuje ključe.
- Izvedi postopek normalizacije v 3NO.
- Skušaj podati tudi primere anomalij pri vstavljanju, brisanju in posodabljanju podatkov.

Funkcionalne odvisnosti:

fo1: (Št. Zap., Datum obiska, Čas obiska)-> Št. Pac., Ime in priimek pac.
fo2: Št. Zap.-> Ime zobozdravnika
fo3: Št. Pac.-> Ime in priimek pac., Številka oddelka
fo4: Št. Zap., Datum obiska)-> Številka oddelka
fo5: (Datum obiska, Čas obiska, Št. Pac)-> Št.Zap, Ime zobozdravnika

Normalizirane relacije:

ObiskPriZobozdravniku (#Št. Zap., Datum obiska, Čas obiska, #Št. Pac.)
Pacient (Št. Pac., Ime in priimek pac.)
Zobozdravnik(Št. Zap., Ime zobozdravnika)
Zaposleni_Oddelek(#Št. Zap., Datum obiska, Številka oddelka)

Vprašanja za ponavljanje

1. Kaj je normalizacija?
2. Zakaj moramo relacije normalizirati?
3. Katere normalne oblike poznaš?
4. Katerih normalnih oblik nikoli ne kršimo?
5. Kdaj je relacija v prvi normalni obliki?
6. Kdaj je relacija v drugi normalni obliki?
7. Kdaj je relacija v tretji normalni obliki?
8. Kdaj je relacija v četrti poslovni normalni obliki?
9. Zakaj včasih ne uporabljamo relacij v najvišjih normalnih oblikah?

Naloge

5.3.1 Določi funkcionalne odvisnosti med atributi in normaliziraj relacijo Najem do 3. NO

Podana je relacija:
Najem (Št_najemnika, Ime_najemnika, (Št_nepr, Naslov_nepr, Datum_z, Datum_k, Cena, Št_lastnika, Ime_lastnika))

Pomen sheme je naslednji:
Izbiramo podatke o najemanju nepremičnin. Termin začetka (Datum_z) in prenehanja najema (Datum_k) je natančno določen s številko najemnika (Št_najemnika) in številko nepremičnine (Št_nepr). Številka najemnika pri tem enolično določa ime najemnika (Ime_najemnika). Vsaka nepremičnina ima svojo identifikacijsko številko (Št_nepr), ki določa naslov nepremičnine (Naslov_nepr) in kdo je njen lastnik (Št_lastnika in Ime_lastnika) ter ceno najema (Cena). Pri tem ima vsak lastnik svojo identifikacijsko številko.

5.3.2 Določi funkcionalne odvisnosti med atributi in normaliziraj relacijo P do 3. NO

Podana je relacija:
P(ŠifraKaseta, ŠifraFilm, NaslovFilm, RežijaFilm, DolžinaFilm, (EMŠO, ImeStranka, UlicaStranka, PoštaStranka, KrajStranka, ČasIzposoje))

Pomen sheme je naslednji:
V videoteki hranijo več videokaset. Vsako kaseto vodijo pod svojo šifro (*ŠifraKaseta*). Na vsaki kaseti je lahko posnet le en film, ki ga določa šifra (*ŠifraFilm*), seveda pa je lahko isti film (z enako šifro) posnet tudi na več različnih kasetah. Poleg šifre za vsak film poznamo še njegov naslov (*NaslovFilm*), ime režiserja (*RežijaFilm*) in dolžino (*DolžinaFilm*). Kasete si izposojajo stranke o katerih hranimo naslednje podatke: EMŠO (*EMŠO*), ime (*ImeStranka*) ter naslov prebivališča (*UlicaStranka*, *PoštaStranka*, *KrajStranka*). Ko si stranka izposodi kaseto, zabeležimo čas

izposoje (ČasIzposoje). Seveda si lahko isti medij izposodi tudi več strank ali pa si ga ista stranka izposodi večkrat, vendar le ob različnih časih izposoje.

5.3.3 Določi funkcionalne odvisnosti med atributi in normaliziraj relacijo R do 3. NO

Podana je relacija:
R(DavčnaŠt, Ime, Priimek, Ulica, PoštnaŠt, Kraj, (ŠifraIzdelka, ImeIzdelka, ŠifraKategorije, ImeKategorije, Cena, Kolicina, DatumČasNakupa)).

Pomen sheme je naslednji:
Neka oseba (DavčnaŠt) ima ime in priimek ter stanuje na naslovu (Ulica) v kraju (PoštnaŠt, Kraj). Oseba v spletni prodajalni kupi izdelke (ŠifraIzdelka) v določeni količini (Količina). Zabeležimo tudi datum in čas nakupa (DatumČas), saj lahko ista oseba večkrat kupi isti izdelek. Šifra izdelka določa ime izdelka (ImeIzdelka), določa pa tudi njegovo ceno (Cena). Vsak izdelek je uvrščen v določeno kategorijo izdelkov npr. oblačila, tehnično blago, ki ima poleg šifre (ŠifraKategorije) tudi svoje ime (ImeKategorije).

5.4 Fizično načrtovanje

Fizično načrtovanje je tretji, zadnji korak pri načrtovanju podatkovne baze. Fizično načrtovanje je proces izdelave opisa implementacije podatkovne baze na zunanjem pomnilnem mediju. Obsega opis relacij, datotečnih organizacij za relacije in indekse, integritetnih omejitev, varnostnih ukrepov in oceno velikosti podatkovne baze (Connolly in Begg, 2010, str. 473).

Če smo se pri logičnem načrtovanju še ukvarjali z vprašanjem KAJ potrebujemo, se pri fizičnem osredotočimo na vprašanje KAKO bomo podatkovno bazo implementirali. Zato so za fizično načrtovanje potrebni ljudje s specifičnimi znanji glede na izbrani SUPB (npr. Oracle 10g). Seveda pa proces fizičnega načrtovanja ne poteka ločeno od logičnega načrtovanja, saj morajo biti določene odločitve, sprejete v okviru fizičnega načrtovanja za izboljšanje učinkovitosti delovanja podatkovne baze, zabeležene tudi na logičnem modelu (npr. združitev določenih relacij).

5.4.1 Izdelava SQL skripte

Prvi korak je pretvorba logičnega modela v jezik za ciljni SUPB. V primeru relacijskega logičnega modela gre za **izdelavo skripte**, ki vsebuje stavke jezika SQL za kreiranje podatkovne baze (DDL stavki za točno določen SUPB). Za vsako relacijo definiramo naziv relacije, listo atributov, primarni ključ ter tuje ključe in omejitve povezav. Pri tem nam je CASE orodje zopet v veliko pomoč, saj najprej preveri sintaktično pravilnost logičnega modela ter nam nato avtomatsko generira skripto (Slika 27). Skripto potem v ciljnem SUPB poženemo in baza se skreira. Druga manj ugodna možnost je ročno kreiranje elementov podatkovne baze, bodisi preko vmesnika SUPB ali z ročnim poganjanjem posameznih SQL stavkov.

Slika 27: Primer skripte v jeziku SQL za kreiranje relacijske podatkovne baze za Oracle 10g

```
/*==============================================================*/
/* Index: "je nosilec2_FK"                                      */
/*==============================================================*/
create index "je nosilec2_FK" on "je nosilec1" (
   "Id_predmet" ASC
);

/*==============================================================*/
/* Table: "Student"                                             */
/*==============================================================*/
create table "Student"  (
   "Vpisna številka"    INTEGER                          not null,
   "Ime"                VARCHAR2(15),
   "Priimek"            VARCHAR2(15),
   "Datum rojstva"      DATE,
   "Naslov"             VARCHAR2(30),
   "Telefon"            VARCHAR2(15),
   "E-poštni naslov"    VARCHAR2(15),
   constraint PK_ŠTUDENT primary key ("Vpisna številka")
);

alter table "Izpit"
   add constraint FK_IZPIT_IZVEDE_PEDAGOŠK foreign key ("ID_delavca")
      references "Pedagoški delavec" ("ID_delavca");

alter table "Izpit"
   add constraint FK_IZPIT_OPRAVLJA_ŠTUDENT foreign key ("Vpisna številka")
      references "Student" ("Vpisna številka");

alter table "Izpit"
   add constraint "FK_IZPIT_SE  NANAŠ_PRIJAVA" foreign key ("ID_prijava")
      references "Prijava" ("ID_prijava");
```

5.4.2 Datotečne organizacije

V okviru fizičnega načrtovanja je potrebno izdelati tudi načrt datotečne organizacije. Potrebno je namreč izbrati optimalno datotečno organizacijo za shranjevanje osnovnih relacij ter različne vrste indeksov za doseganje čim boljše odzivnosti podatkovne baze. Načrtovalec mora zato dobro poznati, kakšne datotečne strukture in organizacije SUPB podpira ter kako deluje, saj med SUPB-ji obstajajo velike razlike. Prvi korak je tako analiza transakcij, ki se bodo v PB izvajale. Ključnega pomena pri tem so tudi uporabniške zahteve v zvezi z želeno/pričakovano učinkovitostjo transakcij.

V splošnem poznamo naslednje **datotečne organizacije** (za relacije in indekse) (Connolly in Begg, 2010, str. 484-485):

- Kopica (Heap),
- Razpršena (Hash),
- Metoda indeksiranega zaporednega dostopa (Indexed Sequential Access Method - ISAM),
- Drevo B+-,
- Gruča (Cluster) in še nekatere druge.

Bralec se lahko podrobneje spozna z značilnostmi navedenih datotečnih organizacij in indeksov, njihovimi prednostmi in slabostmi v Connolly in Begg, 2005, dodatek C, str. 1268-1292 ali v Mohorič, 1992, str. 31-96.

Načeloma lahko za vsako relacijo izberemo zanjo najprimernejšo datotečno organizacijo glede na vrste transakcij, katere se bodo nad njo izvajale. Pri tem smo v praksi žal omejeni, saj nekateri SUPB-ji ne podpirajo vseh navedenih datotečnih organizacij.

5.4.3 Indeksiranje

Naslednji pomemben korak fizičnega načrtovanja je **izbira indeksov**. Navadno generiramo indekse za vse stolpce, ki predstavljajo primarne in tuje ključe. Namen dodatnih indeksov (tako imenovanih sekundarnih indeksov) je povečanje učinkovitosti dela s PB. Dodatne indekse navadno dodamo na stolpce, po katerih bodo uporabniki pogosto iskali podatke ali tam, kjer se tabele med seboj povezujejo, npr. kateri nastopajo v pogojih za selekcijo ali stik: ORDER BY; GROUP BY ali v drugih operacijah, ki vključujejo sortiranje (npr. UNION ali DISTINCT). V primeru table ŠTUDENT bi lahko sekundarni indeks dodali npr. na stolpec Priimek, če ocenimo, da bodo uporabniki po njem pogosto iskali.

Po drugi strani pa ni primerno indeksirati atributov, ki se pogosto spreminjajo ali ki so sestavljeni iz dolgih nizov.

Slika 28: Primer skripte v jeziku SQL za kreiranje indeksov za Oracle 10g

```
create unique index "Študent_PK" on "Študent" ("Vpisna številka" ASC);

create index "Po_priimku" on "Študent" ("Priimek" ASC);
```

Slika 28 prikazuje SQL ukaza za generiranje indeksov. Prvi SQL stavek kreira indeks po stolpcu Vpisna številka, ki predstavlja primarni ključ tabele ŠTUDENT, za to mora biti indeks unikaten (tipa unique)**.** Drugi ukaz kreira dodatni indeks na stolpcu Priimek. V tem primeru ne gre za unikaten indeks.

5.4.4 Analiza transakcij

Da bi fizično načrtovanje kar najuspešneje izvedli, je potrebno analizirati transakcije, ki se bodo izvajale nad podatkovno bazo. Pri tem nas zanima (Connolly in Begg, 2010, str. 479):

- katere transakcije se bodo izvajale zelo pogosto in zato odločilno vplivale na hitrost delovanja PB,
- katere transakcije so kritične z vidika poslovanja,
- v katerih obdobjih znotraj delovnega tedna/dneva bo obremenitev PB največja (ang. peak load).

Z analizo transakcij ugotovimo morebitna ozka grla pri delovanju PB. Rezultati analize predstavljajo podlago za različne odločitve, npr. za izbiro datotečne organizacije in indeksiranja, morebitne združitve nekaterih relacij. Pogosto ne moremo analizirati vseh transakcij, zato se osredotočimo na najpogostejše transakcije (pravilo 80/20). Za analizo lahko uporabimo različne tehnike, npr. matriko transakcija/relacija (Slika 29). V matriki za vsako relacijo prikažemo, katere transakcije do nje dostopajo in za kakšno vrsto dostopa gre (vstavljanje, branje, spreminjanje, brisanje).

Slika 29: Matrika med relacijami in transakcijami

Transaction/ Relation	(A)				(B)				(C)				(D)				(E)				(F)			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch										X				X								X		
Telephone																								
Staff		X				X				X								X				X		
Manager																								
PrivateOwner	X																							
BusinessOwner	X																							
PropertyForRent	X					X	X	X						X				X				X		
Viewing																								
Client																								
Registration																								
Lease																								
Newspaper																								
Advert																								

I = Insert; R = Read; U = Update; D = Delete

Vir: Connolly in Begg, 2010, str. 481.

5.4.5 Ocena velikosti podatkovne baze

Naslednji pomemben korak načrtovanja je tudi **ocena velikosti podatkovne baze**, saj moramo vedeti, koliko prostora na disku bomo zanjo potrebovali. Ocena je odvisna od velikosti posameznega zapisa in števila zapisov ter števila indeksov. Koristna je tudi ocena, koliko zapisov bo v povprečju dodanih na mesec ali leto, da lahko ocenimo tudi potencialno rast PB in prostor, ki ga bomo potrebovali čez leto ali dve.

5.4.6 Varnost podatkovne baze

Ker podatkovna baza predstavlja zelo pomemben informacijski vir vsakega poslovnega sistema, morajo biti v fazi zajema zahtev ustrezno zajete tudi **varnostne zahteve**. V tem koraku pa se je potrebno odločiti, kako bodo realizirane. Ker se SUPB tudi v varnostnih vidikih razlikujejo, je zelo pomembno, da načrtovalec dobro pozna ciljni SUPB. V splošnem relacijski SUPB omogočajo dve vrsti varnosti PB: sistemsko in podatkovno.

Sistemska varnost pokriva dostop in uporabo PB na sistemskem nivoju, npr. jo ščiti z uporabniškim imenom in geslom. Podatkovna varnost ščiti odstop in uporabo posameznih objektov (npr. tabel, pogledov) ter določa akcije, ki jih uporabniki lahko nad temi objekti izvajajo (pregled, ažuriranje, brisanje). Za določanje pravic dostopa SQL jezik pozna ukaza **GRANT** (dodamo pravice) in **REVOKE** (pravice odvzamemo).

5.4.7 Denormalizacija

Normalizacija je postopek, s katerim razvrstimo atribute v relacije na podlagi njihovih funkcionalnih odvisnosti. Rezultat je množica normaliziranih relacij, kar imenujemo logični načrt podatkovne baze. Postopek normalizacije smo obravnavali v poglavju 5.3. Normaliziran logični načrt zagotavlja

minimalno možno redundanco podatkov, vendar pa pogosto ne omogoča najboljše odzivnosti podatkovne baze. Včasih se je tako potrebno odreči višjim normalnim oblikam na račun doseganja boljše učinkovitosti delovanja oziroma hitrejše odzivnosti podatkovne baze. Včasih zavestno uporabljamo relacije, ki ne ustrezajo najvišjim normalnim formam, vendar prve in druge normalne oblike nikoli ne kršimo.

Boljšo učinkovitost tak lahko dosežemo s postopkom denormalizacije, ki pa mora biti nadzorovana. Normalizirane sheme v nekaterih primerih namreč predstavljajo oviro za učinkovitejšo implementacijo programov.

Če je odzivnost podatkovne baze slaba in ugotovimo, da podatkov določene relacije uporabniki ne ažurirajo pogosto, pogosto pa se jih bere oziroma po njih poizveduje, je denormalizacija lahko ključ do izboljšane odzivnosti. V okviru denormalizacije lahko izvedemo različna preoblikovanja. Tako lahko npr. določene relacije združimo (npr. gremo iz 4. poslovne normalne oblike nazaj na 3. NO), kar pomeni manj relacij ter posledično zmanjšano število operacij stika pri izvedbi poizvedb.

5.4.7.1 Primer denormalizacije iz 3. v 2. NO

Imejmo relaciji Oseba in Kraj z naslednjima relacijskima shemama:

Oseba (IDO, Ime, Priimek, Naslov, #Postna_st)

Kraj (Postna_st, Kraj).

Relaciji sta v 3. normalni obliki. Če želimo dostopati do vseh podatkov o osebi, torej imenu, priimku, naslovu, poštni številki in kraju, je pri poizvedbi potrebno izvesti operacijo stika med relacijama, kar pa je z vidika učinkovitosti slabše kot če bi imeli vse podatke v eni relaciji. Če torej želimo izboljšati učinkovitost delovanja podatkovne baze, se lahko odločimo, da bomo relaciji denormalizirali nazaj v 2. normalno obliko, saj se nazivi krajev in poštne številke redko spreminjajo. Tako dobimo naslednjo relacijo:

Oseba (IDO, Ime, Priimek, Naslov, Postna_st, Kraj)

5.4.7.2 Primer denormalizacije iz 4. PNO v 3. NO

Imejmo naslednje tri relacije študentskega informacijskega sistema in sicer Oseba ter njeni specializaciji Student in Pedagog, ki so v 4. poslovni normalni obliki (4. PNO):

Oseba (IDO, Ime, Priimek, Naslov, Postna_st),

Student (IDO,letnik, smer),

Pedagog(IDO, datum_zaposlitve, st_otrok).

Podobno kot prej nastopi težava, če želimo dostopati do vseh podatkov o študentu ali pedagogu, saj je zopet potrebno pri poizvedbi potrebno izvesti operacijo stika med relacijama, kar pa je z vidika učinkovitosti slabše kot če bi imeli vse podatke v eni relaciji. Tako se lahko odločimo, da navedene

relacije denormaliziramo v 3. NO, pri čemer bomo imeli pri študentih v poljih datum_zaposlitve in st_otrok vedno null vrednosti. Podobno bomo vedno imeli null vrednosti pri poljih letnik in smer za pedagoge. Dobimo relacijo v 3.NO:

Oseba (IDO, Ime, Priimek, Naslov, Postna_st, letnik, smer, datum_zaposlitve, st_otrok).

5.4.7.3 Primer denormalizacije iz 3. NO v 2. NO

Imejmo relacijo, v kateri hranimo rezultate smučarskega tekmovanja. V relaciji Rezultat želimo zabeležiti rezultat prvega in drugega teka in tako dobimo relacijo Rezultat, ki je v 3. NO:

Rezultat (IDR, Cas_Prvi_Tek, Cas_Drugi_Tek, #IDTekmovalec).

Pri nadaljnji obdelavi podatkov, npr. kreiranju vrstnega reda tekmovalcev na tekmi, pa večinoma potrebujemo seštevek obeh časov, torej skupni čas. Ker vsakokratno računanje skupnega časa kot vsote časa prvega teka in časa drugega teka pomeni slabšo učinkovitost, se odločimo za denormalizacijo. Dodamo atribut Skupni_cas, ki ga izračunamo enkrat ter zapišemo v podatkovno bazo. Dobimo denormalizirano relacijo Rezultat, ki je tako le v 2. NO.

Rezultat (IDR, Cas_Prvi_Tek, Cas_Drugi_Tek, ***Skupni_cas****, #IDTekmovalec).*

5.4.7.4 Primer denormalizacije v primeru številnosti med relacijama 1:1

Imejmo primer, kjer imamo podatke o stranki in z njo opravljenem intervjuju, ločeni v dveh relacijah.

Stranka *(IDS, Ime, Priimek, Naslov, Postna_st, Telefon,tip_nepremicnine, maxcena)*

Intervju (#IDS, Datum_intervjuja, Komentar, Šifra_zaposlenega).

Če relaciji združimo v relacijo StrankaIntervju dobimo:

StrankaIntervju *(IDS, Ime, Priimek, Naslov, Postna_st, Telefon,tip_nepremicnine, maxcena, Datum_intervjuja, Komentar, Šifra_zaposlenega)*

Ker je sodelovanje v intervjuju opcijsko, bomo imeli v primeru velikega števila strank pri tako denormalizirani relaciji veliko null vrednosti (za vse stranke, s katerimi intervjuja nismo opravili) ter tako veliko izgubljenega prostora.

5.4.7.5 Primer denormalizacije dveh relacij s številnostjo 1:N

Imejmo relacijo Poslovalnica, za katero hranimo telefonske številke, ki jih je seveda lahko več. Zato smo relacijo normalizirali tako, da imamo dve relaciji in sicer Poslovalnica in Telefon:

Poslovalnica (IDPosl, Naslov, Posta)

Telefon (Tel_st, #IDPosl).

Če zopet želimo optimizirati hitrost izvajanja poizvedb, relaciji denormaliziramo v eno samo relacijo. To lahko storimo ob pogoju da: poznamo maksimalno število telefonskih številk vsake poslovalnice, da to število ni preveliko, in da se ne bo spreminjalo. Če na primer vemo, da ima lahko poslovalnica največ tri telefonske številke lahko relaciji denormaliziramo na naslednji način:

Poslovalnica (IDPosl, Naslov, Posta, Tel1, Tel2, Tel3).

Če se odločimo za denormalizacijo, moramo razmisliti glede sprememb indeksov ned podatkovno bazo. Poskrbeti moramo tudi za mehanizme, ki bodo kljub temu zagotavljali integriteto podatkov. Splošni načini so:

- **Uporaba baznih prožilcev** (ang. triggers), ki avtomatizirajo osveževanje izračunanih ali podvojenih podatkov.
- **Vgradnja atomarnih transakcij** v aplikacije, ki zagotovijo osveževanje podatkov.
- **Paketna obdelava**, ki ob določenih časovnih intervalih poskrbi za ponovno konsistentnost podatkovne baze.

Najboljšo integriteto zagotavljajo bazni prožilci, saj osveževanje izvajajo takoj, vendar pa s tem spet zmanjšujejo odzivnost baze.

Tabela podaja prednosti in slabosti, ki se jih moramo zavedati pri denormalizaciji.

Tabela 10: Prednosti in slabosti denormalizacije

Prednosti **Lahko izboljšamo učinkovitost PB z:**	**Slabosti**
Predhodnem izračunu in shranitvi določenih podatkov (primer 3)	Zmanjšanje hitrosti ažuriranj podatkov
Zmanjšanjem števila operacij stika med relacijami (primer 1,2,4)	Slabša fleksibilnost podatkovne baze
Zmanjšanje števila tujih ključev	Povečanje velikosti posamezne relacije
Zmanjšanje števila relacij (primer 2,4,5)	Prilagojeno specifikam določene aplikacije
Zmanjšanje števila indeksov (manjša poraba prostora)	Lahko poenostavi implementacijo v določenih primerih, ter jo naredi bolj kompleksno v drugih primerih

Vir: Connolly in Begg, 2005, str. 531.

Vprašanja za ponavljanje

1. Kaj je cilj fizičnega načrtovanja podatkovne baze?
2. Ali so za to fazo potrebna kakšna specifična znanja, katera?
3. Kaj vsebuje SQL skripta, ki jo dobimo kot rezultat fizičnega načrtovanja?

4. Katere splošne vrste datotečnih organizacij za relacije in indekse poznate?
5. Kaj je namen analize transakcij?
6. Katero tehniko za analizo transakcij poznate?
7. Čemu služijo rezultati analize transakcij?
8. Kaj je namen kreiranja dodatnih indeksov, za katere stolpce jih je smiselno izdelati in za katere ne?
9. Kateri dve vrsti varnostnih mehanizmov za varovanje dostopa do podatkovne baze poznate?
10. Katera dva ukaza za upravljanje dostopnih pravic do objektov podatkovne baze vsebuje jezik SQL?
11. Kaj je denormalizacija in kdaj jo izvedemo?
12. Katere mehanizme za zagotavljanje integritete podatkovne baze uporabljamo v primeru ažuriranja denormaliziranih relacij?
13. Naštejete prednosti in slabosti denormalizacije.

Naloge

Za delo s fizično podatkovno bazo uporabite orodje Oracle APEX, ki je opisano v poglavju 8.3. Gre za spletno orodje, ki v prvem koraku zahteva izdelavo delovne površine.

5.4.1 Izdelava fizične podatkovne baze knjižnice z uporabo SQL skripte v orodju APEX

- SQL skripto, ki jo izdelate za podatkovno bazo Oracle na podlagi relacijskega logičnega modela (glej Slika 60, Slika 61, Slika 62 v poglavju 8.2), uvozite v Apex.
- Preglejte skripto ter jo po potrebi popravite.
- Skripto poženite z ukazom Run.
- Preglejte rezultate kreiranja, odpravite morebitne napake v skripti in postopek generiranja ponovite.
- V brsalniku objektov (Object Browser) preglejte kreirane tabele.
- Vnesite podatke v podatkovno baze, vsaj tri zapise v vsako tabelo: Clanarina, Clan, Knjiga in Izposoja.

Pazite na vrstni red vnosa podatkov!

5.4.2 Izdelava fizične podatkovne baze skladišča z uporabo SQL skripte v orodju APEX

- Konceptualni podatkovni model skladišča (Slika 19) preslikajte v relacijski logični model.
- Iz relacijskega modela kreirajte SQL skripto za izdelavo fizične podatkovne baze.
- SQL skripto uvozite v orodje Apex.
- Preglejte skripto ter jo po potrebi popravite.
- SQL Skripto poženite z ukazom Run.
- Preglejte rezultate kreiranja, odpravite morebitne napake v skripti in postopek generiranja ponovite.

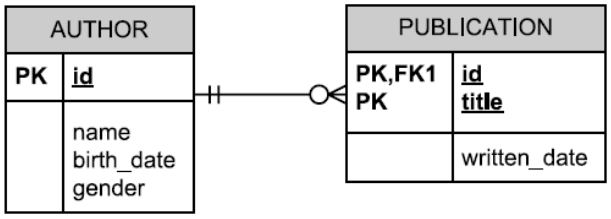
- V brsalniku objektov (Object Browser) preglejte kreirane tabele.
- Vnesite podatke v podatkovno baze, vsaj tri zapise v vsako tabelo.

Pazite na vrstni red vnosa podatkov!

5.4.3 Ročna izdelava fizične podatkovne baze publikacij

- Kreirajte fizično PB v orodju APEX na podlagi logičnega podatkovnega modela publikacij (Slika 30).
- Uporabite ukaze orodja Oracle Apex.
- Ne pozabite kreirati omejitev primarnih in tujih ključev (uporabite ukaze Apexa).
- Kreirajte indekse na stolpce name, birth_date in gender tabele Author (uporabite ukaze Apexa).
- Vnesite podatke v podatkovno baze. Vnesite tri zapise v tabelo Author in šest zapisov v tabelo Publication, tako da ima vsaj en avtor več različnih publikacij.

Slika 30: Logični podatkovni model publikacij

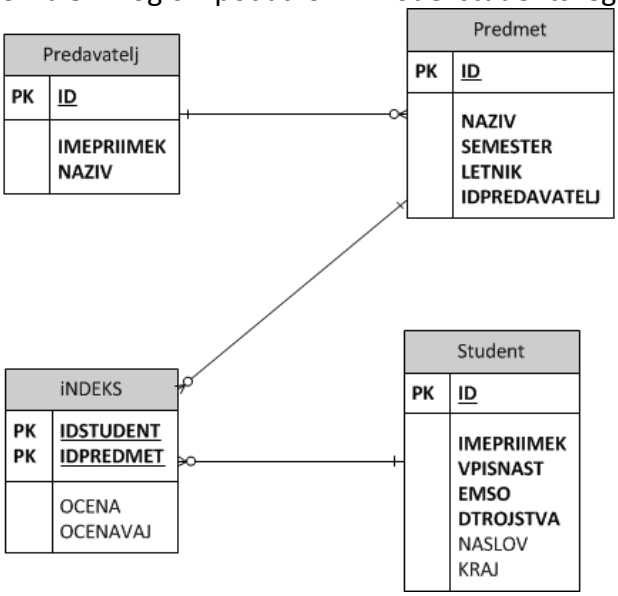


5.4.4 Ročna izdelava fizične podatkovne baze študentskega IS

Kreirajte PB v orodju APEX za naslednji podatkovni model (Slika 31).

- Uporabite ukaze orodja Oracle Apex.
- Ne pozabite kreirati omejitev primarnih in tujih ključev (uporabite ukaze Apexa).
- Vnesite po vsaj tri zapise v tabele Predavatelj, Student in Predmet. Vnesite različno število ocen za posamezne študente v tabeli Indeks.

Slika 31: Logični podatkovni model študentskega IS



5.5 Spremljanje delovanja in optimizacija podatkovne baze

Med delovanjem podatkovne baze je potrebno spremljati njeno učinkovitost, ter na podlagi rezultatov spreminjati določene načrtovalske odločitve (npr. denormalizirati določene relacije). Denormalizacija, ki je podrobno predstavljena v poglavju 5.4.7, je eden od postopkov za izboljšanje učinkovitosti v primeru, da je le-ta slaba.

Cilj spremljanja in optimizacije različnih parametrov, je zagotoviti čim večjo učinkovitost podatkovne baze. Parametri, ki jih spremljamo, so (Connolly in Begg, 2005, str. 532):

- **Propustnost transakcij:** v določenih sistemih, npr. bančnih, rezervacijskih, je pomembno, da sistem lahko obdela veliko število transakcij na časovno enoto.
- **Odzivni čas:** čim hitrejši dostop do podatkov podatkovne baze z vidika uporabnika.
- **Prostor na disku:** ekonomično shranjevanje podatkov podatkovne baze na disku.

Faza fizičnega načrtovanja podatkovne baze se z začetkom njene uporabe ne zaključi, ampak gre za stalen proces, ki vključuje spremljanje navedenih parametrov in njihovo optimizacijo. Pogosto izboljšanje enega parametra vpliva na poslabšanje drugega, zato mora skrbnik podatkovne baze stalno spremljati učinkovitost z različnih vidikov in prilagajati določene parametre. Za spremljanje različnih parametrov delovanja podatkovne baze (gre za administratorska opravila) SUPB-ji vsebujejo različna administratorska aplikativna orodja.

Optimizacija prametrov podatkovne baze nam lahko prinese naslednje koristi :

- Ni potrebno kupiti dodatne strojne opreme,
- Vpliv na večjo učinkovitost uporabnikov in s tem posledično celotnega podjetja, če imamo boljše odzivne čase ali če podatkovna baza lahko obdela več transakcij na časovno enoto.
- Boljši odzivni časi vplivajo na večje zadovoljstvo zaposlenih.
- Boljši odzivni časi lahko vplivajo tudi na večje zadovoljstvo strank.

Da bi administrator lahko povečal učinkovitost delovanja podatkovne baze, mora dobro razumeti, kako različne komponente računalniškega sistema medsebojno vplivajo in kakšen je nadalje njihov vpliv na učinkovitost baze.

Zelo pomemben računalniški vir, ki zagotavlja hitro delovanje aplikacij, je glavni pomnilnik (ang. main memory). Če je glavnega pomnilnika premalo, je potrebno večkrat brati podatke z diska. Če je tega preveč, je delovanje počasno. Za zagotovitev učinkovite rabe glavnega pomnilnika, mora skrbnik podatkovne baze dobro poznati delovanje konkretnega SUPB: kako uporablja glavni pomnilnik, kakšne načine vmesnega pomnjenja uporablja (ang. buffer), kakšne nastavitve njegove velikosti so možne, itd. Potrebno je poznati tudi vzorce dostopa uporabnikov do podatkovne baze: povečanje števila hkratnih uporabnikov podatkovne baze namreč povzroči tudi večje potrebe po glavnem pomnilniku.

Vhodno izhodne operacije (branje in pisanje na disk) predstavljajo množično opravilo pri delu z vsako podatkovno bazo. Diski imajo običajno priporočeno I/O stopnjo. Ko je ta stopnja presežena, se pojavijo I/O ozka grla. Ta nastopijo v primeru, ko želi v določeni časovni enoti do diska dostopati preveč procesov, kar pomeni, da morajo, določeni čakati. V primeru, da prihaja do ozkih grl, je potrebno datoteke razporediti med več diskov tako, da bodo z vidika dostopov čim bolj enakomerno obremenjeni. Priporočljivo je tako ločiti datoteke operacijskega sistema od datotek podatkovne baze, podatke baze od indeksnih datotek baze, log datoteke od podatkov baze in podobno.

Ozka grlo lahko predstavlja tudi računalniško omrežje. Če je prometa na njem veliko, ali je veliko kolizije prometa, lahko nastanejo ozka grla na omrežju preko katerega uporabniki dostopajo do podatkovne baze, ki je navadno nameščena na skupnem strežniku.

Spremembe določenega računalniškega vira lahko pozitivno vplivajo na drug vir. Če tako na primer povečamo količino glavnega pomnilnika, to vpliva na manjše število I/O operacij. Spremljanje delovanja in prilagajanje podatkovne baze in fizičnih računalniških virov, na katerih podatkovna baza deluje, je stalen proces. V času delovanja podatkovne baze namreč prihaja tako do spremenjenih načinov dostopa s strani uporabnikov, povečanja/zmanjšanja števila uporabnikov, ter sprememb v njihovih zahtevah, kar povzroča potrebo po stalnih prilagoditvah. Spremembe pa je potrebno izvajati previdno, saj lahko določene spremembe imajo tudi negativne vplive na druge parametre delovanja. Zato se vedno priporoča spremembe izvajati na testni bazi v testnem okolju ali vsaj izven delovnega časa uporabnikov.

Naloge

Za delo s fizično podatkovno bazo uporabite orodje Oracle APEX, ki je opisano v poglavju 8.3. Gre za spletno orodje, ki v prvem koraku zahteva izdelavo delovne površine.

5.5.1 Prilagoditev fizične podatkovne baze študentskega IS novim poslovnih zahtevam

Fizično podatkovno bazo, ki ste jo izdelali v nalogi 5.4.4, prilagodite novonastalim uporabniškim zahtevam in sicer:

- Za predavatelje želi šola hraniti še: datum zadnje izvolitve v naziv, rojstni datum in število otrok.
- Za študente pa želi šola hraniti še: elektronski naslov in številko mobilnega telefona za obveščanje.
- Šola je začela tudi z mednarodnimi izmenjavami, zato želi shranjevati tudi podatke o njih in sicer: osebo, ki se je udeležila izmenjave (študent ali predavatelj), termin izmenjave, število opravljenih ur predavanj za pedagoške delavce ter podatke o šoli, kjer se izmenjava opravi.

Z orodjem APEX kreirajte nove tabele in atribute, ki bodo omogočali shranjevati navedene podatke ter jih ustrezno umestite ter povežite z obstoječo podatkovno bazo.

6 Jeziki za delo z relacijsko podatkovno bazo

6.1 Relacijska algebra

Relacijska algebra je **postopkovni poizvedovalni jezik,** ki temelji na relacijski teoriji. Sestavljena je iz množice operacij, ki obdelajo eno ali dve vhodni relaciji in kot rezultat vrnejo novo. Operacije relacijske algebre se delijo na **osnovne** in **izvedene**.

Med **osnovne operacije** spadajo (Mohorič, 1992, str. 117-120, Korth in Silberschatz, 1991, str. 60-70):

- **Selekcija**: Rezultat operacije je množica n-teric, ki ustrezajo podanim pogojem. Le-ti so podani v obliki predikata, v katerem se lahko pojavijo primerjalni in logični operatorji (iz tabele se izločijo vrstice, ki ne ustrezajo pogojem).
- **Projekcija**: Operacija ustvari novo relacijo, ki vsebuje le podmnožico atributov prvotne (iz tabele se izločijo nekateri stolpci).
- **Kartezijski produkt**: V relaciji, ki nastane kot rezultat kartezijskega produkta, se nahajajo vsi možni stiki n-teric obeh vhodnih relacij, pri čemer shemo kartezijskega produkta sestavljata obe vhodni shemi.
- **Unija**: V uniji dveh relacij se nahajajo n-terice obeh relacij, pri čemer so dvojne vrstice izločene. Operacija je smiselna le v primeru, ko obe relaciji pripadata isti relacijski shemi.
- **Razlika**: Razlika dveh relacij določa novo relacijo, v kateri se nahajajo zgolj tiste n-terice prve relacije, ki ne nastopajo v drugi.

Izvedene operacije relacijske algebre pa so (Mohorič, 1992, str. 120-122, Korth in Silberschatz, 1991, str. 70-75):

- **Presek**: Presek dveh relacij je množica n-teric, ki pripadajo obema vhodnima relacijama.
- **Naravni stik**: V naravnem stiku se nahajajo vsi možni stiki n-teric, pri katerih so komponente, ki pripadajo enako imenovanim atributom iz obeh shem enake, pri čemer so podvojene komponente izločene. Poleg selekcije in projekcije predstavlja naravni stik najpogosteje uporabljeno operacijo v poizvedovalnih jezikih.
- **Θ stik**: Rezultat operacije je relacija, v kateri se nahajajo vsi možni stiki vhodnih relacij, v katerih sta izbrani komponenti v medsebojni relaciji Θ (podana je s primerjalnimi operatorji).

6.2 Relacijski račun

Za razliko od relacijske algebre, se pri relacijskem računu ne navaja postopka izvajanja operacij nad relacijami, temveč se podajo lastnosti iskane relacije. Relacijo je moč zapisati v naslednji obliki (Mohorič, 1992, str. 124):

$\{t^{(n)}:F(t)\}$ (iskano relacijo sestavljajo vse tiste n-terice, ki zadoščajo formuli F)

Formula F je sestavljena iz atomov različnih oblik, pri čemer pa je potrebno upoštevati določena pravila glede same oblike atomov in njihovega sestavljanja v kompleksne izraze. Izpolnjena mora biti namreč zahteva po varnih izrazih relacijskega računa, ki zagotavlja, da bo rezultat vedno le končna relacija. **Relacijski račun predstavlja temelje splošnim poizvedovalnim jezikom kot je SQL.**

6.3 SQL

Z relacijsko algebro in relacijskim računom je možno skonstruirati obsežne poizvedbe namenjene relacijskim podatkovnim bazam. Ker pa gre pri njiju predvsem za matematično notacijo, nista primerna za praktično uporabo v konkretnih sistemih za upravljanje podatkovnih baz. Eden od vidikov proučevanj relacijskega podatkovnega modela se je zato nanašal na razvoj uporabnejših jezikov. V prvi polovici sedemdesetih let je tako nastalo kar nekaj jezikov, med katerimi velja omeniti SEQUEL (Structured English Query Language), kot predhodnika današnjega jezika SQL. Razvil ga je D. D. Chamberlin s sodelavci v IBM San Jose laboratorijih leta 1974. Prva implementacija SEQUEL-a je bila narejena v okviru IBM-ovega prototipa SEQUEL-XMR v letih 1974-75. Hitremu razvoju jezika SQL in širitvi njegove uporabe v komercialnih sistemih je sledila njegova standardizacija. Leta 1982 je ANSI (American National Standard Institute) ustanovil komite za podatkovne baze z nalogo razviti predlog za **standardiziran relacijski jezik**. Predlog, ki je bil ratificiran leta 1986, je vseboval predvsem IBM-ovo različico jezika SQL z odpravljenimi pomanjkljivostmi. Leto kasneje je standard ANSI za relacijski jezik prevzela še organizacija ISO (International Standards Organization).

Jezik SQL vsebuje vrsto pripomočkov za definiranje, manipuliranje in nadzor podatkov v relacijskih podatkovnih bazah. **SQL je danes najpogosteje uporabljen jezik za dostop do relacijskih podatkovnih baz in vsi pomembni sistemi za upravljanje podatkovnih baz vsebujejo eno izmed njegovih različic.** Osnovna značilnost SQL-a je **neproceduralni pristop**, ki ga je povzel po relacijskem računu, hkrati pa se izogiba matematični notaciji z uporabo angleškega jezika. Na ta način združuje moč relacijske algebre in relacijskega računa s preprostostjo in razumljivostjo angleškega jezika.

Opredelitev jezika SQL kot standarda ima tako prednosti kot slabosti. Med **prednosti** se lahko uvrsti (Date, 1989, str. 4):

- **Zmanjšanje stroškov za izobraževanje:** Programerji lahko svoje znanje uporabljajo na različnih sistemih, dodatno izobraževanje ni potrebno oziroma je omejeno na minimum.
- **Prenosljivost programskih rešitev**: Aplikacije razvite v jeziku SQL lahko delujejo v okviru različnih sistemov za upravljanje podatkovnih baz in na različni strojni opremi.
- **Komunikacija med različnimi sistemi**: Omogoča komunikacijo med različnimi sistemi za upravljanje podatkovnih baz in s tem izgradnjo porazdeljenih podatkovnih sistemov.
- **Poenostavljena izbira:** Če vsi sistemi podpirajo isti vmesnik za dostop do podatkov, se uporabnik lahko osredotoči na druge dejavnike pri izbiri ustreznega sistema za upravljanje podatkovnih baz.

Pomembnejše **slabosti** pa so naslednje (Date, 1989, str. 5):

- **Standardizacija lahko zaduši kreativnost:** Obstaja namreč možnost, da programerji ne bodo iskali najboljše rešitve določenega problema, saj standard že predpisuje neko alternativo, ki pa ni nujno tako učinkovita.
- **SQL je daleč od idealnega relacijskega jezika:** Za načrtovanje formalnih jezikov obstaja kar nekaj dobrih principov, ki pa se jih razvijalci jezika SQL po mnenju mnogih niso držali (pomanjkanje ortogonalnosti vgrajenih funkcij, spremenljivk, praznih množic itd.). Tako jezik vsebuje obilo omejitev in posebnih pravil, kar ga naredi težkega za definiranje, opis, učenje in implementacijo.
- **Standard SQL je na posameznih področjih zelo pomanjkljiv:** Nima definiranih vseh konceptov relacijske teorije, ki bi jih potrebovali v vsakdanji praksi (pomanjkljivosti pri podpori zunanjih ključev, domen, stikov, podatkovnih kurzorjev itd.). Prav zato so se v preteklosti in se še danes pojavljajo različice jezika z bolj ali manj obsežnimi razširitvami.

Čeprav pogosto govorimo o jeziku SQL kot o poizvedovalnem jeziku, pa SQL vsebuje tudi vrsto dodatnih elementov, ki omogočajo celovito obravnavo shranjenih podatkov. V jeziku SQL sta združena tako DDL (Data Definition Language) - jezik za definiranje podatkov (kreiranje elementov podatkovne baze), kot tudi DML (Data Manipulation Language) - jezik za manipulacijo s podatki.

Osnovne elemente jezika tako lahko strnemo v naslednje točke (Rob in Coronel, 2004, str. 226-228):

- **Jezik za definiranje podatkov** (ang. Data Definition Language): Definiranje podatkov je sestavljeno iz večjega števila korakov. Začne se s kreiranjem same podatkovne baze, nadaljuje z vzpostavitvijo delovnih področij in konča s kreiranjem tabel. Medtem ko sta prvi dve opravili tesno povezani s posameznim sistemom za upravljanje podatkovnih baz oziroma njegovo različico jezika SQL, je kreiranje tabel standardizirano in se izvaja s stavkom SQL *Create Table*. V okviru definiranja tabel je potrebno navesti nazive in podatkovne tipe atributov oziroma stolpcev ter vse zahtevane omejitve. Le-te se nanašajo na izbor primarnih in tujih ključev, povezanosti tabele s preostalimi tabelami preko tujih ključev in druge omejitve v obliki različnih prepovedi in vrednostnih omejitev. Pomembno mesto pri definiranju učinkovite podatkovne baze je potrebno nameniti tudi načrtovanju indeksov nad podatki. Indekse definiramo nad vsemi atributi, ki predstavljajo primarni ključ tabele ter nad atributi, po katerih se pogosto izvajajo poizvedbe. SQL v ta namen uporablja stavek *Create Index*.
- **Jezik za manipulacijo s podatki** (Data Manipulation Language): Obstajajo štirje osnovni stavki SQL za manipulacijo s podatki:
    - ***Select***: Uporablja se za kreiranje bolj ali manj obsežnih poizvedb nad podatkovno bazo. Omogoča praktično izvajanje temeljnih operacij relacijske algebre (selekcije, projekcije in stika) nad relacijami oziroma tabelami. Splošna oblika stavka je "*Select - From - Where*", pri čemer se rezultati poizvedbe shranijo v novo, začasno tabelo. Njena oblika je odvisna od uporabljenih parametrov znotraj stavka *Select*, v splošnem pa se najpogosteje uporablja sortiranje in grupiranje.
    - ***Insert***: Omogoča vstavljanje podatkov v tabelo. Preprosto vstavljanje poteka vrstica za vrstico, s kombinacijo stavkov *Insert* in *Select* pa je mogoče vstaviti tudi množico vrstic kot rezultat poizvedbe.
    - ***Update***: Uporablja se za spreminjanje vrednosti posameznih atributov v tabeli.

- ***Delete***: Briše vrstice iz tabele, pri čemer se lahko operacija tako kot pri stavku *Update* izvaja nad posamezno vrstico ali množico vrstic, ki zadoščajo izbranim pogojem.

- **Vgrajeni jezik za manipulacijo s podatki** (ang. Embedded DML): Posebne oblike jezika SQL so namenjene uporabi skupaj s klasičnimi programskimi jeziki kot so C, Fortran, Pascal itd. Uporaba te kombinacije združi poizvedovalno moč jezika SQL pri delu s podatkovnimi bazami s hitrostjo in učinkovitostjo jezikov tretje generacije.
- **Pogledi**: SQL vsebuje dva tipa tabel: osnovne tabele (fizično obstajajo na diskih, v njih so shranjeni dejanski podatki, zanje so definirani različni indeksi) in pogledi (navidezne tabele, ki uporabniku izgledajo enako kot fizične tabele, niso pa vedno fizično zapisane na pomnilniških medijih). Pogledi predstavljajo močno orodje za izgradnjo in predstavitev različnih vidikov podatkov, shranjenih v relacijskih podatkovnih bazah. Pogledi imajo pomembno vlogo tudi pri zagotavljanju varnosti podatkov, ko je potrebno omogočiti vpogled uporabnikom zgolj v nekatere segmente centraliziranega podatkovnega modela, ostale pa zavarovati pred nepooblaščenimi dostopi.
- **Agregatne funkcije**: Uporabljajo se za izvajanje računskih operacij nad izbranimi podatki iz tabel. Osnovne funkcije so: *Avg* (povprečje), *Min* (minimum), *Max* (maksimum), *Sum* (vsota) in *Count* (štetje). Poleg njih vsebujejo različice jezika SQL še veliko dodatnih funkcij, ki omogočajo kompleksno obdelavo podatkov s ciljem zagotoviti uporabnikom želene informacije.
- **Varnost**: SQL vsebuje stavke, ki omogočajo določanje pravic dostopa do tabel in pogledov ter manipulacijo z njimi. V ta namen se uporabljata stavka *Grant* (dodeljevanje pravice) in *Revoke* (odvzemanje pravice).

6.3.1 SQL DDL

SQL DDL je jezik za definiranje podatkov (ang. Data Definition Language). Omogoča kreiranje podatkovne baze in njenih gradnikov (tabel, stolpcev, omejitev, indeksov…).

SQL stavki so sestavljeni iz rezerviranih in uporabniško definiranih besed. Rezervirane besede so natančno določene, napisane morajo biti pravilno, ne smejo se lomiti med vrstice. Uporabniško definirane besede označujejo razne podatkovne objekte, kot so npr. tabele, stolpci, pogledi.

Večina komponent SQL stavkov je neodvisna od velikosti pisave; izjema so tekstovni podatki. Da dosežemo boljšo berljivost, pišemo SQL stavke v več vrsticah in z zamiki:

- Vsak sklop SQL stavka se začne v novi vrstici.
- Sklopi so levo poravnani.
- Če ima sklop več delov, je vsak v svoji vrstici in poravnan z začetkom sklopa.

Za opis sintakse SQL stavkov v nadaljevanju bomo uporabljali razširjeno BNF[1] notacijo:

- REZERVIRANE BESEDE z velikimi črkami,
- uporabniško definirane besede z malimi črkami,
- Znak **|** za izbiro med alternativami,
- {Obvezni elementi} v zavitih oklepajih,
- [Opcijski elementi] v oglatih oklepajih,
- Znak **…** za opcijske ponovitve (0 ali več).

Tabela 11: Ukazi SQL DDL

Ukaz	**Opis**
CREATE TABLE	Kreiranje nove tabele
NOT NULL	Omejitev, ki zagotavlja, da stolpec ne bo vseboval null vrednosti.
UNIQUE	Omejitev, ki zagotavlja, da v stolpcu ne bo podvojenih vrednosti.
PRIMARY KEY	Definira primarni ključ tabele.
SECUNDARY KEY	Definira tuji ključ tabele.
DEFAULT	Definira privzeto vrednost stolpca (se privzame, kadar ni podane druge vrednosti)
CREATE INDEX	Kreiranje indeksa tabele.
CREATE VIEW	Kreiranje pogleda. Gre za dinamično podmnožico vrstic in stolpcev iz določene množice tabel.
ALTER TABLE	Spreminja definicijo tabele potem, ko je ta že kreirana (doda, spremeni, briše stolpce ali omejitve).
DROP TABLE	Trajno zbriše tabelo z vsemi podatki.
DROP INDEX	Trajno zbriše indeks tabele.
DROP VIEW	Trajno zbriše pogled.

SQL stavki podani kot primeri v tem poglavju se nanašajo na primer podatkovne baze knjižnice, katere načrt se nahaja v poglavju 8.2.2 (Slika 61). Relacijski model obsega štiri tabele: CLAN, CLANARINA, IZPOSOJA in KNJIGA.

Podatkovna baza knjižnice je podana z naslednjo relacijsko shemo:

CLAN (ST_izkaznica, Ime, Priimek, Naslov, E_naslov, Datum_placila, #Vrsta_clana)
CLANARINA (Vrsta_clana, Znesek)
KNJIGA (ISBN, Avtor, Naslov, Zalozba, Leto_izida, Dovoljen_cas_izposoje)
IZPOSOJA (#ISBN, #ST_izkaznica, Datum_izposoje, Datum_vrnitve)

[1] BNF = Backus Naur Form

6.3.1.1 Kreiranje tabel

Za kreiranje tabel novih uporabljamo SQL stavek **CREATE TABLE**. Pri kreiranju tabele navedemo ime tabele in imena stolpcev s podatkovnimi tipi kot prikazuje v nadaljevanju podana sintaksa. Navedemo tudi morebitne omejitve stolpcev, npr. NOT NULL.

Sintaksa:

CREATE TABLE <table_name> (
<column_name_1> <data_type_1>,
<column_name_2> <data_type_2>,
<column_name_N> <data_type_N>);

6.3.1.1.1 Primer kreiranja tabele CLAN

Z uporabo stavka CREATE TABLE kreiramo tabelo Clan. Njeni stolpci so ST_izkaznica, Ime, Priimek, Naslov, E_naslov, Datum_placila in Vrsta_clana. Za vsak stolpec podamo tudi podatkovni tip, ki pove, kakšno vrsto podatkov bo vanj možno vpisati (npr. tip VARCHAR2(20) pomeni, da bomo v stolpec Ime lahko vpisali do 20 poljubnih znakov, tip DATE pa da bomo v stolpec Datum_placila lahko vnesli le datume). Kreiramo tudi omejitve NOT NULL za stolpec ST_izkaznica, ki predstavlja primarni ključ tabele, in stolpec Vrsta_clana, ki predstavlja tuji ključ (povezava s tabelo CLANARINA). Omejitvi NOT NULL pomenita, da bo pri vnosu vrstice v tabelo potrebno vnesti vsaj ta dva podatka.

```
CREATE TABLE Clan
  (
    ST_izkaznica  NUMBER (4) NOT NULL ,
    Ime           VARCHAR2 (20) ,
    Priimek       VARCHAR2 (20) ,
    Naslov        VARCHAR2 (40) ,
    E_naslov      VARCHAR2 (40) ,
    Datum_placila DATE ,
    Vrsta_clana   VARCHAR2 NOT NULL
  ) ;
```

6.3.1.1.2 Primer kreiranja tabele KNJIGA

Z uporabo stavka CREATE TABLE kreiramo še tabelo Knjiga. Njeni stolpci so ISBN (primarni ključ), Avtor, Naslov, Zalozba, Leto_izzida, Dovoljen_cas_izp. Za vsak stolpec podamo tudi podatkovni tip. Kreiramo tudi omejitev NOT NULL za stolpec ISBN, ki predstavlja primarni ključ.

```
CREATE TABLE Knjiga
  (
    ISBN             NUMBER (13) NOT NULL ,
    Avtor            VARCHAR2 (20) ,
    Naslov           VARCHAR2 (30) ,
    Zalozba          VARCHAR2 (20) ,
    Leto_izzida      NUMBER (4) ,
    Dovoljen_cas_izp VARCHAR2 (10)
  ) ;
```

6.3.1.2 Kreiranje indeksov

Z uporabo stavka CREATE [UNIQUE] INDEX kreiramo indekse. Poznamo dve vrsti indeksov. Na stolpce, ki predstavljajo primarne ključne, moramo postaviti unikatne indekse (ang. unique), na ostale pa postavimo običajne indekse.

Sintaksa:

CREATE [UNIQUE] INDEX <index_name> on <table_name> (
<column_name_1>,
<column_name_2>,
<column_name_N>);

6.3.1.2.1 Primer kreiranja unikatnega (unique) indeksa na stolpcu ISBN (primarni ključ)

V tem primeru kreiramo unique indeks na stolpcu ISBN, ki predstavlja ključ tabele KNJIGA.

```
CREATE UNIQUE INDEX Knjiga__ISBN ON Knjiga
  (
    ISBN ASC
  );
```

6.3.1.2.2 Primer kreiranja indeksa na stolpcu Avtor

Navadni indeks postavimo na stolpec Avtor, saj predvidevamo, da bodo uporabniki pogosto iskali knjige po avtorju.

```
CREATE INDEX Knjiga__avtor ON Knjiga
  (
    Avtor ASC
  );
```

6.3.1.3 Kreiranje pogledov

Z uporabo stavka CREATE VIEW kreiramo poglede.

Sintaksa:

CREATE [OR REPLACE] VIEW <view_name> AS
<sql_select_statement>;

6.3.1.3.1 Primer kreiranja pogleda plačila članarine

Izdelajmo pogled, ki prikazuje plačilo članarine. Prikazati želimo ime in priimek člana, kdaj je nazadnje plačal članarino in v kakšnem znesku. V pogledu tako združimo podatke iz dveh tabel: CLAN in CLANARINA.

```
CREATE OR REPLACE VIEW placilo_clanarine AS
SELECT ime, priimek, st_izkaznica, datum_placila, znesek
FROM clan, clanarina
WHERE clan.vrsta_clana = clanarina.vrsta_clana;
```

Podatke sedaj lahko pregledujemo preko tega pogleda kot bi bili shranjeni v eni sami tabeli, saj pogled vključuje stolpce obeh tabel (ime, priimek, st_izkaznica, datum_placila iz tabele Clan in znesek iz tabele Clanarina).

Slika 32: Prikaz podatkov dveh tabel z uporabo pogleda placilo_clanarine

IME	PRIIMEK	ST_IZKAZNICA	DATUM_PLACILA	ZNESEK
Peter	Potočnik	4	-	10
Metka	Novak	1	01/28/2015	10
Alenka	Rožanec	2	01/03/2015	30
Tim	Prisel	3	12/12/2014	30

row(s) 1 - 4 of 4

6.3.1.4 *Definiranje omejitev*

OMEJITEV NOT NULL

Zgoraj smo že spoznali omejitev NOT NULL, ki jo moramo dati k stolpcem s ključi, lahko pa jo damo h kateremu koli stolpcu. Omejitev uporabimo, kadar je vnos določenega podatka obvezen.

6.3.1.4.1 Primer obveznosti imena, priimka in naslova (NOT NULL)

Če se odločimo, da so ime, priimek in naslov obvezni podatki, tudi k tem stolpcem dodamo omejitev NOT NULL.

```
CREATE TABLE Clan
  (
    ST_izkaznica  NUMBER (4) NOT NULL ,
    Ime           VARCHAR2 (20) NOT NULL ,
    Priimek       VARCHAR2 (20) NOT NULL ,
    Naslov        VARCHAR2 (40) NOT NULL ,
    E_naslov      VARCHAR2 (40) ,
    Datum_placila DATE ,
    Vrsta_clana   VARCHAR2 (10) NOT NULL
  ) ;
```

OMEJITEV PRIMARNEGA KLJUČA

Omejitvi primernega in tujega ključa dodajamo k tabelam s stavkom ALTER TABLE.

Sintaksa:

ALTER TABLE <table_name> ADD CONSTRAINT <constraint_name>
PRIMARY KEY (
<column_name_1>,
<column_name_2>,...
<column_name_N>);

6.3.1.4.2 Primer omejitve primarnega ključa tabele CLAN

```
ALTER TABLE Clan ADD CONSTRAINT Clan_PK PRIMARY KEY
(
  ST_izkaznica
);
```

6.3.1.4.3 Primer omejitve primarnega ključa tabele IZPOSOJA (sestavljen ključ)

```
ALTER TABLE Izposoja ADD CONSTRAINT Izposoja_PK PRIMARY KEY
(
  ST_izkaznica, ISBN
);
```

OMEJITEV TUJEGA KLJUČA

Omejitev tujega ključa pove, kateri stolpci v posameznih tabelah tvorijo tuje ključe.

Sintaksa:

ALTER TABLE <table_name> ADD
CONSTRAINT <constraint_name>
FOREIGN KEY (
<column_name_1>, …
<column_name_N>)
REFERENCES <referenced_table_name> (
<column_name_1>, …
<column_name_N>);

6.3.1.4.3 Primeri vseh omejitev tujega ključa v bazi knjižnice

```
ALTER TABLE Clan ADD CONSTRAINT Clan_Clanarina_FK FOREIGN KEY (Vrsta_clana)
REFERENCES Clanarina (Vrsta_clana) ;

ALTER TABLE Izposoja ADD CONSTRAINT Izposoja_Clan_FK FOREIGN KEY (ST_izkaznica)
REFERENCES Clan (ST_izkaznica) ;

ALTER TABLE Izposoja ADD CONSTRAINT Izposoja_Knjiga_FK FOREIGN KEY (ISBN)
REFERENCES Knjiga (ISBN) ;
```

6.3.1.5 *Brisanje gradnikov podatkovne baze*

Za **brisanje tabele** podatkovne baze uporabljamo SQL stavek **DROP TABLE** z naslednjo sintakso:
DROP TABLE <table_name>
Izbrišemo tabelo CLAN iz podatkovne baze.

6.3.1.5.1 Primer: Brisanje tabele CLAN

```
DROP TABLE Clan;
```

Za **brisanje pogleda** podatkovne baze uporabljamo SQL stavek **DROP VIEW** z naslednjo sintakso:
DROP VIEW <view_name>
Izbrišemo pogled placilo_clanarine.

6.3.1.5.2 Primer brisanja pogleda placilo_clanarine

```
DROP VIEW placilo_clanarine;
```

Za **brisanje indeksa** podatkovne baze uporabljamo SQL stavek **DROP INDEX** z naslednjo sintakso:
DROP INDEX <index_name>
Izbrišemo indeks na stolpcu avtor.

6.3.1.5.3 Primer brisanja indeksa Knjiga_avtor

```
DROP INDEX Knjiga_avtor;
```

6.3.1.6 *Dodeljevanje in odvzemanje pravic*

Za dodelitev in odvzem pravic v SQL-u poznamo stavka **GRANT** in **REVOKE**.

Za dodeljevanje pravic uporabljamo stavek **GRANT** z naslednjo sintakso:
GRANT {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
TO {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]

PrivilegeList je sestavljen iz ene ali več pravic, ločenih z vejico (INSERT, UPDATE,...). Lahko pa damo vse pravice (*ALL PRIVILEGES*). PUBLIC omogoča dodelitev pravic vsem trenutnim in bodočim

uporabnikom. ObjectName se nanaša na osnovno tabelo, pogled, domeno, znakovni niz, dodelitve in prevedbe. WITH GRANT OPTION dovoljuje, da uporabnik naprej dodeljuje pravice.

6.3.1.6.1 Primer dodelitve pravic vsem knjižničarjem nad tabelo IZPOSOJA

```
GRANT ALL PRIVILEGES
ON Izposoja
TO Knjiznicar WITH GRANT OPTION;
```

Za odvzem pravic uporabljamo stavek **REVOKE** z naslednjo sintakso:
REVOKE [GRANT OPTION FOR]
{PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
[RESTRICT | CASCADE]

6.3.1.6.2 Primer odvzema vseh pravic knjižničarjev nad tabelo CLAN

```
REVOKE ALL on clan FROM Knjiznicar;
```

6.3.2 SQL DML

SQL DML (ang. Data Manipulation Language) **je jezik, ki vsebuje množico operacij za manipulacijo s podatki, shranjenimi v podatkovni bazi**. Omogoča dodajanje, brisanje, spreminjanje ter različna poizvedovanja. DML skupina zajema naslednje SQL stavke (Connolly in Begg, 2010, str. 139):

- INSERT → Dodajanje novih podatkov v podatkovno bazo
- DELETE → Brisanje podatkov iz podatkovne baze
- UPDATE → Spreminjanje podatkov shranjenih v podatkovni bazi in
- SELECT → Izbira določenih skupin podatkov iz podatkovne baze.

Vsi nenumerični podatki morajo biti podani znotraj enojnih narekovajev. SQL ne razlikuje med velikimi in malimi črkami pri pisanju (vseeno je ali ime tabele ali atributa zapišemo kot clan, Clan, ali CLAN).

Tabela 12: Ukazi in operatorji SQL DML

Ukaz	Opis
INSERT	Doda vrstico ali več vrstic v določeno tabelo.
SELECT	Izbere atribute ene ali več tabel/pogledov.
WHERE	Omeji izbor vrstic le na tiste vrstice, ki zadoščajo pogoju.
GROUP BY	Grupira izbrane vrstice po enem ali več atributih.
HAVING	Omeji izbor grupiranih vrstic glede na podani pogoj.
ORDER BY	Uredi izbrane vrstice po enem ali več atributih.
UPDATE	Spremeni vrednosti atributov v eni ali več vrsticah tabele.
DELETE	Briše eno ali več vrstic tabele.
Primerjalni operatorji	
=,<,>,<=,>=,<>	Se uporabljajo pri definiranju pogojev nad številčnimi vrednostmi.
Logični operatorji	
AND/OR/NOT	Se uporabljajo pri definiranju kompleksnejših pogojev.
Posebni operatorji	Se uporabljajo pri definiranju pogojev in sicer:
BETWEEN	Preveri ali je vrednost atributa znotraj podanega intervala.
IS NULL	Preveri, ali je vrednost atributa null (atribut nima vrednosti).
LIKE	Preveri, ali vrednost atributa ustreza podanemu vzorcu (za podatke tipa char).
IN	Preveri, ali je vrednost atributa ustreza vsaj eni od vrednosti iz podanega seznama vrednosti.
EXISTS	Preveri ali vgnezdena poizvedba vrne vsaj eno vrstico (takrat je vrednost true, drugače pa false).
DISTINCT	Omeji vrednosti, ki so rezultat poizvedbe tako, da se ne ponavljajo.
Agregatne funkcije	Se uporabljajo v stavku SELECT za matematične operacije nad stolpci tabel.
COUNT	Vrne število vrstic z ne-nul vrednostmi v določenem stolpcu.
MIN	Vrne najmanjšo vrednost v določenem stolpcu.
MAX	Vrne največjo vrednost v določenem stolpcu.
SUM	Vrne seštevek vseh vrednosti v določenem stolpcu.
AVG	Vrne povprečno vrednost vseh vrednosti določenega stolpca.

6.3.2.1 *Dodajanje podatkov - INSERT*

Za dodajanje vrstic v tabelo podatkovne baze uporabljamo stavek **INSERT**. Navedemo stolpce in vrednosti, ki jih želimo vpisati. Navesti je potrebno vse tiste stolpce in vrednosti, ki zahtevajo vnos (imajo omejitev NOT NULL). Sintaksa in primer sta podana spodaj.

Sintaksa:
INSERT INTO <table_name> (
<column_name_1>,…
<column_name_N>)
VALUES (
<column_value_1>,…
<column_value_N>);

0.1 Primer vnosa novega člana v tabelo CLAN

```
INSERT INTO clan (
St_izkaznica, ime, priimek, naslov, vrsta_clana)
VALUES (
5,'Jože','Jamnik','Na jami 10','Upokojenec');
```

Vnesemo novega člana Jožeta Jamnika v tabelo Clan. Obvezna podatka sta St_izkaznica in vrsta_clana, ki predstavljata primarni oz. tuji ključ (nanju smo postavili omejitev NOT NULL). Vnesli bomo še podatke o imenu, priimku in naslovu.
Po izvedbi SQL stavka - vnosa novega zapisa imamo v tabeli CLAN dodano vrstico s tem članom (Tabela 13) s številko izkaznice 5. Nismo vnesli E_naslov in datum_placila članarine, zato sta ti dve celici v vrstici novo dodanega člana prazni.

Tabela 13: Prikaz podatkov v tabeli CLAN po izvedbi ukaza INSERT

EDIT	ST_IZKAZNICA	IME	PRIIMEK	NASLOV	E_NASLOV	DATUM_PLACILA	VRSTA_CLANA
	4	Peter	Potočnik	-	-	-	Dijak
	1	Metka	Novak	Tržaška c.25	metka.novak@gmail.com	01/28/2015	Študent
	2	Alenka	Rožanec	Viška c. 45	-	01/03/2015	Zaposleni
	3	Tim	Prisel	-	tim.prisel@gmail.com	12/12/2014	Zaposleni
	5	Jože	Jamnik	Na jami 10	-	-	Upokojenec

row(s) 1 - 5 of 5

Podatke v tabelo lahko polnimo tudi tako, da jih prepišemo iz druge, že obstoječe tabele. Tabeli morata imeti takšni relacijski shemi, da je prepis možen (kompatibilni podatkovni tipi).

Sintaksa – prepis iz druge tabele:
INSERT INTO <table_name> (
<column_name_1>,…
<column_name_N>)
SELECT <column_value_1>, <column_value_N>…
FROM <from_table_name> ...;

0.2 Primer vnosa nove osebe v tabelo Oseba s prepisom iz tabele Predavatelj

```
INSERT INTO Oseba (
```

```
ime,
priimek)
SELECT imepriimek, naziv
FROM Predavatelj;
```

6.3.2.2 *Spreminjanje podatkov – UPDATE*

Za spreminjanje podatkov uporabljamo stavek **UPDATE**. Navedemo ime tabele, stolpce in vrednosti, ki jih želimo prepisati. Če želimo spremeniti le določene vrstice, podamo še pogoj WHERE. Sintaksa in primer sta podana spodaj.

Sintaksa:

UPDATE <table_name>
SET <column_name_1> = <column_value_1>,
<column_name_2> = <column_value_2>,...
<column_name_N> = <column_value_N>;
[WHERE]

6.3.2.2.1 Primer spremembe hišne številke znotraj naslova predhodno dodanega člana

```
UPDATE clan
SET naslov = 'Na Jami 20'
WHERE st_izkaznica=5;
```

Tabela 14: Prikaz podatkov v tabeli CLAN po izvedbi ukaza UPDATE

EDIT	ST_IZKAZNICA	IME	PRIIMEK	NASLOV	E_NASLOV	DATUM_PLACILA	VRSTA_CLANA
	4	Peter	Potočnik	-	-	-	Dijak
	1	Metka	Novak	Tržaška c.25	metka.novak@gmail.com	01/28/2015	Študent
	2	Alenka	Rožanec	Viška c. 45	-	01/03/2015	Zaposleni
	3	Tim	Prisel	-	tim.prisel@gmail.com	12/12/2014	Zaposleni
	5	Jože	Jamnik	Na Jami 20	-	-	Upokojenec

row(s) 1 - 5 of 5

6.3.2.3 Brisanje podatkov – DELETE

Za brisanje podatkov iz podatkovne baze uporabljamo stavek **DELETE**. Navedemo ime tabele in pogoje. Sintaksa in primer sta podana spodaj.

Sintaksa:

DELETE FROM <table_name>
WHERE <column_name_N> = <column_value_N>...;

6.3.2.4.1 Primer brisanja predhodno dodanega člana s številko izkaznice 5

```
DELETE FROM clan
WHERE st_izkaznica=5;
```

Tabela 15: Prikaz tabele CLAN po izvedbi ukaza DELETE

EDIT	ST_IZKAZNICA	IME	PRIIMEK	NASLOV	E_NASLOV	DATUM_PLACILA	VRSTA_CLANA
	4	Peter	Potočnik	-	-	-	Dijak
	1	Metka	Novak	Tržaška c.25	metka.novak@gmail.com	01/28/2015	Študent
	2	Alenka	Rožanec	Viška c. 45	-	01/03/2015	Zaposleni
	3	Tim	Prisel	-	tim.prisel@gmail.com	12/12/2014	Zaposleni

row(s) 1 - 4 of 4

6.3.2.4 Poizvedbe – SELECT

Namen **SELECT** stavka je poizvedovanje in prikaz podatkov iz ene ali več medsebojno povezanih tabel. Je zelo močen ukaz, s katerim lahko v enem stavku izvedemo več ukazov relacijske algebre: selekcijo, projekcijo in join več tabel. **SQL SELECT je najpogosteje uporabljan SQL stavek**. Njegova splošna sintaksa je podana spodaj.

Sintaksa:

SELECT [DISTINCT | ALL]
{ | [columnExpression [AS newName]] [,...] }*
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]

Vrstnega reda sklopov ni možno spreminjati. Obvezna sta samo SELECT in FROM sklopa. Pomen sklopov je naslednji:

- **SELECT**: določa stolpce, ki naj se pojavijo v izhodni relaciji (projekcija)
- **FROM**: določa tabele za poizvedbo
- **WHERE**: filtrira vrstice (selekcija), lahko vključimo tudi povezovanje tabel (join)
- **GROUP BY**: združuje vrstice po vrednostih izbranih stolpcev
- **HAVING**: filtrira skupine glede na določene pogoje
- **ORDER BY**: določa vrstni red vrstic na izhodu

POIZVEDBE NAD ENO TABELO

6.3.2.4.1 Primer: Izpiši vse podatke o vseh članih

Poizvedbo lahko zapišemo na dva načina. Lahko naštejemo vse stolpce tabele član (prvi primer) ali uporabimo okrajšavo * (drugi primer). Tabela 16 prikazuje rezultat poizvedbe.

```
SELECT st_izkaznica, ime, priimek, naslov, e_naslov, datum_placila, vrsta_clana
```

```
FROM Clan;
```

ali

```
SELECT *
FROM Clan;
```

Tabela 16: Rezultat poizvedbe primera 6.3.2.4.1

ST_IZKAZNICA	IME	PRIIMEK	NASLOV	E_NASLOV	DATUM_PLACILA	VRSTA_CLANA
4	Peter	Potočnik	-	-	-	Dijak
6	Janez	Jamnik	Jamna ulica 5	-	-	Študent
7	Jaka	Jamnik	Jamna ulica 6	-	-	Študent
1	Metka	Novak	Tržaška c.25	metka.novak@gmail.com	01/28/2015	Študent
2	Alenka	Rožanec	Viška c. 45	-	01/03/2015	Zaposleni
3	Tim	Prisel	-	tim.prisel@gmail.com	12/12/2014	Zaposleni

6.3.2.4.2 Primer: Izpiši vse knjige (ISBN, avtor, naslov)

```
SELECT isbn, avtor, naslov
FROM Knjiga;
```

Pri poizvedbi lahko navedemo samo določene stolpce, ki nas zanimajo (projekcija). Stolpci bodo prikazani v takšnem vrstnem redu kot jih navedemo v poizvedbi. Tabela 17 prikazuje rezultat poizvedbe.

Tabela 17: Rezultat poizvedbe primera 6.3.2.4.2

ISBN	AVTOR	NASLOV
9616046128	Tomaž Mohorič	Podatkovne baze
9788611171982	Tolkien, J. R. R.	Tja in spet nazaj
9788611172088	Tolkien, J. R. R.	Gospodar prstanov -Brat.

6.3.2.4.3 Primer: Izpiši vse člane, ki letos še niso plačali članarine - uporaba WHERE pogoja
V WHERE pogoju lahko uporabimo naslednje vrste testiranja izpolnjevanja pogoja:

- **Primerjava**: primerjamo vrednosti dveh izrazov
- **Obseg**: testiramo, ali je vrednost v podanem obsegu
- **Članstvo v množici**: testiramo, ali je vrednost enaka kateremu od elementov množice
- **Vzorec**: testiramo, ali se string ujema s podanim vzorcem
- **NULL (brez vrednosti):** testiramo, ali je stolpec brez vrednosti.

Podamo lahko več pogojev, ki jih ločimo z logičnimi operatorji in (**AND**), ali (**OR**), negacija (**NOT**).

```
SELECT st_izkaznica, ime, priimek, datum_placila
```

```
FROM clan
WHERE (EXTRACT (year from datum_placila)<2015) or (datum_placila IS NULL);
```

Pri tej poizvedbi dodamo pogoje z uporabo WHERE, in sicer so to tisti člani, ki so se pravkar včlanili in imajo polje datum_placila se prazno ali pa je v tem polju zapisan datum_placila iz katerega od preteklih let. Gre torej za pogoj, ki vsebuje dva dela, ločena z operatorjem ali (ang. or). V prvem delu pogoja testiramo, ali je datum_placila pred letom 2015, v drugem pa ali je datum_placila brez vrednosti. Rezultat poizvedbe so torej vrstice, ki ustrezajo prvemu ali drugemu pogoju (Tabela 18).

Tabela 18: Rezultat poizvedbe primera 6.3.2.4.3

ST_IZKAZNICA	IME	PRIIMEK	DATUM_PLACILA
4	Peter	Potočnik	-
6	Janez	Jamnik	-
7	Jaka	Jamnik	-
3	Tim	Prisel	12/12/2014

6.3.2.4.4 Primer: Izpiši vse ISBN številke iz tabele izposoja – uporaba ukaza DISTINCT

Ker je vsaka knjiga lahko izposojena večkrat imamo v izpisu duplikate. Ukaz DISTINCT nam omogoča izločiti duplikate.

Tabela 19 prikazuje rezultat poizvedbe izpisa ISBN številk knjig iz tabele Izposoja z duplikati in brez duplikatov (DISTINCT).

```
SELECT ISBN
FROM Izposoja;
```

```
SELECT DICTINCT ISBN
FROM Izposoja;
```

Tabela 19: Rezultat poizvedbe primera 6.3.2.4.4 z duplikati in brez duplikatov

ISBN
9616046128
9788611171982
9788611171982
9788611171982
9616046128

ISBN
9788611171982
9616046128

6.3.2.4.6 Primer: Izračunana polja

V SELECT stavku lahko kreiramo stolpce, ki so rezultat izračuna iz podatkov enega ali več drugih stolpcev. Pri izračunu lahko uporabljamo matematične operacije seštevanja, odštevanja, množenja in deljenja in seveda oklepaje, ki nam omogočajo kreiranje kompleksnejših izračunov. Podajamo enostaven primer uporabe izračuna zneska članarine z 20% popustom (Tabela 20). Ukaz **AS** nam omogoča poimenovanje novo nastalega stolpca.

```
SELECT vrsta_clana,znesek, znesek-znesek*0.2 AS Znesek_s_popustom
FROM clanarina;
```

Tabela 20: Rezultat poizvedbe primera 6.3.2.4.6

VRSTA_CLANA	ZNESEK	ZNESEK_S_POPUSTOM
Študent	10	8
Zaposleni	30	24
Dijak	10	8
Upokojenec	10	8
Otrok	0	0

6.3.2.4.7 Primer: Sortiranje izpisa podatkov

Primer prikazuje izpis vseh knjig urejenih po naslovu. Za sortiranje izpisa podatkov uporabljamo ukaz **ORDER BY.** Podamo stolpec ali več stolpcev po katerih želimo urediti izpis ter povemo še smer sortiranja: naraščajoče (**ASC** je privzeto) ali padajoče (**DESC**).

```
SELECT naslov, avtor, ISBN
FROM knjiga
ORDER BY naslov;
```

Tabela 21: Rezultat poizvedbe primera 6.3.2.4.7

NASLOV	AVTOR	ISBN
Gospodar prstanov -Brat.	Tolkien, J. R. R.	9788611172088
Podatkovne baze	Tomaž Mohorič	9616046128
Tja in spet nazaj	Tolkien, J. R. R.	9788611171982

POIZVEDBE NAD VEČ TABELAMI

Če želimo poizvedovati nad več tabelami, moramo dodati ukaze za povezovanje tabel med seboj. Pri tem imamo več možnosti. Najenostavnejši način povezovanja je navedba povezovalnih stolpcev v WHERE pogoju kot prikazuje primer v nadaljevanju.

6.3.2.4.8 Primer: Izpis vseh izposoj za knjige

Ker imamo v tabeli Izposoja samo datum izposoje, datum vrnitve in ISBN knjige, želimo pa izpisati tudi avtorja in naslov, moramo s tabelo Izposoja povezati še tabelo Knjiga. Tabeli sta povezani preko stolpca ISBN.

```
SELECT knjiga.isbn, knjiga.naslov, knjiga.avtor, izposoja.datum_izposoje, datum_vrnitve
FROM knjiga, izposoja
WHERE knjiga.isbn=izposoja.isbn
```

Tabela 22: Rezultat poizvedbe primera 6.3.2.4.8

ISBN	NASLOV	AVTOR	DATUM_IZPOSOJE	DATUM_VRNITVE
9788611171982	Tja in spet nazaj	Tolkien, J. R. R.	01/22/2015	-
9616046128	Podatkovne baze	Tomaž Mohorič	01/05/2015	01/20/2015
9788611171982	Tja in spet nazaj	Tolkien, J. R. R.	01/01/2014	02/20/2014
9788611171982	Tja in spet nazaj	Tolkien, J. R. R.	01/01/2015	01/20/2015
9616046128	Podatkovne baze	Tomaž Mohorič	01/22/2015	-

SQL standard omogoča tudi druge, alternativne načine povezovanja tabel v poizvedbah kot podaja naslednja sintaksa.

SELECT ...
FROM table1 INNER JOIN table2
ON table1.column1 = table2.column1
AND table1.column2 = table2.column2;

SELECT ...
FROM table1 INNER JOIN table2
USING (column1, column2);

6.3.2.4.9 Primer: Izpiši vse člane, ki letos še niso plačali članarine skupaj z zneski.
Znesek je odvisen od vrste člana in je zapisan v tabeli clanarina. Za povezovanje uporabimo ukaz INNER JOIN.

```
SELECT st_izkaznica, ime, priimek, datum_placila, clanarina.znesek
FROM clan INNER JOIN clanarina
ON clan.vrsta_clana = clanarina.vrsta_clana
WHERE (EXTRACT (year from datum_placila)<2015) OR (datum_placila IS NULL);
```

Tabela 23: Rezultat poizvedbe primera 6.3.2.4.9

ST_IZKAZNICA	IME	PRIIMEK	DATUM_PLACILA	ZNESEK
4	Peter	Potočnik	-	10
6	Janez	Jamnik	-	10
7	Jaka	Jamnik	-	10
3	Tim	Prisel	12/12/2014	30

6.3.2.4.10 Primer: Uporaba levega povezovanja (LEFT JOIN)
Izpis knjig z izposojami, pri čemer želimo prikazati tudi knjige, ki še niso bile nikoli izposojene. V izpisu tako vidimo tudi knjigo Gospodar prstanov, ki še ni bila nikoli izposojena (stolpec datum_izposoje nima podatka).

```
SELECT knjiga.ISBN, knjiga.avtor, knjiga.naslov, izposoja.datum_izposoje,
izposoja.datum_vrnitve
FROM knjiga LEFT JOIN izposoja ON knjiga.ISBN=izposoja.ISBN
```

Rezultat poizvedbe je enak tudi, če uporabimo desno povezovanje (RIGHT JOIN), vendar tokrat od tabele Izposoja na tabelo Knjiga.

```
SELECT knjiga.ISBN, knjiga.avtor, knjiga.naslov, izposoja.datum_izposoje,
izposoja.datum_vrnitve
FROM izposoja RIGHT JOIN knjiga ON knjiga.ISBN=izposoja.ISBN
```

Tabela 24: Rezultat poizvedbe primera 6.3.2.4.10

ISBN	AVTOR	NASLOV	DATUM_IZPOSOJE	DATUM_VRNITVE
9616046128	Tomaž Mohorič	Podatkovne baze	01/05/2015	01/20/2015
9616046128	Tomaž Mohorič	Podatkovne baze	01/22/2015	-
9788611171982	Tolkien, J. R. R.	Tja in spet nazaj	01/01/2015	01/20/2015
9788611171982	Tolkien, J. R. R.	Tja in spet nazaj	01/22/2015	-
9788611171982	Tolkien, J. R. R.	Tja in spet nazaj	01/01/2014	02/20/2014
9788611172088	Tolkien, J. R. R.	Gospodar prstanov -Brat.	-	-

UPORABA AGREGATNIH FUNKCIJ

- COUNT: vrne število vrednosti v določenem stolpcu
- SUM: vrne seštevek vrednosti v določenem stolpcu
- AVG: vrne povprečje vrednosti v določenem stolpcu
- MIN: vrne najmanjšo vrednost v določenem stolpcu
- MAX: vrne največjo vrednost v določenem stolpcu

6.3.2.4.11 Primer: Preštej število vseh knjig knjižnice

```
SELECT COUNT (isbn) AS Število_vseh_knjig
FROM knjiga
```

Tabela 25: Rezultat poizvedbe primera 6.3.2.4.11

ŠTEVILO_VSEH_KNJIG
3

6.3.2.4.12 Primer: Izpiši znesek, ki ga je knjižnica letos prejela s članarinami

```
SELECT SUM(znesek)
FROM clan INNER JOIN clanarina
ON clan.vrsta_clana = clanarina.vrsta_clana
WHERE (EXTRACT (year from datum_placila)=2015);
```

Rezultat poizvedbe pokaže, da je knjižnica do sedaj prejela 40€ članarine.
Tabela 26: Rezultat poizvedbe primera 6.3.2.4.12

SUM(ZNESEK)
40

6.3.2.4.13 Primer: Izpiši najvišji znesek članarine

```
SELECT MAX(znesek)
FROM clanarina;
```

Rezultat poizvedbe pokaže, da je najvišji znesek članarine 30€.

Tabela 27: Rezultat poizvedbe primera 6.3.2.4.13

MAX(ZNESEK)
30

6.3.2.4.14 Primer: Izpiši najnižji znesek članarine

```
SELECT MIN(znesek) AS "Minimalni znesek"
FROM clanarina;
```

Rezultat poizvedbe pokaže, da je najnižji znesek članarine 0 €.

Tabela 28: Rezultat poizvedbe primera 6.3.2.4.14

Minimalni znesek
0

6.3.2.4.15 Primer: Preštej število izposoj za posameznega člana, izpiši padajoče po številu izposoj

```
SELECT clan.priimek, clan.ime, COUNT(izposoja.isbn) AS ST_izposoj
FROM izposoja, clan
WHERE izposoja.st_izkaznica=clan.st_izkaznica
GROUP BY clan.priimek, clan.ime
ORDER BY 3 DESC
```

Uporabimo tudi ukaz GROUP BY, ki podatke o izposoji grupira po članu in nato sešteje število izposoj za posameznega člana. Ukaz ORDER BY nam izpis uredi tako, da je na prvem mestu izpisan član z največ izposojami.

Tabela 29: Rezultat poizvedbe primera 6.3.2.4.15

PRIIMEK	IME	ST_IZPOSOJ
Novak	Metka	2
Prisel	Tim	1
Potočnik	Peter	1
Rožanec	Alenka	1

HAVING sklop je namenjen uporabi v kombinaciji z GROUP BY kot omejitev skupin, ki se lahko pojavijo v rezultatu. Deluje podobno kot WHERE in sicer:

- WHERE filtrira posamezne vrstice
- HAVING filtrira skupine.

Stolpci, ki so navedeni v HAVING sklopu, morajo biti tudi v SELECT sklopu ali v agregatih.

6.3.2.4.16 Primer: Izpiši le tiste člane, ki imajo več kot eno izposojo

```
SELECT clan.priimek, clan.ime, COUNT(izposoja.isbn)
FROM izposoja, clan
WHERE izposoja.st_izkaznica=clan.st_izkaznica
GROUP BY clan.priimek, clan.ime
HAVING COUNT(izposoja.isbn)>1;
```

Tabela 30: Rezultat poizvedbe primera 6.3.2.4.16

PRIIMEK	IME	COUNT(IZPOSOJA.ISBN)
Novak	Metka	2

DELO S NIZI

Za delo z nizi (tip STRING) je zelo koristna uporaba operatorja LIKE. Pri tem uporabljamo dva posebna znaka **(velja za podatkovno bazo Oracle, na kateri so narejeni spodnji primeri)**:

- _ nadomešča natanko en znak,
- % nadomešča poljubno število znakov niza.

Pri tem moramo paziti tudi na velike in male črke, saj jih PB Oracle razlikuje: P≠p. Pri delu z nizi vedno uporabimo tudi posebna znaka narekovajev.

V MS Accessu se v povezavi z operatorjem LIKE uporabljata:

- ? nadomešča natanko en znak,
- * nadomešča poljubno število znakov niza.

6.3.2.4.17 Primer: Izpiši vse člane, katerih priimek se začne na črko P

```
SELECT ime, priimek
FROM Clan
WHERE Priimek LIKE 'P%';
```

Tabela 31: Prikaz vseh članov (levo) in rezultat poizvedbe (desno) primera 6.3.2.4.17

IME	PRIIMEK
Peter	Potočnik
Janez	Jamnik
Jaka	Jamnik
Metka	Novak
Alenka	Rožanec
Tim	Prisel

IME	PRIIMEK
Peter	Potočnik
Tim	Prisel

6.3.2.4.18 Primer: Izpiši vse člane, ki imajo kjerkoli v imenu črko e

```
SELECT ime, priimek
FROM Clan
WHERE Ime LIKE '%e%'
```

Tabela 32: Rezultat poizvedbe primera 6.3.2.4.18

IME	PRIIMEK
Peter	Potočnik
Janez	Jamnik
Metka	Novak
Alenka	Rožanec

6.3.2.4.19 Primer: Izpiši vse člane, ki imajo v priimku na drugem mestu črko o in v imenu na tretjem mestu črko t (ostale črko so poljubne)

```
SELECT ime, priimek
FROM Clan
WHERE Priimek LIKE '_o%' AND Ime LIKE '__t%';
```

Tabela 33: Rezultat poizvedbe primera 6.3.2.4.19

IME	PRIIMEK
Peter	Potočnik
Metka	Novak

UPORABA OPERATORJA IN /NOT IN

Primeri v nadaljevanju so podani nad domeno nepremičninske agencije, ki vsebuje tabeli Napremicnina (
Tabela 34) in Lastnik (Tabela 35). Relacijski shemi pa sta naslednji:

Nepremicnina(IDNep, Naslov, Mesto, St_sob,Visina_najem, #Lastnik)
Lastnik (IDLastnik, Naslov_last, Mesto_last,Telefon,ImePriimek)

Tabela 34: Vsebina tabele Nepremicnina

IDNEP	NASLOV	MESTO	STSOB	VISINA_NAJEM	LASTNIK
41	Aškerčeva 15	Maribor	4	350	42
61	Tomažičeva 2	Ljubljana	2	330	61
1	Viška c. 25	Ljubljana	3	500	1
24	Aljaževa 10	Maribor	3	300	41
21	Tržaška c.25	Ljubljana	50	10000	21
22	Tržaška c. 100	Ljubljana	5	700	41
23	Tomažičeva 10	Ljubljana	2	350	41

Tabela 35: Vsebina tabele Lastnik

IDLASTNIK	NASLOV_LAST	MESTO_LAST	TELEFON	IMEPRIIMEK
61	Rozmanova 35	Koper	-	Tone Hrovat
1	Viška c.25	Ljubljana	51233367	Alenka Rožanec
21	Tržaška c.25	Ljubljana	-	Fakulteta za elektrotehniko
41	Mariborska u. 20	Maribor	-	Marko Skače
42	Vinska c. 100	Maribor	-	Tone Novak

6.3.2.4.20 Primer: Uporaba operatorja IN

Članstvo določene vrednosti v množici testiramo z uporabo operatorja IN oz. NOT IN.

Napišimo poizvedbo z uporabo operatorja IN, ki izpiše dvosobne in trosobne tiste nepremičnine iz tabele Nepremicnina.

```
SELECT IDNep, Naslov, Mesto, STSOb
FROM Nepremicnina
WHERE STSob IN (2,3)
```

Poizvedbo lahko napišemo tudi z uporabo operatorja OR na naslednji način:

```
SELECT IDNep, Naslov, Mesto, STSOb
FROM Nepremicnina
WHERE STSob=2 OR STSob=3;
```

V primeru velikega števila pogojev, je uporaba operatorja IN učinkovitejša.

Tabela 36: Rezultat poizvedbe primera 6.3.2.4.20

IDNEP	NASLOV	MESTO	STSOB
61	Tomažičeva 2	Ljubljana	2
1	Viška c. 25	Ljubljana	3
24	Aljaževa 10	Maribor	3
23	Tomažičeva 10	Ljubljana	2

6.3.2.4.21 Primer: Uporaba operatorja NOT IN

Izpišimo še vse večje nepremičnine (vse, razen enosobnih in dvosobnih).

```
SELECT IDNep, Naslov, Mesto, STSOb
FROM Nepremicnina
WHERE StSob NOT IN (1,2);
```

Tabela 37: Rezultat poizvedbe primera 6.3.2.4.21

IDNEP	NASLOV	MESTO	STSOB
41	Aškerčeva 15	Maribor	4
1	Viška c. 25	Ljubljana	3
24	Aljaževa 10	Maribor	3
21	Tržaška c.25	Ljubljana	50
22	Tržaška c. 100	Ljubljana	5

MNOŽICE

Jezik SQL vsebuje naslednje operatorje za delo z množicami:

- Unijo (ang. UNION): vrne vrstice, ki predstavljajo unijo dveh tabel brez duplikatov
- Unijo z duplikati (ang. UNION ALL): vrne vrstice, ki predstavljajo unijo dveh tabel z duplikati
- Presek (ang. INTERSECT): vrne vrstice, ki se pojavijo v obeh tabelah, torej tvorijo presek.
- Razliko (ang. MINUS): vrne vrstice, ki so v prvi in niso v drugi tabeli.

Da lahko izvajamo naštete operacije, morata tabeli A in B biti skladni (domene atributov morajo biti enake).

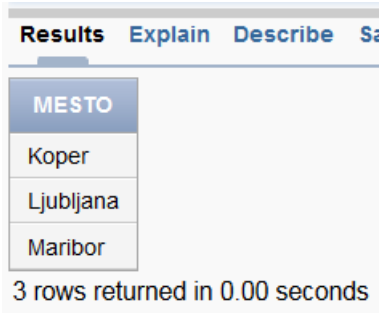
6.3.2.4.22 Unija brez duplikatov (UNION)

Izpiši vsa mesta (brez duplikatov), kjer se nahaja nepremičnina, **ali** kjer je doma vsaj en lastnik.

```
SELECT mesto
FROM Nepremicnina
UNION
SELECT mesto_last
FROM Lastnik;
```

Prvi SELECT stavek vrne vsa mesta, kjer se nahajajo nepremičnine, drugi SELECT stavek pa mesta, kjer se nahaja vsaj en lastnik. Vmes vstavimo ukaz za UNION, s čimer dobimo unijo vseh mest iz obeh tabel brez duplikatov. Tako se izpišejo mesta Koper, Ljubljana in Maribor.

Tabela 38: Rezultat poizvedbe primera 6.3.2.4.22



Results Explain Describe Sa

MESTO
Koper
Ljubljana
Maribor

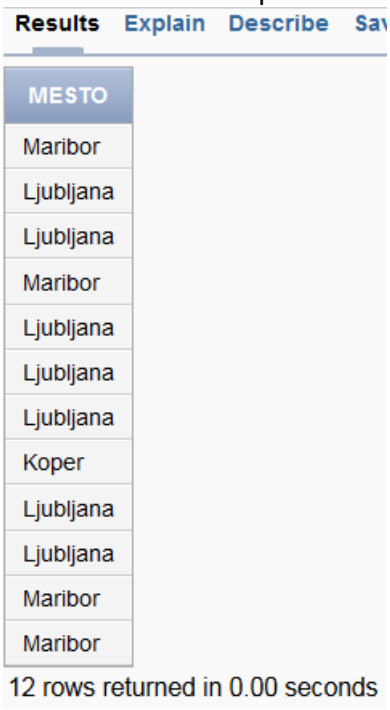
3 rows returned in 0.00 seconds

6.3.2.4.23 Unija z duplikati (UNION ALL)

Če uporabimo ukaz UNION ALL dobimo vsa mesta, ki se nahajajo v eni ali drugi tabeli, skupaj 12 mest, torej so ista mesta zapisana večkrat.

```
SELECT mesto
FROM Nepremicnina
UNION ALL
SELECT mesto_last
FROM Lastnik;
```

Tabela 39: Rezultat poizvedbe primera 6.3.2.4.23



Results Explain Describe Sav

MESTO
Maribor
Ljubljana
Ljubljana
Maribor
Ljubljana
Ljubljana
Ljubljana
Koper
Ljubljana
Ljubljana
Maribor
Maribor

12 rows returned in 0.00 seconds

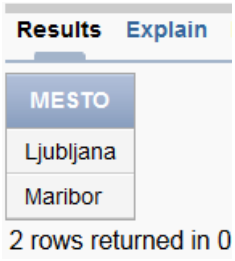
6.3.2.4.24 Presek (INTERSECT)
Izpišimo mesta, kjer se nahaja vsaj ena nepremičnina **in** je doma vsaj en lastnik.

```
SELECT mesto
FROM Nepremicnina
```

```
INTERSECT
SELECT mesto_last
FROM Lastnik;
```

Prvi SELECT stavek vrne mesta nepremičnin (to sta Ljubljana in Maribor), drugi SELECT pa mesta, kjer živijo lastniki (Koper, Ljubljana in Maribor). Z ukazom za presek (ukaz INETERSECT) dobimo presek obeh množic in končni rezultat, ki je prikazan spodaj.

Tabela 40: Rezultat poizvedbe primera 6.3.2.4.24



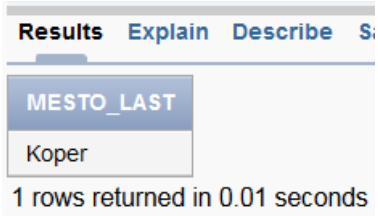
6.3.2.4.25 Razlika (MINUS)

Izpišimo mesta, kjer je doma vsaj en lastnik, a ni nobene nepremičnine. Gre za razliko mest lastnikov in nepremičnin.

```
SELECT mesto_last
FROM Lastnik
MINUS
SELECT mesto
FROM Nepremicnina;
```

Prvi SELECT stavek vrne mesta lastnikov (Koper, Ljubljana in Maribor), drugi SELECT pa mesta nepremičnin (Ljubljana in Maribor). Z ukazom za razliko (MINUS) od mest lastnikov odštejemo mesta nepremičnin in dobimo končni rezultat, ki je prikazan spodaj.

Tabela 41: Rezultat poizvedbe primera 6.3.2.4.25



VGNEZDENE POIZVEDBE

V jeziku SQL lahko med seboj gnezdimo tudi več poizvedb. Nekateri SQL stavki imajo tako lahko vgnezdene SELECT stavke, ki jih imenujemo podpoizvedbe (ang. subquery or nested query). Vgnezdeni SELECT stavki se lahko uporabijo v WHERE ali HAVING sklopih drugega SELECT stavka, lahko pa tudi v INSERT, UPDATE in DELETE stavkih. Vgnezdeni SELECT stavki ne smejo uporabljati ORDER BY sklopa.

Pri vgnezdenih stavkih lahko uporabimo naslednje SQL operatorje:

- ANY, SOME: primerja vrednost z vsaj eno vrednostjo seznama. Pogoj je izpolnjen, če vsaj ena vrednost iz seznama ustreza pogoju.
- ALL: primerja vrednost z vsemi vrednostmi seznama. Pogoj je izpolnjen, če vse vrednosti iz seznama ustrezajo pogoju.
- EXSISTS in NOT EXISTS: vračata vrednosti true in false. EXSISTS vrača true, če vgnezdeni select vrne vsaj eno vrstico. NOT EXISTS pa vrne true, kadar je rezultat vgnezdene poizvedbe prazen.

Poznamo tri tipe vgnezdenih poizvedb:

- **Skalarna poizvedba**: vrne en sam stolpec in eno samo vrstico, oziroma eno samo vrednost (primer 6.3.2.4.26).
- **Tabelarična poizvedba:** vrne več vrstic, lahko tudi več stolpcev. Takšno poizvedbo uporabimo, ko potrebujemo tabelo vrednosti, pogosto v povezavi z operatorjem IN (primer 6.3.2.4.27).
- **Vrstična poizvedba:** vrne več stolpcev, vendar eno samo vrstico.

6.3.2.4.26 Primer vgnezdene poizvedbe nad tabelo Nepremicnina
Izpiši nepremičnine, ker je cena najema višja od povprečne cene najema vseh nepremičnin v tabeli. Vgnezdena poizvedba vrne povprečno ceno najema, glavna poizvedba pa izpiše vse tiste nepremičnine, kjer je visina_najem>povprečne cene.

```
SELECT *
FROM nepremicnina
WHERE visina_najem>(
    SELECT AVG(Visina_najem)
    FROM Nepremicnina)
```

Povprečna cena najema v tabeli Nepremicnina je 1790 (glej podatke v Tabela 34).

Tabela 42: Rezultat poizvedbe primera 6.3.2.4.26

IDNEP	NASLOV	MESTO	STSOB	VISINA_NAJEM	LASTNIK
21	Tržaška c.25	Ljubljana	50	10000	21

6.3.2.4.27 Primer: Tabelarična vgnezdena poizvedba
Izpiši podatke o nepremičninah, kjer je njen lastnik v množici Mariborčanov (uporaba operatorja IN). Vgnezdeni SELECT stavek kreira množico Mariborčanov (id 41, 42). Zunanji SELECT stavek pa izpiše podatke o nepremični, če je vrednost atributa Lastnik (gre za id lastnika iz tabele Lastnik) v množici Mariborčanov.

```
SELECT IDNep, Naslov, Mesto, STSOb
FROM Nepremicnina
WHERE Lastnik IN (
  SELECT IDLastnik
  FROM Lastnik
  WHERE Mesto_last='Maribor')
```

Tabela 43: Rezultat vgnezedene poizvedbe (levo) in glavne poizvedbe(desno) primera 6.3.2.4.27

IDLASTNIK
41
42

IDNEP	NASLOV	STSOB
61	Tomažičeva 2	2
1	Viška c. 25	3
21	Tržaška c.25	50
22	Tržaška c. 100	5
23	Tomažičeva 10	2

6.3.2.4.28 Primer: Uporaba operatorja ANY

V vgnezdenih SELECT stavkih, ki vračajo en sam stolpec, lahko uporabljamo operator ANY. Namesto ANY lahko uporabljamo tudi SOME.

Izpišimo nepremičnine, katerih najemnina je višja od najemnine vsaj ene ljubljanske nepremičnine.

```
SELECT IDNep, visina_najem,Naslov, Mesto,Lastnik
FROM Nepremicnina
WHERE visina_najem>ANY(
   SELECT visina_najem
   FROM Nepremicnina
   WHERE Mesto='Ljubljana');
```

Tabela 44: Rezultat vgnezedene poizvedbe (levo) in glavne poizvedbe(desno) primera 6.3.2.4.28

VISINA_NAJEM
330
500
10000
700
350

IDNEP	VISINA_NAJEM	NASLOV	MESTO	LASTNIK
21	10000	Tržaška c.25	Ljubljana	21
22	700	Tržaška c. 100	Ljubljana	41
1	500	Viška c. 25	Ljubljana	1
41	350	Aškerčeva 15	Maribor	42
23	350	Tomažičeva 10	Ljubljana	41

6.3.2.4.29 Primer: Uporaba operatorja ALL

Izpišimo nepremičnine, katerih najemnina je višja od najemnine vseh ljubljanskih nepremičnin. Rezultat te poizvedbe je prazen, saj takšna nepremičnina ne obstaja.

Izpišimo nepremičnine, katerih najemnina je višja od najemnine vseh mariborskih nepremičnin. Imamo dve mariborski nepremičnini. Vidimo, da so tri ljubljanske nepremičnine dražje od obeh mariborskih.

```
SELECT IDNep, visina_najem,Naslov, Mesto
FROM Nepremicnina
WHERE visina_najem>ALL(
  SELECT visina_najem
  FROM Nepremicnina
  WHERE Mesto='Maribor');
```

Tabela 45: Rezultat vgnezedene poizvedbe (levo) in glavne poizvedbe(desno) primera 6.3.2.4.29

VISINA_NAJEM
350
300

IDNEP	VISINA_NAJEM	NASLOV	MESTO
1	500	Viška c. 25	Ljubljana
22	700	Tržaška c. 100	Ljubljana
21	10000	Tržaška c.25	Ljubljana

6.3.2.4.30 Primer: Uporaba operatorja EXISTS

Operatorja EXISTS in NOT EXISTS se uporabljata izključno v vgnezdenih poizvedbah (ang. subqueries). Vračata vrednosti true oz. false. EXSITS vrača true, če vgnedena poizvedba vrne vsaj eno vrstico. NOT EXISTS je nasproten operatorju EXSITS. Ker EXSITS in NOT EXISTS vračata le vrednosti true in false, lahko vgnezdena poizvedba vrača poljubno število stolpcev.

Imamo relacijo Zaposleni, v kateri hranimo podatke o zaposlenih ter tudi podatek o nadrejenem delavcu z naslednjo relacijsko shemo in naslednjimi podatki:

Tabela 46: Vsebina tabele Zaposleni

EDIT	DS	DELOVNO_MESTO	SIFRA_SEFA	IME_PRIIMEK
	1	Šef kuhinje	-	Marko Šef
	30	Kuhar 1	1	Tine Jaklič
	20	Kuhar 1	1	Janko Novak

Vrni vse zaposlene, ki so nadrejeni vsaj enemu delavcu.

```
SELECT zap.DS, zap.Ime_priimek
FROM Zaposleni zap
WHERE EXISTS
(SELECT sef.DS
FROM Zaposleni sef
WHERE sef.Sifra_sefa = zap.DS);
```

Tabela 47: Rezultat poizvedbe primera 6.3.2.4.30

DS	IME_PRIIMEK
1	Marko Šef

6.3.2.4.31 Primer: Uporaba operatorja NOT EXISTS
Vrni vse zaposlene, ki niso nadrejeni nikomur.

```
SELECT zap.DS, zap.Ime_priimek
FROM Zaposleni zap
WHERE NOT EXISTS
(SELECT sef.DS
FROM Zaposleni sef
WHERE sef.Sifra_sefa = zap.DS);
```

Tabela 48: Rezultat poizvedbe primera 6.3.2.4.31

DS	IME_PRIIMEK
20	Janko Novak
30	Tine Jaklič

6.4 QBE

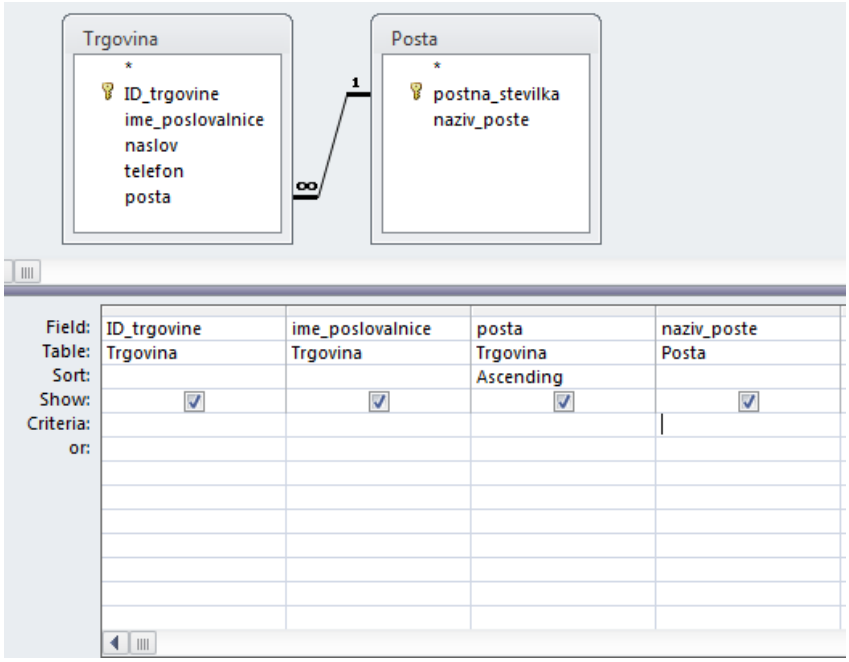
QBE (Query by example) je enostaven grafični poizvedovalni jezik, ki v ozadju naše poizvedbe preslika v SQL. QBE bo na kratko predstavljen z uporabo orodja MS Access, ki je podrobneje opisano tudi v poglavju 8.4.

Poizvedbo nad podatkovno bazo trgovine prikazujejo naslednji primeri. Poizvedbo v MS Accessu lahko kreiramo z uporabo ukaza Create ->QueryDesign.

Primeri v nadaljevanju so izdelani nad podatkovno bazo z naslednjo relacijsko shemo:

Trgovina (ID_trgovine, ime_poslovalnice, naslov, telefon, #posta)
Posta (postna_stevilka, naziv poste)
Zaposleni (ID_zaposlenega, ime, priimek, polozaj, naslov, #posta)
Postavka (ID_postavke, #racun, #koda, kolicina)
Račun (st_racuna, datum, #blagajnik, vrsta_placila, #poslovalnica,…)
Izdelek (koda, naziv, klasifikacija,opis, cena, stopnja DDV)

Slika 33: Delovna površina za kreiranje QBE poizvedb orodja MS Access



Delovno površino sestavljata zgornji in spodnji del. Na zgornji del dodamo tabele, nad katerimi želimo izdelati poizvedbo. Na spodnji del, ki je oblikovan kot preglednica, pa nato »nanašamo« našo poizvedbo. Preglednica vsebuje naslednje osnovne vrstice:

- **Field** (polje): dodajamo stolpce iz zgoraj prikazanih tabel.
- **Table** (tabela): prikazuje tabelo, iz katere je posamezni stolpec.
- **Sort** (urejenost): izberemo sortiranje po določenem stolpcu (naraščajoče ali padajoče).
- **Show** (prikaz): s kljukico označimo stolpec, če ga želimo izpisati.
- **Criteria** (pogoji): vstavimo pogoje. Če sta dva pogoja v isti vrstici, je med njima operator IN**.**
- **Or** (pogoji ločeni z ali): dodamo pogoje, ločene z operatorjem ali.

6.4.1 Enostavne poizvedbe

6.4.1.1 Primer izpisa vseh trgovin

S poizvedbo želimo izpisati vse trgovine s pripadajočimi nazivi pošt. V tabeli Trgovina so shranjeni podatki o trgovini in poštna številka. V tabeli Posta pa poštna številka in naziv pošte oz. ime kraja (npr. 8000 Novo Mesto). Tabeli Trgovina in Posta sta povezani preko poštne številke in sicer: Trgovina.posta=Posta.postna_stevilka. Pošte.

O trgovini želimo izpisati: Id_trgovine, ime_poslovalnice, posta (iz tabele Trgovina) in naziv_poste (iz tabele Posta), zato jih povlečemo v poizvedbo. V vrstici Show morajo biti ti stolpci označeni s kljukico. Če želimo, da bo izpis urejen po poštni številki, v stolpcu posta izberemo ukaz za sortiranje (izbrali smo Ascending za naraščajoče sortiranje). Poizvedbo prikazuje slika (Slika 33). Tabela 49 pa prikazuje rezultat poizvedbe.

Tabela 49: Rezultat poizvedbe primera 6.4.1.1

Oznaka trgo	ime_poslovalnice	posta	naziv_poste
3	Tretja	1002	Ljubljana
2	Za vogalom	2000	Maribor
1	Muca copatarica	3000	Celje

6.4.1.2 Primer dodajanja enega pogoja na poizvedbo
Želimo izpisati podatke le o trgovinah iz Celja. Pogoj dodamo v stolpec naziv_poste.

Slika 34: Poizvedba - izpis vseh trgovin iz Celja

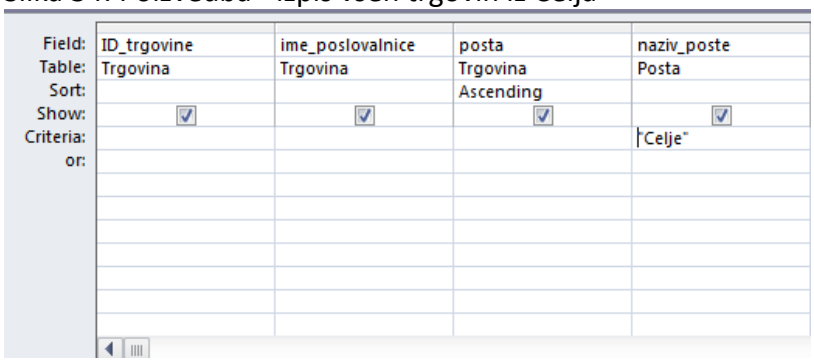


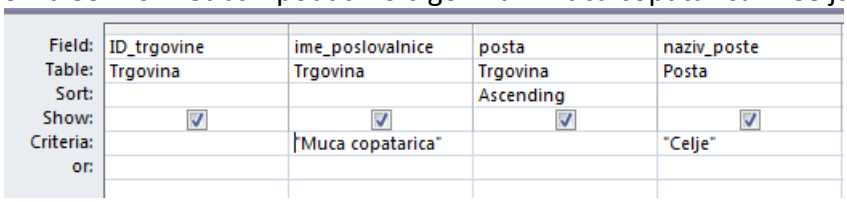
Tabela 50: Rezultat poizvedbe 6.4.1.2

trgovina | muca copatarica

Oznaka trgo	ime_poslovalnice	posta	naziv_poste
1	Muca copatarica	3000	Celje

6.4.1.3 Primer uporabe več pogojev (IN)
Želimo izpisati podatke o trgovinah z nazivom Muca copatarica iz Celja. Dodamo še pogoj v stolpec ime_poslovalnice.

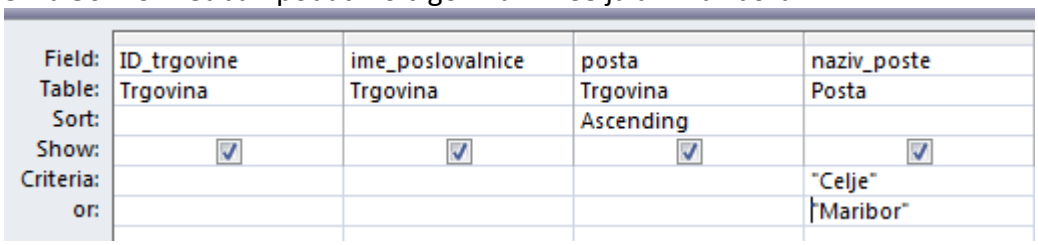
Slika 35: Poizvedba - podatki o trgovinah Muca copatarica iz Celja



Ker je v bazi le ena trgovina iz Celja je izpis enak predhodnemu (Tabela 50).

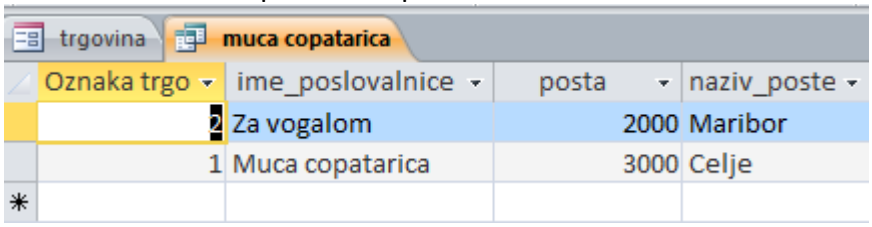
6.4.1.4 Primer uporabe več pogojev (ALI)
Želimo izpisati podatke o trgovinah, ki so iz Celja ali Maribora. Oba pogoja damo v stolpec naziv_poste. Rezultat poizvedbe prikazuje tabela (Tabela 51).

Slika 36: Poizvedba - podatki o trgovinah iz Celja ali Maribora



Field:	ID_trgovine	ime_poslovalnice	posta	naziv_poste
Table:	Trgovina	Trgovina	Trgovina	Posta
Sort:			Ascending	
Show:	☑	☑	☑	☑
Criteria:				"Celje"
or:				"Maribor"

Tabela 51: Rezultat poizvedbe primera 6.4.1.4



trgovina | muca copatarica

Oznaka trgo	ime_poslovalnice	posta	naziv_poste
2	Za vogalom	2000	Maribor
1	Muca copatarica	3000	Celje
*			

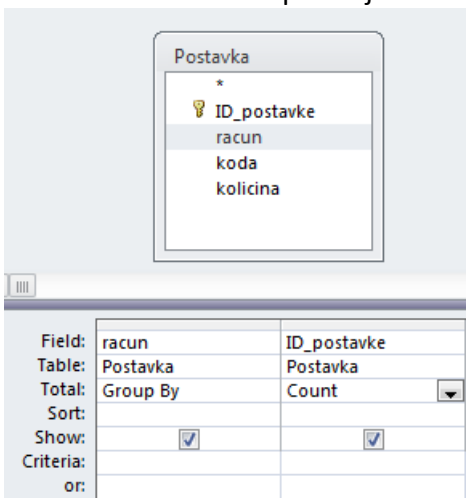
6.4.2 Uporaba agregatnih funkcij

Tudi QBE omogoča izvajanje agregatnih funkcij, ki smo jih spoznali že pri jeziku SQL. Za delo z agregatnimi funkcijami moramo vključiti dodatno vrstico s klikom na ukaz Totals.

6.4.2.1 Primer uporabe agregatne funkcije COUNT

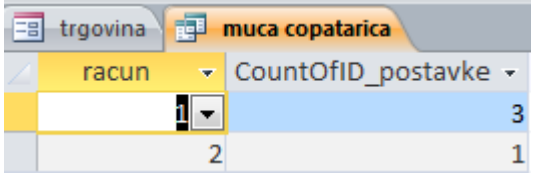
Želimo izpisati, koliko postavk (vrstic računa) je vseboval posamezen račun. Podatke zato grupiramo po številki računa (stolpec racun). Znotraj vsake grupe pa preštejemo postavke (ID_postavke). Ugotovimo, da je imel račun s številko 1 tri postavke, račun s številko 2 pa eno postavko.

Slika 37: Poizvedba – preštej število postavk na posameznem računu



Field:	racun	ID_postavke
Table:	Postavka	Postavka
Total:	Group By	Count
Sort:		
Show:	☑	☑
Criteria:		
or:		

Tabela 52: Rezultat poizvedbe primera 6.4.2.1



trgovina | muca copatarica

racun	CountOfID_postavke
1	3
2	1

6.4.2.2 Primer izračuna izraza in uporaba agregatne funkcije SUM

Želimo izračunati skupni znesek vsakega računa. Na računu imamo eno ali več postavk. Pri vsaki postavki je zapisana prodana količina. Cena artikla pa se nahaja v tabeli Artikel.

Najprej skušajmo izračunati vrednost posamezne postavke. To storimo tako, da dodamo nov stolpec, v katerega zapišemo izraz: **Znesek: [cena]*[kolicina],** pri čemer sta kolicina in cena atributa tabel Postavka oz. Artikel.

Slika 38: Poizvedba – izračun vrednosti posamezne postavke

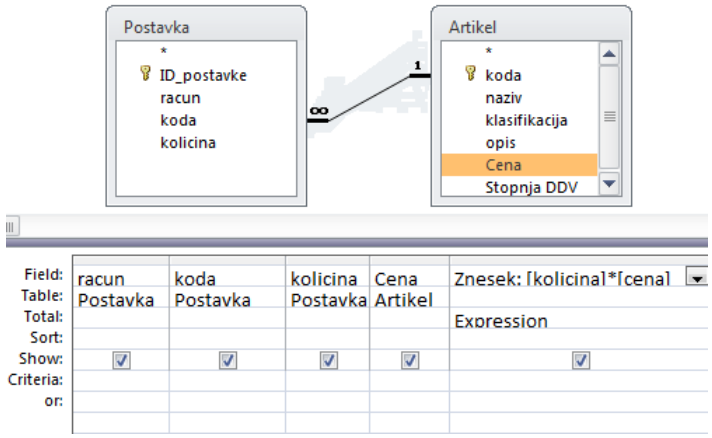
QBE	**SQL**
	SELECT Postavka.racun, Postavka.koda, Postavka.kolicina, Artikel.Cena, [kolicina]*[cena] AS Znesek FROM Artikel INNER JOIN Postavka ON Artikel.koda = Postavka.koda;

Tabela 53: Rezultat poizvedbe primera 6.4.2.2 (izračun vrednosti posamezne postavke)

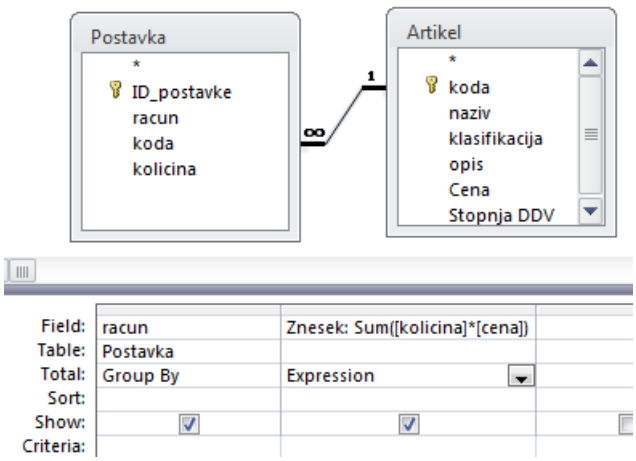
trgovina | Query1 | Skupni znesek računa

racun	koda	kolicina	Cena	Znesek
1	1234a-Lego kocke	5	35,00 €	175
2	1234a-Lego kocke	30	35,00 €	1050
1	6543g-Motorno olje	1	15,00 €	15
1	bc456g-Sobno kolo	7	250,00 €	1750

Sedaj pa bomo vrednosti postavk na istem računu le še sešteli. Izpisali bomo le številko računa ter skupno vrednost, zato stolpce, ki smo jih prej dodali, odstranimo. Dopolnimo poizvedbo z agregatno funkcijo seštevanja (Sum) na pravkar kreiranemu stolpcu Znesek. Tako dobimo končni rezultat. Ogledamo si še SQL stavek, ki se kreira v ozadju.

Slika 39: Poizvedba – izračun skupnega zneska posameznega računa

QBE



SQL

```
SELECT Postavka.racun, Sum([kolicina]*[cena]) AS Znesek
FROM Artikel INNER JOIN Postavka ON Artikel.koda = Postavka.koda
GROUP BY Postavka.racun;
```

Tabela 54: Rezultat poizvedbe primera 6.4.2.2 (izračun skupnega zneska posameznega računa)

trgovina | Query1 | Skupni znesek

racun	Znesek
1	1940
2	1050

Še več primerov poizvedb se nahaja v poglavju 8.4.2 Poizvedovanje z uporabo jezika QBE.

6.5 Izvajanje in optimizacija poizvedb

6.5.1 O izvajanju poizvedb

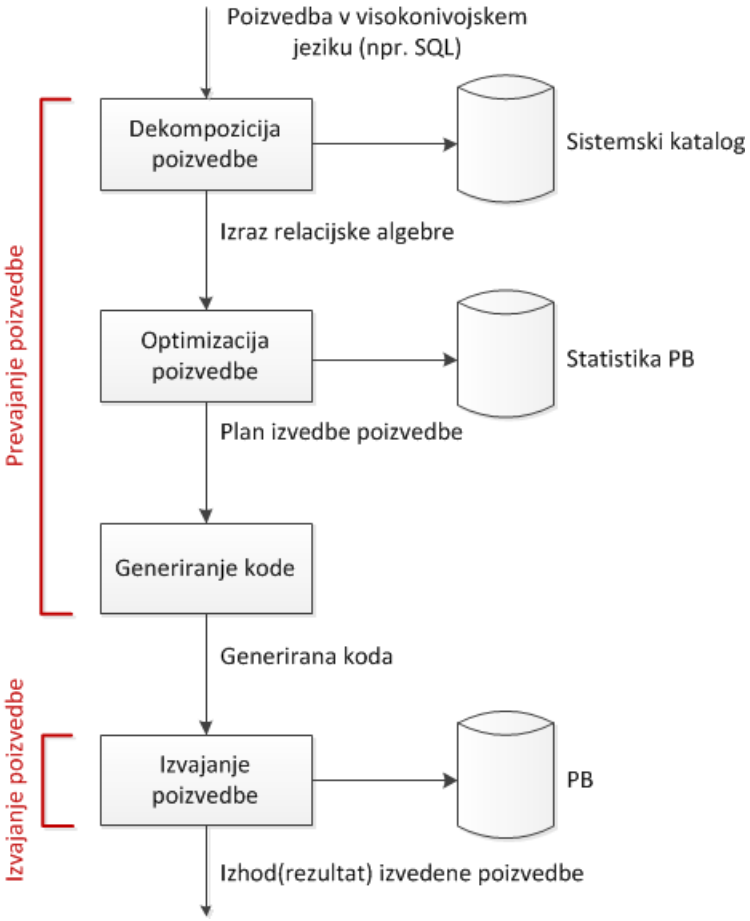
Cilj izvajanja poizvedbe je transformacija poizvedbe, zapisane v visokonivojskem poizvedovalnem jeziku (npr. SQL-u) v pravilno in učinkovito strategijo za izvedbo poizvedbe, v nizkonivojskem jeziku (ki implementira relacijsko algebro) in izvedba strategije za pridobitev potrebnih podatkov. **Izvedba poizvedbe** (ang. Query processing) vključuje dekompozicijo (analizo in validacijo sintakse) poizvedbe, optimizacijo, generiranje kode in izvedbo poizvedbe.

Slika 40 prikazuje faze izvedbe poizvedbe (povzeto po Connolly in Begg, 2005, str. 631-683).

Poizvedbo, zapisano v visokonivojskem jeziku, je možno transformirati v nizkonivojsko poizvedbo na različne načine. Z aktivnostjo optimizacije poizvedbe želimo najti najučinkovitejšo strategijo za izvedbo poizvedbe, z vidika porabe različnih računalniških virov. **Glavni cilj optimizacije je, da se poizvedba izvede čim hitreje.**

V splošnem ločimo **dinamično** in **statično** optimizacijo, ki imata vsaka svoje prednosti in slabosti, zato se v praksi navadno uporablja hibridni pristop. Za dinamično optimizacijo je značilno, da se dekompozicija in optimizacija poizvedbe ponovno izvedeta pred vsako izvedbo poizvedbe. Prednost je, da se optimizacija izvede na ažurnih podatkih statistke podatkovne baze. Slabost pa je čas, potreben za vsakokratno dekompozicijo in optimizacijo. Značilnost statične optimizacije je, da se dekompozicija in optimizacija izvedeta enkrat za posamezno poizvedbo, s čimer se odpravi čas vsakokratne izvedbe teh aktivnosti pred izvedbo poizvedbe, kar pri pogostem zaganjanju istih poizvedb lahko pomeni precejšen prihranek časa. Slabost tega načina pa je, da je poizvedba optimizirana glede na statistiko, ki je bila ažurna v času optimizacije, zato poizvedba čez nekaj časa lahko ni več optimalna. Hibridni pristop je, da se optimizacija izvede in shrani na začetku vsake seje.

Slika 40: Faze izvedbe poizvedbe



Vir: Connolly in Begg, 2005, str. 634.

6.5.2 *Dekompozicija poizvedbe*

Dekompozicija poizvedbe je prva faza izvedbe poizvedbe. Cilj je pretvorba visokonivojske poizvedbe v ukaze relacijske algebre ter preverjanje sintaktične in semantične pravilnosti. Dekompozicija obsega več aktivnosti.

Prva aktivnost je preverjanje sintakse visokonivojskega jezika (uporaba rezerviranih besed, tipov in podobno kot prevajalniki to preverjajo pri drugih programskih jezikih) ter preverjanje obstoja relacij in atributov v sistemskem katalogu podatkovne baze (torej ali smo v poizvedbi pravilno zapisali imena tabel in stolpcev). Poglejmo primer poizvedbe nad tabelo Clan, iz domene knjižnice, z naslednjo relacijsko shemo:

Clan (St_izkaznica, Ime, Priimek, Naslov, E_naslov, Datum_placila, #Vrsta_clana).

S poizvedbo želimo izpisati vse člane (St_izkaznice, Ime, Priimek), ki imajo status študenta. Pravilna poizvedba je:

```
SELECT ST_izkaznica, Ime, Priimek
FROM Clan
WHERE Vrsta_clana='Študent'
```

V primeru napačnega zapisa rezerviranih besed (npr. SELCT) dobimo naslednji izpis napake: invalid SQL statement. Enako napako dobimo tudi v primeru uporabe dvojnih narekovajem namesto enojnih. To napako dobimo tudi v primeru uporabe napačnih operatorjev glede na tip podatkov, na primer:

```
SELCT ST_izkaznica, Ime, Primek
FROM Član
WHERE Vrsta_clana> "Študent"
```

Operatorja > in < namreč ne moremo uporabljati v povezavi z atributi, ki hranijo nize znakov.

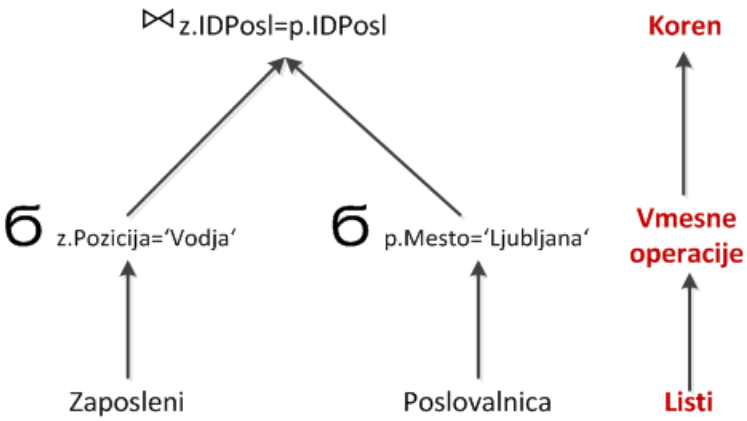
Če se npr. zmotimo pri navedbi atributa Priimek, dobimo izpis napake: "PRIMEK": invalid identifier, saj analizator ugotovi, da atributa s takšnim imenom ni v sistemskem katalogu. Če pa se zmotimo pri navedbi imena tabele in zapišemo Član namesto Clan, dobimo izpis napake: table or view does not exist.

Če je poizvedba sintaktično pravilna se prevede v ukaze relacijske algebre, navadno v obliko drevesa, ki je najprimernejša za nadaljnje aktivnosti. Imejmo tabeli Zaposleni (ID_zap, Ime, Priimek, Pozicija, #IDPosl) in Poslovalnica (IDPosl, Naziv, Naslov, Mesto) ter relacijo, da zaposleni dela v natanko eni poslovalnici (glej tudi Slika 48). Želimo izpisati vse vodje poslovalnic iz Ljubljane. SQL stavek se glasi:

```
SELECT *
FROM Zaposleni inner join Poslovalnica on Zaposleni.IDPosl= Poslovalnica.IDPosl
WHERE (Zaposleni.Pozicija='Vodja') AND (Poslovalnica.Mesto='Ljubljana')
```

Primer drevesa za navedeno poizvedbo prikazuje Slika 41. Poizvedba se izvede od listov proti korenu poizvedbe.

Slika 41: Primer drevesa relacijske algebre



Semantična analiza skuša najti nepravilnosti, npr. poizvedbe, ki nikoli ne vrnejo rezultata. Poizvedba, ki vrne vse člane, ki so dijaki ali študenti, je semantično pravilna.

```
SELECT ST_izkaznica, Ime, Priimek
FROM Clan
WHERE Vrsta_clana='Dijak' OR Vrsta_clana='Študent'
```

Podobna poizvedba, v kateri zamenjamo le operator OR z operatorjem ALI, pa semantično ni pravilna, saj ima vsak član dodeljeno natanko eno vrsto člana. Ne obstaja član, ki bi bil študent in dijak hkrati. Primer sintaktično nepravilne poizvedbe:

```
SELECT ST_izkaznica, Ime, Priimek
FROM Clan
WHERE Vrsta_clana='Dijak' AND Vrsta_clana='Študent'
```

Obvladovanje semantičnih nepravilnosti se med SUPB-ji precej razlikuje.

Nadalje se izvedejo še poenostavitve poizvedbe, pri čemer se uporabijo pravila Boolove algebre pri rabi operatorjev in, ali in negacije. Preverijo se tudi dostopne pravice uporabnika do elementov poizvedbe ter upoštevajo integritetne omejitve. Končna aktivnost je še prestrukturiranje poizvedbe.

6.5.3 Optimizacija poizvedbe

Optimizacija poizvedbe (ang. Query optimization) je aktivnost izbire najučinkovitejše strategije za izvedbo poizvedbe. Pri optimizaciji poizvedbe SUPB uporablja statistiko podatkovne baze (npr. podatke o relacijah, atributih, indeksih).

Za optimizacijo poizvedb se uporabljata dve glavni tehniki: **hevristična tehnika** in **tehnika ocene stroškov**. Prva tehnika uporablja hevristična pravila, ki omogočajo določitev optimalnega vrstnega reda izvajanja operacij relacijske algebre. Druga tehnika omogoča oceno porabe virov, predvsem dostopa do diska zaradi potrebe po branju podatkov in izbiro strategije, kjer je teh dostopov najmanj. Pri tem uporablja statistične podatke o podatkih v podatkovni bazi. SUPB Oracle za optimizacijo uporablja obe vrsti tehnik.

Hevristična tehnika omogoča pretvorbo manj učinkovitih izrazov relacijske algebre v bolj učinkovite. Tako lahko na primer ugotovimo, da je učinkoviteje najprej izvesti selekcijo v vsaki posamezni tabeli in nato rezultate povezati s stikom, kot obratno. Ker obstaja transformacijsko pravilo, ki omogoča zamenjavo operacij selekcije in stika, jih lahko izvedemo v vrstnem redu, ki je z vidika izvedbe učinkovitejši. Pri optimizaciji se tako lahko med drugim uporabijo naslednje hevristične strategije:

- **Izvedi operacije selekcije čim bolj zgodaj**: selekcija zmanjša številnost relacije (to je število n-teric relacije) ter s tem zmanjša nadaljnje procesiranje podatkov.
- **Izvedi operacije projekcije čim bolj zgodaj**: projekcija zmanjša stopnjo relacije (to je število stolpcev) ter s tem zmanjša nadaljnje procesiranje podatkov.
- **Samo enkrat izvedi enake izraze**: rezultati se shranijo in ponovno uporabijo.

Natančen opis vseh transformacijskih pravil in vseh strategij presega obseg učbenika in si jih bralec lahko prebere v (Connolly in Begg, 2005, str. 640-644).

Tehnika ocene stroškov temelji na statističnih podatkih o podatkovni bazi:

- številnost vsake osnovne relacije,
- število blokov za hranjenje relacije,
- število različnih vrednosti vsakega atributa,
- število nivojev vsakega večnivojskega indeksa itd.

Pri tem je pomembno, da so ti podatki v času ocene stroškov čim bolj sveži. Vzdrževanje ažurnih statističnih podatkov predstavlja dodatni problem. V primeru, da se statistični podatki osvežijo ob vsakem dodajanju, brisanju ali ažuriranju podatkovne baze, to seveda bazo dodatno obremenjuje in negativno vpliva na njeno odzivnost v času večjih obremenitev. Splošno sprejet pristop je osveževanje statističnih podatkov v času, ko je podatkovna baza najmanj obremenjena, npr. ponoči. Možen pristop je tudi, da uporabnikom prepustimo odgovornost za proženje osveževanja statistike, kadar se jim to zdi potrebno.

Na podlagi statističnih podatkov je možno oceniti, kakšne stroške predstavlja izvedba posamezne operacije relacijske algebre (selekcije, projekcije, stika…). Pri tem kot stroške praviloma ocenjujemo število blokov, ki jih je potrebno z diska prenesti v glavni pomnilnik, saj je to najpočasnejša operacija. Nadalje je ocena stroškov odvisna tudi od načina urejenosti posamezne relacije. Za operacijo selekcije so strategije oz. ocena stroškov različne, glede na fizično organiziranost podatkovne baze. Odvisno od fizične organiziranosti tako za iskanje n-teric, ki ustrezajo pogoju, uporabimo: zaporedno iskanje (neurejena datoteka, brez indeksa), binarno iskanje (urejena datoteka brez indeksa), pogoj enakosti na primarnem ključu, pogoj neenakosti na primarnem ključu, pogoj enakosti na sekundarnem indeksu (tipa cluster, B+-drevo…,) itd., za katere je ocena stroškov seveda različna. Podobno obstajajo različne strategije za druge operacije relacijske algebre.
Nadalje SUPB potrebuje tudi učinkovit algoritem za iskanje najučinkovitejše strategije. V primeru kompleksnejše poizvedbe je problem lahko dokaj zahteven. V primeru, da imam poizvedbo nad tremi relacijami (n=3), lahko stik med njimi realiziramo na 12 različnih načinov, za n=4 pa že na 120 načinov. V splošnem je za n relacij **$(2(n-1))!/(n-1)!$** različnih možnosti realizacije stika med njimi. Podrobnosti o posameznih strategijah in algoritmih za iskanje najučinkovitejše strategije izvajanja (ang. execution strategy) si bralec lahko prebere v (Connolly in Begg, 2005, str. 647-673).

Vprašanja za ponavljanje

1. Katere jezike za delo z relacijsko bazo poznate?
2. Kaj je SQL?
3. Katere so prednosti in katere slabosti standarda SQL?
4. Kateri dve skupini ukazov jezika SQL poznate?
5. Katere ukaze vsebuje skupina DDL in čemu so namenjeni?
6. Katere ukaze vsebuje skupina DML in čemu so namenjeni?
7. Najmanj katera dva dela mora obsegati SELECT stavek?
8. Katere ukaze za delo z množicami vsebuje SQL?
9. Kaj je QBE?
10. Kako sestavimo poizvedbo v jeziku QBE?
11. V kateri jezik se pred izvedbo poizvedba QBE prevede?
12. Katere agregatne funkcije poznate? Kaj omogočajo?
13. Katere faze obsega izvedba poizvedbe v SUPB?
14. Kaj je cilj dekompozicije poizvedbe? Katere značilnosti poizvedbe se v tej fazi preverijo?
15. Katere aktivnosti se v okviru faze dekompozicije še izvedejo?
16. Kaj je cilj optimizacije poizvedbe?
17. Na podlagi česa deluje hevristični pristop? Ali poznate katero od strategij tega pristopa?
18. Na podlagi česa deluje pristop ocene stroškov?
19. Katere statistične podatke navadno vsebuje sistemski katalog podatkovne baze?

Naloge

6.3.1 Nad podatkovno bazo študentskega IS (glej) izdelajte naslednje poizvedbe nad eno tabelo z uporabo jezika SQL DML:

a. Vsi zapisi in atributi iz tabele Predavatelj.
b. Vsi zapisi in atributi iz tabele Predmet.
c. Naziv, Ime in priimek iz tabele Predavatelj.
d. Vpisna št, emšo, datum rojstva, skupaj naslov in kraj iz tabele Student.
e. Predavatelji, ki se začnejo na črko M.
f. Študentje, ki živijo v Ljubljani in so rojeni po 1.1.1995.
g. Predmeti, ki se izvajajo v prvem ali drugem letniku v zimskem semestru, urejeni naraščajoče po nazivu.
h. Študentje, rojeni med 1.1.1990 in 1.1.2000 urejeni po datumu rojstva padajoče.
i. Študentje, ki imajo vpisan naslov in ima njihova vpisna številka 10 znakov, urejeni po imepriimek.

6.3.2 Nad podatkovno bazo študentskega IS (glej Slika 31**) izdelajte naslednje poizvedbe nad dvema ali več tabelami z uporabo jezika SQL DML:**

a. Nazivi predmetov, nazivi ter imena in priimki predavateljev, ki jih izvajajo, urejeno po nazivu predmeta in imenu, priimku predavatelja.
b. Nazivi predmetov, nazivi ter imena in priimki predavateljev, ki jih izvajajo, urejeno po nazivu predmeta in imenu, priimku predavatelja – TUDI če predmet nima predavatelja!
c. Nazivi predmetov, nazivi ter imena in priimki predavateljev, ki jih izvajajo, urejeno po imenu, priimku predavatelja – TUDI če predavatelj nima predmeta!
d. Seznam vseh študentov, s pripadajočimi ocenami za opravljene predmete, nazivi predmetov in predavatelji.

e. Enako kot d, vendar samo ocene predmetov prvega letnika, ki imajo oceno med 7 in 9.

6.3.3 Nad podatkovno bazo študentskega IS (glej Slika 31**) izdelajte naslednje poizvedbe z uporabo agregatnih funkcij z uporabo jezika SQL DML:**

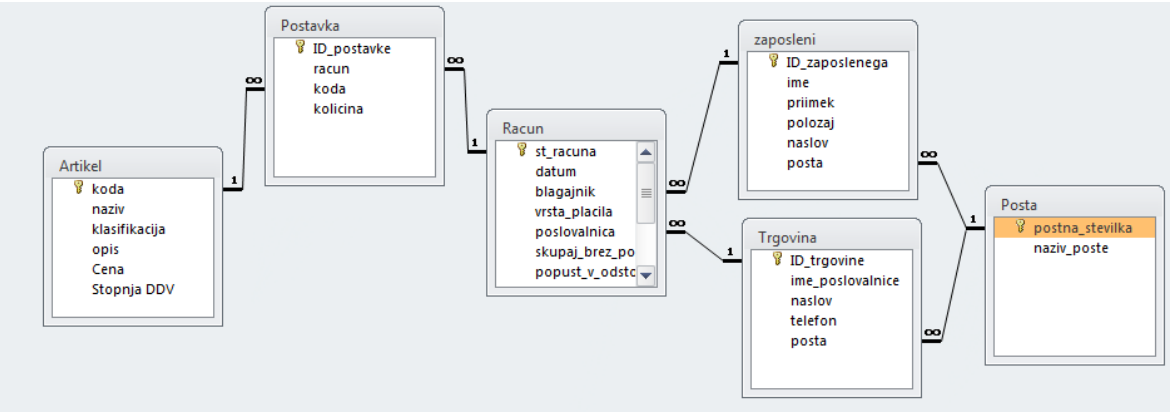
a. Povprečje ocen predavanj za vse študente.
b. Povprečje ocen predavanj za vse študente prvega letnika, ki vaj nimajo opravljenih.
c. Koliko študentov ima kakšno oceno (Ocena, Število študentov).
d. Povprečna ocena po letnikih – samo pozitivne ocene.
e. Samo letniki, kjer imajo študentje povprečno oceno višjo od 8.
f. Seznam predmetov s predavateljem, pri katerih ima vsaj en študent oceno višjo od 8.
g. Seznam študentov, ki imajo vsaj en predmet v zimskem semestru.
h. Seznam študentov, ki imajo povprečno oceno višjo od povprečne ocene celotne šole – štejejo se samo pozitivne ocene.

6.3.4 Kreirajte fizično podatkovno bazo publikacij na podlagi relacijskega modela(Slika 30) z uporabo stavkov SQL DDL:

a. Kreirajte tabeli Author in Publication z atributi.
b. Ne pozabite kreirati omejitev primarnih in tujih ključev.
c. Kreirajte indekse na stolpce name, birth_date in gender tabele.
d. Z uporabo **SQL DDL stavka** DROP TABLE zbrišite tabeli Author in Publication.

6.4.1 Kreiraj podatkovno bazo trgovskega podjetja z orodjem MS Access kot je prikazana na sliki (Slika 42) in vnesi nekaj podatkov.

Slika 42: Relacijski model trgovskega podjetja v orodju MS Access



6.4.2 S pomočjo grafičnega poizvedovalnega jezika QBE (Query by example) in orodja MS Access nad podatkovno bazo iz predhodne naloge izdelaj naslednje poizvedbe in si pri vsaki oglej pretvorbo v SQL:

a. Izpiši vse trgovine.
b. Izpiše podatke o trgovini "Muca copatarica".
c. Izpiši vse zaposlene.
d. Izpiši število vseh zaposlenih v tabeli Zaposleni.
e. Izpiši vse blagajnike (ime, priimek, polozaj) iz tabele Zaposleni.
f. Izpiši vse zaposlene (ime, priimek, naslov,posta,Posta.naziv_poste) iz Novega mesta.
g. Izpiši vse zaposlene, katerih priimek se začne na črko N.
h. Izpiši vse zaposlene, ki imajo v priimku črko š.
i. Izpiši vse zaposlene, ki imajo v priimku kot drugo črko o.
j. Izpiši število vseh artiklov.
k. Izpiši število različnih artiklov za vsak račun (tabeli Račun, Postavka). Potrebno je vključiti vrstico Vsote.
l. Izpiši skupno količino prodanih artiklov za vsak račun (tabli Račun, Postavka).
m. Za vsakega blagajnika izpiši prodano količino izdelkov. Uredi tako, da bo najprej izpisan blagajnik, ki je prodal največ (po količini od največje do najmanjše).
n. Izpiši blagajnika (ime, priimek), ki je prodal več kot 15 izdelkov (količina).
o. Za vsakega blagajnika izpiši prihodek od prodaje.

7 Objektna podatkovna baza

S pojavom in hitro uveljavitvijo objektno usmerjene tehnologije v začetku devetdesetih let, ki danes prevladuje predvsem med programskimi jeziki (Java, C#), so se pojavile tudi prve ideje o novem, objektnem pristopu tudi na področju podatkovni baz. Objektni SUPB so se sprva uveljavili v inženirstvu in načrtovanju ter v zadnjem času tudi na področjih finančnih in telekomunikacijskih rešitev. Trg objektnih SUPB pa je v primerjavi z relacijskimi SUPB-ji še vedno majhen, čeprav so mu v devetdesetih letih napovedovali zelo hitro rast. Objektne baze so danes še vedno prej izjema kot pravilo, je pa opaziti tendenco vpeljevanja posameznih objektnih konceptov in rešitev v obstoječe relacijske sisteme za upravljanje podatkovnih baz s strani vseh najpomembnejših ponudnikov (Oracle, IBM, Microsoft itd) (Lahajnar in Rožanec, 2000). Eden najbolj znanih objektnih SUPB-jev pa je **ObjectStore**.

7.1 Objektni SUPB

Značilnosti objektnega SUPB (O-SUPB) so bile definirane že leta 1989 z manifestom (Object-Oriented Database System Manifesto). V njem je bilo zapisanih trinajst funkcionalnosti, ki jih mora podpirati vsak O-SUPB. Izhajajo iz dveh področij in sicer:

- sistem mora biti objektno usmerjen in
- mora biti sistem za upravljanje s podatkovno bazo.

Tabela 55: Funkcionalnosti, ki jih mora podpirati vsak objektni SUPB

Značilnosti objektne usmerjenosti	**SUPB značilnosti**
Podpora kompleksnim objektom	Zagotavljanje trajnosti podatkov
Podpora objektni identifikaciji (OID)	Obvladovanje velikih količin podatkov
Enkapsulacija (le dostopnost vmesnika, ne pa tudi podatkov in implementacije metod, ki so skriti)	Podpora sočasnemu dostopu uporabnikov
Podpora tipom ali razredom	Sposobnost obnove po podatkovnih in sistemskih nesrečah
Dedovanje atributov in metod tipov ali razredov od nadtipov ali nadrazredov	Enostaven način za poizvedovanje po podatkih v bazi
Podpora dinamičnemu povezovanju (več operacij z istim imenom, a drugačnimi tipi objektov)	
Razširljivost množice podatkovnih tipov	
DML mora biti računsko popoln	

Vir: Connolly in Begg, 2010, str. 826.

Za razvoj objektnega SUPB z navedenimi funkcionalnostmi je možno uporabiti različne strategije (Connolly in Begg, 2010, str. 828):

- **razširitev objektno usmerjenega programskega jezika s funkcionalnostmi SUPB**: funkcionalnosti SUPB se dodajo objektnim jezikom kot so Smalltalk, C++, ali Java. Primer takšnega SUPB je izdelek GemStone, ki razširja navedene tri objektno usmerjene programske jezike.
- **Uporaba razširljivih objektno usmerjenih knjižnic s funkcionalnostmi SUPB**: gre za nekoliko drugačen način razširitve objektno usmerjenih programskih jezikov. Dodatne knjižnice omogočajo trajnost, podporo transakcijam, podporo sočasnemu dostopu, varnostne mehanizme itd. Primera takšnega SUPB sta izdelka Versant in ObjectStore.
- **Razširitev obstoječega jezika za delo s SUPB z objektno usmerjenimi zmogljivostmi:** standard SQL:1999 vsebuje objektne razširitve. Objektni standard ODMG (Object Data Management Group) definira objektni SQL (Object SQL). Izdelki podjetij Ontos and Versant ter številni drugi objektni SUPB-ji vsebujejo objektni SQL.
- **Razvoj novega podatkovnega modela in jezika:** radikalni pristop, ki od začetka v celoti razvije objektno usmerjeni jezik in SUPB z objektno usmerjenimi značilnostmi.

7.2 Načrtovanje objektne podatkovne baze

Različni avtorji so predstavili svoje metode in diagramske tehnike za načrtovanje in analizo razvoja informacijskih sistemov, med katerimi so bile največje pozornosti deležne OMT (Object Modeling Techniques), Object Oriented Analysis and Design in Objectory. Osnoven problem, da se informatiki niso bolj množično odločali za njihovo uporabo, je bila relativna mladost tehnik, odsotnost standardiziranih konceptov in enotnega procesa načrtovanja. V želji po širši uveljavitvi objektnega modeliranja so trije avtorji (Rumbaugh, Booch in Jacobson) staknili glave skupaj, pobrali najboljše iz svojih metod in nastal je UML (Unified Modeling Language).

UML lahko definiramo kot jezik za specifikacijo, vizualno modeliranje, konstrukcijo in dokumentacijo celovitih informacijskih sistemov ali njihovih posameznih komponent. Njegov temeljni cilj je zagotoviti uporabnikom standardiziran, konceptualno močan in razširljiv opisni mehanizem za prikaz vseh možnih vidikov obravnavanega sistema. Uporaba diagramskih tehnik jezika UML ni pogojena s sledenjem nekemu vnaprej definiranemu procesu razvoja informacijskih sistemov, temveč odločitev prepušča samemu uporabniku, pri čemer pa je vseeno priporočljivo upoštevanje splošno znanih načel razvoja informacijskih sistemov kot so iterativnost procesa, inkrementalni razvoj, ponovna uporabljivost itd.

Za načrtovanje relacijskih podatkovnih baz sta se uveljavila model ER in kasneje razširjen model ER, ki je vpeljal koncept generalizacije. Temeljni problem uporabe modela ER izvira iz njegove morda največje prednosti, preprostosti. Kot samo ime pove je model ER zasnovan na vsega dveh osnovnih konceptih (entiteti in razmerju), s katerima ne moremo vedno adekvatno predstaviti podatkov in njihove medsebojne odvisnosti v vse bolj obsežnih in zapletenih informacijskih sistemih.

Če je model ER zaradi svoje bogate tradicije še vedno nadvse uporaben pri načrtovanju relacijskih podatkovnih baz, pa se zadeve drastično spremenijo v kolikor nameravamo implementacijo

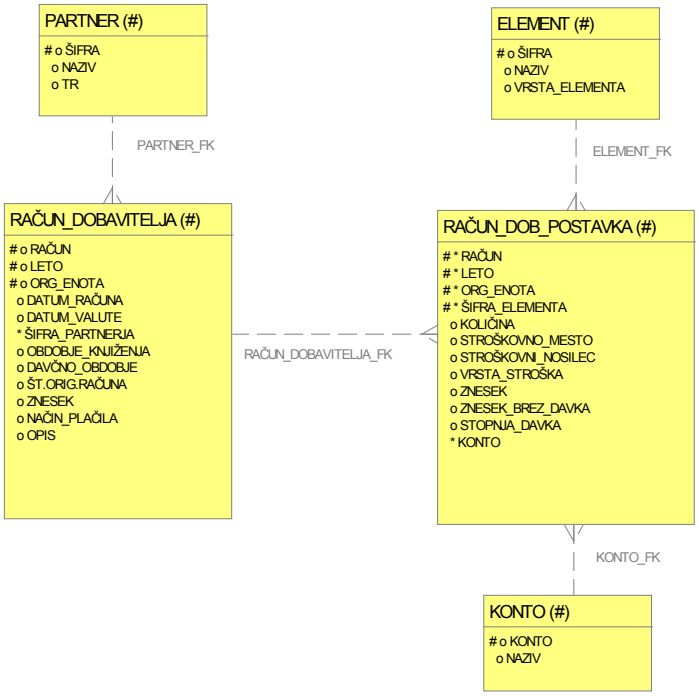
programske rešitve zasnovati na objektni ali objektno-relacijski podatkovni bazi. V tem primeru model ER povsem odpove, saj ne vsebuje možnosti predstavitve operacij in drugih konceptov značilnih za objektno tehnologijo.

Alternativa različnim diagramskim tehnikam modela ER je **UML-ov diagram razredov**. Diagram razredov služi za objektno modeliranje statične strukture sistema, svoje korenine pa ima prav v modelu ER. Sicer pa današnja različica diagrama razredov jezika UML temelji na diagramu razredov metode OMT z dodanimi elementi drugih objektnih metod (Fowler, 1997).

V nadaljevanju je predstavljena primerjalna analiza modela ER z diagramom razredov jezika UML na praktičnih primerih uporabe (povzeto po Lahajnar in Rožanec, 2000).

7.2.1 Razred

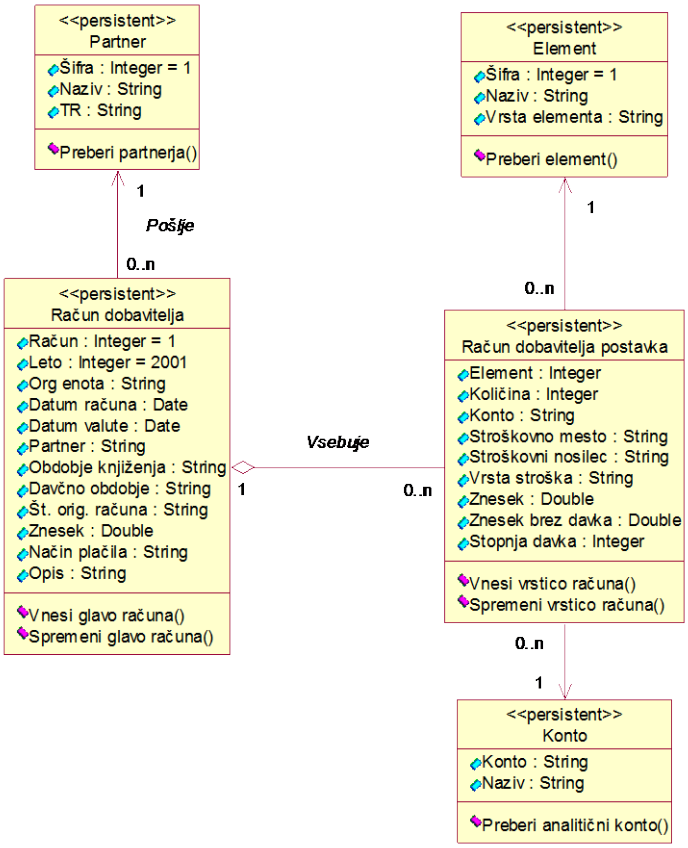
Slika 43: Poenostavljen ER model prejetih računov



Temeljni koncept modela ER je entitetni tip, ki predstavlja množico entitet istega tipa in je določen s pripadajočim naborom atributov. Pri tem pod pojmom entiteta razumemo nek objekt, ki obstaja ali mislimo da obstaja v svetu in ga je moč ločiti od drugih objektov. Entitete delimo na realne (predstavljajo nek realen objekt, dogodek) in abstraktne (predstavljajo idejo). Atributi predstavljajo posamezne lastnosti entitet in jih delimo na osnovne atribute (ni jih mogoče nadalje deliti) in sestavljene (sestavljeni so iz večjega števila osnovnih). Različne notacije prikazujejo entitetne tipe in atribute v različnih grafičnih oblikah (ponavadi v obliki pravokotnika), pri čemer velja da mora entitetni tip vsebovati vsaj naziv, lahko pa tudi nabor atributov in ključ. Slika 43 podaja primer ER modela prejetih računov.

Ekvivalentni koncept entitetnemu tipu v diagramu razredov je razred. **Razred je v jeziku UML definiran kot opis množice objektov, ki si delijo iste atribute, operacije in metode, povezave ter vsebino.** Predstavljen je s pravokotnikom s tremi ločenimi predeli: za naziv in osnovne lastnosti, za atribute ter za operacije. Atribut UML definira kot poimenovano režo znotraj razreda, ki opisuje nabor vrednosti, ki jih posamezen primerek razreda lahko zavzame. Iz navedenih definicij dveh osnovnih konceptov modela ER in razrednega diagrama je razvidno, da med njima obstaja več podobnosti kot razlik, pri čemer izstopa koncept operacije. UML definira operacijo kot storitev, ki jo razred nudi svoji okolici. Operacija torej predstavlja povezavo statične strukture z dinamičnimi elementi sistema (procesi in metodami). V nasprotju z razrednim diagramom je model ER striktno omejen na prikaz statičnih struktur brez možnosti vpeljave dinamičnega vidika, za kar je potrebno uporabiti druge diagramske tehnike (Muller, 1999, str. 136-149). Slika 44 prikazuje razredni diagram prejetih računov.

Slika 44: Poenostavljen razredni diagram prejetih računov



Podrobnejša primerjava entitetnih tipov in razredov prinese na plan še druge, manj opazne razlike. Te izvirajo iz razlik v zasnovi dveh tehnologij (relacijske in objektne), pri čemer je treba poudariti, da razredi ne predstavljajo zgolj naborov objektov podatkovne baze kot je to v primeru entitetnih tipov, temveč gre lahko tudi za objekte, ki nimajo trajnosti, ki pa niso predmet obravnave tega članka. **Razred oziroma primerek razreda v obliki objekta predstavlja torej splošnejši koncept od entitetnega tipa.** UML omogoča podrobnejšo specifikacijo kateregakoli gradnika diagrama z uporabo stereotipa (beseda v dvojnih srednjih narekovajih, praviloma pridana imenu gradnika). V primeru načrtovanja podatkovnih baz se imenu razreda doda **stereotip** <<persistent>>, ki določa, da gre za poseben tip razreda za katerega velja, da sistem ohrani stanja njegovih primerkov tudi po

prenehanju njihovega obstoja. To pa je vsekakor ključna naloga vsakega sistema za upravljanje podatkovnih baz.

Nadalje gre razlike med entitetnim tipom in razredom iskati v lastnostih, s katerimi dodatno opisujemo atribute. V primeru entitetnega tipa imamo možnost dodatno specificirati atribute, ki tvorijo identifikator ključa, večvrednostne atribute s predpisovanjem ustrezne številnosti ter domene vrednosti, ki jih atributi lahko zavzamejo. **Atributom razreda lahko poleg vseh navedenih lastnosti pripišemo še začetno vrednost, stereotip in vidljivost**. Posebej zanimiva je **lastnost vidljivosti, ki predpisuje pravice in način dostopa do vrednosti atributa** (neomejen dostop v primeru javne vidljivosti in omejitve v primeru zaščitene ali zasebne). Vidljivost je tesno povezana z enim od osnovnih načel objektne tehnologije, **ograjevanjem**. Ograjevanje govori o tem, da je dostop do podatkov objekta mogoč le preko njegovih metod, določenih z ustreznim vmesnikom.

Model ER in razredni diagram različno obravnavata **identifikator entitetnega tipa** in **identifikator objekta**. V modelu ER je identifikator definiran kot množica atributov, ki enolično določajo vsako posamezno entiteto znotraj entitetnega tipa. V nasprotju z modelom ER razredni diagram ne vsebuje posebne notacije za prikaz identifikatorja posameznih objektov, saj predpostavlja, da je identifikator objekta ena izmed njegovih bazičnih lastnosti (pravimo, da je identifikator impliciten). Če pa želimo identifikator tudi eksplicitno prikazati, moramo UML notacijo za atribute razširiti z oznako 'OID' (Object identifier), ki pove, da je atribut del identifikatorja.

7.2.2 Asociacija

Drugi bistveni koncept modela ER je razmerje, ki predstavlja skupino istovrstnih povezav in združuje vse povezave istega tipa med dvema ali več entitetami. Podobno kot v primeru entitetnih tipov je prikazovanje razmerja zelo odvisno od izbrane notacije. Chen tako prikazuje razmerje z rombom, v katerega vpišemo naziv in ga povežemo z entitetnimi tipi, medtem ko informacijski inženiring uporablja zgolj poimenovano črto. Vsakemu razmerju lahko določimo številnost, ki pove, koliko entitet drugega entitetnega tipa nastopa v razmerju z izbrano entiteto. Standardne možnosti so 1 proti 1, 1 proti N in M proti N z variantami obvezne ali pogojne prisotnosti. Nekatere notacije omogočajo tudi natančno specifikacijo števila povezav.

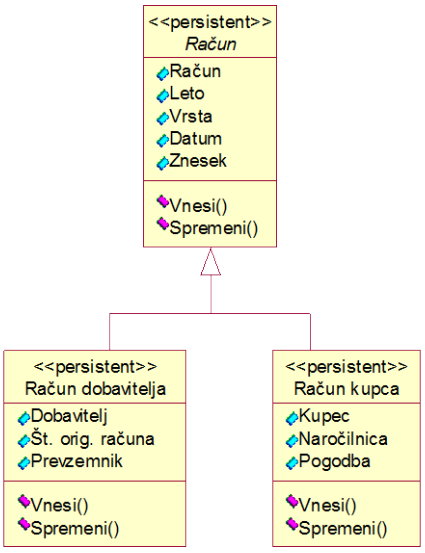
Enakovreden koncept razmerju je v razrednem diagramu **asociacija**. Pod pojmom razmerje namreč v diagramu razredov razumemo kakršenkoli odnos med dvema ali več razredi, kar poleg asociacij vključuje še generalizacijo, odvisnost itd. Binarne asociacije prikazujemo s polno črto, n-terne z rombom, pri čemer jih lahko (ni pa obvezno) zaradi boljšega razumevanja tudi poimenujemo. Posamezno stran asociacije imenujemo vloga (analogno z modelom ER), vlogi pripišemo številnost in s tem omejimo število v asociaciji udeleženih objektov. Podobno kot razmerju modela ER, lahko tudi asociaciji razrednega diagrama pripišemo lastnosti in sicer tako, da ji dodamo razred z ustreznim naborom atributov. Med razmerjem in asociacijo v osnovi torej ni večjih razlik, če odštejemo možnost določitve smeri branja in vidljivosti oziroma usmerjenosti. Asociaciji namreč lahko določimo usmerjenost, ki prikazuje, kako se razredi med seboj vidijo. Tako na primer na sliki (Slika 44) razred Račun dobavitelja ve za obstoj razreda Partner (ga vidi), medtem ko obratno ne velja.

Za razliko od modela ER, ki vsebuje zgolj en tip asociacije, lahko pri konceptualnem načrtovanju podatkovne baze z diagramom razredov uporabimo še več posebnih tipov, s katerimi dodatno specificiramo odnose med razredi. Primera tovrstnih asociacij sta **agregacija** (prikazana kot prazni romb) in **kompozicija** (prikazana kot polni romb). **Asociaciji predstavljata odnos med dvema razredoma, ko en razred poseduje drugega, pri čemer je razlika med njima v moči lastništva**. V primeru kompozicije je posedovani razred ekskluzivno del lastniškega razreda, medtem ko ima lahko razred v primeru agregacije tudi več lastnikov. Koncept agregacije je po svoji funkcionalnosti blizu konceptu šibkega entitetnega tipa modela ER.

Tretji bistveni koncept modela ER je **generalizacija**, ki spada med novosti uvedene leta 1986 v okviru razširjenega modela ER. **Generalizacija specificira odnos tip – podtip med dvema ali več entitetnimi tipi oziroma množica – podmnožica, če obravnavamo entitetni tip kot množico entitet.** Pri generalizaciji lahko določimo še pokritje, ki pove, kako množice podtipov pokrivajo množico nadtipa (kombinacije totalnega ali delnega z ekskluzivnim ali prekrivnim). Tudi diagram razredov vsebuje koncept generalizacije, definiran kot odnos med splošnim in enim ali več specifičnimi razredi in prikazan v obliki puščic od specifičnih razredov k splošnejšemu. Koncepta se praktično ne razlikujeta, seveda pa ne smemo pozabiti, da v primeru diagrama razredov podrazred ne podeduje zgolj atributov temveč tudi operacije, povezave in semantiko.

Dedovanje operacij je v objektnem svetu tesno povezano s konceptom večličnosti, ki predstavlja zmožnost programskega jezika, da uporablja isto ime za več operacij. Poznamo več tipov večličnosti: operacijska (razred vsebuje več operacij z istim imenom, a različnimi parametri), vključitvena (več podrazredov vsebujejo operacijo z istim imenom, a drugačnim obnašanjem). Možnosti, ki nam jih ponuja večličnost, lahko s pridom izkoristimo predvsem, če za implementacijo uporabimo objektno ali objektno relacijsko podatkovno bazo, pri čemer pa tudi nekateri relacijski sistemi za upravljanje podatkovnih baz vsebujejo elemente operacijske večličnosti (na primer programski jezik PL/SQL podatkovnih strežnikov podjetja Oracle).

Slika 45: Primer generalizacije



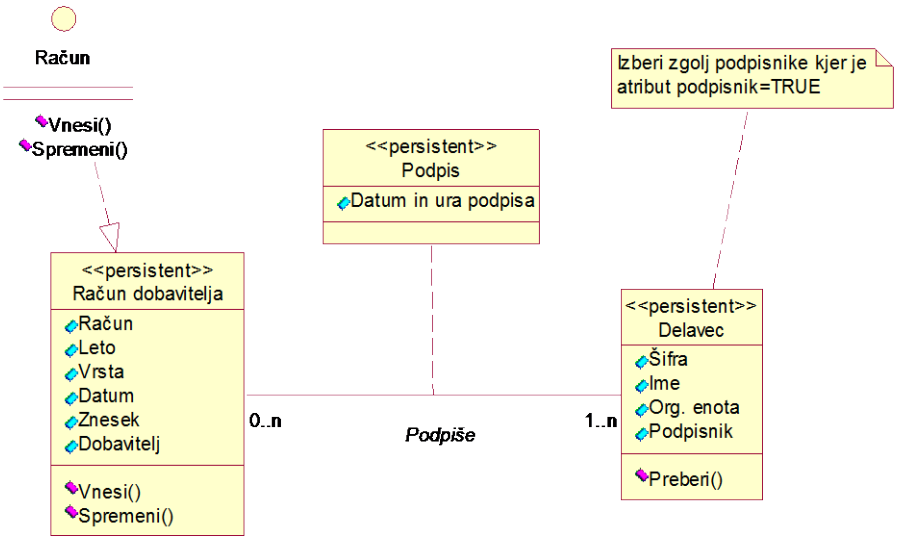
7.2.3 Druge posebnosti objektnega načrtovanja

Sedaj ko smo primerjali vse koncepte modela ER z ekvivalentnimi koncepti razrednega diagrama, se lahko posvetimo posebnostim objektnega pristopa.

Prvi tak koncept je **vmesnik**, ki ga lahko enačimo z abstraktnim razredom brez atributov. **UML definira vmesnik kot deklaracijo nabora navzven vidnih operacij razreda, pri čemer ima lahko razred večje število vmesnikov ali pa en sam vmesnik združuje operacije večjega števila razredov** (Muller, 1999, str. 158-162). Podobno kot abstraktni razredi tudi vmesniki nimajo svojih primerkov, temveč so realizirani z metodami razredov, ki jih implementirajo. Vmesniki torej ne predstavljajo neke obstojne strukture zapisane v podatkovni bazi, ampak operacije, ki s podatki upravljajo. Z njimi lahko na zelo preprost in razumljiv način prikažemo, kako in v kakšnih medsebojnih povezavah bodo podatki uporabljeni v programskih rešitvah. Na sliki (Slika 46) sta prikazana razred Račun dobavitelja in vmesnik Račun, ki implementira operaciji za vnos in spreminjanje podatkov tega razreda.

V razrednem diagramu lahko za modeliranje podatkovnih baz s pridom uporabimo tudi koncept **paketa**, definiranega kot skupina gradnikov modela (predvsem razredov), združenih z nekim namenom in po nekem kriteriju. Pravimo, da med dvema paketoma obstaja odvisnost, če sprememba definicije enega paketa lahko zahteva spremembo drugega. Odvisnosti podrobneje specificiramo z določitvijo stereotipov (<<access>>,<<import>> itd). Pakete prikazujemo kot mape, odvisnosti pa kot črtkane puščice. Tudi nad ime paketa lahko dodamo stereotip, ki dodatno pojasnjujejo vrsto paketa npr. <<subsystem>>. Uporaba tovrstnih paketov je primerna predvsem v začetni fazi izgradnje podatkovne baze, ko na podlagi diagrama primerov uporabe določimo osnovne podsisteme in področja uporabe], ki jih potem lahko razvijamo dokaj neodvisno (Muller, 1999, str. 130-136).

Razredni diagrami prinašajo še eno pomembno novost v primerjavi z modelom ER, možnost **prikaza komentarjev** (Slika 46). V ta namen uporabimo element imenovan opomba, ki ga lahko povežemo s katerimkoli gradnikom diagrama. Opombe se nadvse uporabne pri predstavljanju kompleksnejših poslovnih pravil, ki presegajo določitev objektnih identifikatorjev in domen. Opombe lahko



vsebujejo preprosto besedilo v naravnem jeziku, kot tudi psevdo kodo, strukturirano besedilo, elemente programskih jezikov ali UML-ov jezik za opis omejitev OCL (Object Constraint Language).

Slika 46: Primer uporabe vmesnika, asociacijskega razreda in opombe

7.3 Razširjeni podatkovni tipi

SQL 2003 omogoča opredelitev **uporabniško definiranih tipov** (ang. user-defined types - UDT), imenovanih tudi abstraktni podatkovni tipi (ang. abstract data types ADTs). Uporabljajo se lahko na enak način kot vnaprej določeni standardni podatkovni tipi (na primer CHAR, INT, FLOAT). Uporabniško definirane tipe (UDT) delimo v dve kategoriji: **razločevalni tipi** (ang. distinct types) in **strukturirani tipi** (ang. structured types) (Connolly in Begg, 2005, str. 929-942).

Prva skupna tipov je enostavnejša. **Razločevalni tip** je uporabniško definiran tip, ki svojo notranjo predstavitev deli z notranjo predstavitvijo vgrajenega tipa, na katerem temelji, vendar se pri uporabi šteje, da gre za dve ločena, večinoma med seboj nezdružljiva tipa. Primer kreiranja uporabniško definiranega razločevalnega tipa:

```
CREATE TYPE StevilkaLastnikaTip AS VARCHAR(5) FINAL;
CREATE TYPE StevilkaZaposlenegaTip AS VARCHAR(5) FINAL;
```

Čeprav oba tipa temeljita na vgrajenem tipu VARCHAR in sta dolžine 5, njuni instanci nista združljivi. Čeprav zadeva nekoliko spominja na definiranje domen v osnovnem SQL-u, pa je potrebno poudariti, da imajo domene izključno funkcijo omejevanja veljavnih vrednosti, ki jih je dovoljeno shraniti v nek stolpec.

V splošnem definicija UDT-ja vsebuje enega ali več atributov, nič ali več deklaracij operacij (metod) in tudi deklaracij operatorjev. Poleg tega lahko definiramo tudi enakost in urejenost UDT-ja z uporabo ukaza stavka CREATE ORDERING FOR. V tem primeru gre za **strukturirane tipe**.

Podajmo primer: denimo, da je p instanca strukturiranega tipa **OsebaTip**, ki ima atribut **Ime** tipa VARCHAR. Do tega atributa lahko dostopamo na objekten način (z uporabo .), kot smo tega navajeni iz objektnega programiranja in sicer:

```
p.Ime
p.Ime='A. Novak'
```

Razširjeni podatkovni tipi se uporabljajo v objektno-relacijskih podatkovnih bazah.

7.4 Standard ODMG

Standard na področju objektnih podatkovnih modelov in objektnih SUPB-jev (OSUPB) je bil postavljen leta 1999 s strani organizacije Object Data Management Group (ODMG), ki združuje številna znana podjetja, predvsem ponudnike objektnih SUPB-jev: Sun Microsystems, eXcelon Corporation, Objectivity Inc., POET Software, Computer Associates, and Versant Corporation. Namen standarda ODMG 3.0 je bil določiti semantiko, ki jo bodo razumeli vsi objektni SUPB-ji, kar

bo omogočalo prenosljivost knjižnic in aplikacij med različnimi objektnimi SUPB-ji. Glavne komponente ODMG arhitekture so (Connolly in Begg, 2010, str. 885):

- **Objektni model** (OM),
- **Objektni jezik za kreiranje objektov**: Object Definition Language (ODL); ekvivalenten jeziku SQL DDL v relacijskih SUPB-jih,
- **Objektni poizvedovalni jezik** - Object Query Language (OQL); ekvivalenten SQL DML v relacijskih SUPB-jih,
- **Povezave na objektne programske jezike**: C++, Java, Smalltalk.

ODMG specifikacije pokrivajo tako SUPB-je, ki objekte shranjujejo direktno v objektni SUPB, kot tudi v ODM (Object-to-Database Mappings), ki objekte pretvorijo in shranijo v relacijski ali kateri drug SUPB. ODM-ji omogočajo, da so objekti podatkovne baze dostopni različnim objektno usmerjenim programskim jezikom ter tako razširjajo osnovne funkcije programskega jezika s trajnostjo podatkov, nadzorom dostopa, možnostjo poizvedovanja in drugimi funkcijami podatkovnih baz.

7.4.1 Objektni model

Podajmo nekaj osnovnih objektnih definicij:

- Objekti (ang. objects) in konstante (literale) se razvrščajo v tipe (ang. type). Samo objekti imajo enolični identifikator.
- Vsi objekti in konstante istega tipa imajo skupno obnašanje in stanje. Tip je tudi sam objekt. Objekt je včasih naveden kot instanca svojega tipa.
- Obnašanje je opredeljeno z nizom operacij, ki jih lahko izvedemo nad objektom ali jih objekt izvaja. Operacije imajo lahko seznam vhodnih/izhodnih parametrov in lahko vračajo rezultat določenega tipa.
- Stanje je opredeljeno z vrednostmi objekta za določeno množico lastnosti (ang. property). Lastnost je lahko atribut ali razmerje enega do drugega objekta. Vrednosti lastnosti objekta, se s časom navadno spreminjajo.
- Objektni SUPB shranjuje objekte in jim omogoča, da so v skupni rabi več uporabnikov in aplikacij. OSUPB temelji na shemi, ki je opredeljena z jezikom za definiranje objektov (Object Definition Language) in vsebuje instance tipov, opredeljenih v njegovi shemi.

Objekt je opisan s štirimi značilnostmi:

- **Strukturo**: tipe delimo v dve skupini. V prvi skupini imamo: osnovne tipe (npr. long, short, float, double, string), zbirke (npr. množica, seznam, polje, seznam) in strukturirane tipe (npr. datum, ura). V drugi skupini pa imamo sestavljene objektne tipe.
- **Identifikatorjem**: vsakemu objektu je s strani OSUPB dodeljen enolični identifikator objekta, ki se ne spreminja in se ponovno ne uporabi po brisanju objekta.
- **Imenom**: objektu lahko dodelimo ime, kar nam omogoča razred Database in
- **Življenjsko dobo**: se določi pri kreiranju objekta in je lahko začasna ali trajna. Za shranjevanje trajnih objektov skrbi OSUPB.

V objektnem modelu so definirane vgrajene zbirke, ki omogočajo hranjenje poljubnega števila neimenovanih homogenih elementov. To so:

- **Množica** (ang. Set): neurejena zbirka, ki ne dovoljuje shranjevanja duplikatov,
- **Vreča**: (ang. Bag) – neurejena zbirka, ki dovoljuje shranjevanje duplikatov,
- **Seznam** (ang. List): urejena zbirka, ki dovoljuje shranjevanje duplikatov,

- **Polje** (ang. Array): enodimenzionalno polje prilagodljive dolžine,
- **Slovar** (ang. Dictionary): neurejena zbirka parov ključ-vrednost, brez duplikatov v ključih.

Vsak od navedenih tipov zbirk ima definirani vsaj operaciji za kreiranje objekta ter njegovo vstavljanje v zbirko. Množica in vreča pa imata na primer dodatno definirane še tipične operacije kot so: presek, unija in razlika.

ODMG objektni model podpira koncept podatkovne baze kot področja za hrambo trajnih objektov dane množice tipov. Baza ima shemo, ki vsebuje množico definicij tipov. Vsaka podatkovna baza je instanca tipa *Database* z vgrajenimi operacijami za njeno odpiranje (open), vpogled (lookup) in zapiranje (close).

Slika 47: ODL vmesnik za delo z objektno podatkovno bazo

```
interface DatabaseFactory {
    Database    new();
};
interface Database{
    exception   DatabaseOpen{};
    exception   DatabaseNotFound{};
    exception   ObjectNameNotUnique{};
    exception   ObjectNameNotFound{};
    void        open(in string odms_name) raises(DatabaseNotFound, DatabaseOpen);
    void        close() raises(DatabaseClosed, TransactionInProgress);
    void        bind(in Object an_object, in string name) raises(DatabaseClosed,
                                    ObjectNameNotUnique, TransactionNotInProgress);
    void        unbind(in string name) raises(DatabaseClosed,
                                    ObjectNameNotFound, TransactionNotInProgress);
    void        lookup(in string object_name) raises(DatabaseClosed,
                                    ObjectNameNotFound, TransactionNotInProgress);
    ODLMetaObjects::Module schema() raises(DatabaseClosed, TransactionNotInProgress);
};
```

Vir: Connolly in Begg, 2010, str. 894.

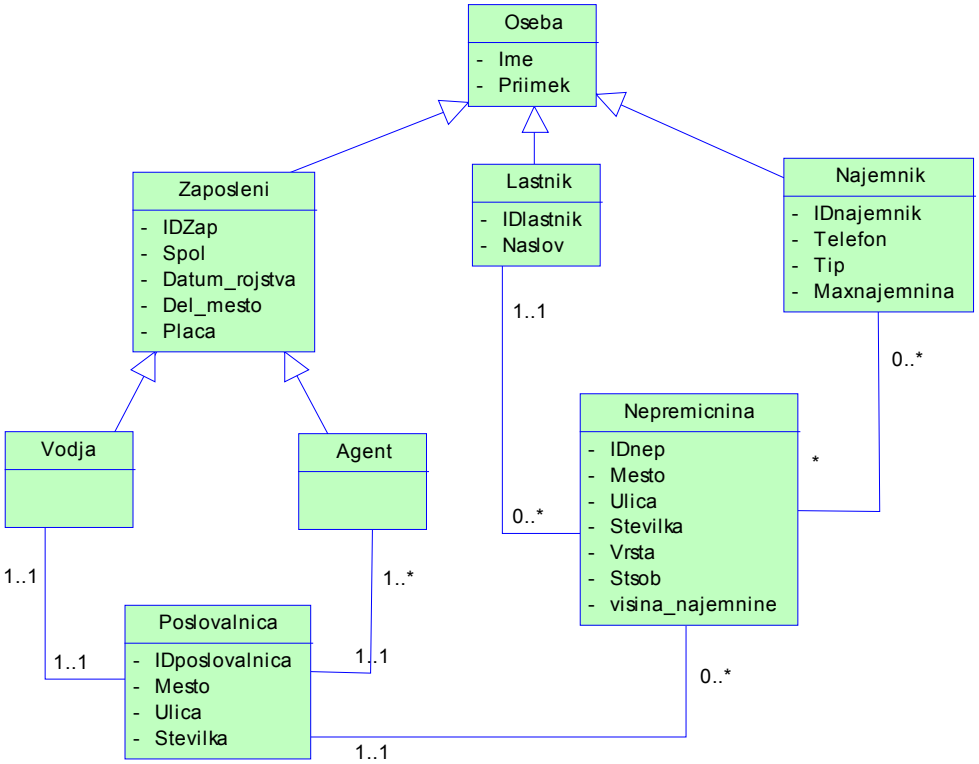
7.4.2 ODL (Object Definition Language)

ODL je objektni jezik, ki omogoča specifikacijo objektnih tipov v objektno usmerjenih sistemih, skladnih s standardom ODMG. Je ekvivalenten jeziku SQL DDL, ki se uporablja v relacijskih SUPB-jih. ODL omogoča interoperabilnost med različnimi objektnimi SUPB-ji. Sintaksa jezika ODL razširja Interface Definition Language (IDL) standarda CORBA (Common Object Request Broker Architecture). Podrobne specifikacije jezika ODL presegajo obseg tega učbenika. Bralci si lahko podrobnosti preberejo v Cattell (2000). V nadaljevanju podajamo primer stavkov jezika ODL za kreiranje dela objektne podatkovne baze, ki temelji na podatkovnem modelu, podanem s sliko (Slika 48).Podatkovni model prikazuje načrt podatkovne baze nepremičninske agencije. V njej želimo hraniti podatke o osebah, katerih pa je več vrst (gre za specializacijo razreda Oseba). Razred Oseba specializiramo na razrede Lastnik, Najemnik in Zaposleni. Razred Lastnik je namenjen hranjenju podatkov o lastnikih nepremičnin, ki jih agencija oddaja. Razred Najemnik hrani podatke o

najemnikih nepremičnin. Tretja specializacija Osebe je razred Zaposleni, ki je nadalje specializiran na Vodje, ki vodijo posamezne poslovalnice agencije (razred Vodja) in agente, ki delajo v posamezni poslovalnici (razred Agent). Razred Nepremičnina je namenjen hranjenju podatkov o nepremičninah (kje se ta nahaja, vrsta nepremičnine (npr. hiša, garsonjera, stanovanje), kakšna je cena najema, število sob). Razred Nepremičnina je z asociacijo povezana z razredom Lastnik, ki pove, kdo je lastnik posamezne nepremičnine (ta je lahko le eden). Nepremičnina je z asociacijo povezana tudi z razredom Najemnik (številnost več). Tako se hrani celotna množica najemnikov. Razred Poslovalnica hrani podatke o posamezni poslovalnici nepremičninske agencije (šifro, lokacijo) ter asociacije do razredov Vodja (poslovalnico vodi natanko en vodja), Agent (poslovalnica ima več agentov) in Nepremicnina (psolovalnica ponuja več nepremičnin).

Atributi posameznih razredov so na sliki (Slika 48) prikazani v razredih, pri čemer ne smemo pozabiti na dedovanje vseh atributov podrazreda od nadrazreda. Tako na primer za vse lastnike nepremičnin hranimo podatke o šifri in naslovu (atributa IDLastnik in Naslov, ki sta navedena v tem razredu) ter ime in priimek (atributa Ime in Priimek, ki se dedujeta od nadrazreda Oseba in v razredu Lastnik zato nista posebej prikazana).

Slika 48: Podatkovni model nepremičninske agencije



Primer 7.4.2.1: Kreiranje dela objektne podatkovne baze nepremičninske agencije

```
module NeprAgencija {
        class Poslovalnica                                  //definiramo razred Poslovalnica
        (extent Poslovalnice   key IDPoslovalnica)
        {
                //definiramo atribute
                attribute string IDPoslovalnica;
                attribute struct NaslovPosl { string Mesto, string Ulica, string Stevilka} naslov;
                //definiramo povezave
                relationship Vodja je_vodena inverse Vodja:: vodi;
                relationship set<Agent> ima inverse Agent:: dela_v;
                relationship set<Nepremicnina> ponuja inverse Nepremicnina:: je_na voljo_v;
                //definiramo operacije
                void Vzeti_v_najem (in string IDNep) raises (NepremicninaZeNajeta);
        };
        class Oseba {                                       //definiramo razred Oseba
         //definiramo atribute
                attribute struct ImePriimek {string Ime, string Priimek} imepriimek;
        };
        class Zaposleni extends Oseba                       //dedovanje iz razreda Oseba
        (extent zaposleni   key IDZap)
        {
                attribute string IDZap;
                attribute enum tipspol{M,Ž} spol;
                attribute date datum_rojstva
                attribute enum vrste_DM {vodja, nadzornik,agent} Del_mesto;
                attribute float Placa;
                //definiramo operacije
                short PridobiStarost();
                void PovecajPlaco(in float znesek_povisanja);
        };
        class Vodja extends Zaposleni       //dedovanje iz razreda Zaposleni
        (extent vodje)
        {
                // definiramo povezave
                relationship Poslovalnica vodi inverse Poslovalnica:: je_vodena;
        };
        class Agent extends Zaposleni       //dedovanje iz razreda Zaposleni
        (extent agenti)
        {
                // definiramo povezave
                relationship Poslovalnica dela_v inverse Poslovalnica:: ima;
                // definiramo operacije
                void PremakniZap(in string izIDPoslovalnica, in string vIDPoslovalnica) raises
                (se_ne_dela_v_poslovalnici);
        };
};
```

7.4.3 OQL (Object Query Language)

OQL je objektni poizvedovalni jezik, ki omogoča odstop do objektne podatkovne baze. Je ekvivalenten jeziku SQL DML v relacijskih SUPB-jih. Sintaksa je podobna sintaksi SQL-la, vendar ne

vsebuje ukazov za ažuriranje pač pa to prepušča operacijam, definiranimi nad objektnimi tipi. OQL lahko uporabljamo kot samostojen jezik ali pa kot jezik, vključen v drug objektni programski jezik. ODMG definira povezave za vključitev v jezike Smalltalk, C++ in Java. OQL lahko kliče tudi operacije, napisane v teh programskih jezikih.

Sintaksa SELECT stavka je podobna sintaksi v SQL-u in je naslednja:

SELECT [DISTINCT] <expression>
FROM <fromList>
[WHERE <expression>]
[GROUP BY <attribute1:expression1, attribute2:expression2,...>]
[HAVING <predicate>]
[ORDER BY <expression>]

Rezultat poizvedbe je množica v primeru SELECT DISTICT, seznam kadar uporabimo ORDER BY, drugače pa vreča. V splošnem potrebujemo vstopno točko do podatkovne baze v primeru vsake poizvedbe. Lahko uporabimo katerikoli poimenovani trajni objekt (extent ali poimenovani objekt) (glej primer 7.4.2.1).

Primer 7.4.3.1: Pridobi množico vseh zaposlenih.
V tem primeru lahko uporabimo »extent« razreda Zaposleni.

```
zaposleni
```

Primer 7.4.3.2: Pridobi množico vseh vodij poslovalnic.
V tem primeru lahko uporabimo »extent« razreda Poslovalnice(poslovalnice) kot vstopno točko do podatkovne baze, nato pa uporabimo povezavo (relationship) je_vodena, da najdemo množico vodij poslovalnic.

```
poslovalnice.je_vodena
```

Primer 7.4.3.3: Pridobi vse poslovalnice iz Ljubljane.

```
SELECT b.IDPoslovalnica
FROM b IN poslovalnice
WHERE b.naslov.Mesto = “Ljubljana”;
```

Primer 7.4.3.4: Kreiraj pogled vseh agentov, ki delajo v Ljubljani.
Za kreiranje pogleda v jeziku OQL uporabimo stavek DEFINE in pogled poimenujemo, v našem primeru LjubljanskiAgenti. Imena pogledov znotraj podatkovne baze morajo biti enolična.

```
DEFINE LjubljanskiAgenti AS
        SELECT s
        FROM s IN agenti
        WHERE s.dela_v.naslov.Mesto = “Ljubljana”;
SELECT s.imepriimek.Priimek FROM s IN LjubljanskiAgenti;
```

Primer 7.4.3.5: Kreiraj pogled vseh agentov, ki delajo v Ljubljani.
Pri kreiranju poizvedb in pogledov lahko uporabljamo tudi parametre, ki naredijo naše poizvedbe dinamične, v našem primeru je parameter imemesta.

```
DEFINE mestniagenti(imemesta) AS
        SELECT s
        FROM s IN agenti
        WHERE s.dela_v.naslov.Mesto = imemesta;
```

Navedeno poizvedbo lahko uporabimo, da pridobimo agente, ki delajo v določenem mestu, npr:

```
mestniagenti("Ljubljana");
mestniagenti("Črnomelj");
```

Primer 7.4.3.5: Uporaba agregatnih funkcij
Tudi OQL ima na voljo agregatne funkcije COUNT, AVG, MIN, MAX kot jih poznamo že iz SQL-a. Želimo prešteti koliko agentov dela v Ljubljani.

```
COUNT(s IN mestniagenti("Ljubljana"));
```

Agregatne funkcije lahko uporabimo znotraj ali zunaj SELECT stavka. Naslednja dva stavka sta ekvivalentna:

```
SELECT COUNT(s) FROM s IN agenti WHERE s.dela_v.IDPoslovalnice = "B003";
COUNT(SELECT s FROM s IN agenti WHERE s.dela_v.IDPoslovalnice = "B003");
```

1. Katero diagramsko tehniko za načrtovanje objektne podatkovne baze poznate?
2. Zakaj modela ER ne moremo uporabiti za načrtovanje objektne ali objektno-relacijske podatkovne baze?
3. Kateri so ključni koncepti, ki nastopajo v razrednem diagramu?
4. Primerjajte koncepte razrednega diagrama s koncepti ER diagrama.
5. Kaj je namen standarda ODMG 3.0?
6. Katere so glavne komponente ODMG arhitekture?
7. Kaj je ODL in čemu je namenjen? Kateremu jeziku relacijskih PB je ekvivalenten?
8. Kaj je OQL in čemu je namenjen? Kateremu jeziku relacijskih PB je ekvivalenten?

8 Orodja za delo s podatkovnimi bazami

V poglavju so opisana tri orodja za delo s podatkovnimi bazami. Poglavje začenjamo z opisom CASE orodja **SAP Sybase PowerDesigne**r (http://sybase-powerdesigner.software.informer.com/16.5/), ki najbolje sledi opisani metodologiji načrtovanja preko treh ravni. Nato bo opisano še CASE orodje **Oracle SQL Developer Data Modeler**, ki je za študijske namene na voljo brezplačno na spletni strani podjetja Oracle (http://www.oracle.com/technetwork/developer-tools/datamodeler/downloads/datamodeler-087275.html). Orodje omogoča vizualno načrtovanje podatkovne baze skozi faze, ki so bile predstavljene v poglavjih 4 in 5. V nadaljevanju predstavljamo drugo Oraclovo orodje in sicer **APEX** (Oracle Application Express). Gre za spletno orodje, katerega namestitev ni potrebna. Orodje omogoča kreiranje podatkovne baze, vnos podatkov, kreiranje poizvedb z uporabo jezika SQL pa tudi razvoj aplikacij (https://apex.oracle.com/i/index.html), kar pa ni več predmet obravnave v tem priročniku. Obe orodji tako skupaj celovito pokrijeta opravila, povezana s podatkovno bazo, od njenega načrtovanja, kreiranja, vnosa podatkov in učenja jezika SQL. V zadnjem poglavju na kratko predstavimo še **MS Access**, ki je del zbirke Office in zato večinoma že prisoten v vsakem računalniku. V tem zadnjem podpoglavju se poleg prikaza kreiranja podatkovne baze za MS Access posvetimo predvsem izdelavi poizvedb z uporabo jezika QBE.

8.1 SAP Sybase PowerDesigner

SAP Sybase PowerDesigner je orodje, ki omogoča modeliranje najrazličnejši vidikov, potrebnih pri razvoju informacijskega sistema: procesnih modelov, UML diagramov, podatkovnih modelov relacijskih baz, modelov podatkovnega skladišča itd. V letu 2002 je orodje PowerDesigner dosegalo 39 % tržni delež na področju modelirnih orodij (http://en.wikipedia.org/wiki/PowerDesigner), saj gre za zelo enostavno, a hkrati zelo zmogljivo orodje. V letu 2012 je bilo podjetje Sybase in samo orodje prevzeto s strani podjetja SAP in se sedaj trži pod imenom SAP Sybase PowerDesigner. V nadaljevanju se bomo osredotočili le na načrtovanje relacijske podatkovne baze s tem orodjem.

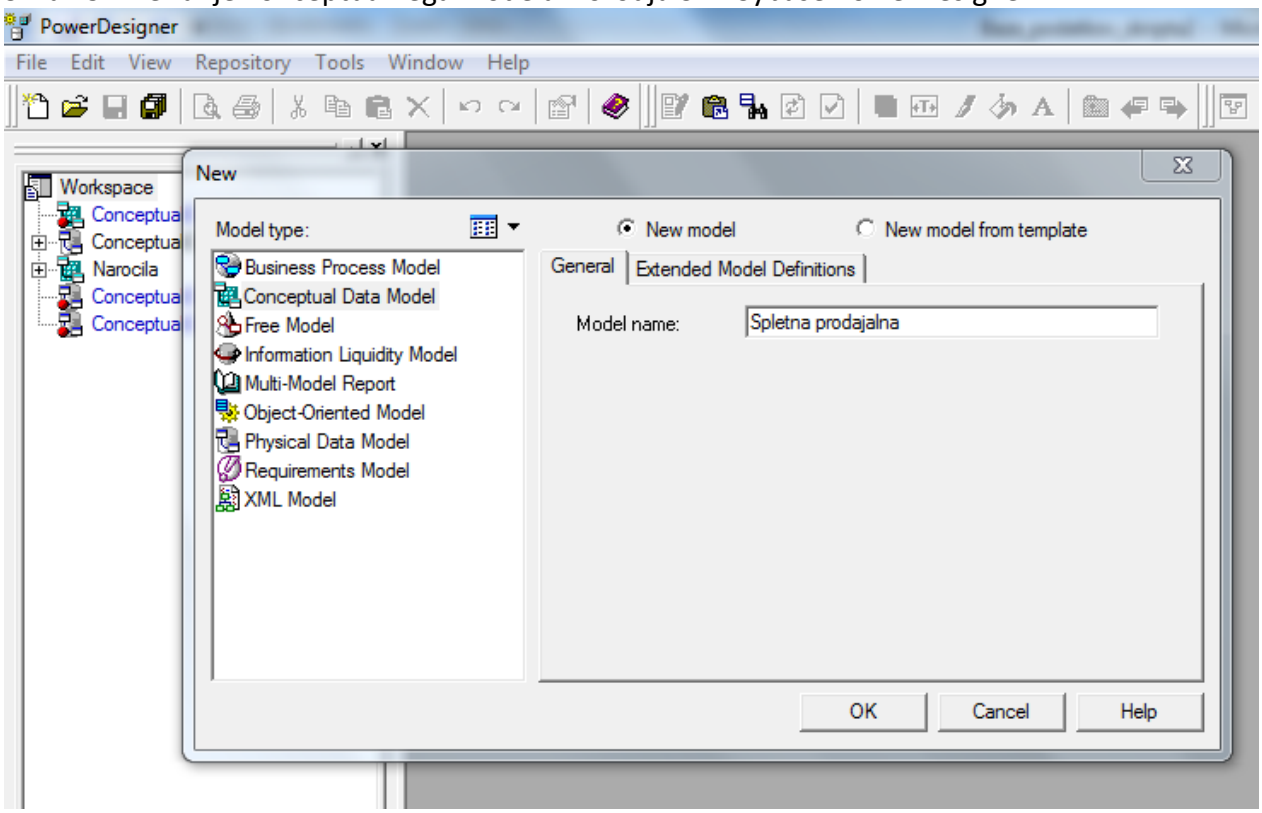
V nadaljevanju bo prikazan primer načrtovanja preprostega podatkovnega modela trgovine skozi faze konceptualnega, logičnega in fizičnega načrta.

8.1.1 Izdelava konceptualnega modela trgovine

Podjetje Spletko d.o.o. potrebuje spletno trgovino, ki bo omogočala prodajo izdelkov. V podatkovni bazi, ki bo temelj tega IS, bo potrebno shranjevati podatke o strankah, izdelkih, ki jih trgovina ponuja ter naročilih strank.

Za vsako stranko želijo hraniti podatke o davčni številki, naslovu, imenu in priimku, kraju in poštni številki. O posameznem izdelku želijo hraniti naziv, opis in ceno. Z vsako naročilo je potrebno zabeležiti datum in čas oddaje naročila, stranko, ki je naročilo oddala, ter seznam izdelkov s količinami, kar bo omogočilo kreiranje računa in odpremo naročenih izdelkov.

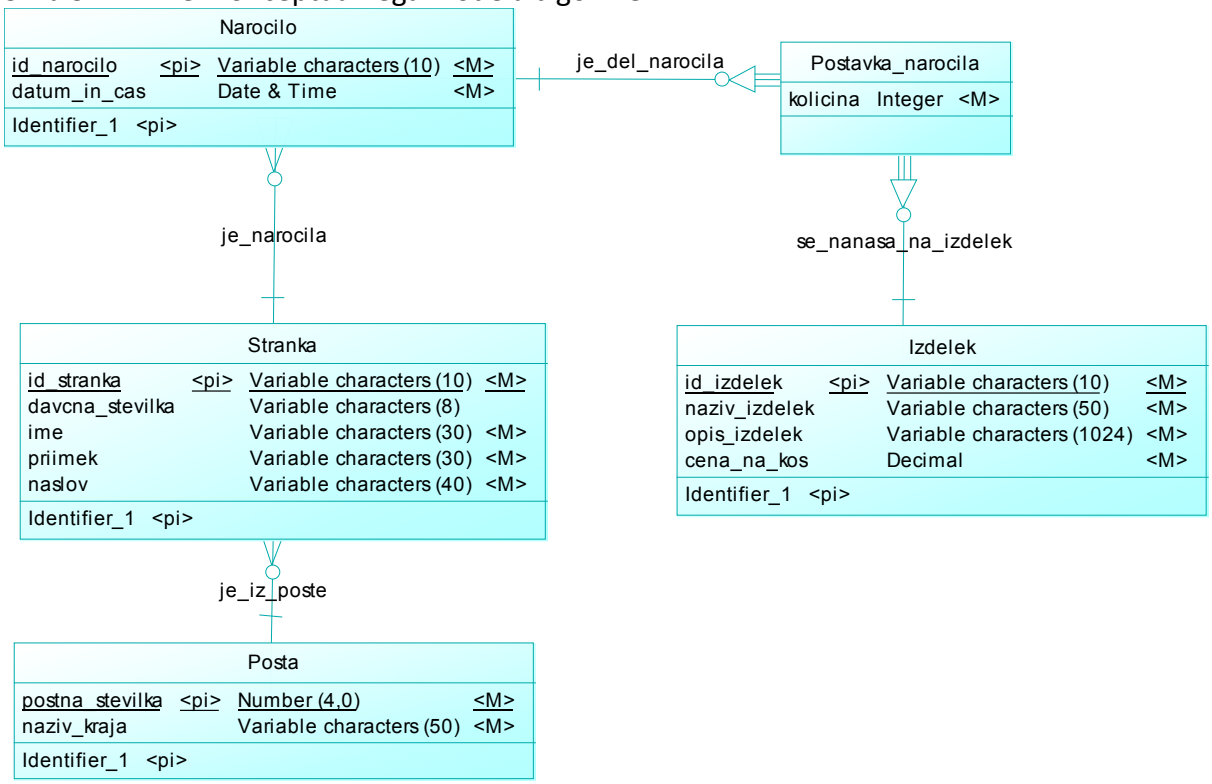
Slika 49: Kreiranje konceptualnega modela v orodju SAP Sybase PowerDesigner



Slika 49 prikazuje orodje PowerDesigner in kreiranje konceptualnega modela v njem. Model poimenujemo Spletna prodajalna. Na delovno površino nato narišemo entitetne tipe, kreiramo atribute in določimo primarne identifikatorje. Nato entitetne tipe povežemo med seboj. Uporabimo grafične gradnike iz palete (Slika 50).

Slika 50: Paleta gradnikov za izdelavo konceptualnega modela v orodju PowerDesigner

Slika 51: Primer konceptualnega modela trgovine



Slika 51 prikazuje konceptualni model PB spletne trgovine, ki ustreza zapisanim poslovnim zahtevam. Model obsega 5 entitetnih tipov s pripadajočimi atributi. Da bi model ustrezal 3. normalni obliki, smo atributa »postna_stevilka in »naziv_kraja« umestili v ločen entitetni tip z nazivom POSTA. Razmerje med entitetnim tipom STRANKA in POSTA pove, da stranka prebiva v natanko enem kraju (z enolično postno številko). V drugo smer pa, da iz določenega kraja (z določeno poštno številko) lahko prihaja nič, ena ali več strank. Razmerje obstaja tudi med entitetnima tipoma STRANKA in NAROCILO. Določena stranka lahko odda nič, eno ali več naročil, medtem ko se določeno naročilo nanaša na natanko eno stranko.

Ker na vsakem naročilu lahko prodamo več izdelkov, za vsakega pa si moramo zabeležiti tudi prodano količino, potrebujemo entitetni tip POSTAVKA_NAROCILA, ki povezuje NAROČILO in IZDELEK. POSTAVKA_NAROCILA nima enoličnega identifikatorja iz vrst lastnih atributov, zato gre za šibki entitetni tip. Njen enolični identifikator sestavljata identifikatorja id_narocilo in id_izdelek. To prikažemo na diagramu s puščicami, ki prikazujejo odvisnost tipa POSTAVKA od tipov NAROČILO in IZDELEK. Ostali entitetni tipi so močni, saj imajo enolični identifikator iz vrst lastnih atributov (postna_stevilka v tipu POSTA, id_stranka v tipu STRANKA, id_izdelek v tipu IZDELEK in id_narocilo v tipu NAROCILO).

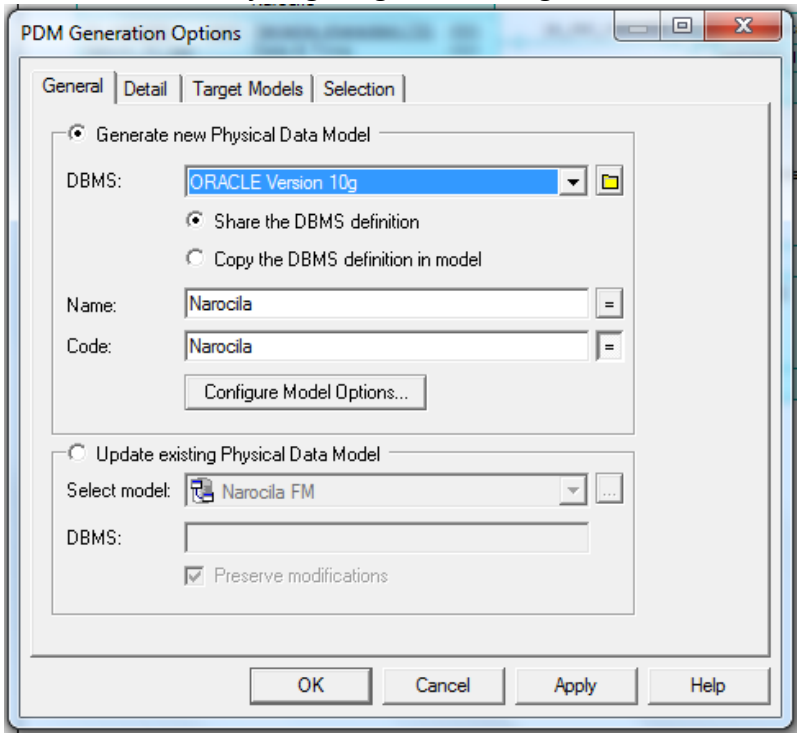
Oznaka <pi> označuje enolični identifikator entitetnega tipa. Oznaka <M> označuje obveznost atributa. Vsi atributi, ki predstavljajo enolični identifikator so vedno obvezni. Poleg teh lahko določimo še druge obvezne atribute. V primeru tipa STRANKA smo določili, da so atributi ime, priimek in naslov obvezni, medtem ko davcna_stevilka ni obvezna (je brez oznake <M>). Na diagramu vidimo tudi podatkovne tipe, ki povedo, kakšne vrste podatkov bo možno vnesti. Tako je

npr. davca_stevila omejena na 8 znakov, medtem ko jih je za vpis imena na voljo 30, za priimek pa 40.

8.1.2 Izdelava logičnega modela trgovine

Naslednji korak je generiranje logičnega modela, ki je v primeru orodja PowerDesigner avtomatizirano. Po izbiri ukaza za generiranje (Tools->Generate Physical Data Model) je potrebno izbrati SUPB, za katerega bomo generirali logični model. Slika 52 prikazuje generiranje logičnega modela za Oracle 10g SUPB. Pri tem je potrebno opozoriti, da terminologija v orodju ni skladna z opisano terminologijo v poglavju 9 in se zato tukaj izbere ukaz za generiranje »fizičnega« podatkovnega modela.

Slika 52: Generiranje logičnega modela trgovine



Slika 53: Primer logičnega modela trgovine

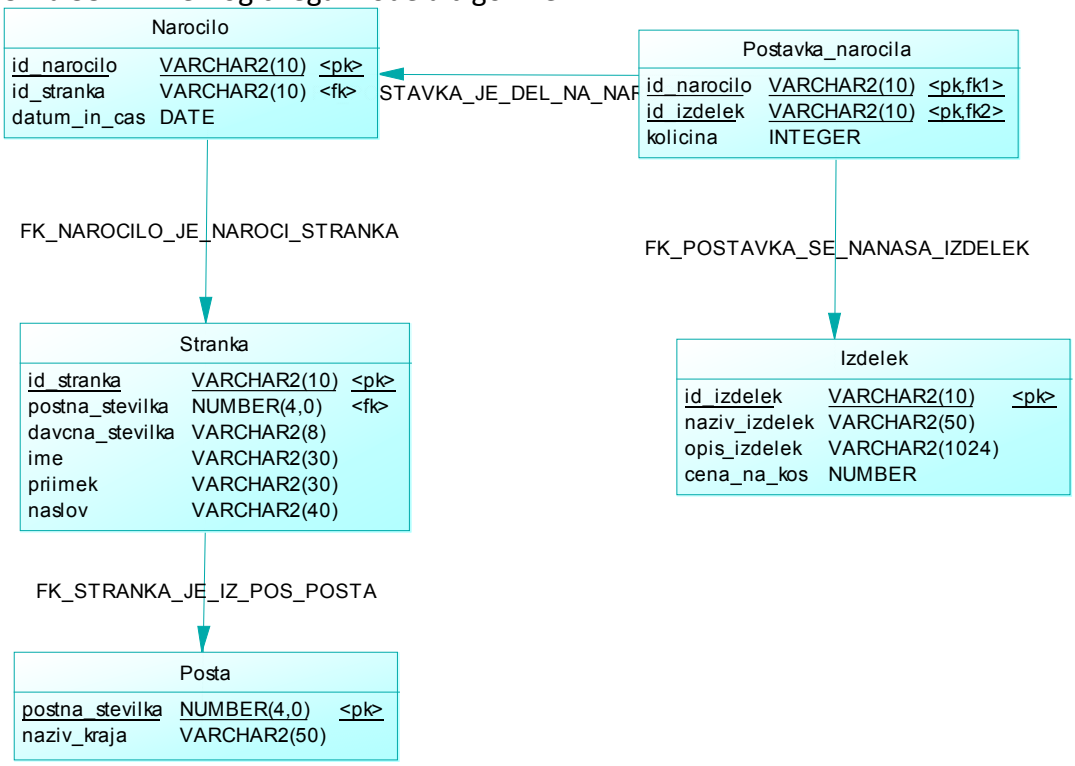
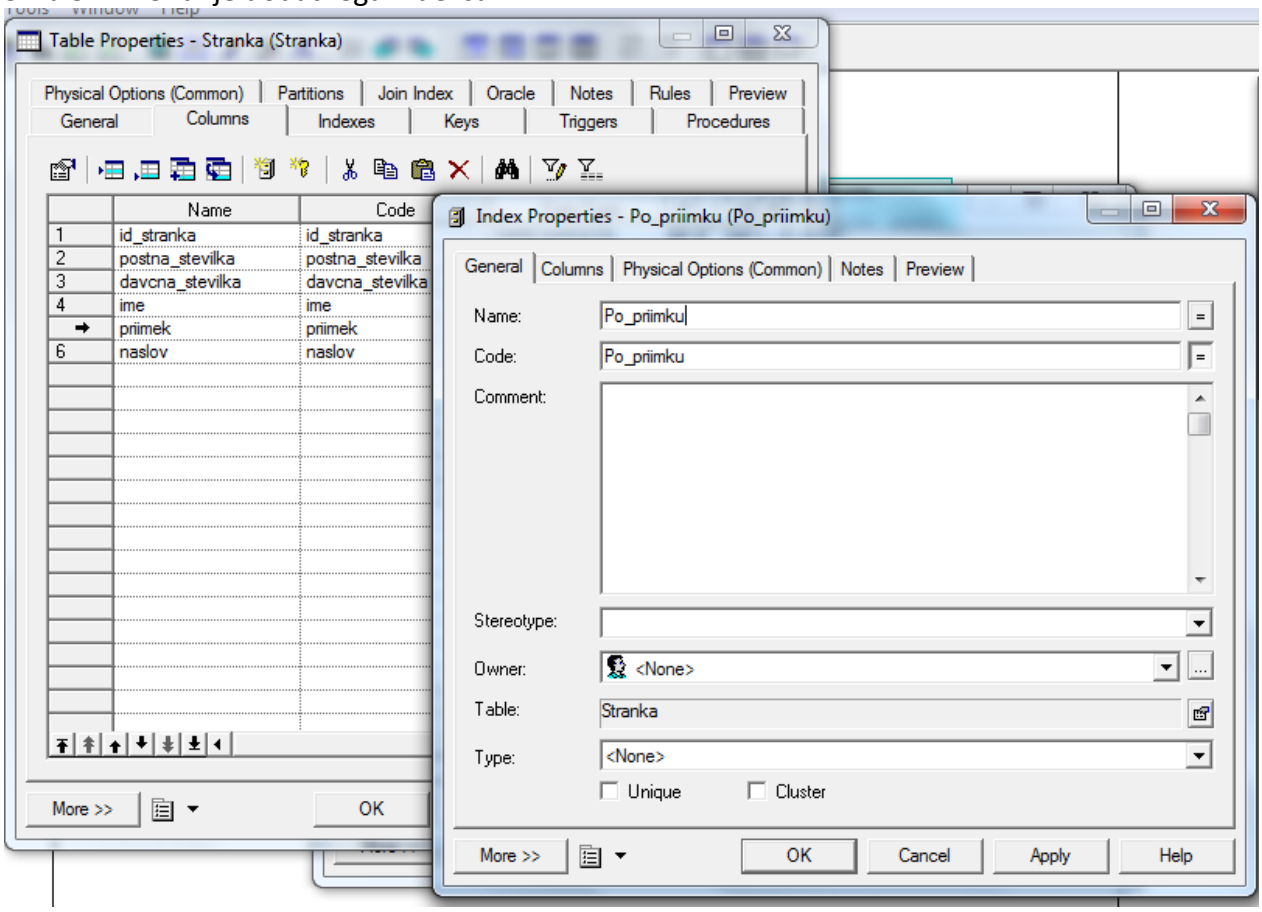


Tabela 53 prikazuje logični podatkovni model spletne trgovine po izvedeni preslikavi. Iz diagrama so razvidne spremembe glede na konceptualni model. Najprej opazimo spremembo pri navedbi podatkovnih tipov atributov, ki sedaj ustrezajo izbranemu SUPB (Oracle 10g). Splošni tip »variable characters« se je tako npr. preslikal v tip »varchar2«, tip »decimal« pa v »number«. Druga sprememba so tuji ključi, ki so označeni z oznako <fk> (ang. foreign key). Tuji ključi nastanejo povsod, ker sta entitetna tipa povezana z razmerjem številnosti ena proti mnogo. Tako je npr. nastal tuji ključ postna_stevilka v tabeli STRANKA. Ta atribut namreč predstavlja primarni ključ tabele POSTA, s katero je povezana tabela STRANKA. Tuji ključi so se generirali tudi v tabeli POSTAVKA_NAROČILA in sicer id_narocilo in id_izdelek. Navedena tuja ključa skupaj tvorita primarni ključ te tabele, saj je bila tabela v konceptualnem modelu modelirana kot šibki tip in ni imela identifikatorja iz vrst lastnih atributov. Vsi atributi, ki so bili modelirani kot primarni identifikatorji (oznaka <pi>) so se preslikali v primarne ključe (oznaka <pk>).

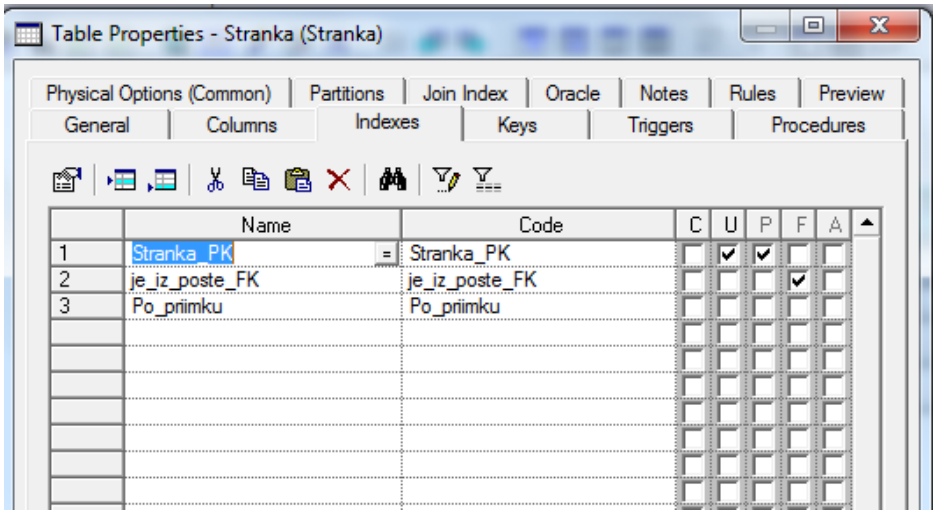
Slika 54: Kreiranje dodatnega indeksa



Na logičnem modelu lahko nadalje definiramo še morebitne dodatne indekse na atributih za katere se nam s stališča učinkovitosti dela z bazo, to zdi potrebno. Tako lahko na primer dodamo indeks na stolpec Priimek tabele STRANKA.

Slika 54 prikazuje, kako v orodju določimo dodatni indeks. Indeksi na primarne in tuje ključe se dodajo avtomatsko.

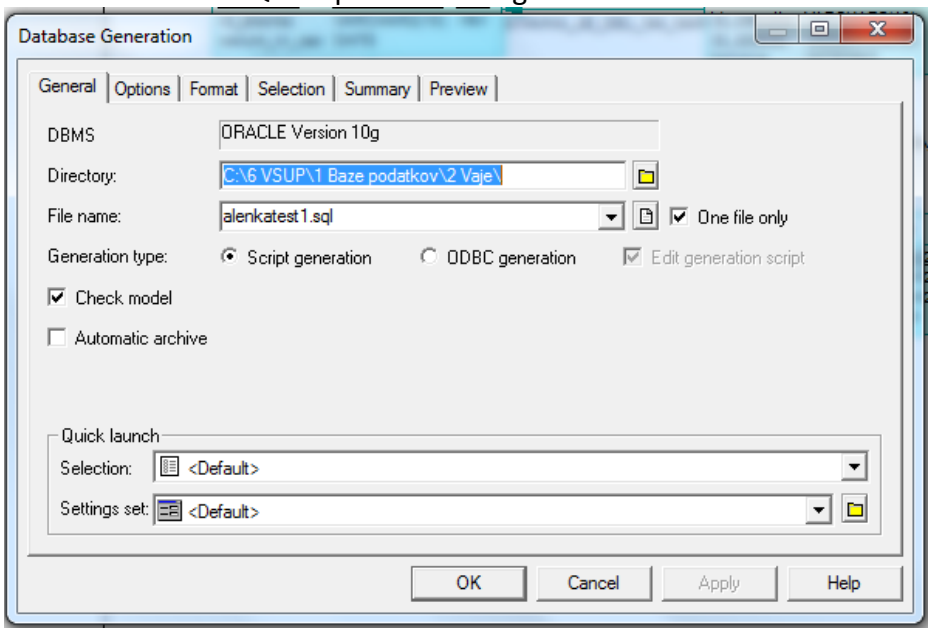
Slika 55: Prikaz indeksov tabele STRANKA



8.1.3 Izdelava fizičnega modela trgovine

Fizični model predstavlja kar SQL skripta, ki jo lahko poženemo v izbranem SUPB. Skripto kreiramo z izbiro ukaza Database->Generate Database. Prikaže se okno, kjer se lahko odločimo za generiranje skripte, ali pa se preko ODBC tudi direktno povežemo z bazo in jo skreiramo. V primeru s slike (Slika 56) smo izbrali generiranje skripte (Script generation), ki se zapiše v datoteko tipa sql.

Slika 56: Izdelava SQL skripte modela trgovine



Nadaljnje delo s podatkovno bazo nato nadaljujemo v ustreznem SUPB (v našem primeru Oracle 10g).

8.2 Oracle SQL Developer Data Modeler

Oracle SQL Developer Data Modeler je brezplačno modelirno orodje podjetja ORACLE, ki omogoča izdelavo logičnih, relacijskih, fizičnih in več-dimenzionalnih podatkovnih modelov. V tem orodju je uporabljena terminologija še nekoliko drugačna kot terminologija, predstavljena v poglavju 9.

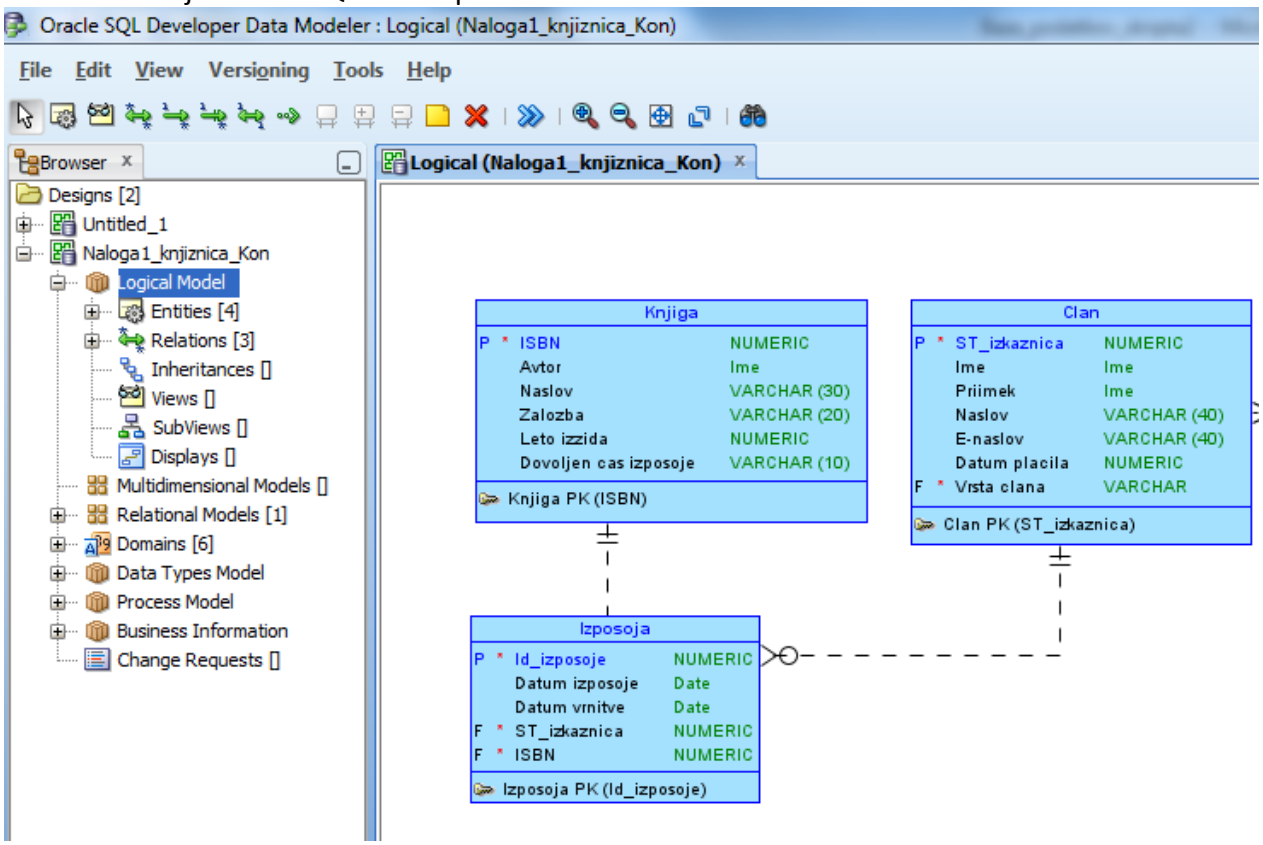
8.2.1 Izdelava logičnega modela knjižnice

Knjižnica Na grbi potrebuje informacijski sistem, ki bo omogočal hranjenje podatkov o knjigah, članih knjižnice in poravnavi članarine. Poleg tega mora IS omogočati beleženje izposoj in vračil izposojenih knjig.O posamezni knjigi želijo hraniti njeno ISBN številko, naslov, avtorja, založbo, leto izida in dovoljen čas izposoje.

Za vsakega člana knjižnice želijo hraniti podatke o številki članske izkaznice, datumu vpisa v knjižnico, ime, priimek, naslov, elektronski naslov, geslo, vrsto člana (npr. učenec, dijak, zaposleni, upokojenec). Od vrste člana je odvisna višina članarine, ki jo mora član vsako leto poravnati.

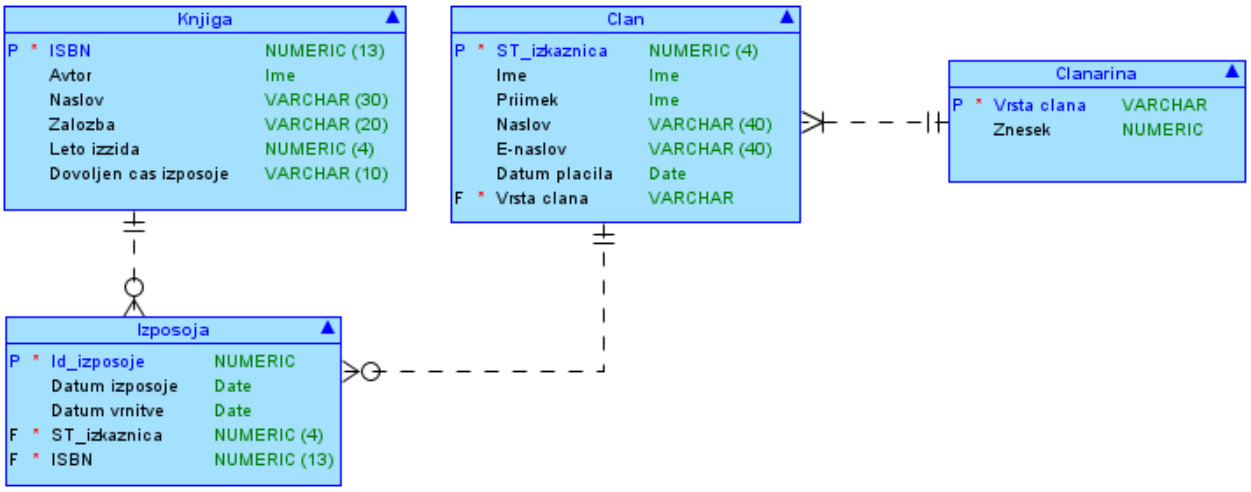
Član si v knjižnici v okviru ene izposoje lahko izposodi eno ali več knjig, prav tako lahko vsakokrat tudi vrne eno ali več predhodno izposojenih knjig. Za vsako knjigo želimo zabeležiti, kdaj je bila izposojena in kdaj je bila vrnjena.

Slika 57: Orodje Oracle SQL Developer Data Modeler



Slika 57 prikazuje delovno površino orodja Oracle SQL Developer Data Modeler ter izdelavo logičnega modela v njem, s katerim začnemo načrtovanje v tem orodju. Model je lahko prikazan na različne načine (možna je izbira treh različnih notacij).

Slika 58: Primer logičnega modela knjižnice (IZPOSOJA kot močni entitetni tip)

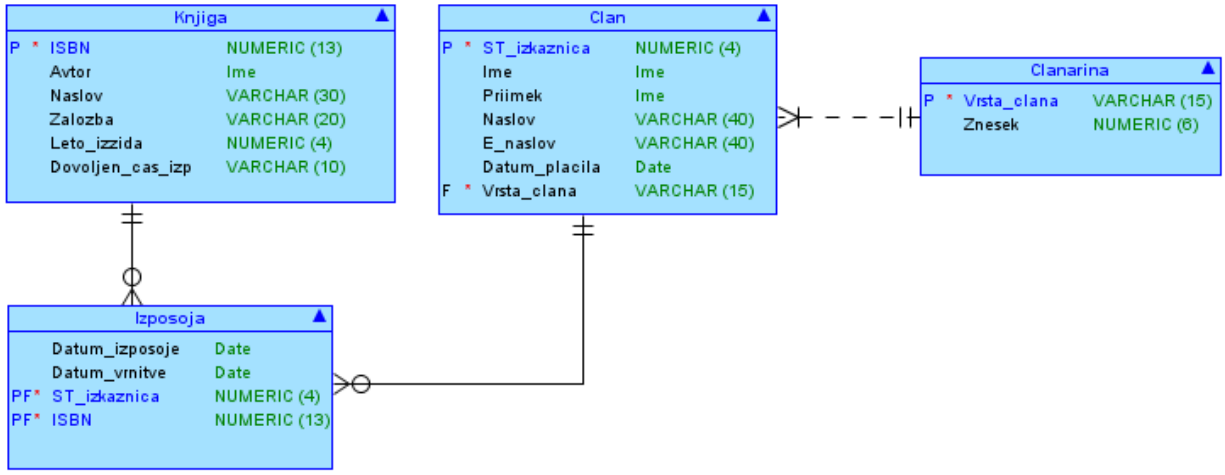


Slika 58 prikazuje logični model knjižnice v notaciji »information engineering«. Pri tem so enolični identifikatorji označeni z oznako P na levi strani atributa. Na desni strani so podani podatkovni tipi ali domene atributov (domene je potrebno predhodno definirati). Na levi strani so z oznako F (Relation UID) označeni atributi preko katerih so entitetni tipi povezani med seboj (ti bodo rezultirali v tuje ključe). Obveznost atributov ni prikazana.

Konceptualni model (ki je v tem orodju imenovan logični model) knjižnice glede na predhodno podane zahteve obsega 4 entitetne tipe: KNJIGA,CLAN, CLANARINA in IZPOSOJA. IZPOSOJA se na eni strani povezuje s KNJIGO, saj povezava pove, da je knjiga lahko nič ali večkrat izposojena. V drugo stran pa, da se vsaka izposoja nanaša na natanko določeno knjigo. Pri vsaki izposoji knjige zabeležimo datum izposoje in datum vrnitve. IZPOSOJA je po drugi strani povezana s CLANOM, saj za vsako izposojo zabeležimo tudi, komu smo knjigo izposodili. ČLAN pa si seveda lahko izposodi nič ali več knjig. Dodatno imamo še šifrant vseh vrst članov CLANARINA s cenikom zneskov članarine. CLAN je povezan s CLANARINO preko atributa vrsta_clana. Vsaj član namreč spada v natanko eno od skupin (otrok, študent, zaposleni, upokojenec…). V primeru s slike (Tabela 58) so vsi entitetni tipi močni, saj imajo enolični identifikator iz vrst svojih atributov. Črtkane črte povezav v tem orodju povezujejo močne entitetne tipe.

Entitetni tip IZPOSOJA lahko modeliramo tudi kot šibki tip (Slika 59). V tem primeru nimamo atributa id_izposoje, ampak označimo, da atributa ST_izkaznice in ISBN skupaj tvorita primarni identifikator entitetnega tipa IZPOSOJA. Ker gre za atributa entitetnih tipov s katerima je IZPOSOJA povezana, in ne za tipu lastni atribut, gre za šibki entitetni tip. Povezave šibkih entitetnih tipov z močnimi tipi imajo polne črte, da se ločijo od črt povezav med močnimi tipi, ki so črtkane.

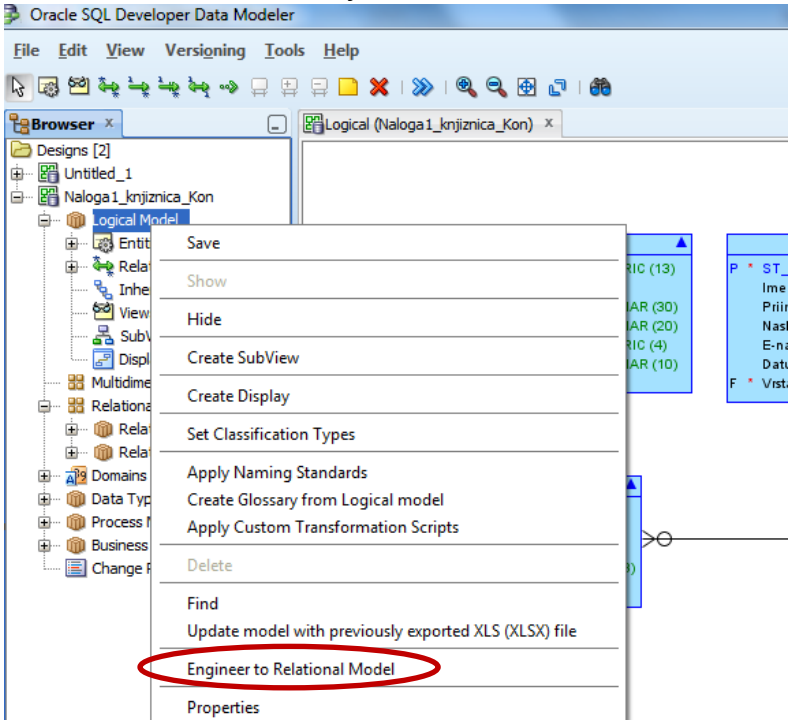
Slika 59: Primer logičnega modela knjižnice (IZPOSOJA kot šibki entitetni tip)



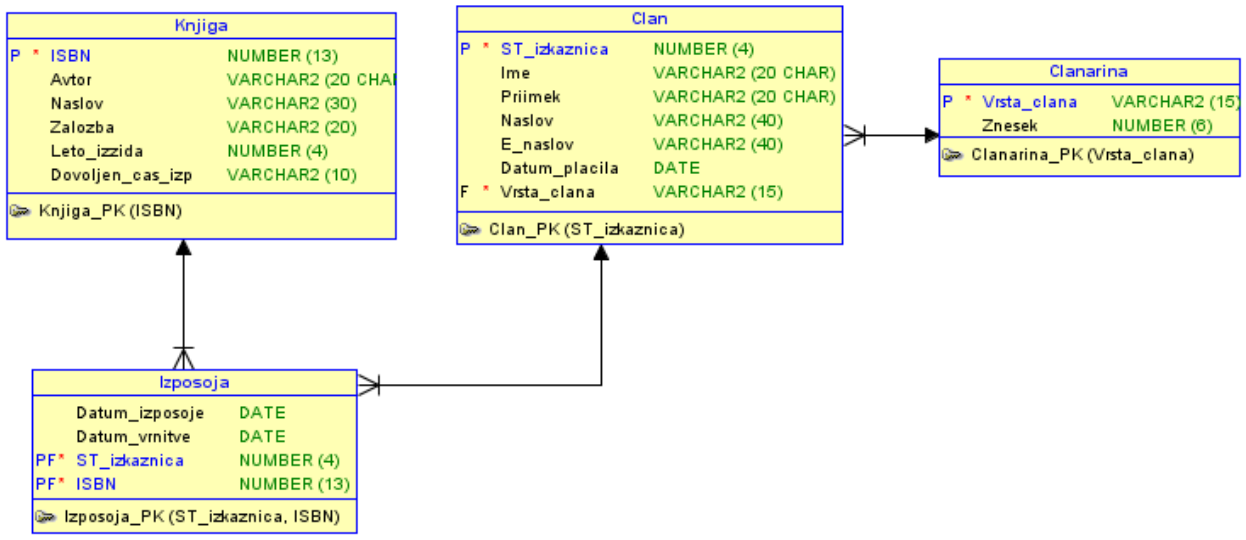
8.2.2 Izdelava relacijskega modela knjižnice

Nadalje preslikamo logični model v relacijskega. Najprej je potrebno kreirati nov relacijski model (postavimo se na Relational models in iz priročnega menija izberemo ukaz New Relational model). Nato se postavimo na izdelan logični model, ki ga želimo preslikati in iz priročnega menija izberemo ukaz Engineer to Relational Model (Slika 60).

Slika 60: Preslikava v relacijski model



Slika 61: Primer relacijskega modela knjižnice (IZPOSOJA kot šibki entitetni tip)

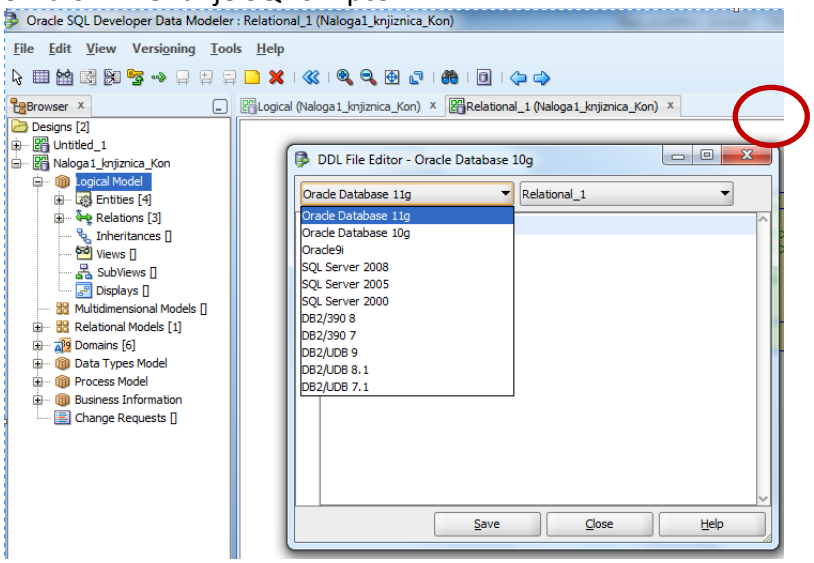


Po preslikavi je ustrezno spremenjena terminologija v oknih za nastavitev lastnosti. Namesto entitetnih tipov imamo tabele, namesto atributov stolpce, namesto primarnega identifikatorja primarni ključ itd. v skladu s terminologijo relacijskega modela. Relacijski model po preslikavi prikazuje slika (Slika 61). Nekoliko spremenjen je grafični prikaz povezav, saj so vse polnih črt, številnost ena pa je prikazana s puščico. Domene so preslikane v konkretne podatkovne tipa. Imeli smo domeno Ime, ki je bila uporabljena pri atributih Ime, Priimek in Avtor. Ker je bila domena definirana kot varchar(20) se to pri vseh treh navedenih atributih preslika v varchar2(20). Drugih bistvenih razlik med modeloma ni, saj smo že pri logičnem modelu upoštevali določene značilnosti relacijskega.

8.2.3 Izdelava fizičnega modela knjižnice

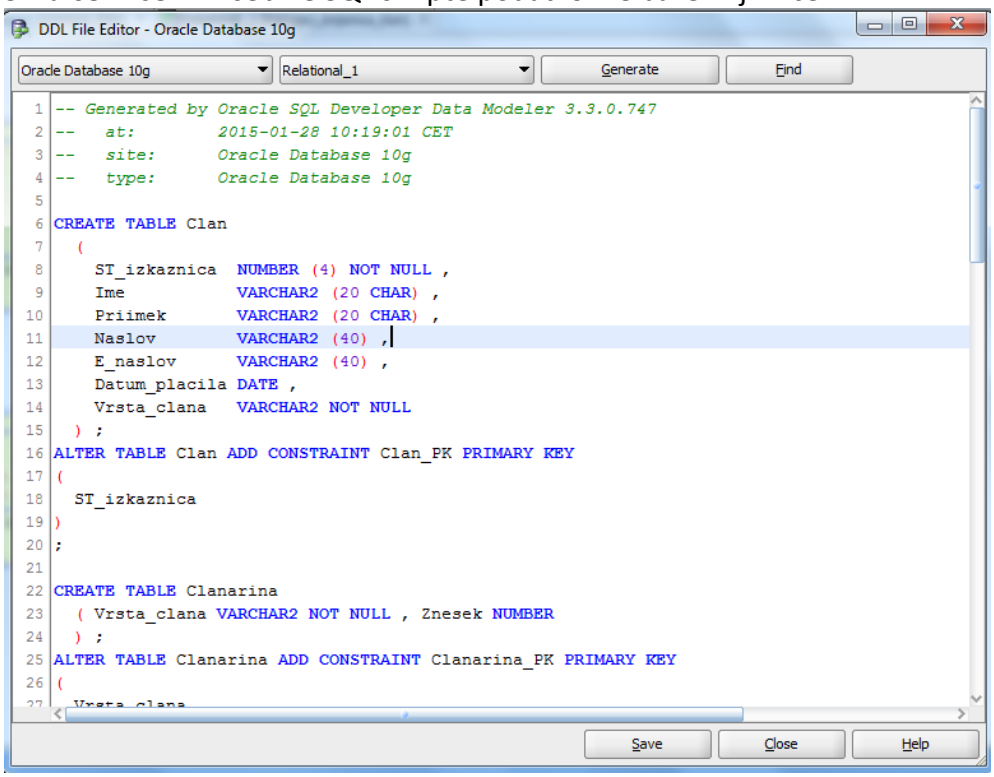
Na koncu še generiramo SQL skripto. Za to opravilo izberemo ukaz Generate DDL.

Slika 62: Kreiranje SQL skripte



V tem orodju šele v tem koraku izberemo enega od relacijskih SUPB. Kot prikazuje slika (Slika 62) imamo na voljo več različic SUPB-jev Oracle, SQL Server in DB2. Generirano skripto (Slika 63) shranimo, da jo bomo kasneje lahko zagnali v izbranem SUPB.

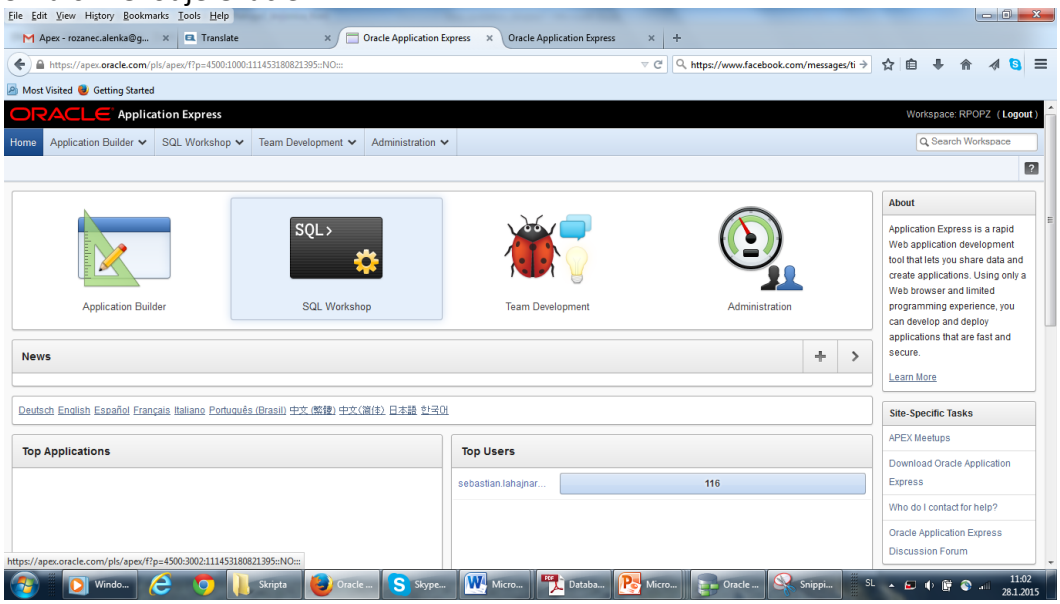
Slika 63: Izsek iz vsebine SQL skripte podatkovne baze knjižnice



8.3 Oracle APEX

Oracle APEX je spletno orodje, ki med drugim omogoča delo s podatkovno baze. Pred uporabo si je potrebno na spletni strani orodja (https://apex.oracle.com/) kreirati svojo delovno površino (ang. workspace). Podati moramo svoj e-naslov ter delovni površini izbrati ime in geslo, s katerima bomo kasneje do nje odstopali. Po zaključeni registraciji in uspešni prijavi se nam prikaže začetno okno delovne površine (Slika 64).

Slika 64: Orodje Oracle APEX

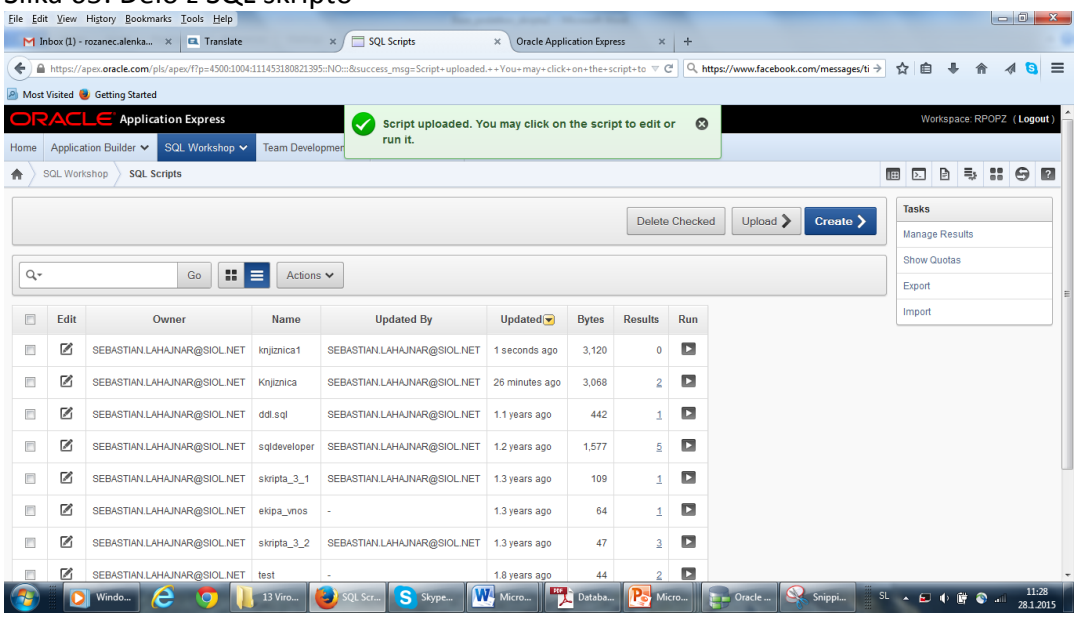


V nadaljevanju bodo opisana tipična opravila nad podatkovno bazo: kreiranje baze iz predhodno pripravljene SQL skripte, pregled in ročno kreiranje objektov baze (tabel, stolpcev, omejitev…) ter uporaba jezika SQL.

8.3.1 Kreiranje baze iz predhodno pripravljene SQL skripte

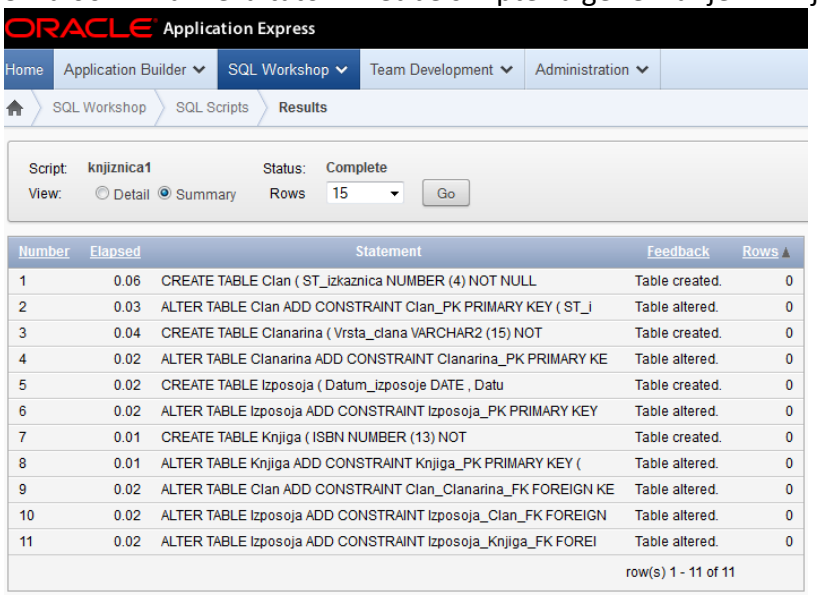
V poglavjih 8.1.3 in 8.2.3 smo pokazali kreiranje SQL skripte. Da bi skripto lahko izvedli, jo moramo najprej naložiti v orodje APEX. To storimo tako, da iz menija SQL workshop izberemo ukaz SQL Scripts. Gumb Upload nam omogoča skripto poiskati in jo naložiti v APEX. Skripta se sedaj nahaja na vrhu seznama skript. Orodje nam omogoča, da skripto prej še pogledamo in popravimo.

Slika 65: Delo z SQL skripto



Slika 65 prikazuje delo z SQL skripto. Po kliku na ukaz Run se skipta požene. Če skripta ni vsebovala napak, dobimo kreirano celotno podatkovno bazo, kot smo jo načrtovali. Če so napake, se deli skripte z napakami ne izvedejo (npr. lahko se nam določena tabela ne kreira) in dobimo samo del podatkovne baze iz načrta. Na koncu nam orodje javi rezultat uspešnosti (kateri objekti so bili kreirani in kateri zaradi morebitnih napak v sintaksi ne). V primeru izvedbe skripte za generiranje baze knjižnice so se uspešno izvedli vsi ukazi. Generirane so bile vse štiri tabele in tudi vse omejitve primarnih in tujih ključev (Slika 66).

Slika 66: Prikaz rezultatov izvedbe skripte za generiranje PB knjižnice



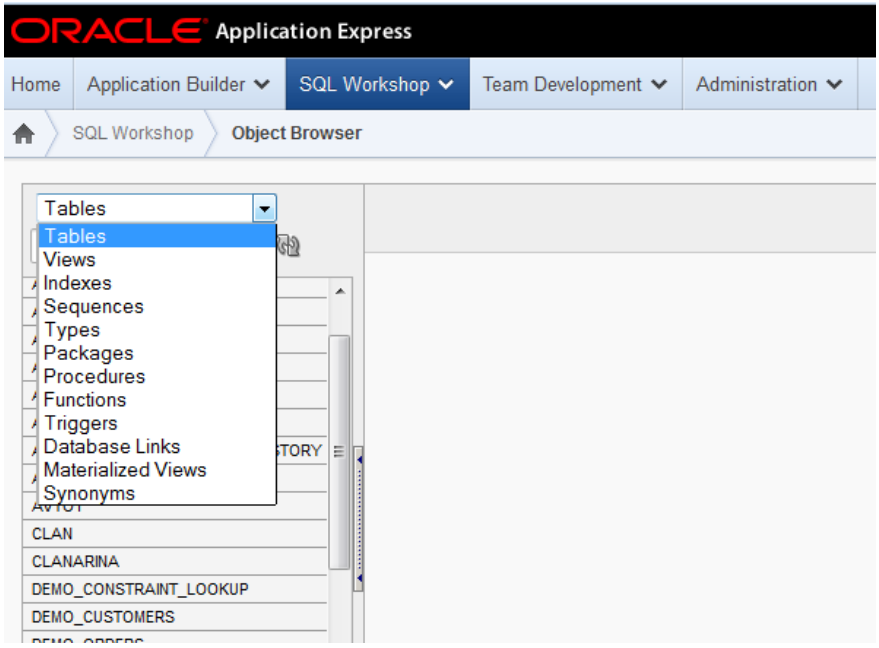
Number	Elapsed	Statement	Feedback	Rows
1	0.06	CREATE TABLE Clan (ST_izkaznica NUMBER (4) NOT NULL	Table created.	0
2	0.03	ALTER TABLE Clan ADD CONSTRAINT Clan_PK PRIMARY KEY (ST_i	Table altered.	0
3	0.04	CREATE TABLE Clanarina (Vrsta_clana VARCHAR2 (15) NOT	Table created.	0
4	0.02	ALTER TABLE Clanarina ADD CONSTRAINT Clanarina_PK PRIMARY KE	Table altered.	0
5	0.02	CREATE TABLE Izposoja (Datum_izposoje DATE , Datu	Table created.	0
6	0.02	ALTER TABLE Izposoja ADD CONSTRAINT Izposoja_PK PRIMARY KEY	Table altered.	0
7	0.01	CREATE TABLE Knjiga (ISBN NUMBER (13) NOT	Table created.	0
8	0.01	ALTER TABLE Knjiga ADD CONSTRAINT Knjiga_PK PRIMARY KEY (	Table altered.	0
9	0.02	ALTER TABLE Clan ADD CONSTRAINT Clan_Clanarina_FK FOREIGN KE	Table altered.	0
10	0.02	ALTER TABLE Izposoja ADD CONSTRAINT Izposoja_Clan_FK FOREIGN	Table altered.	0
11	0.02	ALTER TABLE Izposoja ADD CONSTRAINT Izposoja_Knjiga_FK FOREI	Table altered.	0
			row(s) 1 - 11 of 11	

8.3.2 Pregled in ročno kreiranje različnih objektov podatkovne baze

Vse objekte podatkovne baze lahko pogledamo v brskalniku objektov (ukaz Object Browser menija SQL workshop), kjer lahko izbiramo objekte po vrstah (npr. izberemo pogled tabel, pogledov, indeksov…).

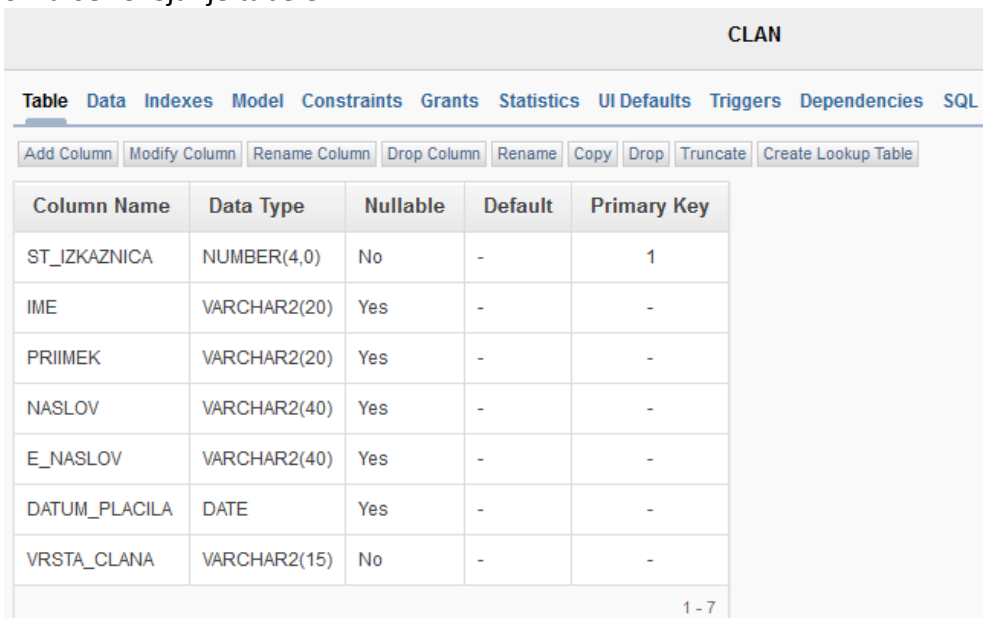
Slika 67 prikazuje brskalnik objektov ter nekatere od tabel (CLAN, CLANARINA ter nekatere DEMO tabele). Po izbiri določenega objekta, ga lahko urejamo. Brskalnik nam omogoča tudi ročno kreiranje novih objektov, npr. dodajanje tabele in njenih elementov, ki jo zaradi novih zahtev uporabnikov pri načrtovanju nismo modelirali.

Slika 67: Brskalnik objektov baze (Object Browser)



V primeru izbire objekta tipa tabela, npr. CLAN se nam prikaže okno kot ga prikazuje slika (Slika 68). Ko stojimo na zavihku Table lahko pregledujemo in urejamo njeno strukturo: dodajamo stolpce, spreminjamo lastnosti stolpcev, brišemo stolpce, preimenujemo, kopiramo ali brišemo celotno tabelo itd.

Slika 68: Urejanje tabele



Zavihek Data nam omogoča vnos podatkov v tabelo. Preko zavihka Indexes pregledujemo indekse, lahko dodajamo tudi nove. Zavihek Model nam grafično prikaže tisti del podatkovnega modela, ki se nanaša na izbrano tabelo (prikaže izbrano tabelo in vse z n njo povezane tabele). Po zavihkom Constraints so zbrane omejitve tabele (nrp. primarni, tuji ključi, atributi ki ne smejo biti NULL). Zavihek Grants omogoča pregled, dodeljevanje in odvzem pravic nad tabelo. Zavihek Triggers pa je namenjen pisanju baznih prožilcev.

Da bi lahko pokazali uporabo poizvedovanja, moramo najprej vnesti v bazo knjižnice nekaj podatkov. To lahko storimo preko zaviha Data ali pa uporabimo kar sam SQL in njegov stavek INSERT. Vrstni red vnosa podatkov v različne tabele je pomemben, saj omejitve tujega ključna zahtevajo vnos ene od vrednosti iz povezane tabele. Zato bomo najprej napolnili tabelo CLANARINA in nato dodali dva člana v tabelo CLAN. Rezultat po nekaj vnosih prikazujeta tabeli (Tabela 56 in Tabela 57).

Tabela 56: Tabela CLANARINA z vnesenimi podatki

Table Data Indexes Model Constraints

Query Count Rows Insert Row

EDIT	VRSTA_CLANA	ZNESEK
	Študent	10
	Zaposleni	30
	Upokojenec	10
	Otrok	0

row(s) 1 - 4 of 4

Tabela 57: Tabela CLAN z vnesenimi podatki

CLAN

Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers Dependencies SQL

Query Count Rows Insert Row

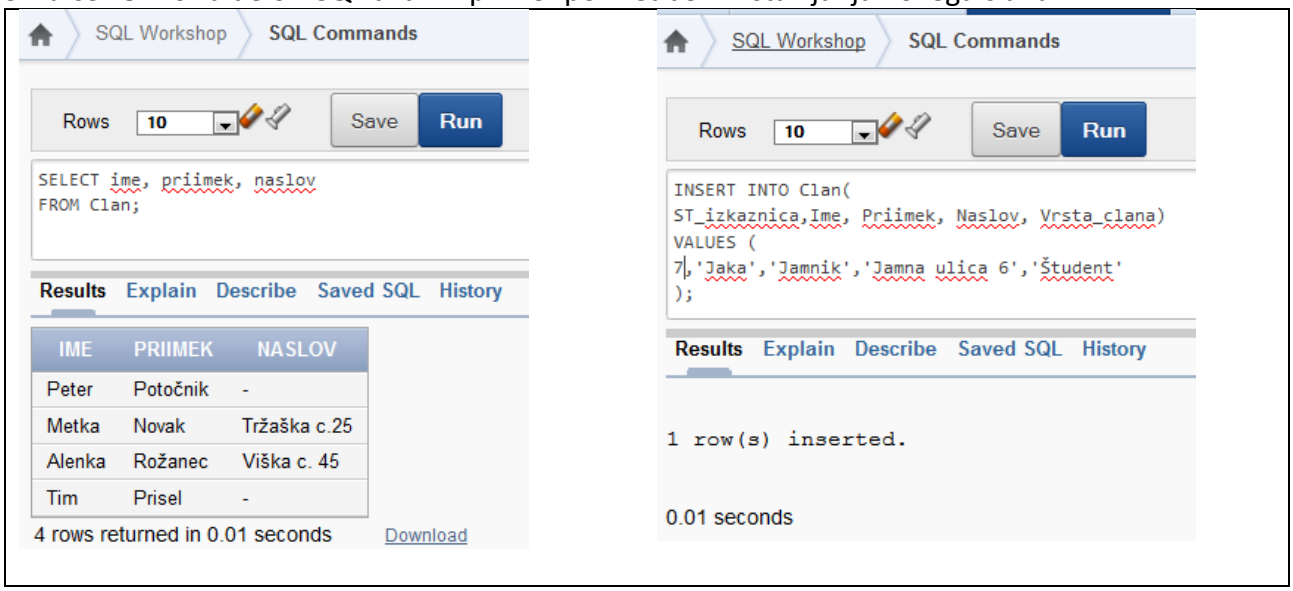
EDIT	ST_IZKAZNICA	IME	PRIIMEK	NASLOV	E_NASLOV	DATUM_PLACILA	VRSTA_CLANA
	1	Metka	Novak	Tržaška c.25	metka.novak@gmail.com	01/28/2015	Študent
	2	Alenka	Rožanec	Viška c. 45	-	01/03/2015	Zaposleni
	3	Tim	Prisel	-	tim.prisel@gmail.com	12/12/2014	Zaposleni

row(s) 1 - 3 of 3

8.3.3 Uporaba jezika SQL v APEX-u

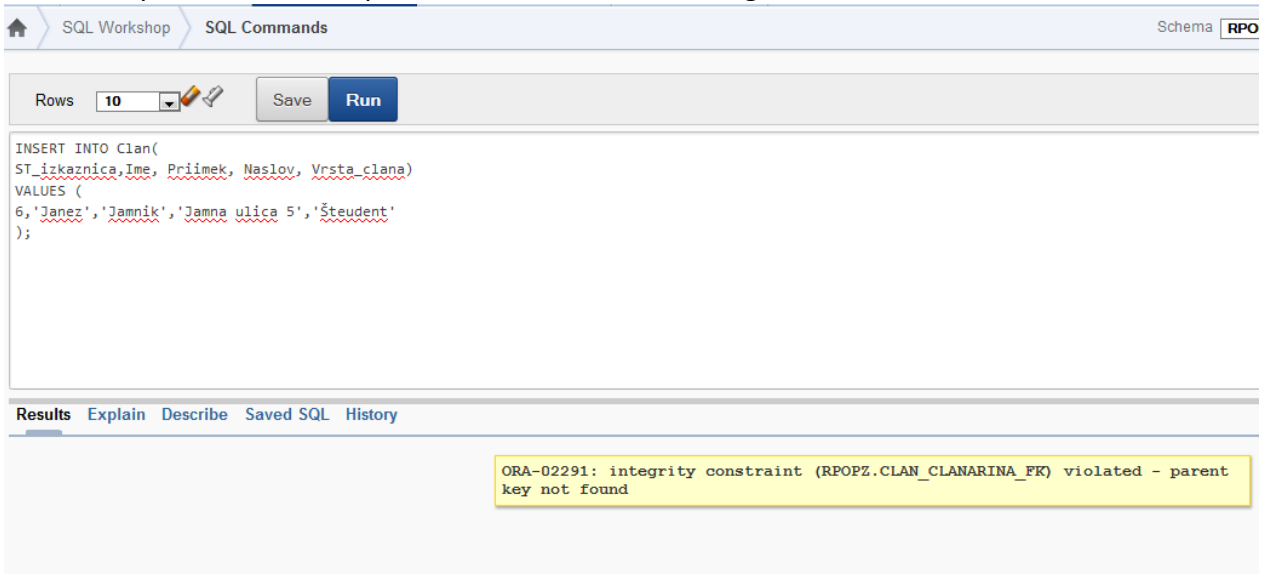
Izvajanje SQL ukazov v orodju APEX najdemo pod ukazom SQL Workshop -> SQL Commands (Slika 69). V primeru, da izvedemo SQL poizvedbo, se rezultat pokaže v spodnjem delu okna. Če pa gre za SQL stavke dodajanja, brisanja itd. se nam izpiše obvestilo o uspešnosti izvedbe določenega stavka (npr. vrstica je bila dodana - dodali smo novega člana). Če je sintaksa napačna se izpiše obvestilo o napaki (
Slika 70).

Slika 69: Okno za delo z SQL ukazi – primer poizvedbe in vstavljanja novega člana



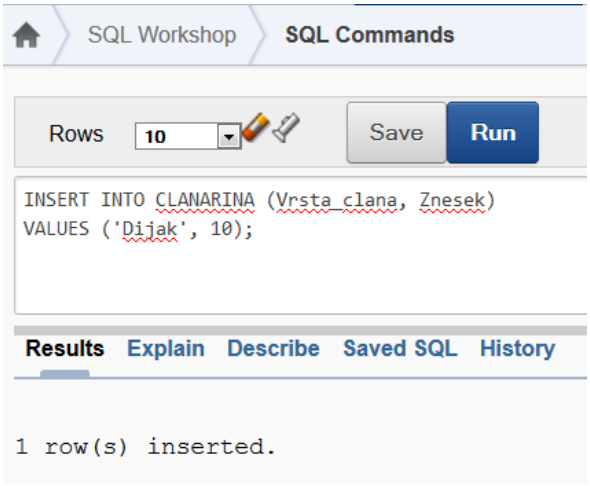
Primer poizvedbe Primer vstavljanja novega člana

Slika 70: Izpis obvestila o napaki – kršitev referenčne integritete



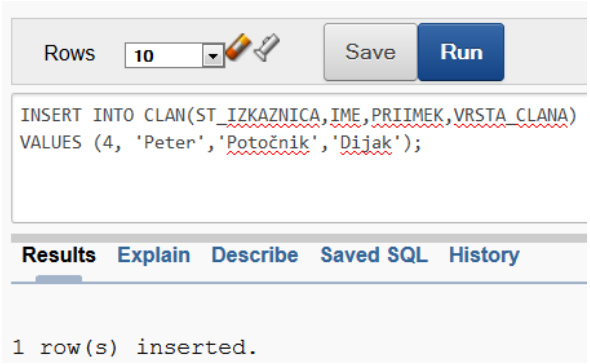
Vnesimo novo vrsto člana v tabelo CLANARINA. V prazno polje vpišemo stavek INSERT, ki bo dodal še skupino dijakov, z ustrezno sintakso: povemo ime tabele, navedemo imena stolpcev in na koncu vrednosti, ki jih želimo v bazo vpisati. Če je sintaksa pravila, se vrstica doda (Slika 71).

Slika 71: Vnos nove vrste člana v tabelo CLANARINA z uporabo SQL stavka INSERT



Sedaj lahko dodamo v tabelo CLAN novega člana, ki je dijak.

Slika 72: Dodajanje dijaka v tabelo CLAN



Pri dodajanju novega zapisa smo dodali dijaka, vendar nismo nastavili vrednosti vseh stolpcev. Ker naslov, e-naslov in datum_placila nimajo definirane omejitve NOT NULL se zapis lahko doda brez da vnesemo te vrednosti. Tabela 58 prikazuje podatke v tabeli CLAN po dodajanju zadnjega zapisa.

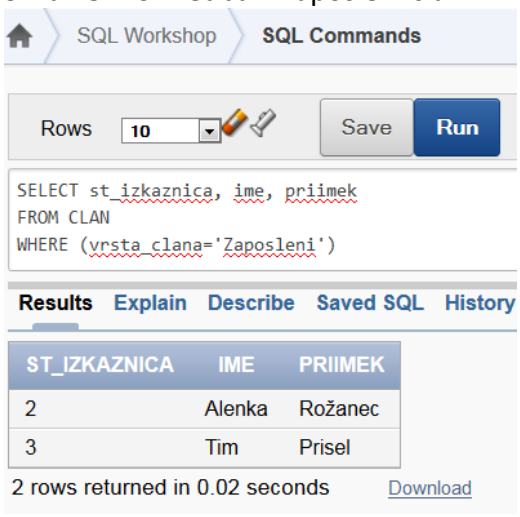
Tabela 58: Tabela CLAN

Query | Count Rows | Insert Row

EDIT	ST_IZKAZNICA	IME	PRIIMEK	NASLOV	E_NASLOV	DATUM_PLACILA	VRSTA_CLANA
	4	Peter	Potočnik	-	-	-	Dijak
	1	Metka	Novak	Tržaška c.25	metka.novak@gmail.com	01/28/2015	Študent
	2	Alenka	Rožanec	Viška c. 45	-	01/03/2015	Zaposleni
	3	Tim	Prisel	-	tim.prisel@gmail.com	12/12/2014	Zaposleni

Sedaj podajmo še primer izdelave poizvedbe nad tabelo CLAN. Izpisali bomo številke izkaznic, imena in priimke vseh članov, ki so zaposleni. Če pogledamo predhodno sliko vidimo, da imamo dva takšna člana. Slika 73 prikazuje poizvedbo (SELECT stavek) in njen rezultat.

Slika 73: Poizvedba – zaposleni člani



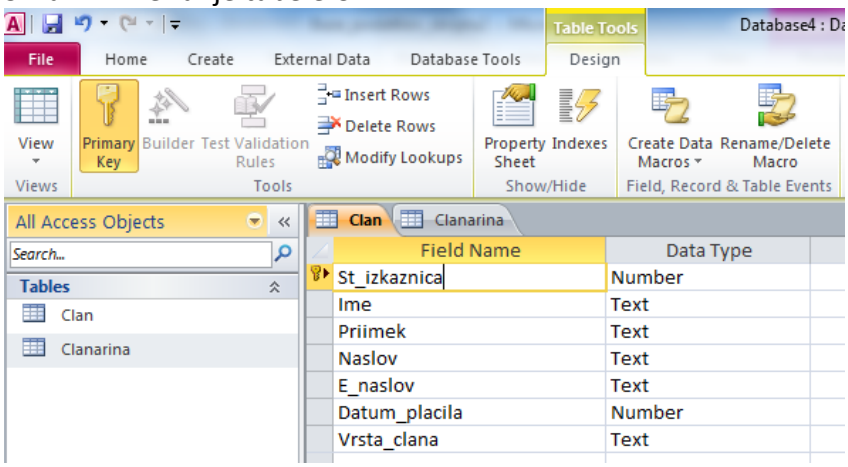
8.4 MS Access

MS Access je danes del zbirke Microsoft Office. Njegov izvor sega v daljno leto 1992. Orodje omogoča kreiranje podatkovne baze pa tudi razvoj aplikacij z uporabo objektno-orientiranega programskega jezika Visual Basic for Applications (VBA). V nadaljevanju se bomo osredotočili le na opravila v povezavi s podatkovno bazo.

8.4.1 Ročno kreiranje podatkovne baze v MS Accesu

Ko zaženemo program MS Access, izberemo novo prazno bazo (ang. blank database), nato se nam privzeto že kreira prva prazna tabela. Za izdelavo podatkovne baze uporabljamo načrtovalski pogled (ang. design view), za vnos podatkov pa pogled preglednice (ang. datasheet view). Spodnja slika (Slika 74) prikazuje okno za izdelavo tabele in sicer zopet kreiramo tabeli CLANARINA in KNJIZNICA kot v predhodnem poglavju oz. primeru. Določimo primarni ključ in podatkovne tipe polj (atributov).

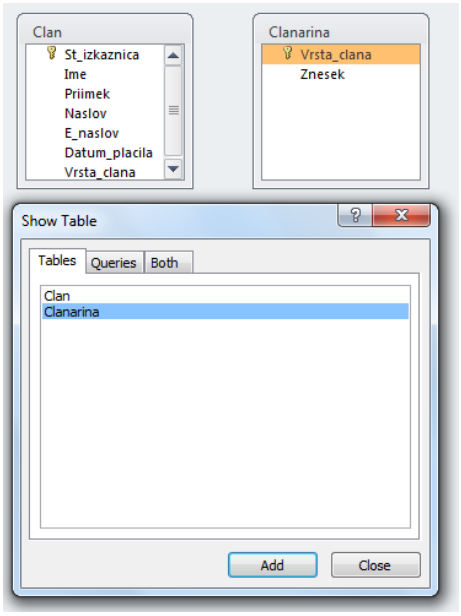
Slika 74: Kreiranje tabele CLAN



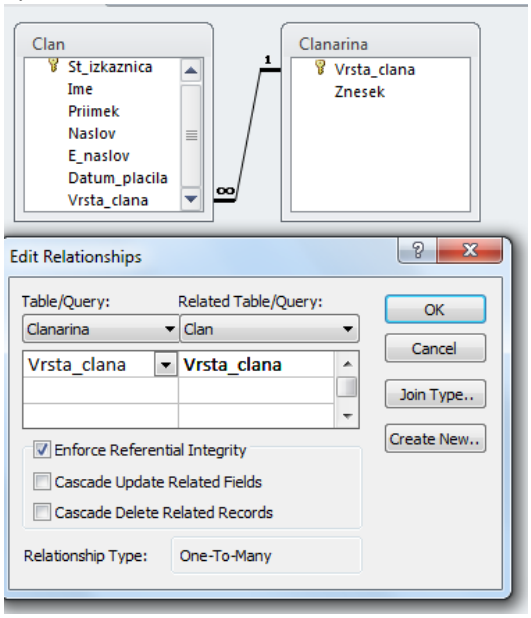
Nato tabeli med seboj še povežemo z uporabo pripomočka Relationships. Najprej dodamo obe tabeli na delovno površino (Slika 75a) nato pa ju povežemo (Slika 75b). Polji (atributa) Vrsta_clana iz obeh tabel povežemo med seboj, pri čemer kot že vemo ta atribut predstavlja primarni ključ tabele

CLANARINA in tuji ključ tabele CLAN. Vključimo še omejitev referenčne integritete, ki bo preprečevala vnos članov takšnih vrst, ki jih nimamo v tabeli CLANARINA.

Slika 75: Dodajanje in povezovanje tabel (Relationships)



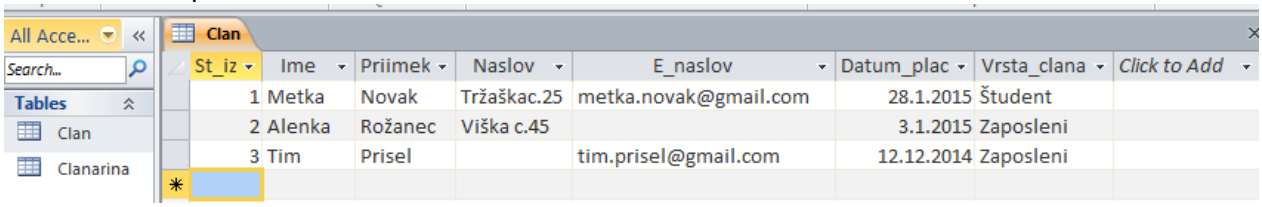
Slika 75a: Dodajanje tabel



Slika 75b: Povezovanje tabel

Sedaj vnesemo podatke kot v predhodnem primeru, da bomo v nadaljevanju lahko prikazali še izdelavo poizvedb. Slika 76 prikazuje tabelo CLAN v pogledu preglednice, ki omogoča vnos podatkov.

Slika 76: Vnos podatkov v tabelo CLAN

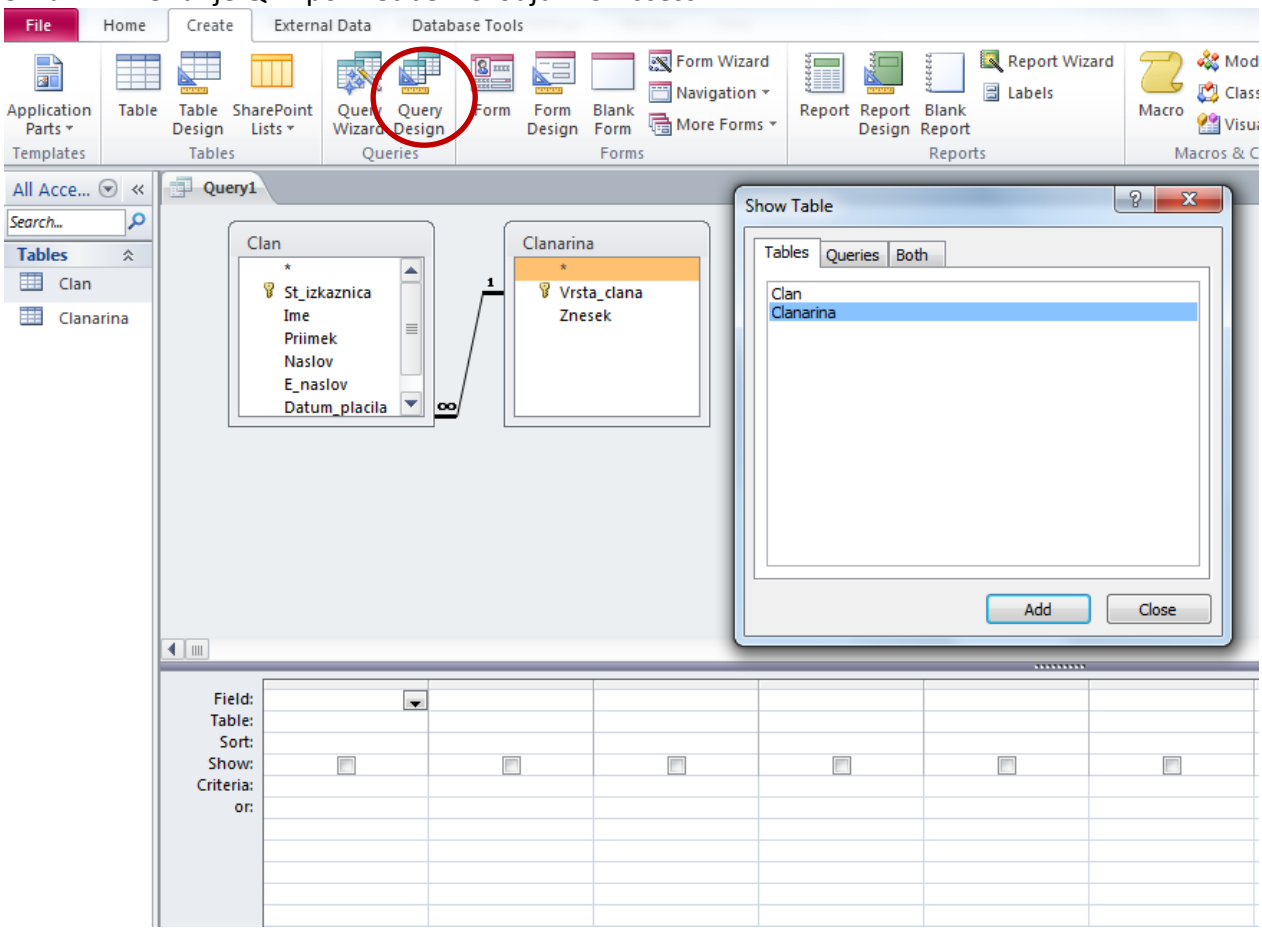


8.4.2 Poizvedovanje z uporabo jezika QBE

MS Access vsebuje orodja za enostavno poizvedovanje s pomočjo jezika QBE (Query by example). Gre za poizvedovanje z uporabo primerov, kar je zelo priročno za tiste uporabnike, ki ne poznajo jezika SQL. Poizvedbe QBE se v ozadju avtomatsko prevedejo v SQL, ki ga lahko pogledamo in tudi nadalje urejamo. Tako orodje QBE pomeni tudi dober pripomoček za učenje jezika SQL.

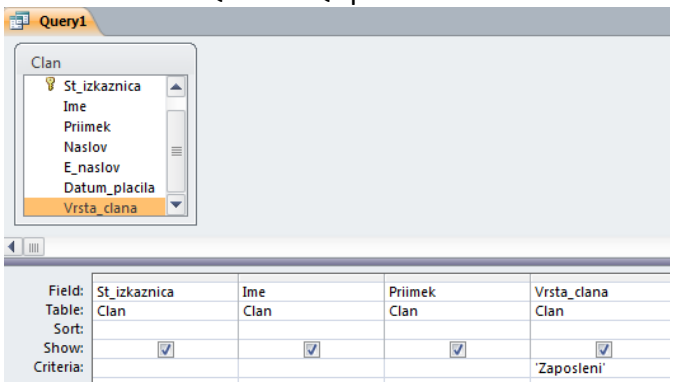
Poizvedbe v Accessu kreiramo v meniju Create z izbiro ukaza Query Design (Slika 77). Lahko pa uporabimo tudi čarovnika za izdelavo poizvedb (Query Wizard).

Slika 77: Kreiranje QBE poizvedbe v orodju MS Access

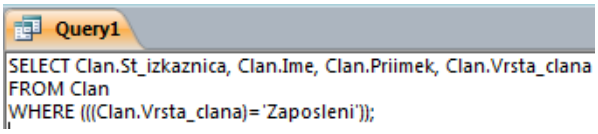


Kreiranje poizvedbe poteka tako, da najprej na delovno površino dodamo tabele, iz katerih bomo poizvedovali (to je lahko ena ali več medsebojno povezanih tabel). V našem primeru smo dodali tabeli CLANARINA in CLAN (Slika 77).

Slika 78: Primer QBE in SQL poizvedbe



Slika 78a: QBE poizvedba

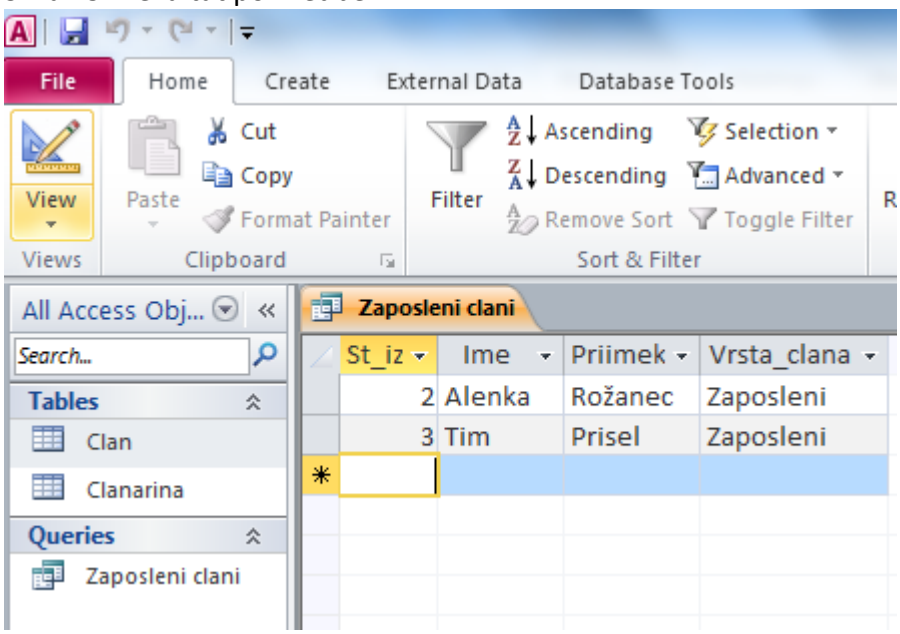


Slika 78b: SQL poizvedba

V spodnjem delu te slike se prikažejo vrstice, s pomočjo katerih izdelamo poizvedbo. Sistematičen opis jezika QBE se nahaja v poglavju 6.4. Izpisali bomo številke izkaznic, imena in priimke vseh članov, ki so zaposleni. Ker potrebujemo le tabelo CLAN, smo tabelo CLANARINA odstranili. Če pogledamo sliko (Slika 76) vidimo, da imamo dva takšna člana. Slika 78 prikazuje QBE poizvedbo in SELECT stavek v SQL-u. Da dobimo rezultat, moramo poizvedbo zagnati s klikom na ukaz Run. Poizvedbo še

poimenujemo npr. Zaposleni_clani in shranimo. Poizvedba se doda v seznam vseh objektov baze v razdelek namenjen poizvedbam (Queries). Rezultat poizvedbe prikazuje Slika 79.

Slika 79: Rezultat poizvedbe



Vprašanja za ponavljanje

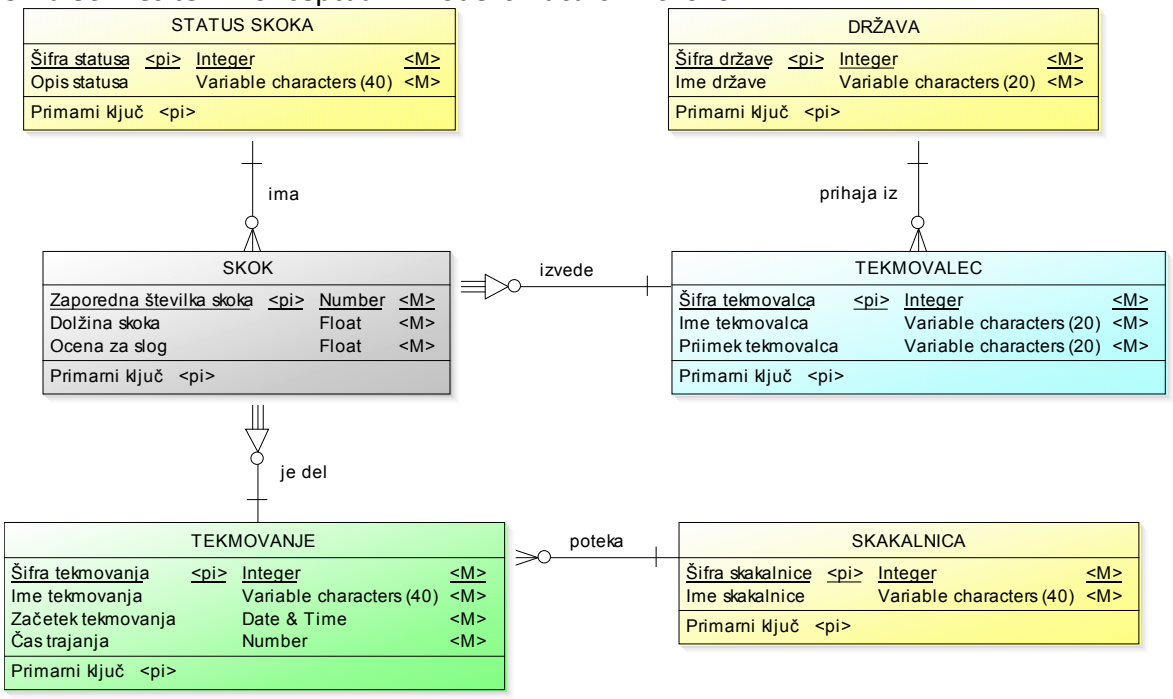
1. Kakšno vrsto računalniških orodij uporabljamo za konceptualno in logično načrtovanje podatkovne baze?
2. Katere so ključne funkcionalnosti, ki jih orodja za načrtovanje podatkovne baze nudijo načrtovalcu oziroma katere so njihove prednosti glede na uporabo risarskih orodij ali lista papirja?
3. Katera orodja za načrtovanje podatkovne baze poznate?
4. Katera orodja za izdelavo fizične podatkovne baze poznate?
5. Kako sta ti dve vrsti orodij med seboj povezani?
6. Katere načine kreiranja elementov podatkovne baze nudi orodje Oracle APEX?
7. Katere druge funkcije poleg kreiranja elementov podatkovne baze orodje Oracle APEX še nudi?
8. Katere načine kreiranja elementov podatkovne baze nudi orodje MS Access?
9. Katere druge funkcije poleg kreiranja elementov podatkovne baze orodje MA Access še nudi?

9 Rešitve nalog

9.1 Rešitve nalog poglavja 4

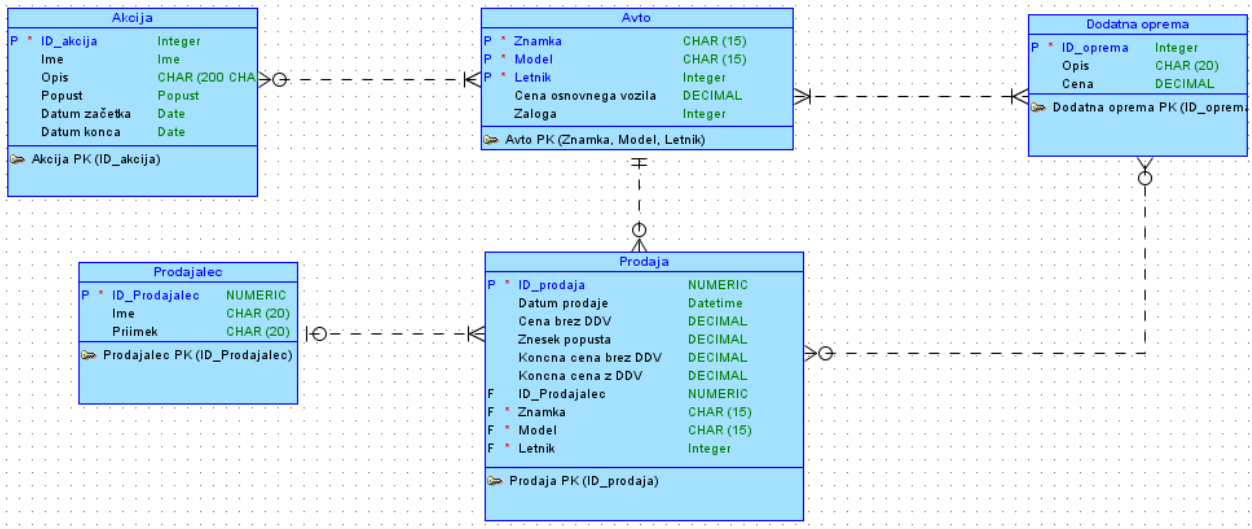
Rešitev naloge 4.1 - Smučarski skoki

Slika 80: Rešitev - konceptualni model smučarskih skokov



Rešitev naloge 4.2 - Prodajalna avtomobilov TineCars d.o.o.

Slika 81: Rešitev - konceptualni model prodajalne avtomobilov



9.2 Rešitve nalog poglavja 5.2

Rešitev naloge 5.2.1 - Logični model (domena smučarskih skokov)

TEKMOVALEC (Šifra tekmovalca, Ime tekmovalca, Priimek tekmovalca, #Šifra države)
DRŽAVA (Šifra države, Ime skakalnice)
SKAKALNICA (Šifra skakalnice, Ime skakalnice)
TEKMOVANJE (Šifra tekmovanja, Ime tekmovanja, Začetek tekmovanja, Čas trajanja tekmovanja, #Šifra skakalnice)
SKOK (Zaporedna številka skoka, #Šifra tekmovalca, #Šifra tekmovanja, Dolžina skoka, Ocena za slog, #Šifra statusa)
STATUS_SKOKA (Šifra statusa, Opis statusa)

Rešitev naloge 5.2.2 - Logični model (domena prodajalna avtomobilov)

AVTO (Znamka, Model, Letnik, Cena, Zaloga)
DODATNA_OPREMA (ID_oprema, Opis, Cena)
PRODAJALEC (ID_Prodajalec, Ime, Priimek)
AKCIJA (ID_Akcija, Ime, Opis, Popust, Datum začetka, Datum konca)
PRODAJA (ID_prodaja, Datum prodaje, Cena brez DDV, Znesek popusta, Koncna cena brez DDV, Koncna cena z DDV, #ID_Prodajalec, #Znamka, #Model, #Letnik)

Dve novi tabeli, kjer so bile povezave številnosti m:n:
DODATNA_OPREMA_ZA_AVTO (#ID_oprema, #Znamka, #Model, #Letnik)
AKCIJA_ZA_AVTO (#ID_akcija, #Znamka, #Model, #Letnik)

Rešitev naloge 5.2.3 Logični model (domena videoteka)

ZVRST (Sifra_zvrsti, Naziv)
FILM (Kat_st, Naslov, Cena_na_dan, Glavni_igralec, Reziser)
IMA_ZANR (#Kat_st, #Sifra_zvrsti) *// nova tabela med ZVRST in FILM zaradi povezave m:n*
FILM_ZA_IZPOSOJO (St_filma, Status, #Kat_st, #ID_podruznice)
IZPOSOJA (St_izposoje, D_izposoje, D_vrnitve, Cena_skupaj, #Clan_st)
ČLAN (Clan_st, Ime, Priimek, Datum_reg, #ID_podruznice)
PODRUZNICA (ID_podruznice, Ulica, Tel_st, #Post_st, #ID_manager)
MESTO (Post_st, Kraj)
OSEBJE (ID_osebja, Ime, Pozicija, Placa, #ID_podruznice)

9.3 Rešitve nalog poglavja 5.3

Rešitev naloge 5.3.1
Nenormalizirana relacija: *Najem (Št_najemnika, Ime_najemnika, (Št_nepr, Naslov_nepr, Datum_z, Datum_k, Cena, Št_lastnika, ime_lastnika))*

Relacije v 3.NO:

Najemnik (*Št_najemnika, Ime_najemnika***)**
Najem(#*Št_najemnika, #Št_nepr, Datum_z, Datum_k***)**
Nepremičnina (*Št_nepr, Naslov_nepr, Cena, #Št_lastnika)*
Lastnik*(Št_lastnika, ime_lastnika)*

Rešitev naloge 5.3.2
Nenormalizirana relacija: *P(ŠifraKaseta, ŠifraFilm, NaslovFilm, RežijaFilm, DolžinaFilm, (EMŠO, ImeStranka, UlicaStranka, PoštaStranka, KrajStranka, ČasIzposoje))*

Relacije v 3.NO:

Kaseta(*ŠifraKaseta, #ŠifraFilm* **)**
Film(*ŠifraFilm, NaslovFilm, RežijaFilm, DolžinaFilm***)**
Izposoja(#*ŠifraKaseta,#EMŠO, ČasIzposoje***)**
Oseba(EMŠO**,** *ImeStranka, UlicaStranka, #PoštaStranka***)**
Kraj (*PoštaStranka, KrajStranka***)**

Rešitev naloge 5.3.3
Nenormalizirana relacija: *R(DavčnaŠt, Ime, Priimek, Ulica, PoštnaŠt, Kraj, (ŠifraIzdelka, ImeIzdelka, ŠifraKategorije, ImeKategorije, Cena, Kolicina, DatumČasNakupa)).*

Relacije v 3.NO:
Oseba (*DavčnaŠt, Ime, Priimek, Ulica, #PoštnaŠt*)
Kraj(*PoštnaŠt, Kraj*)
Izdelek (*ŠifraIzdelka, ImeIzdelka,Cena, #ŠifraKategorije*)
Kategorija (*ŠifraKategorije, ImeKategorije*)
Nakup(*#DavčnaŠt, #ŠifraIzdelka, DatumČasNakupa, Kolicina*)

9.4 Rešitve nalog poglavja 6

Rešitev naloge 6.3.1 – SQL nad domeno študentskega IS:

a. SELECT * FROM predavatelj

b. SELECT * FROM predmet

c. SELECT naziv, imepriimek
   FROM predavatelj

d. SELECT vpisnast,emso,to_char(dtrojstva,'dd.mm.rrrr') dtrojstva, naslov||' ' ||kraj naslov
FROM student

e. SELECT naziv, imepriimek
FROM predavatelj
WHERE imepriimek LIKE 'M%'

f. SELECT *
FROM student
WHERE kraj LIKE '%Ljubljana%'
AND dtrojstva<to_date('1.1.1995','dd.mm.rrrr')

g. SELECT * FROM predmet
WHERE letnik in (1,2) AND semester='Z'
ORDER BY naziv ASC

h. SELECT *
FROM student
WHERE dtrojstva BETWEEN to_date('1.1.1990','dd.mm.rrrr') AND
to_date('1.1.1995','dd.mm.rrrr')
ORDER BY dtrojstva DESC

i. SELECT *
FROM student
WHERE naslov IS NOT NULL AND length(vpisnast)=10
ORDER BY imepriimek

Rešitev naloge 6.3.2 – SQL nad domeno študentskega IS (poizvedbe nad dvema ali več tabelami):

a. SELECT predmet.naziv, predavatelj.naziv, predavatelj.imepriimek
FROM predmet **INNER JOIN** predavatelj ON predmet.idpredavatelj=predavatelj.id
ORDER BY predmet.naziv, predavatelj.imepriimek

b. SELECT predmet.naziv, predavatelj.naziv, predavatelj.imepriimek
FROM predmet **LEFT OUTER JOIN** predavatelj ON predmet.idpredavatelj=predavatelj .id
ORDER BY predmet.naziv, predavatelj.imepriimek

c. SELECT predmet.naziv, predavatelj.naziv, predavatelj.imepriimek
FROM predmet **RIGHT OUTER JOIN** predavatelj ON predmet.idpredavatelj=predavatelj.id
ORDER BY predavatelj.imepriimek

d. SELECT student.imepriimek, predmet.naziv, predavatelj.imepriimek, indeks.ocena
FROM indeks JOIN student ON indeks.idstudent=student.id
JOIN predmet ON indeks.idpredmet=predmet.id
JOIN predavatelj ON predmet.idpredavatelj=predavatelj.id

```
WHERE indeks.ocena is not null
ORDER BY student.imepriimek
```

e.
```
SELECT student.imepriimek, predmet.naziv, predavatelj.imepriimek, indeks.ocena,
indeks.ocenavaj
FROM indeks INNER JOIN student ON indeks.idstudent=student.id
INNER JOIN predmet ON indeks.idpredmet=predmet.id
INNER JOIN predavatelj ON predmet.idpredavatelj=predavatelj.id
WHERE indeks.ocena IN (7,8,9) AND predmet.letnik=1
ORDER BY student.imepriimek
```

Rešitev naloge 6.3.3 – SQL nad domeno študentskega IS (uporaba agregatnih funkcij):

a.
```
SELECT AVG(ocena)
FROM indeks
```

b.
```
SELECT AVG(ocena)
FROM indeks INNER JOIN predmet ON indeks.idpredmet=predmet.id
WHERE predmet.letnik=1 AND indeks.ocenavaj IS NULL
```

c.
```
SELECT  ocena "Ocena", count(indeks.idstudent) "Število študentov z oceno"
FROM indeks
WHERE ocena is NOT NULL
GROUP BY ocena
```

d.
```
SELECT  predmet.letnik "Letnik", AVG(indeks.ocena) "Povprečje"
FROM indeks JOIN predmet ON indeks.idpredmet=predmet.id
WHERE indeks.ocena>5
GROUP BY predmet.letnik
ORDER BY predmet.letnik
```

e. samo letniki, kjer imajo študentje povprečno oceno višjo od 8
```
SELECT  predmet.letnik "Letnik", AVG(indeks.ocena) "Povprečje"
FROM indeks INNER JOIN predmet ON indeks.idpredmet=predmet.id
WHERE indeks.ocena>5
GROUP BY predmet.letnik
HAVING AVG(indeks.ocena)>8
ORDER BY predmet.letnik
```

f.
```
SELECT  predmet.naziv, predavatelj.imepriimek
FROM predmet INNER JOIN predavatelj ON predmet.idpredavatelj=predavatelj.id
WHERE predmet.id IN
 (SELECT indeks.idpredmet
  FROM indeks
```

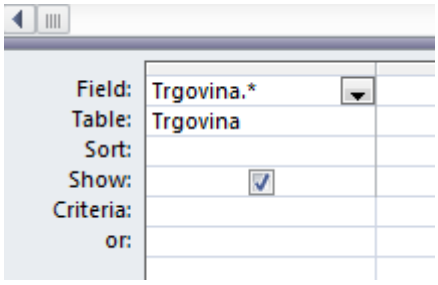
WHERE indeks.ocena>8)

g. SELECT student.vpisnast, student.imepriimek
FROM student INNER JOIN indeks ON student.id=indeks.idstudent
INNER JOIN predmet ON indeks.idpredmet=predmet.id
WHERE predmet.semester='Z';

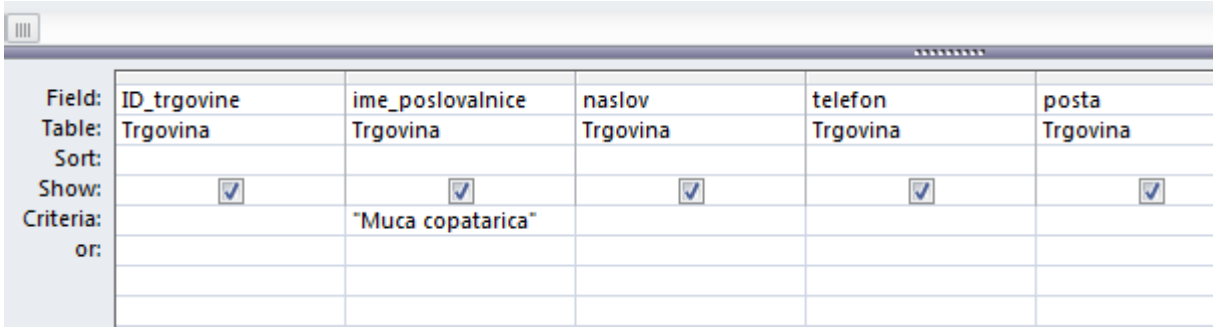
h. SELECT student.vpisnast,student.imepriimek "Študent", avg(indeks.ocena) "Ocena"
FROM indeks INNER JOIN student ON indeks.idstudent=student.id
WHERE indeks.ocena BETWEEN 6 AND 10
GROUP BY student.vpisnast,student.imepriimek
HAVING AVG(indeks.ocena)>
(SELECT AVG(indeks1.ocena)
FROM indeks indeks1
WHERE indeks1.ocena BETWEEN 6 AND 10)

Rešitev naloge 6.3.2 – QBE na domeno trgovskega podjetja

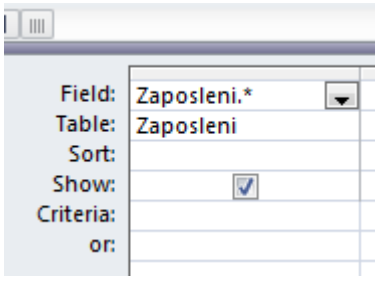
a. Izpiši vse trgovine.



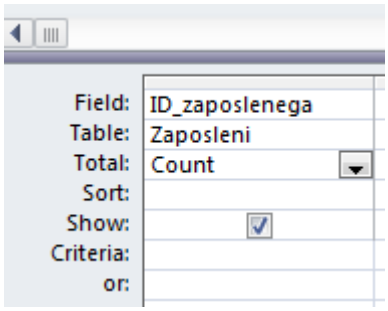
b. Izpiše podatke o trgovini "Muca copatarica".



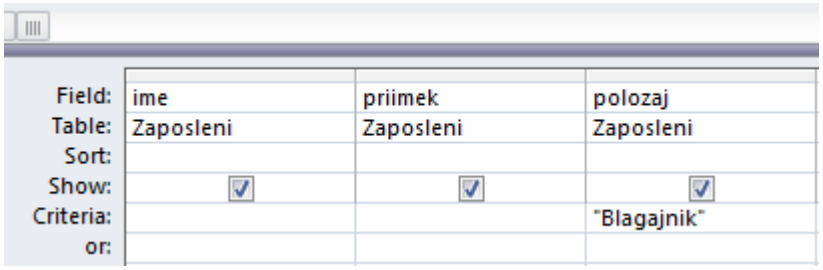
c. Izpiši vse zaposlene.



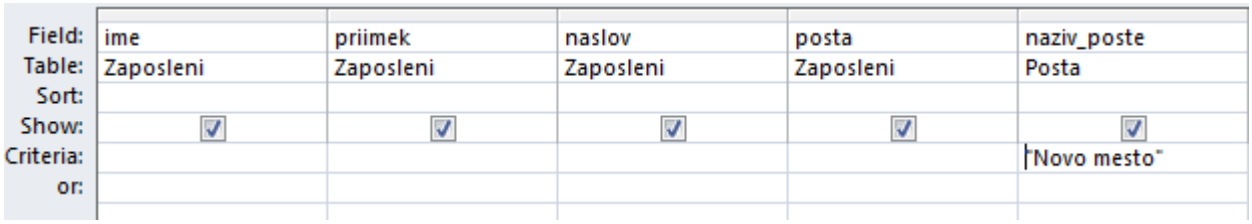
d. Izpiši število vseh zaposlenih v tabeli Zaposleni.



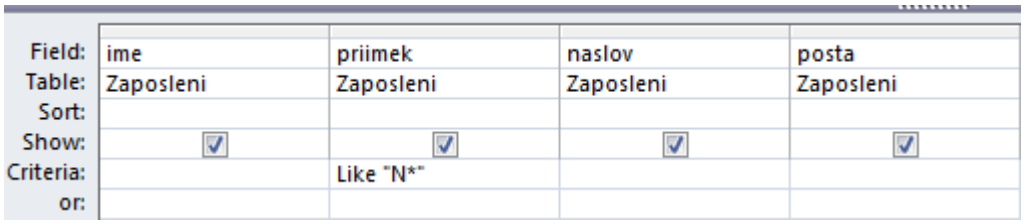
e. Izpiši vse blagajnike (ime, priimek, polozaj) iz tabele Zaposleni.



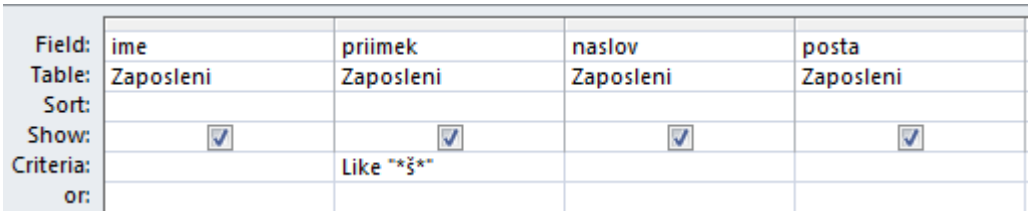
f. Izpiši vse zaposlene (ime, priimek, naslov, posta, Posta.naziv_poste), kjer je naziv poste Novo mesto.



g. Izpiši vse zaposlene (ime, priimek, naslov, posta), katerih priimek se začne na črko N.

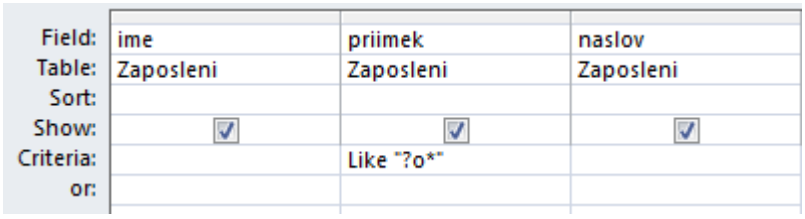


h. Izpiši vse zaposlene, ki imajo v priimku črko š.



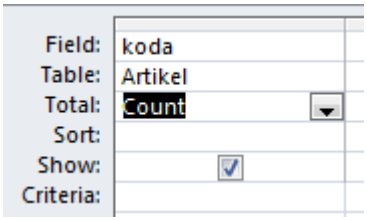
Field:	ime	priimek	naslov	posta
Table:	Zaposleni	Zaposleni	Zaposleni	Zaposleni
Sort:				
Show:	☑	☑	☑	☑
Criteria:		Like "*š*"		
or:				

i. Izpiši vse zaposlene (ime, priimek, naslov), ki imajo v priimku kot drugo črko o.



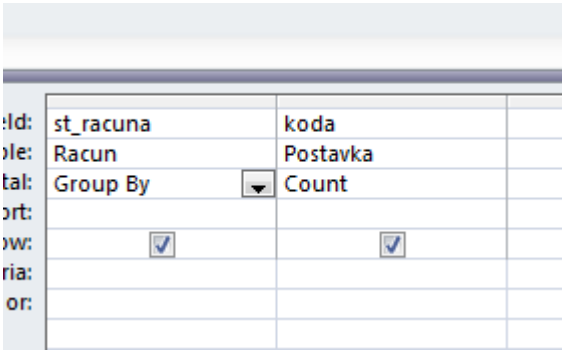
Field:	ime	priimek	naslov
Table:	Zaposleni	Zaposleni	Zaposleni
Sort:			
Show:	☑	☑	☑
Criteria:		Like "?o*"	
or:			

j. Izpiši število vseh artiklov.



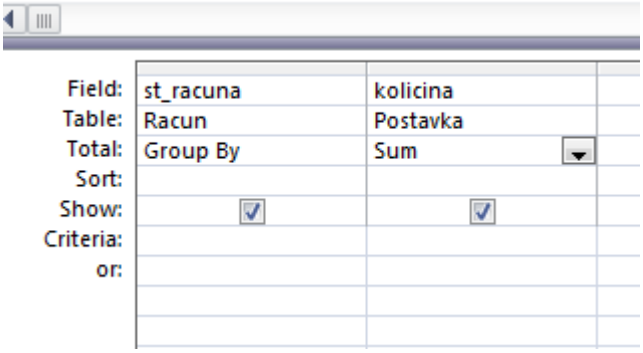
Field:	koda
Table:	Artikel
Total:	Count
Sort:	
Show:	☑
Criteria:	

k. Število različnih artiklov za vsak račun



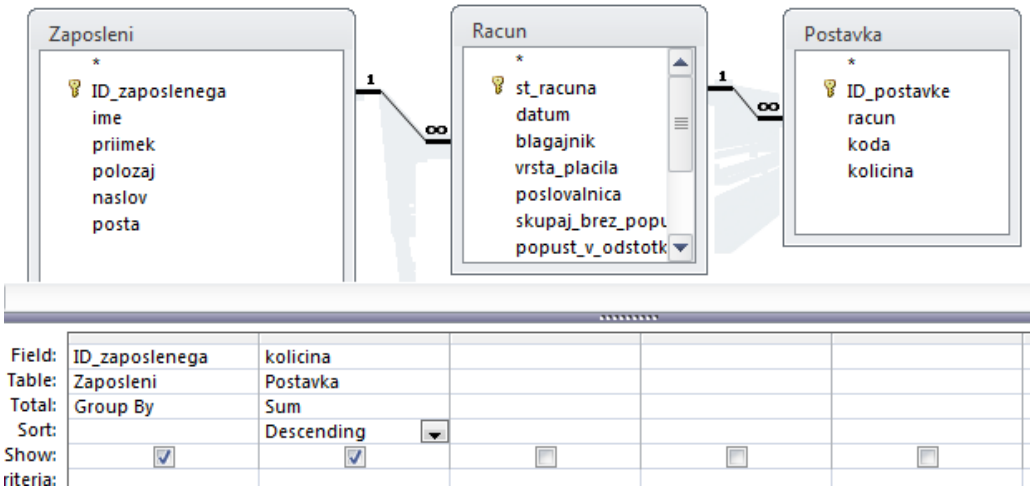
ld:	st_racuna	koda
le:	Racun	Postavka
tal:	Group By	Count
rt:		
w:	☑	☑
ria:		
or:		

l. Skupna količina prodanih artiklov za vsak račun

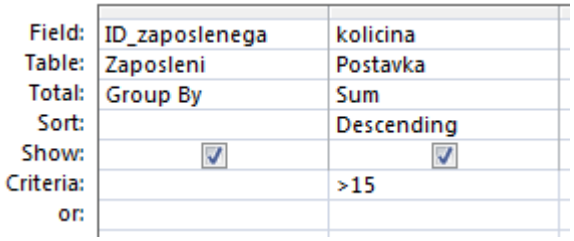


Field:	st_racuna	kolicina
Table:	Racun	Postavka
Total:	Group By	Sum
Sort:		
Show:	☑	☑
Criteria:		
or:		

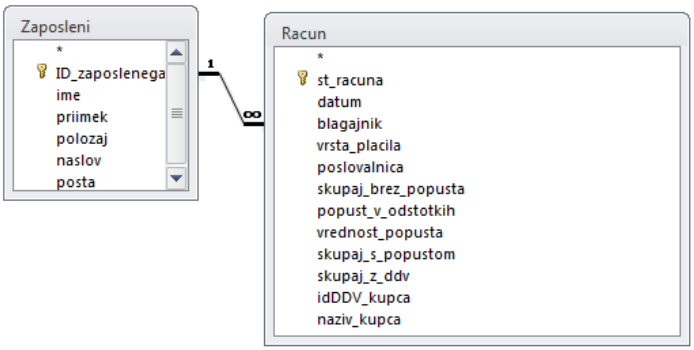
m. Za vsakega blagajnika izpišimo prodano količino izdelkov, urejeno od tistega, ki je prodal največ, navzdol.

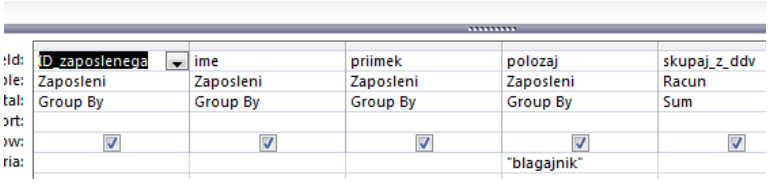


n. Izpiši blagajnika (ime, priimek), ki je prodal več kot 15 izdelkov (količina). Dodamo še Crtiteria: >15 v stolpec količina



o. Za vsakega blagajnika izpiši prihodek od prodaje. Težava:blagajnik, ki še nima računov se ne izpiše. Gremo na join properties (na povezavo med tabelama zaposleni in račun in izberemo opcijo2). V SQL dobimo:
FROM Zaposleni LEFT JOIN Racun ON Zaposleni.ID_zaposlenega = Racun.blagajnik





10 Literatura in viri

1. Cattell, R.G.G. (2000). The Object Database Standard: ODMG Release 3.0. San Mateo, CA: Morgan Kaufmann.
2. Connolly, T. M., Begg, C. E. (2005). Database Systems, A Practical Approach to Design, Implementation and Management, Fourth Edition, Addison-Wesley.
3. Connolly, T. M., Begg, C. E. (2010). Database Systems, A Practical Approach to Design, Implementation and Management, Fifth Edition, Addison-Wesley.
4. Date, C. J. (1989). A Guide to the SQL Standard. Reading: Addison-Wesley.
5. Finkelstein, C. (1992). Information Engineering. Reading: Addison-Wesley.
6. Fowler, M. (1997). UML Distilled. Reading: Addison-Wesley.
7. Grad, J., Jaklič, J.(1996). Podatkovne baze. Ljubljana: Ekonomska Fakulteta.
8. Gradišar, M., Jaklič, J., Turk, T. (2012). Osnove poslovne informatike. Ljubljana: Ekonomska fakulteta.
9. Gradišar, M., Resinovič, G. (1998). Informatika v organizaciji. Kranj: Moderna organizacija.
10. Johnson, L. J. (1997). Database Models, Languages, Design. New York: Oxford University Press.
11. Korth, F. H., Silberschatz, A. (1991). Database System Concepts. New York: McGraw-Hill.
12. Krisper, M. in drugi (2004). Enotna metodologija razvoja informacijskih sistemov. [Zv. 3], Strukturni razvoj. 2. izd. Ljubljana: Vlada Republike Slovenije, Center Vlade RS za informatiko.
13. Lahajnar, S., Rožanec, A. (2000). Načrtovanje večdimenzionalnih podatkovnih baz. Uporabna informatika, let. 8, št. 1, str. 5-13.
14. Mohorič, T. (1992). Podatkovne baze, Ljubljana: FER.
15. Mohorič, T. (1997). Načrtovanje relacijskih podatkovnih baz, Ljubljana: Bi-TIM.
16. Ramakrishnan, R., Gehrke, J. (2003). Database Management Systems. McGraw-Hill.
17. Rob, P., Coronel, C. (2004). Database systems: design, implementation, and management. Thomson.
18. Spletna stran: http://en.wikipedia.org/wiki/Object-relational_database [Citirano 5. 2. 2015 ob 11.51 uri].
19. Spletna stran: http://en.wikipedia.org/wiki/PowerDesigner [Citirano 1. 2. 2015 ob 20.30 uri].

20. Spletna stran: http://sybase-powerdesigner.software.informer.com/16.5/ [Citirano 25. 1. 2015 ob 10.30 uri].
21. Spletna stran: http://www.oracle.com/technetwork/developer-tools/datamodeler/downloads/datamodeler-087275.html [Citirano 27. 1. 2015 ob 8.30 uri].
22. Spletna stran: https://apex.oracle.com [Citirano 25. 1. 2015 ob 17.30 uri].
23. Stonebraker, M. (1996). Object-Relational DBMSs: The Next Great Wave. San Francisco, CA: Morgan Kaufmann Publishers Inc.