DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Matjaž OMAN

IZDELAVA PROGRAMA ZA PRIMERJAVO PODATKOVNIH BAZ

Diplomska naloga

Ljubljana, 2007

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

ZAHVALA

Za pomoč pri izdelavi diplomske naloge se iskreno zahvaljujem mag. Matiji Lokarju.

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

PROGRAM DELA

V diplomski nalogi opišite izdelavo programa za primerjanje dveh podatkovnih baz, ki opisujeta iste podatke.

Mentor:

mag. Matija Lokar

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

POVZETEK

V diplomski nalogi je opisan program, ki primerja podatke dveh različnih podatkovnih baz, ki ju lahko upravljamo z različnima sistemoma za upravljanje podatkovnih baz.

V prvem delu naloge so opisani izrazi, ki jih je potrebno poznati za lažje razumevanje problema. Nato sledi opis načrtovanja programa. V osrednjem delu diplomske naloge so opisani posamezni deli programa in njihova izvedba. V zadnjem delu je navedeno, za katere namene je program že uporaben in kaj bi bilo smiselno še dodati oz. izboljšati.

Math. Subj. Class. (2000): 68N15, 68P05, 68P10, 68P15, 68U15
Computing Review Class. System (1998): D.1.7, D.3.3, D.3.4, H.2.1, H.2.7, H.2.8
Ključne besede: podatkovna baza, sistem za upravljanje podatkovnih baz, primarni ključ, tuji ključ, referenčna integriteta
Key words: database, database management system, primary key, foreign key, referential integrity

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

Kazalo

1 UVOD **6**

1.1 Opis problema 6

2 OSNOVNI POJMI **10**

2.1 Podatkovna baza 10

2.2 Sistem za upravljanje podatkovnih baz (SUPB) 10

2.3 Relacija, atribut, n-terica 12

2.4 Primarni ključ (primary key) 12

2.5 Tuji ključ (foreign key) 13

2.6 Indeks (index) 13

2.7 Referenčna integriteta 13

3 NAČRTOVANJE PROGRAMA **14**

3.1 Kaj mora biti vključeno v program 14

3.2 Zasnova programa 15

4 IMPLEMENTACIJA POSAMEZNIH DELOV PROGRAMA **18**

4.1 Modul za vnos pravil za primerjavo podatkovnih baz 19

4.1.1 Obrazec za vnos skupine in povezovalnih nizov 19

4.1.2 Obrazec za urejanje pravil za primerjavo podatkovnih baz 20

4.1.2.1 Algoritmi in programska koda 23

4.1.2.1.1 Prikaz vseh tabel v podatkovni bazi 23

4.1.2.1.2 Drevesna struktura atributov podatkovne baze . 24

4.1.2.1.3 Prikaz seznama funkcij 26

4.1.2.1.4 Razčlenjevalnik (parser) 28

4.2 Primerjalni modul 37

4.2.1 Obrazec za izbiro skupine 37

4.2.2 Obrazec za primerjavo podatkov 37

4.2.2.1 Algoritmi in programska koda 39

4.2.2.1.1 Poizvedbi, ki vrneta podatke podatkovnih baz . 39

4.2.2.1.2 Primerjanje podatkov podatkovnih baz 45

4.2.2.1.3 Vpis podatkov v primarno podatkovno bazo . . 46

5 ZAKLJUČEK **52**

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

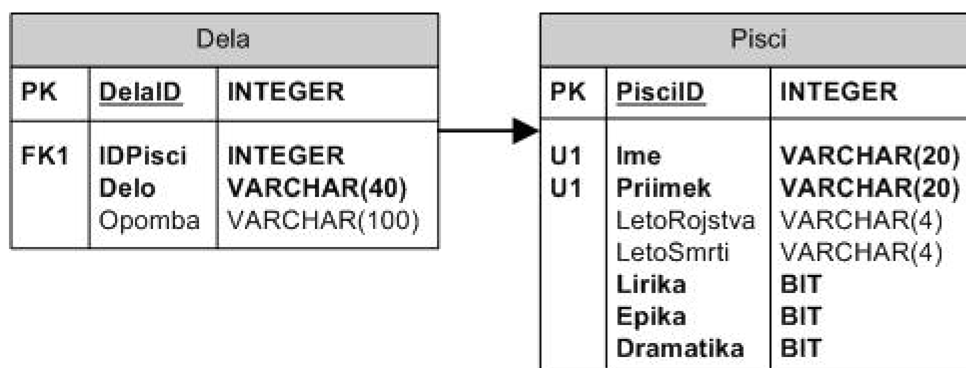
DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

1 UVOD

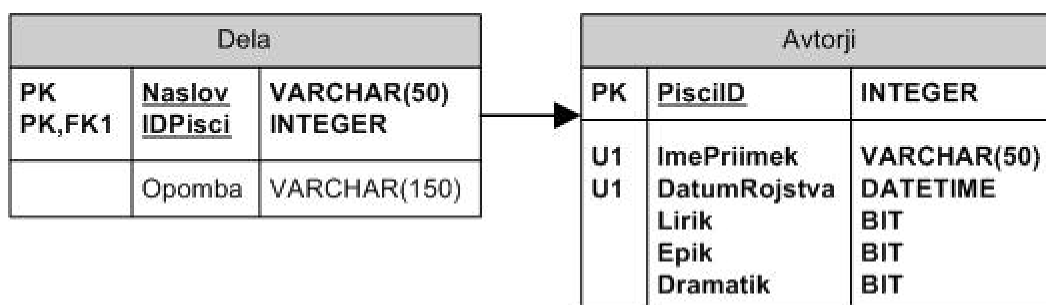
V podjetju, v katerem delam, sem precej časa posvetil delu s podatkovnimi bazami. To področje me zelo zanima, zato sem se odločil, da bom za diplomsko nalogo razvil program, ki mi bo še dodatno razširil znanje in mi bo v pomoč pri delu. Problem, ki sem se ga lotil, je razvoj programa za posodabljanje podatkov iz različnih podatkovnih baz. En način izvedbe takega programa bom predstavil v tej diplomski nalogi.

1.1 Opis problema

Denimo, da imamo na voljo dve podatkovni bazi, ki opisujeta iste podatke. Vendar se načina, kako so ti podatki v posamezni podatkovni baz predstavjeni, nekoliko razlikujeta. Tako so lahko posamezni zapisi sestavljeni nekoliko drugače, ali pa se vrednosti določenih zapisov razlikujejo. Morda določeni zapisi obstajajo le v eni bazi, v drugi jih pa ni. Spet določeni zapisi v eni podatkovni bazi ne vsebujejo vseh podatkov, v drugi podatkovni bazi pa jih in podobno. Zelo poenostavljen primer vidimo na slikah Slika 1 in Slika 2. Sliki prikazujeta strukturo dveh podatkovnih baz s podatki o delih in njihovih avtorjih oz. piscih.



Slika 1: Možna struktura podatkovne baze



Slika 2: Še ena struktura podatkovne baze

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

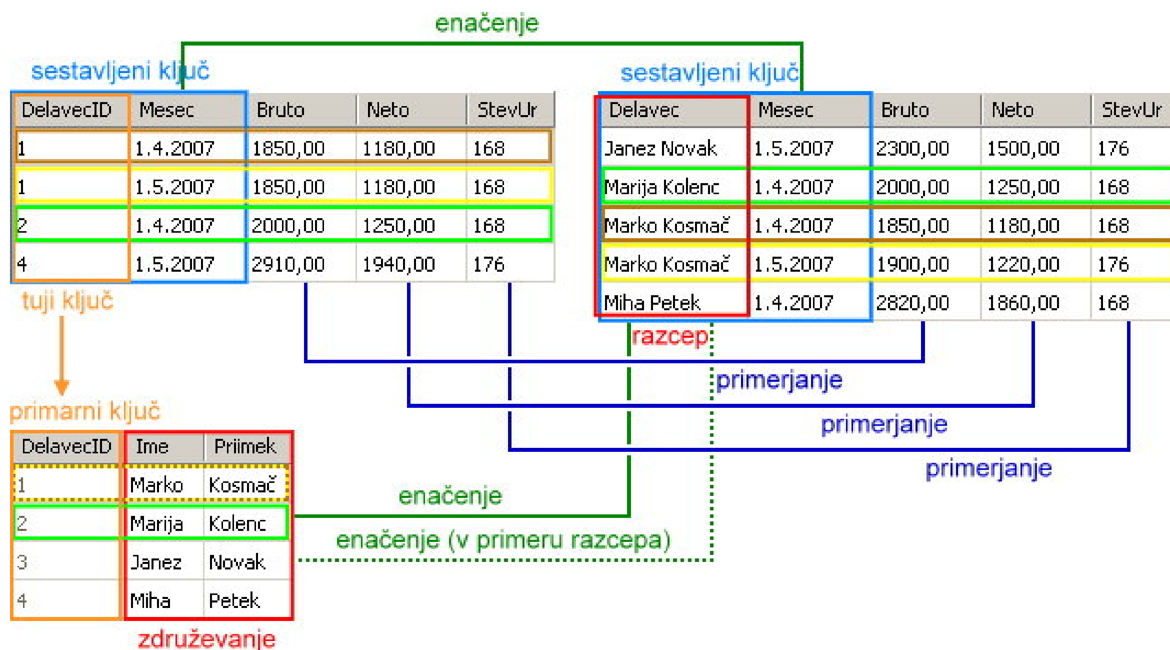
Denimo, da bazi opisujeta iste podatke. Sedaj bi ju radi primerjali in poenotili. Pri tem glavni problem predstavlja različna struktura obeh podatkovnih baz. Tako obe podatkovni bazi sicer vsebujeta iste podatke, vendar so ti v zapisih združeni drugače, posamezni deli zapisov so poimenovani različno in podobno. Tako v prvi podatkovni bazi v tabeli *Dela* enoličnost zapisa določa podatek *DelaID* (polje je označeno z PK, kar pomeni, da je na njem nastavljen primarni ključ), medtem ko v tabeli *Dela* v drugi strukturi enoličnost zapisa določata podatka *Naslov* in *IDPisci*. Prav tako na primeru tabele *Dela* opazimo, da ima lahko v različnih podatkovnih bazah polje, ki predstavlja enak podatek, različen naziv in je različne dolžine. Tako se polje, ki predstavlja naslov dela, v prvi podatkovni bazi imenuje *Delo*, v drugi pa *Naslov*. Ti dve polji se razlikujeta tudi v dolžini. Polje *Delo* namreč dovoljuje vnos 40 znakov, medtem ko v polje *Naslov* lahko shranimo do 50 znakov. Dovoljena dolžina vnosa je različna tudi na polju *Opomba*. Tudi strukturi druge tabele *Pisci* (ki se v eni bazi imenuje *Avtorji*) se razlikujeta. Tako v prvi strukturi enoličnost zapisa določata polji *Ime* in *Priimek*, v drugi strukturi pa enoličnost določata polji *ImePriimek* in *DatumRojstva*. To pomeni, da v prvo strukturo ne moremo vpisati več piscev z enakim imenom in priimkom, med tem ko v drugo strukturo lahko zapišemo več piscev z enakim imenom in priimkom, ki pa morajo imeti različne datume rojstva. Hkrati tudi opazimo, da je podatek o imenu in priimku v drugi strukturi združen v enem polju. Tabeli *Pisci* (oziroma Avtorji) se razlikujeta tudi v načinu hranjenja podatka o rojstvu in smrti pisca. Tako v prvi podatkovni bazi podatka o datumu rojstva sploh ne hranimo, ampak hranimo le podatek o letu rojstva. Podatka o smrti pisca v drugi podatkovni bazi sploh nimamo, v prvi pa imamo sam podatke o letu smrti pisca, ne pa točnega datuma. Ne nazadnje pa se obe strukturi razlikujeta tudi v tem, kateri podatki so obvezni (obvezni podatki so na sliki 1 in sliki 2 zapisani poudarjeno).

Že na enostavnem primeru vidimo, da je lahko predstavitev istih podatkov v dveh podatkovnih bazah zelo različna. Iz primera lahko razberemo, kateri so glavni cilji, ki sem jih želel doseči s programom. To so primerjava podatkov dveh podatkovnih baz z različno (lahko tudi enako) strukturo, uvoz podatkov iz ene podatkovne baze v drugo podatkovno bazo in popravljanje podatkov, ki v obeh podatkovnih bazah niso enaki. Pri tem sem želel upoštevati tudi to, da podatkovni bazi lahko nadzorujeta različna sistema za upravljanje podatkovnih baz.

Kaj vse je bilo potrebno upoštevati pri izdelavi programa, si poglejmo še na primeru z dejanskimi podatki. Primer prikazuje podatke o zaposlenih in njihovih plačah. Leva polovica slike 3 prikazuje podatke ene podatkovne baze, desna polovica pa podatke druge podatkovne baze. V desni podatkovni bazi so vsi podatki v eni tabeli, v levi pa v dveh. Tako do kompletnih podatkov delavca v levi strukturi dostopamo preko povezave (relacije). Zato sem moral v programu omogočiti primerjanje podatkovnih baz, ki vsebujejo relacije med tabelami (relacijske podatkovne baze). V obeh strukturah enoličnost zapisa določata dve polji, Delavec (oz. polje DelavciID) in polje Mesec. Zato sem moral upoštevati, da se lahko podatki enačijo po več poljih. Da je možno enačenje ali primerjanje zapisov, je včasih potrebno podatke ene struk-

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO



Slika 3: Primer dveh podatkovnih baz, ki opisujeta iste podatke (levo je prva, desno pa druga baza)

ture prilagoditi podatkom druge strukture. Tako moramo podatek o imenu in priimku v levi strukturi združiti, da ga lahko primerjamo s podatkom v desni strukturi oz. moramo podatek o imenu in priimku v desni strukturi razcepiti, da ga lahko primerjamo s podatkom v levi strukturi. To pomeni, da sem moral v programu omogočiti izvajanje funkcij na podatkih. Na ta način lahko podatke ene podatkovne baze prilagodimo podatkom druge podatkovne baze z uporabo funkcij, ki so možne na posamezni podatkovni bazi (funkcij, ki jih pozna sistem za upravljanje podatkovnih baz). Tako je možno primerjanje podatkov podatkovnih baz, ki se ne razlikujejo v tolikšni meri, da se jih ne bi dalo prilagoditi s funkcijami. Če zapise na sliki 3 enačimo po poljih, ki so povezani z temno zeleno barvo (če podatek o delavcu združimo, sta to dve polji, če pa ga razcepimo, pa so to tri polja), vidimo, da moramo enačiti zapise, ki so označeni z enako barvo (rjava, rumena, svetlo zelena). Posamezne podatke zapisov pa primerjamo po poljih, ki so povezani z temno modro barvo.

Rešitev navedenih problemov sem razdelil v dva dela. V prvem delu uporabnik vnese pravila, po katerih se podatki primerjajo. S tem uporabnik določi, kdaj bo program določene zapise v podatkovnih bazah imel za enake. V tem delu je bilo potrebno sestaviti obrazec, preko katerega uporabnik vnaša pravila za primerjavo podatkov. S pravili, ki jih uporabnik vnese na tem obrazcu, določi kateri zapisi iz prve podatkovne baze so enaki zapisom v drugi podatkovni bazi. Neposredna primerjava dveh zapisov ni vedno možna. Zato sem uporabniku omogočil, da lahko definira funkcije, ki se izvajajo na atributih, ki sestavljajo zapis. Tako lahko na primer uporabnik vnese pravilo, s katerim dva atributa iz ene podatkovne

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

baze združi in ju primerja z atributom v drugi podatkovni bazi. Seveda mora pred tem definirati funkcijo, ki zna vrednosti dveh atributov združiti v eno vrednost. V tem delu sem moral implementirati tudi preverjanje pravilnosti pravil, ki jih vnese uporabnik. Zato sem napisal razčlenjevalnik ukazov, ki uporabniku v primeru nepravilnega vnosa javi napako. Za lažji vnos pravil sem sprogramiral tudi nekaj komponent. Uporabil sem jih pri gradnji uporabniškega vmesnika, s katerim uporabnikom pomagamo pri izbiranju tabel, atributov in funkcij.

V drugem delu sem sestavil obrazec, ki primerja podatke po pravilih, ki jih je vnesel uporabnik. S pomočjo pravil sem moral sestaviti dve poizvedbi, ki se izvedeta na podatkovnih bazah. S pomočjo podatkov, dobljenih iz teh dveh poizvedb, podatke primerjamo in različne oz. manjkajoče na ustrezen način prikažemo uporabniku. V tem delu sem uporabniku omogočil, da požene operacijo, ki podatke v prvi podatkovni bazi ustrezno posodobi. To pomeni, da operacija podatke, ki jih ni v prvi podatkovni bazi, v drugi pa so, zapiše v prvo podatkovno bazo. Tistim podatkom, ki so v obeh podatkovnih bazah in se razlikujejo le v nekaterih vrednostih, v prvi podatkovni bazi popravi vrednosti na vrednosti, ki so zapisane v ustreznih atributih zapisov druge podatkovne baze.

Za razvoj programa sem uporabil orodje Microsoft Visual Studio 2005, program pa sem napisal v programskem jeziku C♯. Od bralca te diplomske naloge se pričakuje osnovno poznavanje jezika SQL in osnovno poznavanje programskega jezika C♯. Zato osnov teh dveh jezikov nisem razlagal, čeprav jih v sklopu študija nismo podrobneje obravnavali, oziroma jih sploh nismo.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA FAKULTETA ZA MATEMATIKO IN FIZIKO

2 OSNOVNI POJMI

V tem razdelku so predstavljeni pojmi, ki so uporabljeni v nadaljevanju diplomske naloge.

2.1 Podatkovna baza

Podatkovna baza je množica povezanih podatkov. Sestavlja jo ena sama ali pa skupina datotek, ki so shranjene na trdem disku ali podobnem podatkovnem nosilcu. Na zahtevo uporabnika se podatki iz podatkovne baze prek sistema za upravljanje podatkovnih baz prenesejo na lokalni računalnik. Tam jih uporabnik prek ustreznega programa pregleduje. V podatkovno bazo lahko podatke vnašamo, jih shranjujemo, brišemo in beremo. Vsi podatki so sočasno dosegljivi različnim uporabnikom oz. programom. Podatkovne baze lahko med seboj ločimo glede na vrsto podatkov, ki jih hranijo (tekstovne, numerične, slikovne, zvočne . . .), glede na način dostopa (lokalne, mrežne), glede na strukturo (sekvenčne, hierarhične, relacijske), glede na fizične nosilce, kjer so podatki shranjeni (trdi disk, magnetni trak . . .) in podobno.

2.2 Sistem za upravljanje podatkovnih baz (SUPB)

Sistem za upravljanje podatkovnih baz (v nadaljevanju bomo uporabljali kar kratico SUPB) je skupek programske opreme, ki omogoča kreiranje, vzdrževanje in nadzor nad dostopom do podatkov v podatkovni bazi.

Sistem običajno delimo na več delov, ki se glede na vrsto SUPB lahko nekoliko razlikujejo. Ti deli so:

Del, ki omogoča kreiranje in vzdrževanje podatkovne baze

Glavni namen tega dela SUPB je, da omogoča kreiranje podatkovnih baz in izvajanje poizvedb na njih. Podatkovne baze običajno upravljamo s povpraševalnim jezikom SQL. Ukaze tega jezika delimo na tri področja.

- SQL DDL (Data Definition Language for SQL)
  V to skupino spadajo ukazi, ki omogočajo kreiranje podatkovnih baz in tabel in spreminjanje ter brisanje le teh. Najbolj značilni ukazi v tej skupini so *CREATE DATABASE, CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX* in *DROP INDEX*.

- SQL DML (Data Manipulation Language for SQL)
  V to skupino spadajo ukazi za vzdrževanje podatkov. Za to skupino so značilni ukazi *SELECT, INSERT, UPDATE in DELETE*. Pri delu z bazami podatkov ukaze iz te skupine uporabljamo najbolj pogosto.

- SQL DCL (Data Control Language for SQL)

DIPLOMSKA NALOGA FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA FAKULTETA ZA MATEMATIKO IN FIZIKO

V to skupino ukazov uvrščamo ukaze, ki skrbijo za nadzor transakcij. Transakcija je skupek ukazov, ki jih mora SUPB izvršiti v celoti ali pa jih sploh ne sme izvršiti. Na ta način SUPB skrbi za konsistentnost podatkov v podatkovni bazi (torej, da so shranjeni podatki v skladu z določenimi nastavitvami in omejitvami). Najbolj značilna ukaza te skupine sta *COMMIT* in *ROLLBACK*.

Del, ki skrbi za nadzor dostopa do podatkov

Glavna naloga tega dela SUPB je skrb za varnost in dostopnost podatkov. Delimo ga na več delov:

- Sistem varnosti
  Za ta del je najbolj pomembno, da v skladu z avtorizacijo omogoča nastavljanje pravic za dostop do podatkov. To pomeni, da omogoča nastavljanje pravic za posamezne uporabnike ali pa skupino uporabnikov. Tako lahko preprečimo nedovoljene posege v podatkovno bazo in s tem zagotovimo večjo varnost podatkov.

- Sistem nadzora integritete (veljavnosti)
  Za ta del je najbolj pomembno zagotavljanje konsistentnosti podatkov. Pred izvedbo kakršnekoli spremembe podatkov SUPB (pred dodajanjem, brisanjem ali spreminjanjem) preveri vrednosti novih podatkov glede na domeno (veljavno zalogo vrednosti) podatkov in glede na omejitve, ki so opredeljene glede na povezave z drugimi podatki. Pri ohranitvi integritete podatkovne baze si SUPB pomaga tudi s transakcijami.

- Sistem nadzora sočasnega dostopa
  Za ta del je najbolj pomembno, da v primeru sočasnega dostopa do podatkovne baze zagotavlja pravilnost transakcij. SUPB torej zagotavlja konsistentnost podatkov v primeru, ko več uporabnikov ali aplikacij hkrati dostopa do istih podatkov.

- Sistem obnove podatkovne baze (recovery)
  Pri delu s podatkovnimi bazami prihaja tako do tehničnih kot človeških napak. Zato je pomembno, da SUPB v vsakem trenutku omogoča vrnitev celotne podatkovne baze v zadnje konsistentno stanje.

- Sistemski katalog (data dictionary)
  Ta del SUPB zagotavlja dostop in upravljanje s sistemskim katalogom. Sistemski katalog hrani podatke o tabelah, atributih, primarnih ključih, tujih ključih, indeksih ...

Del, ki komunicira s pomnilnikom in z zunanjimi pomnilniškimi enotami

Na najnižjem nivoju SUPB skrbi za upravljanje s pomnilnikom in z zunanjimi pomnilniškimi enotami. Na tem nivoju se vse operacije iz višjih plasti pretvorijo v nizko-nivojske ukaze za delo tako z notranjim, kot tudi z zunanjim pomnilnikom.

DIPLOMSKA NALOGA FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

2.3 Relacija, atribut, n-terica

Relacija je dvodimenzionalna tabela s stolpci in vrsticami (relacija je matematični pojem, tabela pa fizični pojem).

Atribut predstavlja ime stolpca relacije. Stopnja relacije je število vseh atributov v relaciji.

N-terica je vrstica relacije, ki predstavlja posamezen zapis. Števnost relacije je število vseh n-teric v relaciji, torej število zapisov v posamezni tabeli.

Na sliki 4 so atributi Ime, Priimek in Naslov. Stopnja relacije je 3, ker tabela vsebuje tri atribute (stolpce). Števnost relacije je prav tako 3, ker tabela vsebuje tri n-terice (vrstice).

Slika 4: Relacija, atribut, n-terica

2.4 Primarni ključ (primary key)

Primarni ključ je en ali več (takrat govorimo o sestavljenem ključu) atributov v tabeli (relaciji), katerih vrednosti enolično določajo vsako n-terico v tabeli. S primarnim ključem dosežemo,

Slika 5: Primarni ključ, tuji ključ, sestavljeni ključ

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

da se zapisi (n-terice) med sabo razlikujejo vsaj v vrednosti primarnega ključa. Primarni ključi so pomembni tudi pri povezovanju s tujimi ključi v drugih tabelah. Na sliki 5 je primarni ključ prve tabele nastavljen na atributu *Sifra* v tabeli *ARTIKLI*, pri drugi tabeli pa je primarni ključ sestavljen in sicer iz atributov *Racun* in *SifraArtikla*.

2.5 Tuji ključ (foreign key)

Tuji ključ je en ali več (takrat govorimo o sestavljenem ključu) atributov v tabeli (relaciji), ki omogoča povezovanje zapisov te tabele z zapisi druge tabele s pomočjo primarnega ključa te druge (izhodiščne) tabele. Na Sliki 5 je v tabeli *RAČUNI* tuji ključ atribut *SifraArtikla* in kaže na primarni ključ, ki ga predstavlja atribut *Sifra* v tabeli *ARTIKLI*.

2.6 Indeks (index)

Indeks je objekt (datoteka), ki omogoča hitrejše iskanje zapisov v tabeli. Večina SUPB sama nastavi indekse na tiste atribute, ki jih določimo kot ključe. Indeks je koristno postaviti tudi na tiste atribute, za katere vemo, da bomo po njih pogosto iskali zapise. Vedeti pa moramo, da indeksi operacije posodabljanja in brisanja podatkov upočasnijo, zato jih uporabljamo samo tam, kjer jih resnično potrebujemo.

2.7 Referenčna integriteta

Referenčna integriteta je mehanizem, ki omogoča, da SUPB povemo, kako naj postopa ob brisanju, dodajanju in popravljanju zapisov na tabelah, ki so s pomočjo relacij povezane z drugimi tabelami. Z njim povemo SUPB, kaj naj naredi ob brisanju zapisov, ki vsebujejo primarni ključ oz. kaj naj naredi ob spreminjanju primarnih ključev. Nastavimo lahko, da sistem ne dovoli brisanja tistih zapisov, ki so v relaciji z drugimi zapisi (so povezani preko ključev). Lahko pa predpišemo, da sistem ob brisanju podatka briše tudi podatke v vseh odvisnih tabelah ali pa jim tuje ključe nastavi na prevzeto vrednost. Različne možnosti nastavitve referenčne integritete si poglejmo s pomočjo slike 6. Primer na sliki 6 prikazuje dve tabeli. V prvi hranimo podatke o osebah, v drugi pa podatke o poštnih številkah. V prvi tabeli je stolpec (atribut) *PosteID*, na katerem je postavljen tuji ključ, označen z modro barvo. Pripadajoči primarni ključ je v drugi tabeli nastavljen na stolpcu *PosteID* in je prav tako obarvan modro. Z barvami je na sliki prikazano, kateri zapis iz tabele poštnih številk je preko tujega in primarnega ključa povezan s tabelo oseb. Poglejmo si, kaj se zgodi ob brisanju zapisa v tabeli poštnih številk glede na različne nastavitve referenčne integritete.

- Če SUPB ne dovolimo brisanja podatkov takrat, ko so podatki uporabljeni v odvisnih tabelah, obarvanih štirih zapisov v tabeli poštnih številk ne bomo mogli brisati.
- Če SUPB dovolimo, da podatke briše, ne dovolimo mu pa, da podatke briše tudi v

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

odvisnih tabelah, bomo obarvanavane zapise v tabeli poštnih številk lahko zbrisali. V tabeli oseb pa bo SUPB enako obarvanim zapisom vrednost atributa *PosteID* nastavil na privzeto vrednost (v našem primeru bo to null).

- Če nastavimo, da SUPB ob brisanju podatkov briše tudi podatke v odvisnih tabelah, bo SUPB dovolil brisanje zapisov v tabeli poštnih številk, hkrati pa bo brisal tudi pripadajoče zapise v tabeli oseb. Tako bo v primeru, da bomo v tabeli poštnih številk brisali zapis s poštno številko Ajdovščine, SUPB v tabeli oseb izbrisal pripadajoči zapis (Janeza Novaka).

OSEBE

Ime	Priimek	Naslov	PosteID
Franci	Oblak	Triglavska 14	425
Janez	Novak	Gasilska 10	319
Južica	Dolenc	Slovenska 45	75
Igor	Kranjc	Koroška 23a	387

POŠTNE ŠTEVILKE

PosteID	Sifra	Kraj
425	8341	Adlešiči
319	5270	Ajdovščina
368	6281	Ankaran/Ancarano
463	9253	Apače
387	8253	Artiče
287	4275	Begunje na Gorenjskem
75	1382	Begunje pri Cerknici

Slika 6: Referenčna integriteta

3 NAČRTOVANJE PROGRAMA

V tem poglavju bom opisal, kako sem načrtoval program in na kakšen način sem prišel do primera, ki je bil osnova za implementacijo programa.

3.1 Kaj mora biti vključeno v program

Pri načrtovanju programa sem si postavil nekaj osnovnih ciljev, ki sem jih želel doseči s svojim programom. Te cilje lahko povzamem v nekaj točkah:

- Program mora omogočati primerjavo podatkov dveh podatkovnih baz, ki ju upravljamo z enakima ali različnima SUPB.

- Program mora omogočati primerjavo podatkov dveh podatkovnih baz z enako ali različno strukturo.

- Program mora omogočati popravljanje in dodajanje podatkov iz ene podatkovne baze v drugo podatkovno bazo.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

- Določanje pravil, po katerih se bodo primerjali podatki, mora biti prepuščeno uporabniku.

- Pri vnašanju pravil, po katerih se podatki primerjajo, mora program uporabniku pomagati.

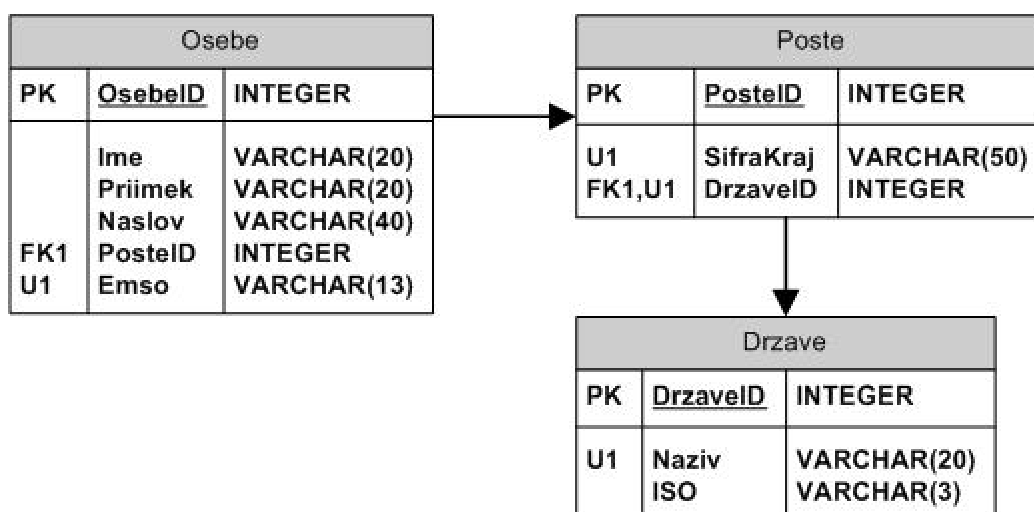
- Vključitev podpore novemu SUPB mora biti enostavno.

- Pridobivanje podatkov in obdelava le teh mora biti prepuščena SUPB.

3.2 Zasnova programa

Tako kot pri vsakem problemu oz. nalogi sem se moral naprej seznaniti s tem, za kakšen problem sploh gre. Potrebno je bilo določiti kakšne so zahteve, kako obširen je problem in kakšno je moje poznavanje problema. Ko sem problem okvirno spoznal, sem se lotil načrtovanja. Pri načrtovanju programa sem se najprej osredotočil na konkreten primer, ko sta bili obe bazi nadzorovani samo z enim tipom sistema za upravljanje podatkovnih baz (SUPB). Ko sem proučil ta primer, sem premislil, koliko se zasnova programa spremeni v primeru drugačnih in morda bolj zapletenih struktur podatkov in koliko v primeru, ko sta SUPB različna. Na koncu sem prišel do primera, ki je bil osnova za implementacijo programa. Nekoliko spremnjeno različico tega primera si oglejmo in jo uporabimo za razlago zasnove programa.

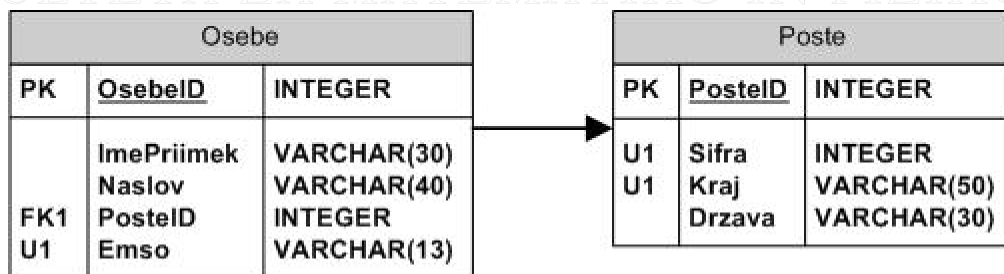
Denimo, da želimo v podatkovni bazi hraniti osnovne podatke o osebi. Predpostavimo, da mora podatkovna baza vsebovati podatke o imenu, priimku, enotni matični številk, naslovu, poštni številki, kraju in državi osebe. Poglejmo si dve podatkovni baz s takimi podatki.



Slika 7: Diagram strukture podatkovne baze s podatki o osebi

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO



Slika 8: Diagram še ene strukture podatkovne baze s podatki o osebi

Iz slike 8 je razvidno, da ta struktura ne vsebuje tabele *Drzave*, kot jo vsebuje struktura podatkovne baze na sliki 7. Podatek (atribut) o državi v tej bazi dobimo v tabeli *Poste*. Do podatka ISO koda države pa v primeru strukture na sliki 8 sploh ne moramo dostopati, ker atribut v strukturi ne obstaja. Iz navedenega je razvidno, da v določenih primerih ni možno primerjati vseh podatkov, ker v neki podatkovni bazi podatek (atribut) obstaja, v drugi pa podatka (atributa) ni.

Iz slik Slika 7 in Slika 8 je tudi razvidno, da nekateri atributi v obeh strukturah niso enakega tipa (npr.: *Sifra* v tabeli *Poste* v strukturi na sliki 7 je v atributu *SifraKraj*, ki je tipa varchar, v tabeli *Poste* v strukturi na sliki 8 pa je *Sifra* tipa int). Razvidno je tudi, da nekateri atributi nimajo enake dolžine (npr.: *Naziv* v tabeli *Drzave* na sliki 7 je dolžine 20, *Drzava* v tabeli *Poste* na sliki 8 pa dolžine 30).

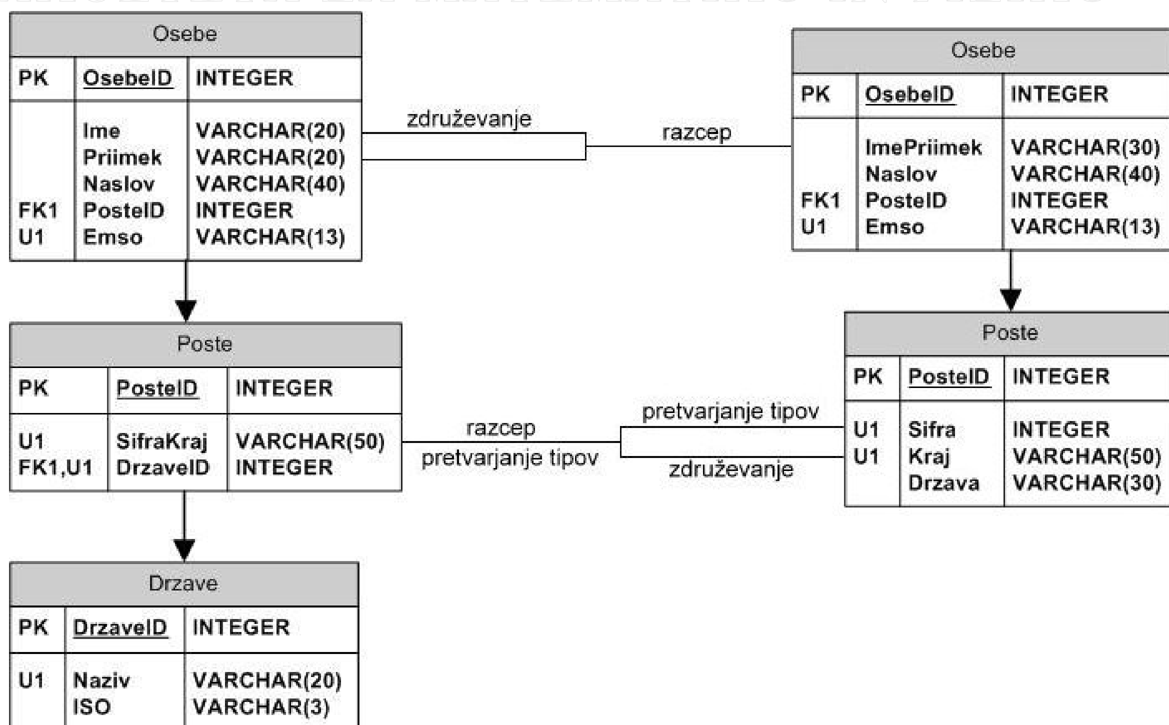
Ko želimo podatke struktur na sliki 7 in sliki 8 primerjati, moramo v določenih primerih podatke združevati ali pa razcepiti. Na sliki 9 vidimo dva primera, ko sta v eni strukturi dva atributa enaka združenemu atributu v drugi strukturi. V tem primeru je potrebno, če gledamo iz ene smeri, podatke združevati, če gledamo iz druge smeri, pa je potrebno podatke razcepiti.

Pri razcepu podatkov naletimo na problem, ki se ga v splošnem ne da rešiti. V določenih primerih namreč ne vemo kako oz. kje podatek razdeliti, da bo delitev pravilna (npr.: kot delilni znak je določen presledek, podatek pa vsebuje več presledkov). Kot bomo videli v nadaljevanju, je, če želimo podatke le primerjati, ta problem enostavno rešljiv. Kadar pa želimo podatke vpisati (insert) ali popraviti (update), pa se moramo odločiti za neko metodo. Ta naj bi bila taka, da podatek razdelila tako, da bo rezultat deljenja v večini primerov pravilen.

Iz slike 9 je razvidno, da so v obeh podatkovnih bazah vzpostavljene relacije med tabelami. Zato sem pri načrtovanju in izvedbi programa upošteval tudi to možnost in uporabniku omogočil lažji vnos pravil za primerjanje podatkov. Zato sem razvil komponento, ki uporabniku pomaga tako, da prikazuje tabele in atribute na podlagi ustrezno vzpostavljenih povezav med tabelami. Nekateri sistemi za upravljanje podatkovnih baz ne omogočajo nastavljanja relacij med tabelami. Spet drugi podpirajo relacije, pa jih načrtovalci podatkovne baze niso

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO



Slika 9: Primer diagrama podatkovnih baz

nastavili ali pa so relacije implementirali programsko in ne na nivoju podatkovne baze. Zato sem dele program, ki so odvisni od relacij med tabelami, poskušal pustiti čimbolj odprte, da jih je možno prilagajati.

Pomemben del implementacije programa je določitev ključa, ki določa, kateri zapis v prvi podatkovni bazi je enak zapisu v drugi podatkovni bazi. V primeru na sliki 9 bi lahko ključ nastavili na atribut *Emso*. S tem bi določili, da je zapis v drugi bazi enak zapisu v prvi bazi takrat, ko se vrednosti atributov *Emso* ujemata. Poenostavljeno lahko rečemo, da bo zapis v drugi bazi enak zapisu v prvi bazi, če bosta zapisa imela enak *Emso*. Ključ lahko nastavimo tudi tako, da ga sestavlja več podatkov (več atributov). To pomeni, da bomo enačili dva zapisa, ki bosta imela enake podatke glede na pravilo, na katerega nastavimo ključ. Torej, bi v našem primeru lahko postavili ključ tako, da bi enačili zapisa, ki bi se ujemala tako v imenu kot priimki, ne pa samo v imenu ali samo v priimku (glej Slika 10). V tem primeru bi sestavili pravilo, ki bi za desno podatkovno bazo vsebovalo funkcijo, ki bi razbila atribut *ImePriimek* na ime in priimek. Torej lahko ključ postavljamo na pravila, ki vsebujejo funkcije. V primeru, da ključ nastavimo na polja, ki ne določajo enoličnosti zapisov, programu pri primerjanju podatkov ne bo uspelo nastaviti primarnega ključa na ta polja. V tem primeru bo program javil napako, da polja, na katerih je nastavljen ključ, ne določajo zapisov enolično.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

Ime	Priimek	Naslov	...		Ime	Priimek	Naslov	...
Franci	Oblak	Triglavska 15	...	ne enači	Franci	Petek	Slovenska 26	...
Janez	Novak	Gasilska 10	...	enači	Janez	Novak	Tržaška 5	...
Nina	Koren	Slovenska 24	...	ne enači	Maja	Koren	Krekova 12	...

Slika 10: Enačenje zapisov

4 IMPLEMENTACIJA POSAMEZNIH DELOV PROGRAMA

Pred opisom implementacije programa, si oglejmo osnovne korake pri njegovi uporabi.

Ko se uporabnik odloči, podatke katerih dveh podatkovnih baz želi primerjati, mora najprej določiti pravila, po katerih bo primerjal podatke, torej določil, kdaj je en zapis v prvi bazi enak zapisu v drugi bazi. Do operacije, ki mu omogoča vnos pravil, uporabnik dostopa preko menija (*Datoteka**Nova*). Program vpraša po imenu skupine in po povezovalnih nizih za dostop do obeh podatkovnih baz. Ime skupine določa, kakšno ime bo dobila datoteka, v katero se bodo shranila primerjalna pravila. Povezovalna niza določata, kako bo program dostopal do podatkov podatkovnih baz. Ko uporabnik vnese zahtevane podatke in potrdi vnos, se odpre okno, v katerem definiramo primerjalna pravila. S pravili, ki jih označi s ključem, določi, kdaj sta dva zapisa enaka (en iz prve in drugi iz druge podatkovne baze). S pravili, ki jih ne označi s ključem, pa določi, kako se bodo podatki prve podatkovne baze primerjali s podatki druge podatkovne baze in kako se bodo podatki v prvo podatkovno bazo dodali oz. popravili na podlagi podatkov druge podatkovne baze. Po vnosu primerjalnih pravil uporabnik vnesena pravila shrani.

Ko so primerjalna pravila določena, je vse pripravljeno, da lahko podatke podatkovnih baz primerjamo. Do operacije, ki omogoča primerjanje podatkovnih baz, uporabnik dostopa preko menija (*Datoteka**Primerjaj*). Program najprej prikaže seznam vseh definiranih skupin, torej skupkov pravil. Ko uporabnik iz seznama izbere želeno skupino, program odpre okno, v katerem se primerjajo podatki. V oknu program prikazuje rezultat primerjanja podatkov na podlagi vnesenih pravil izbrane skupine. Uporabnik lahko v vsakem trenutku ponovno primerja podatke in program mu bo prikazal samo tiste zapise, ki v tistem trenutku niso enaki v obeh podatkovnih bazah. Možnost primerjanja podatkov v vsakem trenutku je potrebna, saj se primerjava podatkov lahko izvaja na podatkovnih bazah, katerih podatki se ves čas spreminjajo. Prav tako lahko uporabnik v vsakem trenutku zahteva vpis (dodajanje in popravljanje) zapisov iz druge podatkovne baze v prvo. Pred vpisom oz. popravljanjem zapisov program ponovno primerja podatke, da vpiše oz. popravi le tiste zapise, ki se v tistem trenutku razlikujejo. Razlog za to je, da so lahko drugi programi v času od zadnjega primerjanja podatke v podatkovnih bazah spremenili oz. dodali. Tisti zapisi, ki so v drugi podatkovni

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

bazi, v prvi pa jih ni, se vpišejo v prvo podatkovno bazo. Tiste zapise, ki niso enaki v obeh podatkovnih bazah (to so tisti zapisi, pri katerih je bila vrednosti pravil označenih s ključem enaka, razlikujejo pa se vsaj glede na eno od pravil, na katerih ni bil nastavljen ključ), pa program v prvi podatkovni bazi popravi.

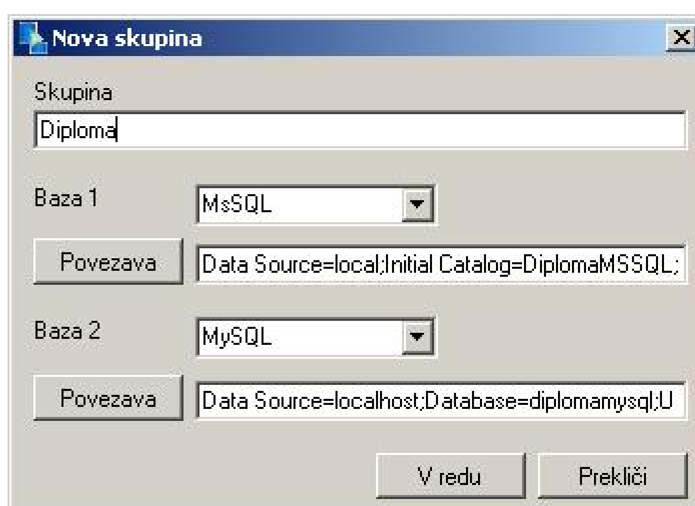
Natančnejši opis posameznih delov programa bom opisal v nadaljevanju tega poglavja. Pri opisu se bom osredotočil na razlago pomena posameznih obrazcev in vnosnih polj na njih. Predstavil bom komponente, ki sem jih sprogramiral, da je program tak, da je njegova uporaba čim enostavnejša. Zaradi lažjega razumevanja bom navedel konkretne primere. Navedel bom le pomembne dele programske kode. Kjer se sklicujem na poizvedbe, ki jih izvaja SUPB, bom zgolj zaradi prikaza različnega pristopa glede na SUPB, navajal poizvedbi za dva konkretna SUPB in sicer za Microsoft SQL in MySQL.

4.1 Modul za vnos pravil za primerjavo podatkovnih baz

Modul je razdeljen na dva glavna dela. Prvi del je obrazec za vnos nove skupine. Skupina določa pod kakšnim imenom se bodo pravila za primerjavo podatkovnih baz shranila na disk in na kakšen način dostopamo do obeh podatkovnih baz. V drugi del sodi obrazec, preko katerega vnašamo pravila za primerjavo podatkov.

4.1.1 Obrazec za vnos skupine in povezovalnih nizov

Na obrazcu (Slika 11) je potrebno vnesti ime skupine, tipa obeh SUPB in povezovalna niza (connection string) za dostop do podatkovnih baz. Z imenom skupine določimo, pod kakšnim imenom bodo primerjalna pravila shranjena in s kakšnim imenom bomo dostopali do njih. S tem ko izberemo tipa SUPB, programu povemo, katere razrede oz. objekte za



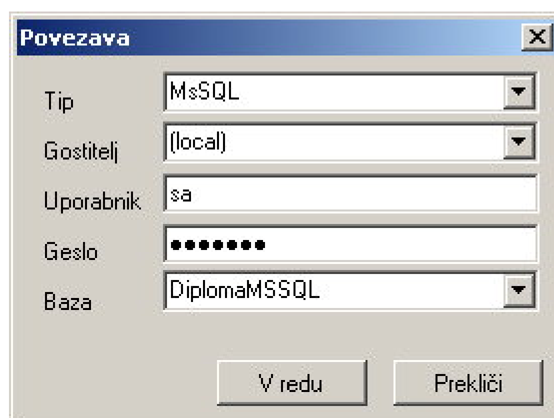
Slika 11: Obrazec za vnos skupine in povezovalnih nizov

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

komunikacijo z podatkovnima bazama bo moral uporabljati v nadaljevanju. S povezovalnima nizoma programu posredujemo ustrezne podatke o tem, kako bomo dostopali do obeh podatkovnih baz.

Povezovalni niz lahko vnesemo kar direktno ali pa si pomagamo z obrazcem, ki ga vidimo na sliki 12. Do njega pridemo preko gumba Povezava.



Slika 12: Obrazec za vnos povezovalnega niza

Na obrazcu, ki ga prikaže klik na gumb Povezava, najprej izberemo tip SUPB oz. tip podatkovne baze, na katero se bomo povezovali. Nato izberemo gostitelja, preko katerega bomo dostopali do podatkovne baze (npr.: localhost, URL strežnika ali pa kar IP naslov računalnika, na katerem je podatkovna baza). Zatem izberemo uporabnika, preko katerega imamo dostop do podatkovne baze in njegovo geslo. Če so vsi podatki pravilno vneseni, na koncu iz seznam podatkovnih baz, ki jih programu vrne strežnik (in z njimi napolni izbirni seznam v polju Baza), izberemo želeno podatkovno bazo. Po potrditvi se s podatkov sestavi povezovalni niz, ki se prepiše v obrazec za vnos skupine (Slika 11) in povozi morebiten že vnesen podatek (če smo na obrazcu povezava izbrali drug SUPB, se povozi tudi ta podatek).

V primeru na sliki 12 se bomo povezali na Microsoftovo podatkovno bazo z imenom *DiplomaMSSQL*, ki se nahaja na lokalnem računalniku. Do podatkovne baze pa bomo dostopali preko uporabniškega računa uporabnika *sa*.

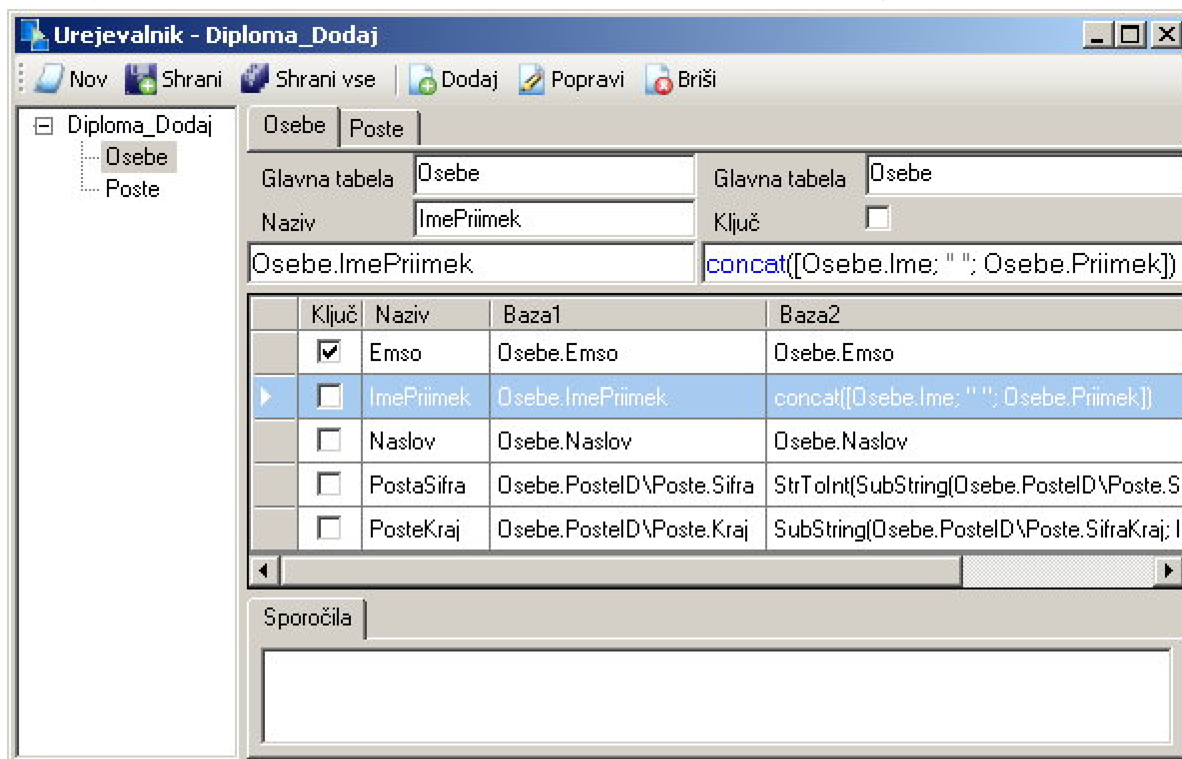
Seveda pa lahko povezovalni niz vnesemo kar neposredno v obrazec, ki je na sliki 11. Na [5] najdemo seznam povezovalnih nizov za najrazličnejše SUPB.

4.1.2 Obrazec za urejanje pravil za primerjavo podatkovnih baz

Obrazec na sliki 13 je poimenovan Urejevalnik. Gre za okno, v katerem nastavimo (urejamo) pravila za primerjavo zapisov v podatkovnih bazah. Obrazec sestavlja orodna vrstica z gumbi, seznam projektov, vnosna polja in polje sporočil.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO



Slika 13: Urejanje pravil za primerjavo podatkovnih baz

Orodna vrstica vsebuje šest gumbov. Ti gumbi so:

- Novo: Če izberemo to izbiro, se odpre novo okno, ki od nas zahteva vnos naziva novega projekta znotraj skupine. Ime skupine in vsi projekti znotraj nje se izpisujejo v levem delu obrazca Urejevalnik.
- Shrani: Če izberemo to izbiro, se v datoteko XML shranijo podatki trenutno izbranega projekta.
- Shrani vse: Če izberemo to izbiro, se v datoteko XML shranijo podatki vseh projektov v skupini.
- Dodaj: Ta gumb je vezan na zgornji del desnega dela programskega okna Urejevalnik. Ob kliku na ta gumb se vnešeno pravilo razčleni. Če razčlenjevalnik ne ugotovi napak in če ne obstaja pravilo z enakim nazivom, pravilo vpiše v tabelo. Če pa pri preverjanju pravila razčlenjevalnik ugotovi napako, v polje Sporočila na dnu obrazca izpiše opis napake.
- Popravi: Ko izberemo vrstico v tabeli, se podatki prepišejo v vnosna polja. Tam jih lahko popravimo in ob kliku na gumb Popravi se bodo popravljeni podatki ponovno razčlenili. Uspešno razčlenjeno pravilo se nato vpiše v izbrano vrstico v tabeli in s tem popravi obstoječega.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

- Briši: Ta izbira izbriše trenutno izbrano vrstico (torej pravilo) iz tabele.

Seznam projektov v skupini, viden na levem delu obrazca, prikazuje vse projekte, ki smo jih vnesli za določeno skupino. V skupino je možno dodajati več projektov. Na ta način lahko smiselno ločimo posamezna primerjanja (npr.: primerjanje samo podatkov o osebah, primerjanje samo podatkov o državi . . .). Tako tudi zagotovimo, da bomo pri vpisu podatkov lahko izbirali vrstni red, po katerem se bodo podatki vpisali v podatkovno bazo. Tako na primer za primer na sliki 9, kjer imamo v strukturi na levi strani tri table, definiramo tri projekte. Prvi projekt bo vpisal podatke o državah, drugi projekt bo vpisal podatke o poštnih številkah in tretji projekt bo vpisal podatke o osebah. Če bomo najprej pognali vpis podatkov na projektu, ki vpiše podatke o osebah in bo program naletel na zapis o osebi, ki se sklicuje na poštno številko, ki je še ni v tabeli poštnih številk, bo program javil, da moramo pred vpisom oseb vpisati podatke o poštnih številkah. To pomeni, da moramo pred vpisom podatkov o osebah pognati vpis na projektu, ki vpiše podatke poštnih številk. Podobno bo program javil napako pri vpisu poštnih številk, če bo naletel na zapis, ki se sklicuje na državo, ki jo še ni v tabeli držav.

Glavni del obrazca so vnosna polja, preko katerih vnašamo primerjalna pravila. Ta polja so:

- Glavni tabeli: Glavni namen teh dveh polj je, da določimo, katera tabela v posamezni podatkovni bazi je osnovna. Preko nje z uporabo relacij dostopamo do podatkov iz ostalih tabel. Da uporabnik lažje izbira tabele, lahko na tem polju s kombinacijo tipk *Ctrl + t* prikliče seznam vseh tabel. Potem iz seznama le še izbere ustrezno tabelo, ki se prepiše v polje s podatkom o glavni tabeli.

- Naziv: Vrednost tega polja predstavlja kratek opis oz. pomen posameznega pravila. Zato se nazivi pravil znotraj enega projekta ne smejo ponavljati.

- Ključ: S ključem označimo tista primerjalna pravila, s katerimi določimo, po katerih podatkih se bodo enačili zapisi. Vsa tista pravila, ki jih označimo kot ključ, pri primerjanju podatkov določajo, kdaj sta dva zapisa enaka. Če s ključem označimo več pravil, bo program enačil le tista zapisa, ki se bosta ujemala v vseh pravilih, ki so označeni s ključem.

- Primerjalni polji: V ti dve polji vnašamo pravila, s katerimi povemo, kdaj sta dva zapisa enaka in pravila, ki povedo, kako se zapis iz ene baze pretvori v zapis v drugi bazi. Prva pravila (ki določajo enakost zapisov) so tista, ki jih označimo kot ključ (v polju *Ključ* je potrditveni znak), druga pa vsa ostala. Kadar je pravilo ključ (torej določa enakost), lahko v obe primerjalni polji napišemo poljubna izraza, v katerih nastopajo atributi in funkcije nad njimi. Če pa pravilo ni ključ in ga torej uporabljamo za popravljanje prve podatkovne baze, moramo iz prve podatkovne baze izbirati samo atribute (primerjalno polje sme vsebovati samo en atribut - ne sme biti izraz). Program

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

dovoli vnos sestavljenih oz. izračunanih izrazov tudi za prvo podatkovno bazo. Tako lahko primerjamo podatke na osnovi sestavljenih izrazov, ne moremo pa na osnovi sestavljenih izrazov potem podatkov uvažati ali spreminjati. Če poskusimo izvesti uvoz podatkov na osnovi sestavljenih izrazov, program to ugotovi in vrne sporočilo, da vpis ni mogoč.

Za lažje izbiranje atributov lahko uporabnik na teh dveh poljih s kombinacijo tipk *Ctrl + d* prikliče komponento, ki predstavlja drevesno strukturo atributov v podatkovni bazi. Po izbiri ustreznega atributa se le ta doda (prepiše) v primerjalno polje. S kombinacijo tipk *Ctrl + f* pa uporabnik pokliče seznam funkcij, ki jih glede na izbrano vrsto SUPB lahko uporabi.

- Tabela: Prikazuje ves seznam primerjalnih pravil trenutno izbranega projekta.

Pomemben del obrazca je tudi polje s sporočili, ki se nahaja na dnu obrazca. V tem polju se prikazujejo vsa sporočila oz. napake, ki jih ugotovi razčlenjevalnik pri vnosu primerjalnih pravil.

V primeru na sliki 13 vidimo primerjalna pravila za projekt z imenom *Osebe*, ki se nahaja v skupini *Diploma_Dodaj*. Poleg projekta *Osebe* je v skupini tudi projekt *Poste*. V projektu *Osebe* je kot glavna tabela za obe podatkovnih bazah izbrana tabela *Osebe*. Za projekt *Osebe* je definiranih pet primerjalnih pravil. Med njimi je pravilo z nazivom *Emso* označeno kot ključ, po katerem se bodo enačili zapisi. S tem pravilom smo povedali, da so enaki tisti zapisi, ki se ujemajo v atrubutu *Emso* v obeh podatkovnih bazah. Seveda bi bilo lahko pravilo, ki bi določalo enakost zapisov, bolj kompleksno (pravilo je lahko sestavljeno iz več atributov in lahko vsebuje funkcije).

4.1.2.1 Algoritmi in programska koda

V tem razdelku si bomo ogledali nekatere algoritme, poizvedbe in programsko kodo, ki sem jo uporabil pri implementaciji posameznih delov obrazca za vnos primerjalnih pravil. V prvih treh razdelkih si bomo pogledali vidne komponente, ki uporabniku pomagajo pri vnosu pravil. V zadnjem razdelku je opisan razčlenjevalnik, ki preverja pravilnost primerjalnih pravil.

4.1.2.1.1 Prikaz vseh tabel v podatkovni bazi

Da dobimo seznam vseh tabel v podatkovni bazi, moramo klicati poizvedbo, ki nam vrne seznam vseh tabel. Različni SUPB imajo dostop do seznama tabel rešen na različne načine. Tako na primer sistem Microsoft SQL hrani podatke o strukturi podatkovnih baz v sistemskih tabelah (sys oz. information_schema). Sistem MySQL uporablja sistemske tabele šele v novejših verzijah. V starejših verzijah lahko večino tovrstnih podatkov dobimo s pomočjo ukaza *Show*.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

Navedimo primere poizvedb, ki vrnejo seznam vseh tabel v podatkovni bazi.

- Microsoft SQL

  ```
  SELECT table_name FROM information_schema.tables WHERE table_catalog = @DatabaseName
  ```

  ali

  ```
  SELECT name FROM sys.tables
  ```

- MySQL

  ```
  SELECT table_name FROM information_schema.tables WHERE table_schema = @DatabaseName
  ```

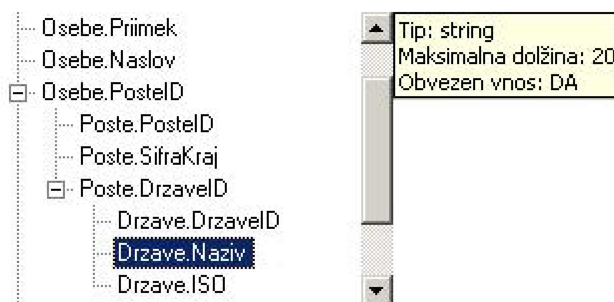
  ali

  ```
  Show tables
  ```

V trenutni verziji programa sem tako za Microsoft SQL kot za MySQL uporabil prvo možnost, ker je univerzalna in enaka za vse tipe bolj znanih podatkovnih baz. Če bi se izkazala potreba po podpori tudi podatkovnim bazam, upravljanimi s starejšim različicami MySQL, pa bi bilo potrebno v program vključiti izbiro novega tipa podatkovne baze, ki bi do seznama tabel dostopal z ukazom *Show*.

4.1.2.1.2 Drevesna struktura atributov podatkovne baze

Za lažje izbiranje atributov podatkovne baze sem sprogramiral komponento, katere uporabo vidimo na sliki 14. Ta na podlagi relacij med tabelami zgradi drevo atributov. Prvi nivo drevesa je sestavljen iz atributov (vozlišč), ki jih vsebuje glavna tabela. To izberemo na vrhu obrazca za vnos primerjalnih pravil. Ob kliku na atribute, na katerih je nastavljen tuji ključ, se dinamično naloži naslednji nivo. Ta je sestavljen iz atributov (vozlišč) tabele, na katero kaže izbrani atribut (tuji ključ).



Slika 14: Drevo atributov podatkovne baze

Dinamično nalaganje nivojev (torej sproti glede na zahtevo uporabnika) je potrebno zaradi dveh razlogov. Pri podatkovnih bazah z velikim številom tabel in velikim številom atributov

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

bi bilo nalaganje celotne strukture naenkrat prepočasno. Še večji problem pa se pojavi pri podatkovnih bazah, kjer so relacije med tabelami vzpostavljene tako, da vsebuje cikle. V tem primeru je nalaganje celotne strukture naenkrat dejansko nemogoče. Slabost dinamičnega nalaganja nivojev pa je v tem, da je ob odpiranju naslednjega nivoja potrebna komunikacija s strežnikom, na katerem je podatkovna baza.

Komponenta ob premikanju po atributih (vozliščih) v drevesni strukturi sproti prikazuje podrobne podatke o atributu, na katerem se nahajamo (tip, dolžina, največja vrednost, največja natančnost, ali gre za obvezen vnos, prevzeta vrednost).

V primeru na sliki 14 je izbran atribut Naziv v tabeli Drzave. Vidimo, da je atribut tipa string (niz), njegova maksimalna dolžina je 20 znakov in da je vnos podatka v to polje obvezen. Ker je atribut tipa string, podatka o največji vrednost in največja natančnost nista smiselna. Zato ta dva podatka nista prikazana. Prav tako ni podatka o prevzeti vrednosti, saj na tem atributu prevzeta vrednost ni nastavljena. Če bi izbrali ta atribut, bi program sestavil niz *Osebe.PosteID\Poste.DrzaveID\Drzave.Naziv*. Ta niz bi vpisal v polje za vnos primerjalnih pravil.

Navedimo ustrezne poizvedbe, ki jih moramo izvesti ob izvajanju posameznih funkcij tega dela programa.

Poizvedba, ki vrne tujemu ključu pripadajoči primarni ključ

To poizvedbo moramo klicati v primeru, ko izberemo atribut, ki ima nastavljen tuji ključ. Takrat se morajo naložiti atributi tabele, na katero se sklicuje tuji ključ. Poizvedba za podan atribut, ki je vpisan v parametru *@FKColumnName* in za dano ime tabele (*@FKTableName*) v kateri se nahaja atribut, vrne ime tabele *PKTable* in atribut (v *PKColumn*) te tabele, na katerega kaže tuji ključ.

- Microsoft SQL

```
SELECT KeyColumn2.COLUMN_NAME AS PKColumn, KeyColumn2.TABLE_NAME AS PKTable
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KeyColumn INNER JOIN
  INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS AS Constraints ON
  KeyColumn.CONSTRAINT_NAME = Constraints.CONSTRAINT_NAME INNER JOIN
  INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KeyColumn2 ON
  Constraints.UNIQUE_CONSTRAINT_NAME = KeyColumn2.CONSTRAINT_NAME
WHERE KeyColumn.COLUMN_NAME = @FKColumnName AND KeyColumn.TABLE_NAME = @FKTableName
```

- MySQL

```
SELECT KeyColumn.REFERENCED_COLUMN_NAME AS PKColumn,
  KeyColumn.REFERENCED_TABLE_NAME AS PKTable
FROM information_schema.key_column_usage AS KeyColumn
WHERE KeyColumn.POSITION_IN_UNIQUE_CONSTRAINT = 1 AND
  KeyColumn.COLUMN_NAME = @FKColumnName AND KeyColumn.TABLE_NAME = @FKTableName
```

Vidimo, da sta ustrezna stavka SQL različna. Zato je očitno, da je potrebno izvesti ustrezno poizvedbo glede na tip SUPB.

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Poizvedba, ki vrne atribute v podani tabeli

Ta poizvedba vrne vse atribute v podani tabeli (@*TableName*). Kličemo jo vsakič, ko dinamično nalagamo naslednji nivo drevesa. To se zgodi takrat, ko prvič izberemo atribut, na katerem je nastavljen tuji ključ.
V tem primeru sta poizvedbi za oba SUPB, tako za Microsoft SQL kot za MySQL, enaki

```
SELECT COLUMN_NAME AS ColName
FROM INFORMATION_SCHEMA.COLUMNS
WHERE (TABLE_NAME = @TableName)
```

Poizvedba, ki vrne podrobne podatke o podanem atributu

Ta poizvedbo kličemo, ko se premikamo po zapisih v drevesu (atributih). Poizvedba vrača tip, dolžino, največjo vrednost, največjo natančnost, obvezen vnos in prevzeto vrednost atributa, na katerem je klicana. @*ColumnName* je ime atributa, @*TableName* pa ime tabele, v kateri se nahaja atribut.

- Microsoft SQL

```
SELECT data_type AS DataType, character_maximum_length AS CharMaxLen,
  numeric_precision AS NumPrec, numeric_precision_radix AS NumPrecRadix,
  is_nullable AS IsNullable, column_default AS DefaultValue
FROM information_schema.columns
WHERE table_name = @TableName AND column_name = @ColumnName
```

- MySQL

```
SELECT data_type AS DataType, character_maximum_length AS CharMaxLen,
  numeric_precision AS NumPrec, numeric_scale AS NumPrecRadix,
  is_nullable AS IsNullable, column_default AS DefaultValue
FROM information_schema.columns
WHERE table_name = @TableName AND column_name = @ColumnName
```

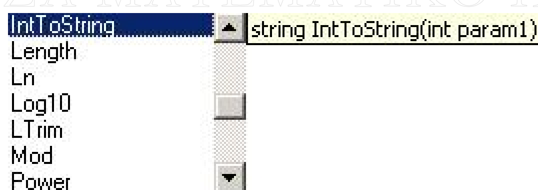
Poizvedbi sta skoraj enaki. Le ime lastnosti s katero zvemo osnovo natančnosti števila, se pri Microsoft SQL imenuje *numeric_precision_radix*, pri MySQL pa *numeric_scale*.

4.1.2.1.3 Prikaz seznama funkcij

Ker moramo v nekaterih primerih na atributu ene podatkovne baze izvesti določeno operacijo, da podatke lahko primerjamo s podatki druge podatkovne baze, sem v program vključil možnost izvajanja funkcij na atributih. Tu sem se omejil samo na funkcije, ki jih posamezen SUPB omogoča (podpira). Definicije funkciji, ki so dovoljene, uporabnik vnese še pred uporabo programa. Definicije so navedene v označevalnem jeziku XML. Za prikaz funkcij, ki so na voljo, sem sprogramiral komponento na sliki 15, ki uporabniku pomaga pri vnašanju funkcij.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO



Slika 15: Seznam funkcij

Definicije funkcij uporabnik vnese (s poljubnim tekstovnim urejevalnikom) v datoteko `function.xml`, ki se nahaja v mapi, v kateri je nameščen program. Struktura te datoteke je vnaprej predpisana. Tako z značko <*name*> določimo ime funkcije. To ime program prikazuje na seznamu v komponenti na sliki 15. Značka <*declare*> predstavlja opis deklaracije funkcije. Iz deklaracije je razvidno, kakšnega tipa je rezultat funkcije in kakšnih tipov so parametri. Parametre moramo še natančno opisati. Določimo jih z značko <*param*>. Značka <*param*> vsebuje dve podznački, ki določata opis (<*desc*>) in tip (<*type*>) parametra. Značka <*param*> se mora ponoviti tolikokrat, kolikor parametrov ima funkcija. V znački <*resultDesc*> podamo kratek opis rezultata funkcije, v znački <*resultType*> pa tip rezultata funkcije. Za tip parametrov in tip rezultata funkcije, ne vnašamo tipov, ki jih poznajo SUPB, ampak tipe, s katerimi operira program. To so tipi string, int, decimal, bool, time, date in datetime. Znački <*resultType*> sledi toliko značk <*funcName*>, za kolikor različnih tipov SUPB oz. podatkovnih baz definiramo funkcijo. Navajanje definicij funkcij za vse različne SUPB je potrebno zato, ker vsi SUPB ne podpirajo enakih funkcij. Prav tako določeni SUPB nekaterih funkcij nimajo. V znački <*funcName*> določimo dve podznački. Z značko <*dataBase*> določimo tip podatkovne baze oz. SUPB. V znački <*definition*> pa podamo klic funkcije v obliki, ki jo pozna SUPB, ki ga navedemo v znački <*dataBase*>. Klic funkcije je lahko sestavljena iz več funkcij, ki jih pozna SUPB. Pri vnosu klica funkcije je pomembno, da so imena parametrov oblike *paramX*, kjer je *X* >= *1*. Le na ta način bo program parametre pravilno zamenjal z dejanskimi vrednostmi.

Poglejmo si dva primera deklaracij funkcij.

```
<function>
  <name>SubString</name>
  <declare>string SubString(string param1; int param2; int param3)</declare>
  <param>
    <desc>niz</desc>
    <type>string</type>
  </param>
  <param>
    <desc>začetek podniza</desc>
    <type>int</type>
  </param>
  <param>
    <desc>dolžina podniza</desc>
    <type>int</type>
  </param>
  <resultDesc>Podniz niza param1</resultDesc>
```

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
  <resultType>string</resultType>
  <funcName>
    <dataBase>MsSQL</dataBase>
    <definition>SUBSTRING(param1, param2, param3)</definition>
  </funcName>
  <funcName>
    <dataBase>MySQL</dataBase>
    <definition>SUBSTRING(param1, param2, param3)</definition>
  </funcName>
</function>
```

```
<function>
  <name>SubString2</name>
  <declare>string SubString2(string param1, string param2)</declare>
  <param>
    <desc>niz</desc>
    <type>string</type>
  </param>
  <param>
    <desc>podniz</desc>
    <type>string</type>
  </param>
  <resultDesc>
   Vrne podniz od začetka niza param1, do tam, kjer se v nizu param1 začne niz param2.
   Če niz param1 ne vsebuje niza param2, vrne prazen niz.
  </resultDesc>
  <resultType>string</resultType>
  <funcName>
    <dataBase>MsSQL</dataBase>
    <definition>SUBSTRING(param1, 0, CHARINDEX(param2, param1))</definition>
  </funcName>
</function>
```

Kot vidimo, prva funkcija vrne podniz parametra *param1*, ki se začne z indeksom *param2* in je dolžine *param3* oz. se konča z indeksom *param2 + param3*. Vidimo, da je definicija funkcije podana za dva tipa SUPB (Microsoft SQL in MySQL). Definiciji smo morali zapisati za oba SUPB, čeprav sta enaki.

Druga funkcija vrne podniz od začetka niza *param1*, do tam, kjer se v nizu *param1* začne niz *param2*. Če pa niz *param1* ne vsebuje niza *param2*, vrne prazen niz. Vidimo, da je funkcija definirana samo za Microsoftov SUPB. Torej v primeru, da kot tip SUPB izberemo MySQL, te funkcije ne bomo mogli izbrati. Vidimo tudi, da je klic funkcije sestavljen iz dveh funkcij (*SUBSTRING* in *CHARINDEX*), ki jih pozna Microsoftov SUPB.

Če pravilno vnesemo vse funkcije, zagotovimo, da bo program pravilno sestavil ustrezne stavke SQL. Na ta način se izognemo programskemu preverjanju ali je vnesena funkcija dovoljena in pravilno definirana. Če funkcije ne deklariramo pravilno, program sestavi napačen stavek SQL in SUPB vrne napako. To napako program prestreže in jo prikaže.

4.1.2.1.4 Razčlenjevalnik (parser)

Pomemben del modula za vnos pravil za primerjavo podatkovnih baz je razčlenjevanje ukazov ob dodajanju oz. urejanju posameznih ukazov. Za razčlenjevanje ukazov skrbi razčlen-

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

jevalnik. Razčlenjevalnik je program oz. del programa, ki zaporedje znakov pretvori v ustrezno podatkovno strukturo, hkrati pa preveri pravilnost podatkov.

Poglejmo si nekaj primerov uporabe razčlenjevalnika.

- Uporabnik vnese pravilen izraz, razčlenjevalnik ga razčleni in vrne rezultat.
- Uporabnik vnese vrednost, ki ni število. Razčlenjevalnik ugotovi, da bi na tem mestu moralo biti število, zato vrne napako.
- Uporabnik vnese število s preveč decimalnimi mesti. Razčlenjevalnik vrne napako.
- Uporabnik vnese izraz, v katerem sešteva števila in nize. Tudi tu razčlenjevalnik lahko ugotovi, da struktura izraza ni ustrezna, zato vrne napako.
- Uporabnik vnese izraz z napačno postavljenimi oklepaji in razčlenjevalnik vrne napako.

Običajno razčlenjevalnik sestavlja več delov. Sam sem ga razdelil na tri dele.

V prvem delu so definirani razredi, ki predstavljano osnovne enote. Osnovne enote imenujemo leksični izrazi. Leksični izrazi sestavljajo strukturo leksičnih izrazov. V našem primeru je struktura leksičnih izrazov sestavljena iz operatorjev (+, -, *, /), oklepajev ((,)), celih števil, decimalnih števil, nizov, funkcij, imen atributov in konca. Vsi razredi, ki predstavljajo leksične izraze, vsebujejo konstruktor in metodo, ki vrne tabelo. Tabela vsebuje sestavljen izraz, iz katerega v nadaljevanju sestavimo stavek *SELECT* in tip rezultata sestavljenega izraza.

Navedimo programsko kodo, ki predstavlja razrede leksičnih izrazov s katerimi operira razčlenjevalnik.

```
// abstrakten razred, ki predstavlja izraz
public abstract class Term
{
    // Metoda, ki vrača tabelo. Prvi element je dejanski izraz, drugi
    // pa opis tipa izraza.
    public abstract object[] Value();
}

// razred, ki predstavlja niz
class TString : Term
{
    string Str;

    public TString(string pStr) { this.Str = pStr; }

    public override object[] Value()
    {
        return new object[] { "'" + Str + "'", "string" };
    }
}
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

```
// razred, ki predstavlja atribut v tabeli podatkovne baze
class TAttribute : Term
{
    string Str;
    DbTable TableDB; // objekt s funkcijami vezanimi na podatkovno bazo

    public TAttribute(string pStr, DbTable pTableDB)
    {
        this.Str = pStr; TableDB = pTableDB;
    }

    public override object[] Value()
    {
        // vrne tip atributa v podatkovni bazi
        string TypeDB = TableDB.GetAttributeType(Str);
        return new object[] { Str, fType };
    }
}

// razred, ki predstavlja funkcijo
class TFunction : Term
{
    string FuncName;

    public TFunction(string pFuncName) { this.FuncName = pFuncName; }
    public override object[] Value()
    {
        // ...
        // Izvorna koda, ki preveri, če je število parametrov funkcije pravo in
        // če so parametri pravih tipov glede na definicijo v datoteki XML.
        // V primeru, da vse ustreza definiciji, vrne ustrezno sestavljen del
        // stavek SQL in tip rezultata, sicer pa izvede prekinitev, ki vrne
        // ustrezno obliko napake.
        // ...
    }
}

// razred, ki predstavlja število
class TNumber : Term
{
    int Number;

    public TNumber(long pNumber) { this.Number = pNumber; }

    public override object[] Value() { return new object[] { Number, "int" }; }
}

// razred, ki predstavlja decimalno število
class TDecimal : Term
{
    double DecNumber;

    public TDecimal(double pDecNumber) { this.DecNumber = pDecNumber; }

    public override object[] Value()
    {
        return new object[] { DecNumber, "decimal" };
    }
}
```

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
// razred, ki predstavlja vsoto
class TPlus : Term
{
    Term NumA;
    Term NumB;

    public TPlus(Term pNumA, Term pNumB)
    {
        this.NumA = pNumA;
        this.NumB = pNumB;
    }

    public override object[] Value()
    {
        object[] ValueA = NumA.Value();
        object[] ValueB = NumB.Value();

        return new object[] {
            ValueA[0].ToString() + " + " + ValueB[0].ToString(), "decimal"
        };
    }
}

// razred, ki predstavlja razliko
class TMinus : Term
{
    // ...
    // podobno kot pri TPlus
    // ...
}

// razred, ki predstavlja produkt
class TProduct : Term
{
    // ...
    // podobno kot pri TPlus
    // ...
}

// razred, ki predstavlja količnik
class TDivide : Term
{
    // ...
    // podobno kot pri TPlus
    // ...
}

// razred, ki predstavlja predznak
class TUnarniMinus : Term
{
    Term Num;

    public TUnarniMinus(Term pNum) { this.Num = pNum; }
    public override object[] Value()
    {
        object[] ValueA = Num.Value();
        return new object[] { "-" + NumA[0].ToString(), "decimal" };
    }
}
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

Drugi del razčlenjevalnika vsebuje razred, ki celoten izraz razdeli na leksične izraze. Tako na primer izraz *2 * (5,3 + 3)* razdeli na dele: celo število, *, *(*, decimalno število, +, celo število in *)*.

Poglejmo si programsko kodo razreda, ki celoten izraz razdeli na leksične izraze.

```
public class TLexer
{
    private string Str; // niz, ki ga delimo na leksične izraze
    private int Poz; // indeks, na katerem se nahajamo

    private string oString;
    private double oDecNumber;
    private long oNumber;
    private string oFunction;
    private string oTableAttribute;

    // konstante, ki označujejo enote
    public const int cPLUS = 0;         // plus
    public const int cMINUS = 1;        // minus
    public const int cPRODUCT = 2;      // krat
    public const int cDIVIDE = 3;       // deljeno
    public const int cBRACKET_B = 4;    // oklepaj
    public const int cBRACKET_E = 5;    // zaklepaj
    public const int cNUMBER = 6;       // število
    public const int cDECIMAL = 7;      // decimalno število
    public const int cSTRING = 8;       // niz
    public const int cFUNCTION = 9;     // funkcija
    public const int cATTRIBUTE = 10;   // atribut
    public const int cEND = 11;         // konec

    public TLexer(string pStr)
    {
        Str = pStr;
        Poz = 0; // smo na začetku izraza
    }

    // vrne kodo naslednjega leksičnega izraza
    public int LexTerm()
    {
        // preskočimo presledke
        while (Poz < Str.Length && Str[Poz] == ' ') { Poz++; }

        // preverimo, če smo na koncu izraza
        if (Poz == Str.Length) { return cEND; }

        char Symbol = Str[Poz];

        if (Symbol == '+') { return cPLUS; }
        else if (Symbol == '-') { return cMINUS; }
        else if (Symbol == '*') { return cPRODUCT; }
        else if (Symbol == '/') { return cDELJENO; }
        else if (Symbol == '(') { return cBRACKET_B; }
        else if (Symbol == ')') { return cBRACKET_E; }
        else if (Symbol == '"') // začetek niza
        {
            // regularni izraz, ki zajema nize
            string fRegEx = "\"[^\"\\\\\\r\\n]*(?:\\\\.[^\"\\\\\\r\\n]*)*\"";
            oString = ProcessRegex(Str.Substring(Poz), fRegEx);
```

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
        if (oString != String.Empty)
        {
            Poz += oString.Length;
            return cSTRING;
        }
        else
            throw new TNapaka("Niz se mora zaključiti z \" ali " +
                      "pa izraz vsebuje preveč znakov \"");
    }
    else if ('0' <= Symbol && Symbol <= '9') // začetek števila
    {
        // regularni izraz, ki zajema cela in decimalna števila
        string fRegEx = "\\b(?:[0-9]*\\,)?[0-9]+\\b";
        string fStr = ProcessRegex(Str.Substring(Poz), fRegEx);
        Poz += fStr.Length;
        if (fStr.IndexOf(',') > -1)
        {
            oDecNumber = Double.Parse(fStr);
            return cDECIMAL;
        }
        else
        {
            oNumber = long.Parse(fStr);
            return cNUMBER;
        }
    }
    else if (('a' <= Symbol && Symbol <= 'z') || ('A' <= Symbol && Symbol <= 'Z')) {
        string fRegEx = GetFunctions(); // funkcija, ki iz datoteke XML prebere vse tam
                                        // definirane funkcije in sestavi regularni izraz
        string fStr = ProcessRegex(Str.Substring(Poz), fRegEx);
        if (fStr != String.Empty)
        {
            Poz += fStr.Length;
            oFunction = fStr;
            return cFUNCTION;
        }
        else
        {
            bool fIsDot = false;
            int fStart = Poz;
            while (Str[Poz].Equal('.') || ('a' <= Str[Poz] && Str[Poz] <= 'z') ||
                Str[Poz].Equal('\\') || ('A' <= Str[Poz] && Str[Poz] <= 'Z'))
            {
                if (Str[Poz].Equal('.')) fIsDot = true;
                Poz++;
            }
            // Če najdemo piko, ki loči tabelo in atribut, predpostavimo, da smo
            // naleteli na del izraza, ki predstavlja atribut. Da bi bili popolnoma
            // prepričani, bi morali klicati poizvedbo, ki bi preverila, če atribut
            // resnično obstaja v podatkovni bazi
            if (isDot)
            {
                oTableAttribute = Str.Substring(fStart, Poz - fStart);
                return cATTRIBUTE;
            }
            throw new TNapaka("Izraz je neveljaven!");
        }
    }
    else new TNapaka("Neveljaven znak!");
}
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
    /// <summary>
    /// Vrne del niza pStrLine, ki ustreza regularnemu izrazu pStrRegex
    /// </summary>
    /// <param name="pStrLine">niz</param>
    /// <param name="pStrRegex">regularni izraz</param>
    private string ProcessRegex(string pStrLine, string pStrRegex)
    {
        Regex fKeywords = new Regex(pStrRegex, RegexOptions.IgnoreCase|RegexOptions.Compiled);
        Match fRegMatch = fKeywords.Match(pStrLine);
        if (fRegMatch != null && pStrLine.IndexOf(fRegMatch.ToString()) == 0)
            return fRegMatch.ToString();
        return String.Empty;
    }
}
```

Tretji del razčlenjevalnika vsebuje razred, ki določa slovnična (logična) pravila. Vsako slovnično pravilo predstavlja ena metoda, ki razčleni to pravilo.

```
public class TParser
{
    // objekt, ki celoten izraz razdeli na leksične izraze
    private TLexer Lex;
    // koda (konstanta) iz razreda TLexer, ki predstavlja tekoči leksični izraz
    private int LexTerm;
    // objekt, ki vsebuje funkcije za komunikacijo s podatkovno bazo
    private DbTable TableDB;

    public TParser(string pStr, DbTable pTableDB)
    {
        this.Lex = new TLexer(pStr);
        this.TableDB = pTableDB;
    }

    // prebere naslednji leksični izraz
    public void Next() { LexTerm = Lex.LexTerm(); }

    public Term Parse()
    {
        Next();
        Term fTerm = Term();
        Next();
        // če smo na koncu izraza in se ne nahajamo na znaku, ki označuje konec izraza
        if (LexTerm != TLexer.cEND)
            throw new Exception("Nepričakovani konec");
        else
            return fTerm;
    }

    public Term Term()
    {
        Term fTermA = Term1();
        while (LexTerm == TLexer.cPLUS || LexTerm == TLexer.cMINUS)
        {
            int fLex = LexTerm;
            Next();
            Term fTermB = Term1();
            if (fLex == TLexer.cPLUS)
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
                // vrne objekt TPlus, ki združi objekt, ki ga vrne izraz
                // pred znakom + in objekt, ki ga vrne izraz za znakom +
                fTermA = new TPlus(fTermA, fTermB);
            else
                // vrne objekt TMinus, ki združi objekt, ki ga vrne izraz
                // pred znakom - in objekt, ki ga vrne izraz za znakom -
                fTermA = new TMinus(fTermA, fTermB);
        }
        return fTermA;
    }

    public Term Term1()
    {
        Term fTermA = Term2();
        while (LexTerm == TLexer.cPRODUCT || LexTerm == TLexer.cDIVIDE)
        {
            int fLex = LexTerm;
            Next();
            Term fTermB = Term2();
            if (fLex == TLexer.cPRODUCT)
                // vrne objekt TMinus, ki združi objekt, ki ga vrne izraz
                // pred znakom * in objekt, ki ga vrne izraz za znakom *
                fTermA = new TProduct(fTermA, fTermB);
            else
                // vrne objekt TMinus, ki združi objekt, ki ga vrne izraz
                // pred znakom / in objekt, ki ga vrne izraz za znakom /
                fTermA = new TDivide(fTermA, fTermB);
        }
        return fTermA;
    }

    public Term Term2()
    {
        if (LexTerm == TLexer.cMINUS)
        {
            Next();
            Term fTerm = Term3();
            // vrne objekt TUnarniMinus, ki izrazu doda predznak -
            return new TUnarniMinus(fTerm);
        }
        else return Term3();
    }

    public Term Term3()
    {
        // Če je trenutni leksični izraz predklepaj, vrne objekt, ki ga
        // dobi kot rezultat klica na izrazu znotraj oklepajev.
        if (LexTerm == TLexer.cBRACKET_B)
        {
            Next();
            Term fTerm = Term();
            if (LexTerm != TLexer.cBRACKET_E)
                throw new Exception("Manjka pričakovani )");
            Next();
            return fTerm;
        }

        // Če je trenutni leksični izraz celo število, vrne objekt tipa TNumber.
        else if (LexTerm == TLexer.cNUMBER)
        {
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
            Term fTerm = new TNumber(Lex.Number);
            Next();
            return fTerm;
        }

        // Če je trenutni leksični izraz decimalno število, vrne objekt tipa TDecimal.
        else if (LexTerm == TLexer.cDECIMAL)
        {
            Term fTerm = new TDecimal(Lex.Decimal);
            Next();
            return fTerm;
        }

        // Če je trenutni leksični izraz niz, vrne objekt tipa TString.
        else if (LexTerm == TLexer.cSTRING)
        {
            Term fTerm = new TString(Lex.String);
            Next();
            return fTerm;
        }

        // Če je trenutni leksični izraz funkcija, vrne objekt tipa TFunction.
        else if (LexTerm == TLexer.cFUNCTION)
        {
            Term fTerm = new TFunction(Lex.Function);
            Next();
            return fTerm;
        }

        // Če je trenutni leksični izraz atribut, vrne objekt tipa TAttribute.
        else if (LexTerm == TLexer.cATTRIBUTE)
        {
            Term fTerm = new TAttribute(Lex.Attribute, TableDB);
            Next();
            return fTerm;
        }

        else
            throw new Exception("Nepredvideni znak");
    }
}
```

Na enostavnem primeru si oglejmo, kako uporabljamo posamezne metode zgoraj navedenih razredov. Recimo, da uporabnik vnese ukaz *2 * (5,3 + 3)*. To pomeni, da konstruktor razreda *TParser* sprejme niz "2 * (5,3 + 3)". Ob klicu metode *Parse()* se kličejo metode *Term*, *Term1*, *Term2* in *Term3*, ki s pomočjo metode *LexTerm()* iz razreda *TTerm* izraz razbijejo na dele. Pri tem ustvarijo ustrezne objekte. Ko se metoda *Parse()* izvrši do konca, vrne objekt tipa *TProduct*. Ta vsebuje spremenljivki tipa *TNumber* in *TPlus*. Druga spremenljivka (tipa *TPlus*) vsebuje spremenljivki tipov *TDecNumber* in *TNumber*. Če rezultat zapišemo kot en izraz, dobimo:

```
new TProduct(new TNumber(2), new TPlus(new TDecNumber(5,3) + new TNumber(3)))
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

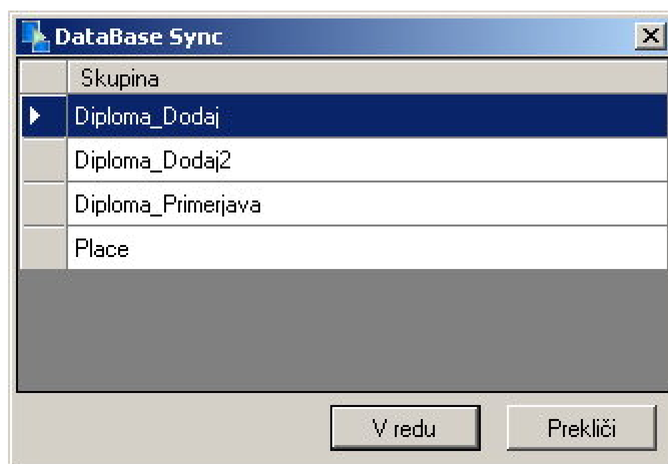
DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

4.2 Primerjalni modul

Primerjalni modul vsebuje dva obrazca. Na prvem obrazcu so prikazane vse skupine, ki smo jih vnesli v modulu za vnos primerjalnih pravil. Na drugem obrazcu se za skupino, ki jo izberemo na prvem obrazcu, prikazuje rezultat primerjanja podatkov.

4.2.1 Obrazec za izbiro skupine

Obrazec na sliki 16 vsebuje seznam skupin, ki so bile definirane v modulu za vnos pravil za primerjavo podatkovnih baz. Ko izberemo skupino in potrdimo izbiro, se odpre obrazec za primerjavo podatkov (Slika 17).



Slika 16: Seznam skupin

4.2.2 Obrazec za primerjavo podatkov

Po izbiri skupine se odpre obrazec (Slika 17). Njegov glavni del je tabela, v kateri je prikazan rezultat primerjave podatkov za izbran projekt. Seznam projektov se nahaja v levem delu programskega okna. V desnem delu so prikazani zapisi, v katerih se podatkovni bazi med seboj razlikujeta. Če se podatki ustreznih zapisov ujemajo (kako, je določeno s pravili, ki so označena kot ključ) v vseh atributih, program zapisa ne prikaže. Če pa program najde razliko v vsaj enem atributu, za tak zapis prikaže vse atribute. Atribute, pri katerih je našel razlike, v tabeli obarva oranžno, atribute, kjer pa ni našel razlik, obarva zeleno. Na sliki 17 vidimo, da je program našel štiri razlike med podatki v obeh podatkovnih bazah. Pri prvih dveh gre za zapisa, ki sta v obeh bazah, a se podatki v niju razlikujejo. Pri tretji gre za zapis, ki ga v prvi bazi ni, v drugi pa je. Pri četrti razliki pa gre za primer, ko zapisa ni v drugi bazi, v prvi pa je. Enakost, oziroma ujemanje zapisov, je določena na osnovi ujemanja atributa *Sifra* (to pravilo je bilo pri vnosu pravil označeno s ključem). Pri prvih dveh razlikah se ne ujemata atributa *Kraj*, medtem ko se atributa *Drzava* ujemata. Ne pozabimo pa, da je zapis

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

sestavljen, torej da je atribut *Drzava* v prvi podatkovni bazi pridobljen iz druge tabele preko ustrezne relacije.

Sync - Diploma_Dodaj

Diploma_Dodaj
Osebe
Poste

Osebe | Poste

Izvedi | Osveži

Naziv	Baza 1	Baza 2
Sifra	2319	2319
Kraj	Poljčana	Poljčane
Drzava	Slovenija	Slovenija
Sifra	4290	4290
Kraj	Tržč	Tržič
Drzava	Slovenija	Slovenija
Sifra		8275
Kraj		Škocjan
Drzava		Slovenija
Sifra	4281	
Kraj	Mojstrana	
Drzava	Slovenija	

Slika 17: Primerjava podatkov

V orodni vrstici programskega okna se nahaja gumb Izvrši, ki podatke zapiše oz. jih popravi v primarni podatkovni bazi. Primarna podatkovna baza je tista, ki je bila kot prva izbrana pri vnosu primerjalnih pravil. V obrazcu na sliki 17 so podatki iz te baze, navedeni v stolpcu *Baza1*. Vpis oz. popravljanje podatkov se uspešno izvrši le v primeru, če so vsa pravila nad primarno podatkovno bazo taka, da so atributi fiksi (niso sestavljeni in se ne izračunavajo). Če bi pognali vpis zapisov v primeru na sliki 17, bi program v primarni bazi prvima dvema zapisoma popravil podatek *Kraj*. V prvem primeru bi namesto vrednosti *Poljčana* vpisal vrednost *Poljčane*, v drugem primeru pa namesto vrednosti *Tržč* vpisal vrednost *Tržič*. Program bi nato v primarno bazo vpisal še nov zapis z vrednostmi *Sifra = 8275*, *Kraj = 'Škocijan'* in *DrzavaID* = vrednost atributa s primarnim ključem, kjer je naziv države enak *'Slovenija'* (ker je atribut *Drzava* v primarni podatkovni bazi dostopen preko ustrezne relacije). Zadnji četrti zapis bi program še vedno prikazal v tabeli (torej kot zapis, kjer se bazi razlikujeta), ker program ne omogoča vpisovanj zapisov v sekundarno podatkovno bazo. Pred vpisom zapisov se podatki še enkrat primerjajo, saj so se med tem podatki v bazah že lahko spremenili. Na ta način zagotovimo, da program po vpisu prikaže samo tiste

DIPLOMSKA NALOGA FAKULTETA ZA MATEMATIKO IN FIZIKO

zapise, ki so v primarni podatkovni bazi, v sekundarni jih pa ni in tiste zapise, pri katerih je prišlo do napake pri vpisu. Do napake pa lahko pride v primeru, ko so bila pravila napačno vnesena (npr.: ključ ni bi nastavljen na pravih pravilih, v pravilih je bila uporabljena funkcija, ki ni bila pravilno definirana ...). V tem primeru SUPB s pomočjo transakcij poskrbi (če je potrebno) za vrnitev zapisa v prejšnje stanje. Tu je potrebno omeniti, da v trenutni različici programa ni mogoče vpisati samo tistih zapisov, ki bi jih izbral uporabnik. Uporabnik torej lahko zahteva le vnos vseh sprememb in ne le določenih.

4.2.2.1 Algoritmi in programska koda

V prvem delu tega razdelka si bomo ogledali, kako se sestavita poizvedbi, ki vrneta podatke iz obeh podatkovnih baz. V drugem delu je predstaviljen algoritem in programska koda, ki v rezultatu obeh poizvedb poišče tiste zapise, ki se v bazah razlikujejo oz. ki ne obstajajo v obeh podatkovnih bazah. V zadnjem delu pa si bomo ogledali, kako se podatki vpišejo v primarno podatkovno bazo, torej kako se sestavita poizvedbi, ki podatke vpišeta oz. jih popravita.

4.2.2.1.1 Poizvedbi, ki vrneta podatke podatkovnih baz

Za primerjav podatkov obeh podatkovnih baz najprej sestavimo dve poizvedbi (za vsako podatkovno bazo ena). Algoritem izgradnje poizvedbe je v obeh primerih enak. Za razlago algoritma vzemimo konkreten primer.

Recimo, da smo pri vnosu pravil za eno izmed baz vnesli naslednje izraze (levo je ime pravila, desno pa vsebina primerjalnega polja za ustrezno bazo):

ImePriimek	Concat([Osebe.Ime; ” ”; Osebe.Priimek])
Naslov	Osebe.Naslov
Posta	Osebe.PosteID\Poste.SifraKraj
Drzava	Osebe.PosteID\Poste.DrzaveID\Drzave.Naziv
Emso	Osebe.Emso
Naslov2	Osebe.Naslov2
Posta2	Osebe.PosteID2\Poste.SifraKraj
Drzava2	Osebe.PosteID2\Poste.DrzaveID\Drzave.Naziv

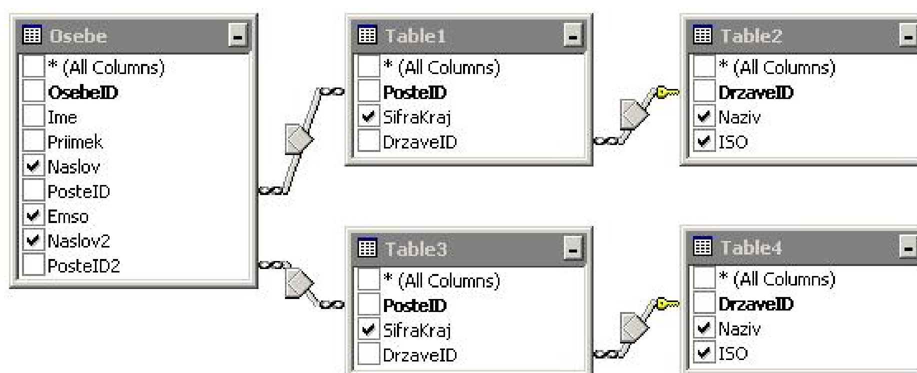
Funkcija *Concat* v prvem pravilu združi atributa Ime in Priimek, med niju pa vstavi presledek. Za podana pravila moramo sestaviti sledečo poizvedbo (grafični prikaz vidimo na sliki 18).

```
SELECT Osebe.Ime + ' ' + Osebe.Priimek AS Osebe_ImePriimek,
  Osebe.Naslov AS Osebe_Naslov, Osebe.Emso AS Osebe_Emso,
  Osebe.Naslov2 AS Osebe_Naslov2, Table1.SifraKraj AS Table1_SifraKraj,
  Table2.Naziv AS Table2_Naziv, Table2.ISO AS Table2_ISO,
  Table3.SifraKraj AS Table3_SifraKraj, Table4.Naziv AS Table4_Naziv,
  Table4.ISO AS Table4_ISO
FROM Osebe LEFT OUTER JOIN
  Poste AS Table1 ON Osebe.PosteID = Table1.PosteID LEFT OUTER JOIN
  Drzave AS Table2 ON Table1.DrzaveID = Table2.DrzaveID LEFT OUTER JOIN
```

DIPLOMSKA NALOGA FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
  Poste AS Table3 ON Osebe.PosteID2 = Table3.PosteID LEFT OUTER JOIN
  Drzave AS Table4 ON Table3.DrzaveID = Table4.DrzaveID
```



Slika 18: Grafični prikaz poizvedbe

Algoritem, ki sestavi poizvedbo

Najprej s pomočjo pravil za primerjavo podatkovnih baz razčlenjevalnik napolni seznam atributov. V podanem primeru ta seznam vsebuje sledeče atribute:

Osebe.Ime
Osebe.Priimek
Osebe.Naslov
Osebe.PosteID\Poste.SifraKraj
Osebe.PosteID\Poste.DrzaveID\Drzave.Naziv
Osebe.Emso
Osebe.Naslov2
Osebe.PosteID2\Poste.SifraKraj
Osebe.PosteID2\Poste.DrzaveID\Drzave.Naziv

Nato iz seznama atributov zgradimo drevo, kjer so vozlišča sestavljena iz podatkov o atributu, seznama kazalcev na sinove atributa in kazalca na očeta atributa.

Navedimo najprej programsko kodo razreda (objekta), ki predstavlja vozlišče drevesa in hrani podatke o atributu.

```
// objekt, ki predstavlja vozlišče drevesa
public class Node
{
    private Node oFather;                // kazalec na očeta
    private Attribute oAttribute;        // objekt s podatki o atributu
    private List<string> oChildrenKeys;  // seznam ključev otrok atributa
    private List<TNode> oChildren;       // seznam otrok atributa
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
    // konstruktor s katerim kreiramo glavno vozlišče
    public Node()
    {
        oAttribute = new Attribute();
        oChildrenKeys = new List<string>();
        oChildren = new List<Node>();
    }

    public Node(Node pFather, Attribute pAttribute)
    {
        oFather = pFather;
        oAttribute = pAttribute;
        oChildrenKeys = new List<string>();
        oChildren = new List<TNode>();
    }

    // ...
    // običajne lastnosti (property)
    // ...
}

// objekt s podatki o atributu
public class Attribute
{
    private string oFieldName;      // ime atributa
    private string oTableName;      // ime tabele v kateri se nahaja atribut
    private bool oIsSelect;         // ali atribut dodamo v stavek SELECT
    private string oTableAlias;     // anonimno ime, ki ga dobi tabela v stavka SELECT
    private string oFieldFullPath; // celotno ime atributa (skupaj z imeni očetov)
    private string oPrimaryField;  // ime atributa na katerem je nastavljen primarni
                                    // ključ - le v primeru, če je atribut z
                                    // imenom oFieldName tuji ključ

    public Attribute(string pTableName, string pFieldName, string pFieldFullPath,
        bool pIsSelect, string pPrimaryField)
    {
        oTableName = pTableName;
        oFieldName = pFieldName;
        oFieldFullPath = pFieldFullPath;
        oPrimaryField = pPrimaryField;
        oIsSelect = pIsSelect;
    }

    public string FieldAlias
    {
        get { return oTableAlias + "." + oFieldName; }
    }

    public string PrimaryFieldAlias
    {
        get { return oTableAlias + "." + oPrimaryField; }
    }

    // ...
    // običajne lastnosti (property)
    // ...
}
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

Sedaj, ko imamo definiran razred, s katerim predstavimo vozlišče drevesa, lahko iz seznama atributov napolnimo drevo. Zato sledi metoda, ki iz seznama atributov zgradi drevo.

```
/// <summary>
/// Iz seznama atributov napolni drevo
/// </summary>
/// <param name="pFields">seznam atributov</param>
/// <returns>Drevo atributov</returns>
public Node FillTree(List<string> pFields)
{
    Node fTree = new Node();
    foreach (string fField in pFields) // gremo čez seznam atributov
    {
        // celotno pot atributa (T1.A1\T2.A2\T3.A3) razdelimo na posamezne dele
        string[] fTableFields = fField.Split(new char[] { '\\' });
        Node fNode = fTree;
        for (int i = 0; i < fTableFields.Length; i++)
        {
            int fIndex = fNode.ChildrenKeys.IndexOf(fTableFields[i]);
            // če je atribut že v drevesu
            if (fIndex > -1) {
                fNode = fNode.Children[fIndex];
                if (i == fTableFields.Length - 1) fNode.Attribut.IsSelect = true;
                continue;
            }
            // če atributa še ni v drevesu
            else
            {
                bool fIsSelect = (i == fTableFields.Length - 1);
                string[] fFieldTable = fTableFields[i].Split('.');
                // Novemu atributu nastavimo polno ime. Če smo na prvem nivoju, je
                // to enako kratkemu imenu, sicer mu dodamo polno ime očeta.
                string fFieldFullPath = ((fNode.Attribut.FieldFullPath != null) ?
                 (fNode.Attribut.FieldFullPath + "\\") : "") + fTableFields[i];
                fNode.ChildrenKeys.Add(fTableFields[i]);
                // poiščemo atribut, na katerem je nastavljen primarni ključ
                // (če nanj kaže tuji ključ nastavljen na atributu)
                string fPrimaryField = GetPrimaryField(fFieldTable[0]);
                // atribut dodamo v drevo
                fNode.Children.Add(new TNode(fNode, new TAttribute(fFieldTable[0],
                 fFieldTable[1], fFieldFullPath, fIsSelect, fPrimaryField)));
                // dokler ne pridemo do zadnjega dela atributa, se premikamo proti
                // listu drevesa
                if (i < fTableFields.Length - 1)
                    fNode = fNode.Children[fNode.Children.Count - 1];
            }
        }
    }
    return fTree;
}
```

Ko je drevo napolnjeno z atributi, lahko z rekurzivnim pregledom drevesa sestavimo posamezne dele stavka *SELECT*.

```
public void Recursion(Node pTree, int pIndexAliasTable, List<string> pSelect,
    List<string> pFields, ref string pJoin)
{
    int fInx = 0;
```

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
    foreach (Node fNode in pTree.Children)
    {
        if (fNode.Father.Father == null)
            // če smo na glavnem nivoju je sinonim tabele kar ime tabele
            fNode.Attribute.TableAlias = TableName;
        else
            fNode.Attribute.TableAlias = "Table_" + pIndexAliasTable;

        // dodamo atribut v seznam atributov, ki bodo v stavku SELECT
        if (fNode.Attribute.IsSelect)
        {
            pSelect.Add(fNode.Attribute.FieldAlias);
            pFields.Add(fNode.Attribute.FieldFullPath);
        }

        // dodamo v join (v from)
        if (fNode.Father.Father != null)
        {
            if (oFirst)
            {
                // prvič dodamo glavno tabelo na začetek stavka FROM (pJoin)
                pJoin += fNode.Father.Attribute.TableName +
                " AS [" + fNode.Father.Attribute.TableAlias + "] LEFT OUTER JOIN " +
                fNode.Attribute.TableName + " AS [" + fNode.Attribute.TableAlias +
                "] ON " + fNode.Father.Attribute.FieldAlias + " = " +
                fNode.Attribute.PrimaryFieldAlias;
                oFirst = false;
            }
            else if (fInx == 0)
                pJoin += " LEFT OUTER JOIN " + fNode.Attribute.TableName + " AS [" +
                fNode.Attribute.TableAlias + "] ON " +
                fNode.Father.Attribute.FieldAlias + " = " +
                fNode.Attribute.PrimaryFieldAlias;
        }

        fInx++;
        // v drevesu gremo v globino, dokler ne pridemo do listov
        if (fNode.Children.Count > 0)
        {
            oIndexAliasTable++;
            // rekurzivni klic metode
            Recursive(fNode, oIndexAliasTable, pSelect, pFields, ref pJoin);
        }
    }
}
```

Kot rezultat navedene rekurzivne metode (rekurzivnega pregleda) dobimo seznam atributov (*List<string> pFields*), seznam atributom pripadajočih aliasov (*List<string> pSelect*) in ustrezno sestavljen del *FROM* (*pJoin*) stavka *SELECT*.

Da sestavimo celotno poizvedbo na sliki 18, moramo v rezultatu, ki ga vrne razčlenjevalnik, zamenjati atribute s pripadajočimi aliasi, ki jih vrne rekurzivna metoda. Tako dobimo glavni del stavka *SELECT* (del z atributi, ki so rezultat poizvedbe). Temu delu stavka *SELECT* dodamo del *FROM*, ki ga vrača rekurzivna metoda in dobimo končno poizvedbo.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

Poglejmo si programsko kodo metode, ki iz rezultata rekurzivne metode sestavi končno poizvedbo.

```
/// <summary>
/// Zgradi stavek SELECT
/// </summary>
/// <param name="pSelect">seznam ukazov, ki jih vrne razčlenjevalnik</param>
/// <param name="pFields">seznam atributov, ki jih vrne razčlenjevalnik</param>
/// <returns>SQL SELECT</returns>
public string SelectSQL(List<string[]> pSelect, List<string> pFields)
{
    string fResult = "SELECT ";

    Node fTree = FillTree(pFields); // zgradimo drevo

    List<string> fSelect = new List<string>();
    List<string> fFields = new List<string>();
    string fJoin = String.Empty;

    fTree.Attribut.TableName = TableName;
     // rekurzivni pregled drevesa
    Recursion(fTree, 0, ref fSelect, ref fFields, ref fJoin);

    int fIndex = 0;
    foreach (string[] fSel in pSelect) // za vse ukaze
    {
        string fSelectAtrib = fSel[0];
        int fSelLen = Convert.ToInt32(fSel[1]);
        // zamenjava atributov z aliasi
        for (int i = 0; i < fSelLen; i++)
        {
            fSelectAtrib = fSelectAtrib.Replace(pFields[fIndex],
                fSelect[fFields.IndexOf(pFields[fIndex])]);
            fIndex++;
        }
        // doda v stavek SELECT
        fResult += fSelectAtrib + " AS" + fSel[2] + ", ";
    }

    // če so vsi atributi samo iz glavne tabele
    if (fJoin == String.Empty) fJoin = TableName;
    // dodamo še del FROM
    fResult = fResult.Substring(0, fResult.Length - 2) + " FROM " + fJoin;

    return fResult;
}
```

Na enak način dobimo poizvedbo tudi za drugo podatkovno bazo. Če od SUPB zahtevamo, da nam poizvedbi izvrši, dobimo zapise baze sestavljene tako, kot smo določili s pravili na obrazcu za vnos primerjalnih pravil. Vendar naš cilj ni dobiti vse zapise, ampak samo tiste, ki nimajo enakega para v drugi bazi. Zato moramo rezultata poizvedb, ki jih vrne(ta) SUPB, med sabo primerjati in zapise, ki so enaki, izločiti. Ker poizvedbi sestavimo iz pravil, ki jih vnesemo na obrazcu za vnos primerjalnih pravil na sliki 13, obe vrneta enako število stolpcev z enakimi imeni. Tako je primerjanje zapisov poizvedb enostavno, saj vemo, da moramo med sabo le primerjati podatke v stolpcih z enakim imenom. Ker sta SUPB lahko

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

različna ali pa določen SUPB ne podpira izvedbe poizvedb na dveh podatkovnih bazah, sem ta korak primerjanja podatkov rešil programsko in ne na nivoju SUPB.

4.2.2.1.2 Primerjanje podatkov podatkovnih baz

Rezultat poizvedb, ki jih vrne SUPB, dobimo v objektu tipa DataTable. To je objekt, ki je del Microsoftovega ogrodja .Net Framework. Stolpce v objektu DataTable, ki prikazujejo podatke glede na pravila, ki so bila v modulu za vnos primerjalnih pravil nastavljena kot ključi, nastavimo kot primarni ključ. Na ta način dosežemo, da lahko podatke iščemo po indeksu. Nato z zanko pregledamo vse zapise rezultata poizvedbe na sekundarni podatkovni bazi. Vsak zapis poskusimo poiskati v rezultatu poizvedbe na primarni podatkovni bazi. Če zapisa ne najdemo ali pa ga najdemo in se zapisa ne ujemata vsaj v enem atributu, podatek prikažemo v tabeli na obrazcu za primerjavo podatkov. Vsakič, ko zapis najdemo, ga iz objekta tipa DataTable, kjer hranimo rezultat poizvedbe na prvi podatkovni bazi, zbrišemo. Tako vse podatke, ki na koncu ostanejo v rezultatu poizvedbe na primarni podatkovni bazi, v tabeli lahko prikažemo kot manjkajoče v sekundarni podatkovni bazi.

Programska koda

```
/// <summary>
/// izvede primerjavo podatkov v dveh podatkovnih bazah
/// </summary>
/// <param name="pPrimaryKeys">seznam indeksov polj na katerih je nastavljen ključ</param>
/// <param name="pSelectBaza1">podatki iz poizvedbe na primarni podatkovni bazi</param>
/// <param name="pSelectBaza2">podatki iz poizvedbe na sekundarni podatkovni bazi</param>
public void CompareProject(List<int> pPrimaryKeys, DataTable pSelectBaza1,
    DataTable pSelectBaza2)
{
    object[] fPKKeys = new object[pPrimaryKeys.Count]; // tabela ključev
    for (int i = 0; i < pSelectBaza2.Rows.Count; i++)
    {
        // seznam atributov označenih s ključem
        for (int k = 0; k < pPrimaryKeys.Count; k++)
            fPKKeys[k] = pSelectBaza2.Rows[i][pPrimaryKeys[k]];

        // poiščemo zapis v primarni podatkovni bazi
        DataRow fRow = pSelectBaza1.Rows.Find(fPKKeys);
        if (fRow != null) //zapis obstaja tako v prvi kot v drugi bazi
        {
            // primerjamo po vseh poljih
            for (int j = 0; j < pSelectBaza2.Columns.Count; j++)
            {
                // če najdemo polje, v katerem se zapisa razlikujeta,
                // zapis prikažemo v tabeli
                if (!pSelectBaza2.Rows[i][j].ToString().Equals(fRow[j].ToString()))
                {
                    for (int k = 0; k < pSelectBaza2.Rows[i].ItemArray.Length; k++)
                    {
                        mDataGridView1.Rows.Add(new object[] {
                            pSelectBaza2.Columns[k].Caption,  // naziv podatka
                            fRow.ItemArray[k],                // vrednost v prvi p. b.
                            pSelectBaza2.Rows[i].ItemArray[k] // vrednost v drugi p. b.
                        });
```

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA:
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
                    }
                    break;
                }
            }
            // v poizvedbi na primarni podatkovni bazi brišemo zapis
            pSelectBaza1.Rows.Remove(fRow);
        }
        else // če zapisa ne najdemo, ga v tabeli prikažemo
        {
            for (int j = 0; j < pSelectBaza2.Rows[i].ItemArray.Length; j++)
            {
                mDataGridView1.Rows.Add(new object[] {
                    pSelectBaza2.Columns[j].Caption,  // naziv podatka
                    String.Empty,                     // v prvi p. b. zapisa ni
                    pSelectBaza2.Rows[i].ItemArray[j] // vrednost v drugi p. b.
                });
            }
        }
    }

    // zapisi, ki še ostanejo v poizvedbi primarne podatkovne baze prikažemo kot
    // manjkajoče v sekundarni podatkovni bazi
    for (int i = 0; i < pSelectBaza1.Rows.Count; i++)
    {
        for (int j = 0; j < pSelectBaza1.Rows[i].ItemArray.Length; j++)
        {
            mDataGridView1.Rows.Add(new object[] {
                pSelectBaza1.Columns[j].Caption,   // naziv podatka
                pSelectBaza1.Rows[i].ItemArray[j], // vrednost v prvi p. b.
                String.Empty                       // v drugi p. b. zapisa ni
            });
        }
    }
}
```

4.2.2.1.3 Vpis podatkov v primarno podatkovno bazo

Če na obrazcu za primerjavo podatkov izberemo možnost Izvrši, program izvede primerjavo podatkov. V primeru, da program najde zapisa, ki imata enak ključ (enaki vrednosti vseh tistih izrazov v primerjalnih poljih, kjer je pravilo označeno kot ključ), razlikujeta pa se vsaj v vrednosti enega atributa, izvrši popravljanje (*UPDATE*) zapisa na primarni podatkovni bazi. V primeru, da zapis obstaja v sekundarni podatkovni bazi, v primarni podatkovni bazi pa zapisa ni, program izvrši vpis (*INSERT*) zapisa v primarno podatkovno bazo.

Prav tako kot pri primerjanju podatkov, tudi pri vpisu podatkov stolpce (atribute) v objektu DataTable, ki so bili v modulu za vnos primerjalnih pravil nastavljeni kot ključi, nastavimo kot primarni ključ. Nato z zanko pregledamo vse zapise rezultata poizvedbe na sekundarni podatkovni bazi. Posamezen zapis poskusimo poiskati v rezultatu poizvedbe na primarni podatkovni bazi. Če zapisa ne najdemo, ga vpišemo v primarno podatkovno bazo. Če se pri vpisu transakcija ne zaključi uspešno (npr.: ključ ni bi nastavljen na pravih pravilih, v

DIPLOMSKA NALOGA:
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

pravilih je bila uporabljena funkcija, ki ni bila pravilno definirana ...), zapis prikažemo v tabeli na obrazcu za primerjavo podatkov. Če pa zapis najdemo in se zapisa ne ujemata vsaj v enem atributu, kličemo popravljanje podatkov. V primeru, da se transakcija pri popravljanju zapisa ne zaključi uspešno, podatek prikažemo v tabeli na obrazcu za primerjavo podatkov. Vsakič, ko najdemo zapis, ga v DataTable zbrišemo. Vse podatke, ki na koncu ostanejo v rezultatu poizvedbe na primarni podatkovni bazi, v tabeli prikažemo kot manjkajoče v sekundarni podatkovni bazi.

Programska koda

```
/// <summary>
/// izvede vpis ali popravljanje podatkov v primarno podatkovno bazo
/// </summary>
/// <param name="pPrimaryKeys">seznam indeksov polj na katerih je nastavljen ključ</param>
/// <param name="pSelectBaza1">podatki iz poizvedbe na primarni podatkovni bazi</param>
/// <param name="pSelectBaza2">podatki iz poizvedbe na sekundarni podatkovni bazi</param>
/// <param name="pFields">seznam atributov, ki sestavljajo stavka UPDATE in INSERT</param>
public void InsertUpdateProject(List<int> pPrimaryKeys, DataTable pSelectBaza1,
    DataTable pSelectBaza2, List<TField> pFields)
{
    object[] fPKKeys = new object[pPrimaryKeys.Count];
    for (int i = 0; i < pSelectBaza2.Rows.Count; i++)
    {
        // seznam atributov označenih s ključem
        for (int k = 0; k < pPrimaryKeys.Count; k++)
        {
            fPKKeys[k] = pSelectBaza2.Rows[i][pPrimaryKeys[k]];
        }
        // poiščemo zapis v primarni podatkovni bazi
        DataRow fRow = pSelectBaza1.Rows.Find(fPKKeys);
        if (fRow != null)
        {
            // primerjamo po vseh poljih
            for (int j = 0; j < pSelectBaza2.Columns.Count; j++)
            {
                // če najdemo polje, v katerem se zapisa razlikujeta,
                // izvedemo popravljanje podatkov na primarni podatkovni bazi
                if (!pSelectBaza2.Rows[i][j].ToString().Equals(fRow[j].ToString()))
                {
                    object[] fOldValues = fRow.ItemArray; // stare vrednosti
                    object[] fNewValues = pSelectBaza2.Rows[i].ItemArray; // nove vrednosti
                    for (int m = 0; m < pFields.Count; m++)
                    {
                        // objektom, ki hranijo podatke o atributih nastavimo stare
                        // in nove vrednosti
                        pFields[m].OldValue = fOldValues[m];
                        pFields[m].Value = fNewValues[m];
                    }
                    // Če se popravljanje zapisa ne izvede uspešno, zapis prikažemo v tabeli.
                    // oDbTable1 je objekt, ki vsebuje funkcije, ki so potrebne
                    // za komunikacij s podatkovno bazo (primarno).
                    if (!oDbTable1.UpdateSQL(pFields))
                    {
                        for (int k = 0; k < pSelectBaza2.Rows[i].ItemArray.Length; k++)
                        {
                            mDataGridView1.Rows.Add(new object[] {
                                pSelectBaza2.Columns[k].Caption,  // naziv podatka
```

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
                              fRow.ItemArray[k],                // vrednost v prvi p. b.
                              pSelectBaza2.Rows[i].ItemArray[k] // vrednost v drugi p. b.
                          });
                      }
                  }
                  break;
              }
          }
          // v poizvedbi na primarni podatkovni bazi brišemo zapis
          pSelectBaza1.Rows.Remove(fRow);
      }
      else // če zapisa ne najdemo, ga vpišemo v primarno podatkovno bazo
      {
          object[] fNewValues = pSelectBaza2.Rows[i].ItemArray;
          for (int m = 0; m < pFields.Count; m++)
          {
              // objektom, ki hranijo podatke o atributih, nastavimo nove vrednosti
              pFields[m].Value = fNewValues[m];
          }
          // Če se vpis zapisa ne izvede uspešno, zapis prikažemo v tabeli.
          // oDbTable1 je objekt, ki vsebuje funkcije, ki so potrebne
          // za komunikacij s podatkovno bazo (primarno).
          if (!oDbTable1.InsertSQL(pFields))
          {
              for (int j = 0; j < pSelectBaza2.Rows[i].ItemArray.Length; j++)
              {
                  mDataGridView1.Rows.Add(new object[] {
                      pSelectBaza2.Columns[j].Caption,  // naziv podatka
                      String.Empty,                     // v prvi p. b. zapisa ni
                      pSelectBaza2.Rows[i].ItemArray[j] // vrednost v drugi p. b.
                  });
              }
          }
      }
  }

  // zapise, ki še ostaneja v poizvedbi po primarni podatkovni bazi, prikažemo kot
  // manjkajoče v sekundarni podatkovni bazi
  for (int i = 0; i < pSelectBaza1.Rows.Count; i++)
  {
      for (int j = 0; j < pSelectBaza1.Rows[i].ItemArray.Length; j++)
      {
          mDataGridView1.Rows.Add(new object[] {
              pSelectBaza1.Columns[j].Caption,   // naziv podatka
              pSelectBaza1.Rows[i].ItemArray[j], // vrednost v prvi p. b.
              String.Empty                       // v drugi p. b. zapisa ni
          });
      }
  }
}
```

V metodi *InsertUpdateProject* smo dejansko popravljanje oziroma vnos podatkov izvedli z metodama *UpdateSQL* oziroma *InsertSQL*. Oglejmo si ju.

Metoda *UpdateSQL* najprej vzpostavi povezavo s primarno podatkovno bazo, nato sestavi ustrezen stavek *UPDATE* in ga izvrši.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
/// <summary>
/// Izvrši popravljanje podatkov na podatkovni bazi
/// </summary>
/// <param name="pFields">seznam objektov s podatki o atributih
/// (ime atributa, ime tabele, tip atributa, stara vrednost, nova vrednost)</param>
/// <returns>true, če se transakcija izvede uspešno, sicer false</returns>
public override bool UpdateSQL(List<Field> pFields)
{
    // s pomočjo povezovalnega niza vzpostavimo povezavo s podatkovno bazo
    SqlConnection fSqlConn = new SqlConnection(base.Connection.ConnString);
    SqlCommand fSqlComm = new SqlCommand();
    fSqlComm.Connection = fSqlConn;

    pFields = AddForeignKeys(pFields); // nastavi prave vrednosti tujim ključem
    // sestavimo stavek SQL UPDATE
    string fSQL = "UPDATE [" + pFields[0].TableName + "] SET ";
    string fWhere = " WHERE ";
    for (int i = 0; i < pFields.Count; i++)
    {
        // iz tipa atributa ugotovimo pripadajoči tip v podatkovni bazi. S tem dosežemo, da
        // se vrednost iz niza pretvori v ustrezno obliko glede na tip
        SqlDbType fType = (SqlDbType)(Enum.Parse(typeof(SqlDbType),
            pFields[i].DataType, true));
        // dodamo parametre v del SET stavka UPDATE
        fSQL += "[" + pFields[i].FieldName + "]=@" + pFields[i].FieldName + ", ";
        // parametrom nastavimo nove vrednosti
        fSqlComm.Parameters.Add("@" + pFields[i].FieldName, fType).Value = pFields[i].Value;
        if (pFields[i].OldValue != null)
        {
            // dodamo parametre v del WHERE stavka UPDATE
            fWhere += "[" + pFields[i].FieldName + "]=@W" + pFields[i].FieldName + " AND ";
            // parametrom nastavimo stare vrednosti
            fSqlComm.Parameters.Add("@W" + pFields[i].FieldName, fType).Value =
                pFields[i].OldValue;
        }
    }
    // sestavimo celoten stavek UPDATE, pri čemer odrežemo vejico in presledek na koncu
    // dela SET (zato Length - 2) in " AND " na koncu dela WHERE (zato Length - 5)
    fSqlComm.CommandText = fSQL.Substring(0, fSQL.Length - 2) +
        fWhere.Substring(0, fWhere.Length - 5);
    SqlTransaction fTransaction = oSqlConn.BeginTransaction("UpdateTransaction");
    fSqlComm.Transaction = fTransaction;
    try
    {
        fSqlConn.Open(); // odpremo povezavo
        // izvršimo popravljanje podatkov na podatkovni bazi
        fSqlComm.ExecuteNonQuery();
        fTransaction.Commit(); // potrdimo transakcijo
        return true;
    }
    catch // če popravljanje ni uspelo
    {
        try
        {
            // transakcija vzpostavi prejšnje stanje
            fTransaction.Rollback();
            return false;
        }
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

```
        catch (Exception e)
        {
            // izvrši le v primeru težav na strežniku
            throw new Exception("Napaka: " + e.Message);
        }
    }
    finally
    {
        fSqlComm.Dispose();
        fSqlConn.Dispose();
    }
}
```

Druga metodo *InsertSQL*, izvrši vpis zapisov v primarno podatkovno bazo. Tudi ta metoda najprej vzpostavi povezavo s primarno podatkovno bazo. Nato sestavi ustrezen stavek *INSERT* in ga izvrši.

```
/// <summary>
/// Izvrši vpis podatkov v podatkovno bazo
/// </summary>
/// <param name="pFields">seznam objektov s podatki o atributih
/// (ime atributa, ime tabele, tip atributa, vrednost)</param>
/// <returns>true, če se transakcija izvede uspešno, sicer false</returns>
public override bool InsertSQL(List<TField> pFields)
{
    // s pomočjo povezovalnega niza vzpostavimo povezavo s podatkovno bazo
    SqlConnection fSqlConn = new SqlConnection(base.Connection.ConnString);
    SqlCommand fSqlComm = new SqlCommand();
    fSqlComm.Connection = oSqlConn;

    pFields = AddForeignKeys(pFields); // nastavi prave vrednosti tujim ključem

    // sestavimo stavek SQL INSERT
    string fSQLFields = "INSERT INTO [" + TableName + "](";
    string fSQLValues = ") VALUES(";
    foreach (TField fField in pFields)
    {
        // dodamo atribut v stavek INSERT
        fSQLFields += "[" + fField.FieldName + "], ";
        // dodamo parametre v del VALUES stavka UPDATE
        fSQLValues += "@" + fField.FieldName + ", ";
        // iz tipa atributa ugotovimo pripadajoči tip v podatkovni bazi, da se vrednost
        // iz niza pretvori v ustrezno obliko glede na tip
        SqlDbType fType = (SqlDbType)(Enum.Parse(typeof(SqlDbType), fField.DataType, true));
        // napolnimo parametre z vrednostmi
        fSqlComm.Parameters.Add("@" + fField.FieldName, fType).Value = fField.Value;
    }
    // sestavimo celoten stavek INSERT, pri čemer odrežemo vejico in presledek na koncu niza
    // atributov in na koncu niza vrednosti (zato Length - 2)
    fSqlComm.CommandText = fSQLFields.Substring(0, fSQLFields.Length - 2) +
        fSQLValues.Substring(0, fSQLValues.Length - 2) + ")";
    SqlTransaction fTransaction = oSqlConn.BeginTransaction("InsertTransaction");
    fSqlComm.Transaction = fTransaction;
    try
    {
        fSqlConn.Open(); // odpremo povezavo
        // izvršimo vpis v podatkovno bazo
        fSqlComm.ExecuteNonQuery();
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

```
        fTransaction.Commit(); // potrdimo transakcijo
        return true;
    }
    catch // če vpis ni uspel
    {
        try
        {
            // transakcija vzpostavi prejšnje stanje
            fTransaction.Rollback();
            return false;
        }
        catch (Exception e)
        {
            // izvrši le v primeru težav na strežniku
            throw new Exception("Napaka: " + e.Message);
        }
    }
    finally
    {
        fSqlComm.Dispose();
        fSqlConn.Dispose();
    }
}
```

Kot vidimo, metodi UpdateSQL in InsertSQL prekrijeta (override) virtualni metodi iz nadrazreda, ki vsebuje vse metode, ki jih program potrebuje za komunikacijo s podatkovno bazo. Metodi sta v nadrazredu definirani takole:

```
public virtual bool UpdateSQL(List<TField> pFields) { return false; }
public virtual bool InsertSQL(List<TField> pFields) { return false; }
```

Metodi ne moremo napisati že v nadrazredu, saj je od SUPB odvisno, kakšen je povezovalni niz in kateri *SQLConnection* oz. *SQLCommand* se uporablja. V obeh povoženih metodah kličemo metodo *AddForeignKeys*, ki atributom na katerih je nastavljen tuji ključ, dodeli pravo vrednost. Vrednost teh atributov nastavimo glede na vrednost atributov, ki enolično določajo zapis v tabeli, na katero kaže tuji ključ. V primeru, da podatka o vrednosti atributov program ne uspe najti, vrne napako. Zato je pomembno, da imamo pravila za primerjavo podatkov pravilno porazdeljena po projektih. Tako lahko najprej poženemo vpis podatkov na projektih, ki vpišejo podatke v tabele, na katere se sklicujejo tuji ključi. V primeru na sliki 17 imamo dva projekta *Osebe* in *Poste*. Ker se v projektu *Osebe* pravila sklicujejo na tabelo poštnih številk, ki pa jih vpišemo na projektu *Poste*, moramo najprej pognati vpis s projektom *Poste* in šele nato izvesti projekt *Osebe*. Na ta način zagotovimo, da imamo pri vpisu oseb že na voljo vse podatke o poštnih številkah.

DIPLOMSKA NALOGA : FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

5 ZAKLJUČEK

Kaj program že omogoča

Sedanja različica programa nam omogoča, da na enostaven način med sabo primerjamo dve bazi podatkov, ki ju upravlja bodisi Microsoftov SUPB bodisi sistem za upravljanje baz podatkov MySQL. Pregledamo lahko razlike med podatki v obeh podatkovnih bazah in tiste podatke, ki so le v eni od obeh baz. med njimi. Potem lahko v prvo podatkovno bazo vpišemo tiste podatke (zapise), ki v drugi podatkovni bazi so, v prvi jih pa ni. Podatke (posamezne atrubute) prve podatkovne baze lahko prepišemo z ustreznimi podatki druge podatkovne baze. Pri primerjanju, vpisu in popravljanju podatkov nismo omejeni le na podatkovne baze s povsem enako strukturo. Program namreč omogoča vnos pravil, s katerimi določimo operacije, ki se bodo izvajale na atributih. Tako lahko podatke ene podatkovne baze prilagodimo strukturi podatkov druge podatkovne baze in obratno.

Ideje, ki bi jih bilo smiselno vključiti v program

V prihodnje različice programa bi bilo smiselno vgraditi možnost vpisovanja oz. popravljanja podatkov v obe smeri. To pomeni možnost uvoza podatkov iz prve podatkovne baze v drugo in obratno. V tem primeru uporabniku ne bi bilo potrebno definirati primerjalnih pravil za vsako smer posebej, ampak bi bila lahko primerjalna pravila enaka za obe smeri. Koristno bi bilo tudi dodati možnost izbiranja zapisov, ki se pri vpisu podatkov smejo vpisati oz. popraviti. Ta možnost bi prišla prav takrat, kadar želimo prepisati samo določene podatke oziroma takrat, kadar so pri posameznih zapisih pravilni podatki v prvi, pri posameznih pa podatki v drugi bazi. Koristno bi bilo tudi dodati uporabniški vmesnik, ki bi uporabniku omogočil lažji vnos deklaracij funkcij. Vnašanje funkcij direktno v jeziku XML je namreč nepregledno in lahko prihaja do napak pri vnosu. Program bi lahko v naslednjih verzijah omogočal tudi primerjavo in uvoz podatkov iz drugih tipov podatkovnih struktur, kot je na primer uvoz podatkov iz navadnih tekstovnih datotek (npr.: txt, xml).

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA
FAKULTETA ZA MATEMATIKO IN FIZIKO

Viri

1. Dokumentacija za Microsoft SQL Server
   http://msdn2.microsoft.com/en-us/library/bb545450.aspx (dostopano 11.08.2007)

2. Dokumentacija za MySQL
   http://dev.mysql.com/doc/ (dostopano 11.08.2007)

3. Sistem za upravljanje podatkovnih baz
   http://colos1.fri.uni-lj.si/ERI/RACUNALNISTVO/PODATKOVNE_BAZE/supb2.html
   (dostopano 11.08.2007)

4. Regularni izrazi
   http://www.regular-expressions.info/ (dostopano 11.08.2007)
   http://slo-tech.com/clanki/03041/03041.shtml (dostopano 11.08.2007)

5. Seznam povezovalnih nizov
   http://www.connectionstrings.com/ (dostopano 11.08.2007)

6. Primerjava sintakse nekaterih SUPB
   http://troels.arvin.dk/db/rdbms/ (dostopano 11.08.2007)

7. Seznam funkcij, ki jih uporablja Microsoft SQL Server
   http://msdn2.microsoft.com/en-us/library/aa172604(SQL.80).aspx
   (dostopano 11.08.2007)

8. Seznam funkcij, ki jih uporablja MySQL
   http://dev.mysql.com/doc/refman/5.1/en/functions.html (dostopano 11.08.2007)

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO