UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI
FAKULTET
DEPARTMAN ZA MATEMATIKU
I INFORMATIKU

mr Srđan Škrbić

UPOTREBA FAZI LOGIKE U RELACIONIM BAZAMA PODATAKA

- doktorska disertacija -

Novi Sad, 2008.

PREDGOVOR

Doktorska disertacija pripada oblasti informacionih sistema, odnosno podoblasti koja se bavi upravljanjem skladištenjem i pretraživanjem informacija. Osnovni cilj disertacije je modeliranje i implementacija skupa alata koji omogućavaju upotrebu fazi logike u radu sa relacionim bazama podataka.

Da bi se do tog skupa alata došlo, najpre je relacioni model podataka proširen elementima teorije fazi skupova, a zatim je definisano fazi proširenje upitnog jezika SQL – PFSQL. Interpreter za taj jezik je implementiran u okviru fazi JDBC drajvera koji, osim implementacije interpretera, sadrži i elemente koji omogućavaju jednostavnu upotrebu ovih mehanizama iz programskog jezika Java. Skup alata je zaokružen implementacijom CASE alata za razvoj fazi-relacionog modela baze podataka. Osim toga, razmatrane su i mogućnosti za upotrebu PFSQL jezika u višeslojnim aplikacijama.

Disertacija se sastoji od sledećih sedam poglavlja i zaključka:

1. Uvod
2. Fazi skupovi i CSP
3. Fazi baze podataka
4. Skladištenje fazi podataka
5. PFSQL i fazi JDBC drajver
6. CASE alat za rad sa fazi bazama podataka
7. Upotreba jezika PFSQL u višeslojnim sistemima

Prvo poglavlje predstavlja osvrt na razvoj ideja koje su dovele do realizacije ove doktorske disertacije. Prikazane su osnove organizacije skupa alata za rad sa fazi logikom u relacionim bazama podataka. Osim toga, dat je i pregled tema koje se obrađuju u pojedinim poglavljima.

Sledeće poglavlje predstavlja uvod u teoriju fazi skupova i fazi logiku. U prvom delu poglavlja su dati osnovni pojmovi bitni za realizaciju

ciljeva ove doktorske disertacije. Drugi deo poglavlja predstavlja uvod u fazifikaciju problema zadovoljenja ograničenja koja predstavlja teorijsku osnovu za realizaciju PFSQL interpretera.

U trećem poglavlju je dat pregled istraživanja u oblasti upotrebe fazi logike u bazama podataka koja su prethodila ovoj doktorskoj disertaciji. Četvrto poglavlje sadrži detaljan opis proširenja relacionog modela podataka elementima fazi logike. Osim strukture samih proširenja, dat je i obiman primer ovakvog modela baze podataka koji predstavlja fazifikaciju jednog segmenta modela baze podataka informacionog sistema studentske službe na Prirodno-matematičkom fakultetu u Novom Sadu.

Peto poglavlje se bavi osobinama PFSQL jezika, kao i implementacijom fazi JDBC drajvera koji sadrži interpreter za taj jezik. Data je analiza mogućnosti za proširenje SQL jezika fazi konstrukcijama i sintaksa PFSQL jezika nastala na osnovu te analize. U nastavku je dat opis implementacije fazi JDBC drajvera koji u sebe uključuje interpreter za jezik PFSQL.

Modeliranje i implementacija CASE alata za rad sa fazi bazama podataka su prikazani u šestom poglavlju. Poslednje, sedmo poglavlje sadrži diskusiju o mogućnostima upotrebe PFSQL jezika na srednjem sloju višeslojnih aplikacija koje se oslanjaju na EJB 3.0 tehnologiju.

Na kraju disertacije je dat zaključak o značaju i doprinosu teze, kao i predlozi za dalje pravce istraživanja u ovoj oblasti.

U disertaciji je usvojen i korišćen APA (American Psychological Association) stil citiranja referenci.

Zahvaljujem se komisiji koja je detaljno pregledala rad i svojim korisnim predlozima uticala na konačnu verziju disertacije. Posebnu zahvalnost dugujem mentoru prof dr Milošu Rackoviću na nesebičnoj podršci u toku izrade disertacije.

Zahvaljujem se Svetlani i porodici na razumevanju i podršci.

Novi Sad, 2008. Srđan Škrbić

SADRŽAJ

Poglavlje 1

UVOD

Ova doktorska disertacija predstavlja rezultat višegodišnjih napora autora u istraživanju inovativnih i nedovoljno istraženih načina za upravljanje podacima.

U svojoj magistarskoj tezi (Škrbić, 2004), autor daje detaljan prikaz XML (eXstensible Markup Language) native baza podataka i mogućnosti njihove primene u radu sa XML bibliografskim zapisima. Tamino baza podataka i alati koje ona nudi su upotrebljeni za skladištenje XML bibliografskih zapisa i implementaciju aplikacije za upravljanje XML bibliografskim zapisima u XML native tehnologiji. Ovi rezultati su kasnije u sažetom obliku publikovani u (Škrbić & Surla, 2008).

U ovoj doktorskoj disertaciji je pažnja usmerena na upravljanje nepreciznim i nepotpunim informacijama u radu sa relacionim bazama podataka. Polazi se od pretpostavke da vrednosti atributa u relacionim bazama podataka ne moraju biti egzaktne. Ponekad postoji potreba da se zadaju neprecizne i nejasne, ali i nepotpune vrednosti. Upotreba fazi skupova i fazi logike predstavlja jedan od načina da se postigne ovaj cilj. Fazi skupovi su uopštenje klasičnih skupova u smislu da mera pripadnosti elementa nekom skupu može uzimati vrednosti iz jediničnog intervala. Uveo ih je Lotfi Zadeh 1965. godine (Zadeh, 1965). Od tada su se teorija fazi skupova i fazi logika razvile u moćne alate sa širokim spektrom primene. Rezultati iz oblasti fazi logike se koriste na mnogim poljima: upravljanje i kontrola sistema, prognoza finansijskih trendova, medicinska dijagnostika, geološka istraživanja itd. U skladu sa tim, kratko rečeno, cilj ove doktorske disertacije je razvoj skupa alata koji omogućava upotrebu fazi logike u relacionim bazama podataka.

Da bi se takav cilj dostigao, definisano je nekoliko potciljeva. Prvo, potrebno je proširiti i prilagoditi relacionu bazu podataka aparaturom koja omogućava upotrebu fazi vrednosti. Od različitih mogućnosti i mnoštva opcija koje nudi teorija fazi skupova, odabrani su oni elementi koji se najčešće koriste u praksi. Zatim su za te elemente definisani mehanizmi za skladištenje u relacionoj bazi podataka. Na taj način je nastao model za skladištenje fazi informacija kojim se bavi četvrto poglavlje.

Pored toga, potrebno je proširiti i sam upitni jezik SQL (Structured Query Language), tako da omogući rad sa fazi vrednostima i postavljanje fazi upita. Osim standardnih konstrukcija fazi logike, u proširenja su uključene i mogućnosti za definisanje prioriteta pojedinih iskaza. Slično kao i kod modela za skladištenje fazi informacija, i ovde je urađena analiza mogućnosti za proširenje jezika SQL. Imajući u vidu ovu analizu i definisani model za skladištenje fazi informacija, definisan je jezik PFSQL (Priority Fuzzy Structured Query Language). Osim toga, za rad sa takvom bazom podataka je implementiran drajver koji omogućava njenu upotrebu pomoću objektno-orijentisanih jezika, konkretno, programskog jezika Java. Drajver u sebe uključuje i mehanizme za interpretaciju PFSQL upita. Ova tema je detaljno obrađeno u petom poglavlju.

Kako modeliranje fazi baze podataka pomoću postojećih CASE (Computer Aided Software Engineering) alata nije moguće, realizovan je i odgovarajući CASE alat. Alat je realizovan tako da podržava pomenuti model za skladištenje fazi informacija. Time je omogućeno jednostavno projektovanje fazi baze podataka bez potrebe da se poznaju detalji interne realizacije proširenja relacionog modela fazi elementima. Detaljan prikaz modeliranja, implementacije i mogućnosti ovog alata je dat u šestom poglavlju.

Za modeliranje i implementaciju svih komponenata sistema je korišćena objektno-orijentisana metodologija razvoja softvera. Modeliranje je urađeno koristeći jezik UML (Unified Modeling Language), dok je za implementaciju korišćen programski jezika Java.

Kako je direktno programiranje baze podataka retko u većim informacionim sistemima, razmotrene su i mogućnosti upotrebe fazi baze podataka i PFSQL jezika u višeslojnom okruženju. Diskusiju na ovu temu

sadrži sedmo poglavlje. Kao tehnologija za razvoj srednjeg sloja višeslojnih aplikacija je izabrana Enterprise JavaBeans 3.0 tehnologija. Iako su predložene neke mogućnosti za upotrebu PFSQL jezika u ovoj tehnologiji, najbolje moguće rešenje je samo nagovešteno kao dalji pravac istraživanja.

U pokušaju implementacije jezika se postavilo pitanje metoda za izračunavanje fazi stepena zadovoljenja pojedinih torki u rezultatu upita. Zbog toga su, kao teorijska osnova, proučeni fazifikovani problemi zadovoljenja ograničenja koji uključuju prioritetnu logiku - PFCSP (Priority Fuzzy Constraint Satisfaction Problem). Definisano je proširenje PFCSP-a koje uključuje i operator OR, odnosno, t-konormu, koje je nazvano GPFCSP (Generalized Priority Fuzzy Constraint Satisfaction Problem). Ovim sistemima, kao i osnovnim pojmovima teorije fazi skupova i fazi logike se bavi drugo poglavlje.

Detaljnim prikazom postojeće literature i dostignuća u oblasti upotrebe fazi logike u relacionim bazama podataka se bavi treće poglavlje. Između ostalog, ovde je dat kratak prikaz FIRST-2 modela i FSQL jezika, kao dva najozbiljnija dosadašnja pokušaja da se fazi koncepti približe upotrebi u bazama podataka. Takođe, dat je pregled prethodnih radova autora i istraživačke grupe kojoj pripada iz predmetne oblasti.

Istraživanje prikazano u ovoj disertaciji se nastavlja na disertaciju (Takači, Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja, 2006) u kojoj je dat predlog proširenja relacionog modela elementima fazi logike. Osim toga, u toj disertaciji je opisan podskup SQL-a sa fazi proširenjima za koga je implementirana i softverska podrška. U ovoj disertaciji je usavršen metod za implementaciju fazi baze podataka i upitnog jezika za tu bazu u odnosu na rezultate postignute u gore pomenutoj disertaciji. Dato je i proširenje teorijske osnove za izračunavanje rezultata upita. U pomenutoj disertaciji je istražen PFCSP, dok je ovde kao teorijska osnova prikazan GPFCSP sistem. Izvršena je kompletna rekonstrukcija modela za skladištenje fazi informacija, tako da je dobijen opširniji, robusniji i efikasniji model. Takođe, urađena su i brojna proširenja SQL jezika koja ranije nisu razmatrana ni praktično ni teoretski. Najvažnije proširenje je mogućnost postavljanja upita sa prioritetom gde je uslov proizvoljan logički izraz. U prethodnoj varijanti su bile moguće samo određene konstrukcije sa ograničenim skupom operatora. Da bi se olakšala

praktična upotreba realizovanih mehanizama, razvijen je i prateći skup alata, fazi JDBC (Java Database Connectivity) drajver i CASE alat, koji to omogućavaju. Kao dodatak, date su smernice za razvoj višeslojnih aplikacija koje koriste ovakvu bazu podataka.

Postoji nekoliko tačaka koje se mogu navesti kao ograničenja dobijenih rezultata, ali i kao smernice za dalja istraživanja. Kao prvo, tu su mogućnosti daljih proširenja samog jezika PFSQL. PFSQL jezik, za sada, podrazumeva samo proširenja SELECT klauzule. Pored mogućnosti za dalje proširenje funkcionalnosti koje su opisane u ovoj disertaciji, istraživanje se može usmeriti u cilju proširenja ostalih naredbi SQL jezika koje služe za unos, brisanje i modifikaciju podataka. Model za skladištenje fazi informacija je još jedno mesto gde ima prostora za unapređenje. Opisani model je ograničen na određene konstrukcije koje se najčešće koriste u praksi, ali budući modeli bi mogli da uključe i dodatne mogućnosti. Na kraju, kao što je to prikazano u sedmom poglavlju, u segmentu upotrebe fazi mehanizama na srednjem sloju višeslojne aplikacije ima širokog prostora za dalji rad.

Detaljnim uvidom u literaturu se došlo do zaključka da je skup alata predstavljen u ovoj disertaciji prvi pokušaj da se upotreba fazi logike u relacionim bazama podataka zaokruži u sistem i na taj način približi krajnjem korisniku. Iako postoje pokušaji, pa čak i implementacije jezika zasnovanih na proširenju SQL jezika fazi mehanizmima, PFSQL je prvi jezik koji uključuje i upotrebu prioriteta u kombinaciji sa fazi iskazima. Ideja da se interpreter za takav jezik uobliči kao fazi JDBC drajver se takođe sreće prvi put. Na taj način je postignuta nezavisnost u odnosu na bazu podataka, a upotreba PFSQL jezika je učinjena bližom korisniku. Na kraju, implementacija CASE alata za projektovanje modela baza podataka koji uključuju fazi logiku je takođe originalna.

U skladu sa interesovanjima autora usmerenim ka XML tehnologijama iskazanim na početku ovog poglavlja, razmatrane su i mogućnosti primene fazi logike u radu sa XML dokumentima i XML native bazama podataka. Ispostavilo se da i na ovom polju postoje prethodna istraživanja opisana u (Ma, 2005) i (Ma & Yan, 2007). Dalje istraživanje ovih mogućnosti je ostavljeno kao jedna od smernica za dalji rad.

Poglavlje 2

FAZI SKUPOVI I CSP

U ovom poglavlju je dat uvod u teoriju fazi skupova i fazi logiku. U prvom delu poglavlja su dati osnovni pojmovi bitni za realizaciju ciljeva ove doktorske disertacije. Drugi deo poglavlja predstavlja uvod u fazifikaciju problema zadovoljenja ograničenja koja predstavlja teorijsku osnovu za realizaciju PFSQL interpretera koja je opisana u petom poglavlju. Osnovne reference korišćene u ovom poglavlju su (Bistarelli, Montanari, Rossi, Schiex, Verfaillie, & Fargier, 1999), (Bodenhofer, 2003), (Dubois & Prade, 1980), (Klement, Mesiar, & Pap, 2000) i (Takači, 2006).

2.1 FAZI SKUPOVI

Koristeći klasičnu logiku, moguće je raditi jedino sa informacijama koje su ili u potpunosti tačne ili u potpunosti pogrešne. Nije moguće upravljati informacijama koje su neprecizne ili nekompletne, iako ovakve informacije mogu pružiti bolje rešenje nekog problema. Pripadnost skupu, u terminima klasične logike, je predstavljena ili sa vrednošću 0, kada element ne pripada skupu, ili sa vrednošću 1, kada element pripada skupu. Originalna interpretacija fazi skupova je nastala generalizacijom klasičnog koncepta skupa tako što je kodomen karakteristične funkcije skupa proširen sa skupa {0, 1} na jedinični interval. Pojam fazi skupa je u svom radu uveo Zadeh (Zadeh, 1965), koji se smatra i tvorcem fazi logike. Fazi logika je logika koja stoji iza približnog umesto preciznog zaključivanja. Njen značaj i veliki potencijal leže u činjenici da je ljudsko razmišljanje po prirodi aproksimativno. Dobar izvor vezan za teoriju fazi skupova i fazi logiku predstavlja (Dubois & Prade, 1980).

2.1.1 Uvodna razmatranja

Formalno, fazi skup se može definisati ovako:

Definicija 2.1: Fazi skup A nad univerzumom X je određen svojom karakterističnom funkcijom $\mu_A : X \to [0,1]$, gde se za svako $x \in X$ $\mu_A(x)$ interpretira kao stepen pripadnosti elementa x fazi skupu A.

Vrednost $\mu_A(x) = 0$ označava da element x uopšte ne pripada skupu A, dok vrednost $\mu_A(x) = 1$ označava da element x u potpunosti pripada skupu A. Često se za univerzum uzima skup realnih brojeva. Slede definicije niza osnovnih koncepata vezanih za fazi skupove.

Definicija 2.2: Dva fazi skupa A i B su jednaki, u oznaci A=B ako i samo ako važi $\forall x \in X, \mu_A(x) = \mu_B(x)$.

Definicija 2.3: Za fazi skup A kažemo da je podskup fazi skupa B, u oznaci $A \subseteq B$ ako i samo ako važi $\forall x \in X, \mu_A(x) \leq \mu_B(x)$.

Definicija 2.4: Nosač fazi skupa A je skup:
$supp(A) = \{x \in X \mid \mu_A(x) > 0\}$.

Definicija 2.5: Jezgro fazi skupa A je skup:
$ker(A) = \{x \in X \mid \mu_A(x) = 1\}$.

Definicija 2.6: Visina fazi skupa A je broj: $h(A) = \sup_{x \in X} \mu_A(x)$.

Definicija 2.7: Za fazi skup A kažemo da je normalizovan ako i samo ako važi $\exists x \in X, \mu_A(x) = h(A) = 1$.

Definicija 2.8: α-odsečci fazi skupa A, u oznaci $[A]_\alpha$, za svako $\alpha \in [0,1]$, definišu se na sledeći način: $[A]_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}$.

Definicija 2.9: Fazi skup A se naziva konveksnim ako i samo ako su za svako $\alpha \in [0,1]$ α-odsečci konveksni podskupovi od X.

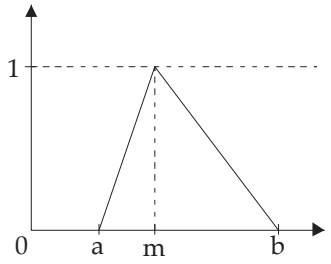
Definicija 2.10: Fazi broj je fazi skup A koji je normalizovan, konveksan i ima ograničeno jezgro.

2.1.2 Vrste karakterističnih funkcija

U zavisnosti od tipa karakteristične funkcije, dobijaju se različiti fazi skupovi. Zadeh je klasifikovao ove funkcije u dve kategorije: linearne i nelinearne. Sledi prikaz najčešćih vrsta karakterističnih funkcija, odnosno, fazi skupova.

1. Trougaoni fazi broj, prikazan na slici 2.1, se definiše na sledeći način:

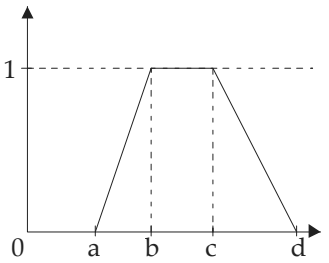
$$\mu_A(x) = \begin{cases} 0, x \le a, x \ge b \\ \dfrac{(x-a)}{(m-a)}, x \in (a,m] \\ \dfrac{(b-x)}{b-m}, x \in (m,b) \end{cases}$$



Slika 2.1. Trougaoni fazi broj.

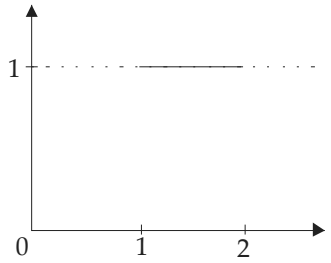
2. Trapezoidni fazi broj je prikazan na slici 2.2, a definiše se na sledeći način:

$$\mu_A(x) = \begin{cases} 0, x \le a, x \ge d \\ \dfrac{(x-a)}{(b-a)}, x \in (a,b) \\ 1, x \in (b,c) \\ \dfrac{(d-x)}{(d-c)}, x \in (b,d) \end{cases}$$



Slika 2.2. Trapezoidni fazi broj.

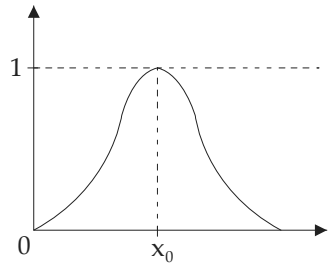
3. Interval se takođe može predstaviti kao fazi broj. Na slici 2.3 je prikazan interval [1,2] kao fazi broj.



Slika 2.3. Interval kao fazi broj.

4. Gausov fazi broj je prikazan na slici 2.4, a njegova karakteristična funkcija je:

$$\mu_A(x) = e^{\frac{-(x-x_0)^2}{d}}$$

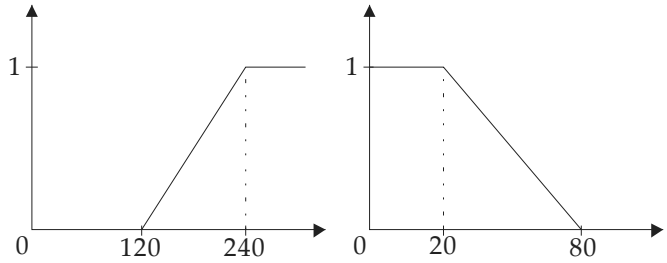


Slika 2.4. Gausov fazi broj.

Ponekad je potrebno modelirati veličine koje su ograničene samo sa jedne strane. Na primer, pojam „brzi automobili“. Ako je maksimalna brzina automobila 240 km/h svi će se složiti da je on brz, a ako je 120 km/h, svi će se složiti da nije. Što je automobil brži, to je njegova pripadnost fazi skupu „brzi automobili“ veća i obrnuto. Zbog toga se definišu fazi veličine.

Definicija 2.11: Fazi veličine su normalizovani fazi skupovi čija je karakteristična funkcija neprekidna i monotona, a jezgro interval ograničen sa jedne strane.

Na slici 2.5 su prikazane fazi veličine „brzi automobili“ i „niske zgrade“.



Slika 2.5. Fazi veličine „brzi automobili“ i „niske zgrade“.

2.1.3 Trougaone norme i konorme

Iz činjenice da je teorija fazi skupova generalizacija klasične teorije skupova sledi da se i operacije unije, preseka i komplementa mogu definisati i nad fazi skupovima. Da bi se došlo do ove generalizacije, potrebno je prvo definisati pojmove negacije i trougaonih normi i konormi, koje su nastale kao proširenje operacija $\neg$, $\wedge$ i $\vee$, respektivno. Ovde su prikazane samo najosnovnije definicije i osobine vezane za ove operacije, dok se detaljan opis može naći u (Klement, Mesiar, & Pap, 2000).

Sledeća definicija daje uopštenje klasične negacije.

Definicija 2.12:

1. Nerastuća funkcija je negacija ako važi: $N(0) = 1 \wedge N(1) = 0$.
2. Negacija $N : [0,1] \to [0,1]$ je stroga negacija ako je neprekidna i strogo opadajuća.
3. Stroga negacija $N : [0,1] \to [0,1]$ je jaka negacija ako je involucija, odnosno, ako važi $N \circ N = id_{[0,1]}$.

Najčešće korišćena negacija je tzv. standardna negacija: $N_S(x) = 1 - x$. Dokazano je da je svaka jaka negacija transformacija standardne negacije.

Definicija 2.13: Funkcija $T : [0,1] \times [0,1] \to [0,1]$ je trougaona norma, ili t-norma, ako zadovoljava sledeće osobine:

1. Komutativnost: $T(x, y) = T(y, x)$
2. Asocijativnost: $T(x, T(y, z)) = T(T(x, y), z)$

3. Monotonost: ako je , $y \leq z$ onda je $T(x,y) \leq T(y,z)$
4. Granični uslovi: $T(x,1) = x$ i $T(x,0) = 0$.

Četiri osnovne t-norme su:

1. $T_M(x,y) = \min(x,y)$,
2. $T_P(x,y) = x \cdot y$,
3. $T_L(x,y) = \max(x + y - 1,0)$ i
4. $T_D(x,y) = \begin{cases} 0, (x,y) \in [0,1)^2 \\ \min(x,y), inače \end{cases}$

Osobina asocijativnosti omogućava da t-normu proširimo na n-arni operator koristeći indukciju:

$$T_{i=1}^{n} x_i = T(T_{i=1}^{n-1} x_i, x_n)$$

Sada četiri osnovne t-norme postaju:

1. $T_M(x_1,...,x_n) = \min(x_1,...,x_n)$,
2. $T_P(x_1,...,x_n) = x_1 \cdots x_n$,
3. $T_L(x_1,...,x_n) = \max(\sum_{i=1}^{n} x_i - (n-1),0)$ i
4. $T_D(x_1,...,x_n) = \begin{cases} x_i, ako\ \forall j \neq i, x_j = 1 \\ 0, inače \end{cases}$

Može se pokazati da važi $T_D \leq T_L \leq T_P \leq T_M$ i $T_D \leq T \leq T_M$ za svaku t-normu T.

Trougaone konorme se koriste kao uopštenje operatora $\vee$, a od t-normi se razlikuju samo po graničnim uslovima.

Definicija 2.14: Funkcija $S : [0,1] \times [0,1] \to [0,1]$ je trougaona konorma, ili t-konorma ili s-norma, ako zadovoljava sledeće osobine:

1. Komutativnost: $S(x,y) = S(y,x)$
2. Asocijativnost: $S(x,S(y,z)) = S(S(x,y),z)$
3. Monotonost: ako je , $y \leq z$ onda je $S(x,y) \leq S(y,z)$
4. Granični uslovi: $S(x,1) = 1$ i $S(x,0) = x$.

Negacija se koristi za definiciju komplementa fazi skupa kao i dualnosti t-normi i t-konormi. Za svaku t-normu T dobijamo njenu dualnu konormu S koristeći strogu negaciju N na sledeći način:

$$S(x,y) = N(T(N(x),N(y)).$$

Ako N zamenimo standardnom negacijom, dobijamo:

$$S(x,y) = 1 - T(1-x,1-y).$$

Slično, za svaku t-konormu S dobijamo dualnu t-normu T datu sa:

$$T(x,y) = N(S(N(x),N(y)).$$

Odnosno, u slučaju da je N standardna negacija:

$$T(x,y) = 1 - S(1-x,1-y).$$

Imajući u vidu da je t-norma uopštenje konjunkcije, a t-konorma uopštenje disjunkcije postaje jasno da su ove formule uopštenja De Morganovih obrazaca. Dakle, ako je N stroga negacija, T t-norma, a S njena dualna konorma, onda (T,S,N) čini De Morganovu trojku.

Koristeći ove transformacije dobijamo četiri osnovne t-konorme:

1. $S_M(x,y) = \max(x,y)$,
2. $S_P(x,y) = x + y - xy$,
3. $S_L(x,y) = \min(x+y,1)$ i
4. $S_D(x,y) = \begin{cases} 1, (x,y) \in [0,1)^2 \\ \max(x,y), inače \end{cases}$

Analogno t-normama, i t-konorme se mogu proširiti na n-arne operatore na sledeći način:

1. $S_M(x_1,...,x_n) = \max(x_1,...,x_n)$,
2. $S_P(x_1,...,x_n) = 1 - \prod_{i=1}^{n}(1-x_i)$,
3. $S_L(x_1,...,x_n) = \min(\sum_{i=1}^{n} x_i, 1)$ i

4. $S_D(x_1,...,x_n) = \begin{cases} x_i, ako\ \forall j \neq i, x_j = 0 \\ 1, inače \end{cases}$.

2.1.4 Fazi logika

Proučavanje logike sa više od dve istinitosne vrednosti je započeo Lukašijevič uvođenjem treće istinitosne vrednosti – delimično tačno (Lukasiewicz, 1920). Uvođenjem pojma fazi skupa (Zadeh, 1965), Zadeh je otvorio put za primenu logika sa više istinitosnih vrednosti.

Formalna definicija i različite aksiomatike fazi logike se mogu naći u (Takači, Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja, 2006) i, šire, u (Klement, Mesiar, & Pap, 2000), ali one izlaze van okvira ove disertacije. Bez zalaženja u detalje se može reći da fazi logiku dobijamo fazifikacijom klasične logike. Skup istinitosnih vrednosti se proširuje sa {0,1} na jedinični interval [0,1], konjunkcija se zamenjuje t-normom, disjunkcija t-konormom, a negacija negacijom kakva je definisana u 2.1.3. Ovim je dat potreban aparat za izračunavanje istinitosnih vrednosti iskaza u fazi logici dovoljan za potrebe ove disertacije.

2.1.5 Skupovna algebra fazi skupova

Definicije negacije i trougaonih normi i konormi iz prethodnog paragrafa omogućavaju uvođenje elementarnih operacija nad fazi skupovima – unije, preseka i komplementa.

Neka su date t-norma T, t-konorma S i negacija N, kao i fazi skupovi A i B i njihove karakteristične funkcije $\mu_A(x)$ i $\mu_B(x)$. Skupovi $A \cup B$, $A \cap B$ i $C_N(A)$ definišu se preko njihovih karakterističnih funkcija na sledeći način.

Definicija 2.15: Unija dva fazi skupa A i B je data sa:

$$\mu_{A\cup B}(x) = S(\mu_A(x), \mu_B(x)) \ ,$$

presek dva fazi skupa A i B je data sa:

$$\mu_{A\cap B}(x) = T(\mu_A(x), \mu_B(x)) \ ,$$

dok je komplement fazi skupa A dat sa:

$$\mu_{C_N(A)}(x) = N(\mu_A(x)) \, .$$

Osobine ovih operacija se mogu izvesti direktno iz osobina trougaonih normi, konormi i negacije.

Interesantno je da, čak i u slučaju da je (T,S,N) De Morganova trojka, jednakosti $A \cap C_N(A) = \varnothing$ i $A \cup C_N(A) = X$ ne važe za svaki fazi podskup A univerzuma X. Sa druge strane, ako je data De Morganova trojka (T_L, S_L, N_S) , ili takva De Morganova trojka da važi $T \le T_L$, tada za svako A važi $A \cap C_N(A) = \varnothing$ i $A \cup C_N(A) = X$.

Ako uzmemo u obzir da karakteristična funkcija označava meru pripadanja nekog elementa skupu, relacija podskupa se prirodno proširuje na fazi slučaj. Ona je data u definiciji 2.3. Na osnovu te definicije sledi da, u slučaju da je $\mu_A(x) = \alpha$, element x zadovoljava ne samo kriterijume pripadnosti skupu A sa vrednošću α, nego i sa svakom vrednošću koja je manja od α.

Slično kao sa karakterističnom funkcijom fazi podskupa, može se uvesti mera koliko je jedan fazi skup podskup nekog drugog fazi skupa. Za ove potrebe se najčešće koristi relacija kompatibilnosti dva fazi skupa.

Definicija 2.16: Neka su A i B dva fazi skupa nad univerzumom X. Mera kompatibilnosti skupa A sa skupom B je data na sledeći način

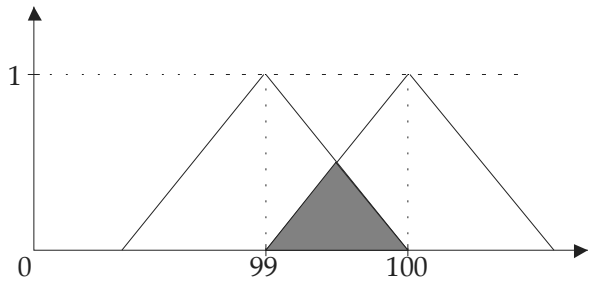
$$C_{A,B} = \frac{P(A \cap B)}{P(A)} .$$

Gde je sa P predstavljena mera skupa, odnosno površina u dvodimenzionalnom prostoru. Potrebno je uočiti da mera kompatibilnosti nije simetrična ni tranzitivna, ali jeste refleksivna. U slučaju da je $A \subseteq B$, važi $C_{A,B} = 1$. Što je više elemenata za koje važi $\mu_A(x) > \mu_B(x)$, to je mera kompatibilnosti manja. Odnosno, što su dva skupa „bliža“ jedan drugom,

to je mera kompatibilnosti veća. Pošto je podskup antisimetrična relacija, važi:

$$\text{ako je } C_{A,B} = 1 \wedge C_{B,A} = 1, \text{ onda } A = B.$$

Posmatrajmo primer mere kompatibilnosti dva trougaona fazi broja. Neka to budu trougaoni fazi brojevi koji predstavljaju broj „približno 99“ i broj „približno 100“ sa tolerancijom 1 (slika 2.6).



Slika 2.6. Kompatibilnost fazi skupova.

Očigledno, mera kompatibilnosti ova dva trougaona fazi broja je 0.25.

2.1.6 Poredak

Uvođenje poretka među fazi skupovima nije jednostavan problem. Tokom godina su proučene različite vrste poredaka, a neki od rezultata se mogu naći u (Dubois & Prade, 1980), (Zadeh, 1971) i (Bodenhofer, 2003). Kompletan problem uvođenja poretka izlazi van okvira ove disertacije.

U ovoj sekciji će biti opisan najčešći poredak koji nastaje kao uopštenje dobro poznate relacije $\leq$.

Neka je I skup svih podintervala skupa realnih brojeva. Generalizacija poretka $\leq$ nad I za intervale $[a_1, b_1]$ i $[a_2, b_2]$ je sledećeg oblika:

$$[a_1, b_1] \leq [a_2, b_2] \Leftrightarrow a_1 \leq a_2 \wedge b_1 \leq b_2.$$

Dakle, što je interval više desno na x osi, on se smatra većim. Imajući ovu generalizaciju u vidu, opišimo poredak nad fazi skupovima.

Definicija 2.17: Neka je $\leq$ poredak nad univerzumom X i neka je A fazi skup nad X. Fazi nadskup od A, u oznaci LTR(A) se definiše na sledeći način:

$$\mu_{LTR(A)}(x) = \sup\{\mu_A(y)\} \mid y \leq x\},$$

analogno, RTL(A) se definiše kao:

$$\mu_{RTL(A)}(x) = \sup\{\mu_A(y)\} \mid x \leq y\}.$$

LTR(A) je, u stvari, najmanji fazi nadskup od A sa neopadajućom karakterističnom funkcijom, dok je RTL(A) najmanji fazi nadskup od A sa nerastućom karakterističnom funkcijom.

Definicija 2.18: Neka su A i B fazi skupovi nad univerzumom X. Tada se poredak $\leq_F^{'}$ nad skupom svih fazi skupova nad univerzumom X, F(X), definiše na sledeći način:

$$A \leq_F^{'} B \Leftrightarrow LTR(B) \subseteq LTR(A) \wedge RTL(A) \subseteq RTL(B).$$

Odmah se primećuje da se dva fazi skupa sa različitim visinama ne mogu ovako porediti. Da bi se ovaj problem prevazišao uvođenjem novog poretka, definiše se skup $\overline{A}$ na sledeći način:

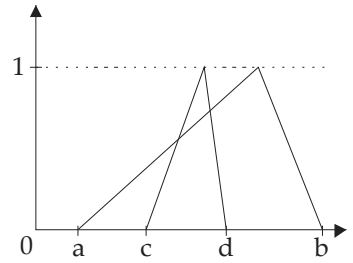
$$\mu_{\overline{A}}(x) = \begin{cases} 1, \mu_A(x) = Hgt(A) \\ \mu_A(x), \textit{inače} \end{cases}.$$

Sada se može dati definicija nove vrste poretka (Bodenhofer, 2003).

Definicija 2.19: Za proizvoljne fazi skupove A i B nad univerzumom X, poredak $\leq_F$ se uvodi na sledeći način:

$$A \leq_F B \Leftrightarrow \overline{A} \leq_F^{'} \overline{B}.$$

Poredak $\leq_F$ uređuje fazi skupove u zavisnosti od njihove horizontalne pozicije na grafiku. Što je karakteristična funkcija više desno, veći je fazi skup. Ovo poređenje je takođe samo parcijalno. Fazi skupovi koji se njime ne mogu uporediti su prikazani na slici 2.7.



Slika 2.7. Neuporedivi fazi skupovi.

Uvođenje ovakvog poretka sa stanovišta implementacije ne predstavlja naročit problem zbog toga što je jedino potrebno naći maksimum sa leve i desne strane za svaku tačku. Kada je dat fazi skup, ovo može da se uradi jednim prolazom kroz domen. Međutim, ako se ograničimo na uzak skup mogućih vrsta karakterističnih funkcija (detaljno o ovoj temi se diskutuje u poglavlju 4), onda se, za fazi skup A, LTR(A) i RTL (A) pronalaze jednostavnije. Na primer:

$$LTR(triangle(180,10,10)) = fq(170,180,inc),$$

$$RTL(triangle(180,10,10)) = fq(180,190,dec),$$

gde je sa *triangle(a,b,c)* predstavljen trougaoni fazi broj sa maksimumom u *a* i levim i desnim otklonom *b* i *c*, respektivno, a sa *fq(a,b,inc/dec)* opadajuća ili rastuća fazi veličina sa prevojima u *a* i *b*.

Koristeći poredak $\leq_F$ i pojam jednakosti fazi skupova, možemo da izvedemo relacione operatore $<_F$ i $>_F$ na isti način kao što izvodimo operatore < i > od operatora $\leq$ nad realnim brojevima.

2.2 PFCSP SISTEMI

Problemi zadovoljenja ograničenja (Constraint Satisfaction Problem), u daljem tekstu CSP, su izučavani u različitim oblastima matematike. Cilj je pronalaženje rešenja koje zadovoljava sva ograničenja za optimalno vreme (Dubois & Fortemps, 1999). U slučaju da zadovoljenje ograničenja ima istinitosnu vrednost iz intervala [0,1], odnosno, ako postoji više nivoa zadovoljenja ograničenja, postaje jasno da se u CSP može ugraditi fazi logika. Ograničenja se mogu modelirati kao fazi skupovi, te se

na taj način dobijaju fazi problemi zadovoljenja ograničenja (Fuzzy Constraint Satisfaction Problem, FCSP). U tom slučaju će mera zadovoljenja ograničenja biti vrednost karakteristične funkcije u datoj tački (Pap & Takači, 2005). Mera zadovoljenja svih ograničenja se dobija agregacijom vrednosti zadovoljenja pojedinačnih ograničenja. Agregacija se vrši pomoću operatora fazi logike.

2.2.1 CSP i FCSP

CSP se može posmatrati kao skup domena i ograničenja nad tim domenima koja su n-arne relacije nad Dekartovim proizvodom tih domena. Sledi definicija CSP-a (Bistarelli, Montanari, Rossi, Schiex, Verfaillie, & Fargier, 1999).

Definicija 2.20: Problem zadovoljenja ograničenja je uređena trojka (X,D,C), gde su:

1. Skup $X = \{x_i \mid i = 1,2,...,n\}$ je konačan skup promenljivih.
2. Skup $D = \{d_i \mid i = 1,2,...,n\}$ je konačan skup domena. Svaki domen d_i je skup koji sadrži moguće vrednosti promenljive x_i iz X.
3. Skup C je skup ograničenja:
$$C = \{R_i : (\prod_{x_j \in \text{var}(R_i)} d_j) \rightarrow \{0,1\}, i = 1,2,...,n\}$$

gde $\text{var}(R_i)$ predstavlja skup promenljivih ograničenja R_i.

Dodela vrednosti promenljivoj iz odgovarajućeg domena se naziva valuacijom promenljive. Ako uzmemo promenljivu x, njena valuacija se označava sa v_x. Kombinovanom valuacijom se naziva istovremena valuacija svih promenljivih u skupu. Na primer, ako skup X sadrži promenljive $x_1, x_2, ..., x_n$, onda kombinovanu valuaciju označavamo sa $v_X = (v_{x_1}, v_{x_2}, ..., v_{x_n})$.

Iz kombinovane valuacije se izračunava vrednost karakteristične funkcije R_i za svako ograničenje. U slučaju CSP-a, ograničenje može biti ili potpuno zadovoljeno ili potpuno nezadovoljeno.

Agregacijom pojedinačnih zadovoljenja se dobija globalno zadovoljenje ograničenja. U slučaju da želimo da sva ograničenja budu zadovoljena, kao operator agregacije koristimo konjunkciju, a u slučaju da nam treba zadovoljenje bar jednog ograničenja – disjunkciju. Takođe, zadovoljenje možemo računati kao proizvoljnu logičku formulu čiji su simboli ograničenja.

Proširenje CSP-a fazi ograničenjima otvara nove mogućnosti njihove primene (Takači, 2005). Sledi definicija FCSP-a – problema zadovoljenja ograničenja proširenog upotrebom fazi skupova i fazi logike.

Definicija 2.21: Problem zadovoljenja fazi ograničenja je uređena trojka (X, D, C^f), takva da je:

1. Skup $X = \{x_i \mid i = 1,2,...,n\}$ je konačan skup promenljivih.
2. Skup $D = \{d_i \mid i = 1,2,...,n\}$ je konačan skup domena. Svaki domen d_i je skup koji sadrži moguće vrednosti promenljive $x_i \in X$.
3. Skup C^f je skup fazi ograničenja:
$$C^f = \{R_i^f \mid \mu_{R_i^f} : (\prod_{x_j \in \text{var}(R_i^f)} d_j) \to [0,1], i = 1,2,...,n\}$$
gde je $\text{var}(R_i^f)$ skup promenljivih ograničenja R_i^f.

Potrebno je uočiti jasnu razliku između CSP i FCSP sistema. Kod CSP-a su ograničenja funkcije koje preslikavaju Dekartov proizvod jednog ili više domena iz D u skup {0,1}. Drugim rečima to su relacije nad tim domenima. U definiciji FCSP-a je skup {0,1} zamenjen intervalom [0,1]. Dakle, na neki način je dvovalentna logika zamenjena fazi logikom. Ovako su dobijene funkcije koje preslikavaju Dekartov proizvod jednog ili više domena iz D u jedinični interval, a takve funkcije se mogu posmatrati kao karakteristične funkcije fazi skupova. Dakle, ograničenja u FCSP su fazi skupovi, a mera zadovoljenja ograničenja je vrednost karakteristične funkcije fazi skupa koji opisuje to ograničenje.

Da bi se dobila globalna mera zadovoljenja ograničenja, koristi se fazi logika. U slučaju da sva ograničenja treba da budu zadovoljena, za operator agregacije $\oplus$ se koristi t-norma. Ako najmanje jedno ograničenje

treba da bude zadovoljeno, koristi se t-konorma. Slično kao kod CSP-a, dodavanjem negacije, zadovoljenje ograničenja se može računati kao proizvoljna formula fazi logike.

Ako za FCSP (X, D, C^f) zadamo kombinovanu valuaciju v_X za sve promenljive u X, tada se stepen globalnog zadovoljenja ograničenja za tu valuaciju može izraziti kao:

$$\alpha(v_X) = \oplus\{\mu_{R_i^f}(v_{\text{var}(R^f)}) \mid R^f \in C^f\},$$

gde je $\oplus$ operator agregacije. Rešenje FCSP-a je kombinovana valuacija v_X za koju važi:

$$\alpha(v_X) \geq \alpha_0,$$

gde je α_0 predefinisani prag zadovoljenja.

Ako želimo da utvrdimo da li je valuacija v_{X_1} bolje rešenje od valuacije v_{X_2} potrebno je uvesti poredak na skupu valuacija koji označavamo sa $\succ$. Najočiglednije uređenje je upotreba globalnog stepena zadovoljenja ograničenja:

$$v_{X_1} \succ v_{X_2} \Leftrightarrow \alpha(v_{X_1}) > \alpha(v_{X_2}).$$

Ovo poređenje ima svojih mana. Na primer, ako uzmemo da je $\oplus = T_M$, onda nastupa takozvani efekat urušavanja (drowning effect) koji označava da dve valuacije neće biti ispravno upoređene čak i ako su potpuno različite. Na primer $\min(0,...,0) = \min(0,1,...,1)$. Da bi se ovaj efekat ispravio mogu se uvesti drugi poreci. Neki od njih su opisani u (Fortemps & Pirlot, 2004).

Već na ovom mestu je potrebno istaći sličnosti problema zadovoljenja ograničenja sa SQL jezikom. Osnovna struktura SQL naredbe se sastoji od klauzula SELECT, FROM i WHERE. Promenljive koje se navode iza SELECT se mogu posmatrati kao promenljive CSP-a, a nazivi tabela koji se navode posle FROM kao domeni tih promenljivih. Klauzula WHERE sadrži niz ograničenja povezanih logičkim veznicima, pa se, baš kao i u CSP sistemu zadovoljenje ograničenja računa tako što se izračunaju zadovoljenja svakog pojedinog ograničenja, a zatim se izračuna istinitosna

vrednost formule. Ako CSP možemo fazifikovati i proširiti na FCSP koristeći gore opisane mehanizme, onda sledi da i SQL možemo fazifikovati i dobiti FSQL. Tada bi uslovi u WHERE klauzuli mogli biti fazi skupovi, a izračunavanje istinitosne vrednosti bi se vršilo pomoću operatora fazi logike – t-norme, t-konorme i negacije. Takođe, u tom slučaju bi istinitosna vrednost bila broj iz jediničnog intervala, pa bi torka ulazila u skup rezultata upita samo ako njena istinitosna vrednost prelazi unapred zadati prag, baš kao i kod FCSP-a. Ta ideja se u narednim poglavljima detaljno razrađuje. No, pre toga je potrebno opisati dalja proširenja problema zadovoljenja ograničenja – PFCSP i GPFCSP koja se dobijaju uvođenjem prioriteta.

2.2.2 PFCSP

Može se desiti da ograničenja nemaju isti prioritet, odnosno da su neka važnija, a neka manje važna. Zbog toga se koncept FCSP-a može proširiti uvođenjem prioriteta i na taj način se dobija Prioritised Fuzzy Constraint Satisfaction Problem ili PFCSP. Pored stepena zadovoljenja, svako ograničenje dobija stepen važnosti, odnosno prioritet. Što je ograničenje važnije, više će uticati na agregirani rezultat. PFCSP se uvodi aksiomatski (Luo, Lee, Leung, & Jennings, 2003).

Definicija 2.22: PFCSP je uređena četvorka (X, D, C^f, ρ), gde je (X, D, C^f) FCSP, a ρ je funkcija prioriteta $\rho : C^f \to [0, \infty)$.

Definicija 2.23: Neka je dat PFCSP (X, D, C^f, ρ) i kombinovana valuacija svih promenljivih iz X, v_X. Tada se $\alpha_\rho(v_X)$, dato sa

$$\alpha_\rho(v_X) = \oplus_\rho\{g(\rho(R^f), \mu_{R^f}(v_{\mathrm{var}(R^f)})) \mid R^f \in C^f\},$$

gde je $\oplus_\rho : [0,1]^n \to [0,1]$, a $g : [0, \infty) \times [0,1] \to [0,1]$, naziva globalnim stepenom zadovoljenja ako su zadovoljene sledeće osobine – aksiome:

1. Ako za fazi ograničenje $R^f_{\max}$ važi

$$\rho_{\max} = \rho_{\max}(R^f_{\max}) = \max\{\rho(R^f \mid R^f \in C^f)\},$$

tada važi i

$$\mu_{R^f_{\max}}(v_{\mathrm{var}(R^f_{\max})}) = 0 \Rightarrow \alpha_\rho(v_X) = 0\,.$$

2. Ako $\exists \rho_0 \in [0,1]$ takvo da $\forall R^f \in C^f$ važi da je $\rho(R^f) = \rho_0$, onda je globalni stepen zadovoljenja ograničenja dat sa:

$$\alpha_\rho(v_X) = \oplus_\rho\{\mu_{R^f}(v_{\mathrm{var}(R^f)}) \mid R^f \in C^f\},$$

gde je $\oplus_\rho$ t-norma.

3. Pretpostavimo da za $R^f_i, R^f_j \in C^f$ važi $\rho(R^f_i) \geq \rho(R^f_j)$, neka je dato $\delta > 0$ i neka su date dve kombinovane valuacije v_X i v'_X, takve da za $\forall R^f \in C^f$ važi:

   a. ako $R^f \neq R^f_i$ i $R^f \neq R^f_j$, onda $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = \mu_{R^f}(v'_{\mathrm{var}(R^f)})$

   b. ako $R^f = R^f_i$, onda $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = \mu_{R^f}(v'_{\mathrm{var}(R^f)}) + \delta$

   c. ako $R^f = R^f_j$, onda $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = \mu_{R^f}(v'_{\mathrm{var}(R^f)}) + \delta$.

Tada, ako važi:

$$g(\rho(R^f_i), \mu_{R^f_i}(v_{\mathrm{var}(R^f_i)})) \leq g(\rho(R^f_j), \mu_{R^f_j}(v_{\mathrm{var}(R^f_j)})),$$

onda sledi: $\alpha_\rho(v_X) \geq \alpha_\rho(v'_X)$.

4. Neka su date dve različite kombinovane valuacije v_X i v'_X sa osobinom da $\forall R^f \in C^f$ važi:

$$\mu_{R^f}(v_{\mathrm{var}(R^f)}) \geq \mu_{R^f}(v'_{\mathrm{var}(R^f)})\,.$$

   Tada je $\alpha_\rho(v_X) \geq \alpha_\rho(v'_X)$.

5. Ako postoji kombinovana valuacija v_X takva da $\forall R^f \in C^f$ važi $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = 1$, onda sledi $\alpha_\rho(v_X) = 1$.

Ovakva definicija svakako zahteva dodatna objašnjenja. Funkcija ρ predstavlja prioritet pridružen svakom ograničenju. Veća vrednost $\rho(R)$

znači da ograničenje R ima veći prioritet. Funkcija g agregira prioritet svakog ograničenja sa vrednošću toga ograničenja, odnosno stepenom njegovog zadovoljenja. Ovako agregirane vrednosti za sva ograničenja se agregiraju u globalni stepen zadovoljenja ograničenja pomoću operatora agregacije $\oplus_\rho$.

Aksiome takođe zaslužuju dodatno pojašnjenje. One u stvari tačno definišu šta to znači prioritet ograničenja i kako se on ponaša. Prva aksioma kaže da, ako ograničenje sa maksimalnim prioritetom ima stepen zadovoljenja 0, onda je i globalni stepen zadovoljenja ograničenja 0. Druga aksioma se bavi situacijom kada su svi prioriteti jednaki. Po njoj, PFCSP sistem tada prelazi u FCSP.

Treća aksioma je veoma važna, u njoj se postavlja važna osobina ovakvog poimanja prioriteta. Taj uslov se može interpretirati na sledeći način: ako jedno ograničenje ima veći prioritet, onda povećanje stepena zadovoljenja tog ograničenja rezultuje većim povećanjem globalnog stepena zadovoljenja ograničenja nego kada se za istu vrednost poveća ograničenje sa manjim prioritetom. Ovaj koncept saglasan je opštem pojmu prioriteta koji meri relativnu važnost među stvarima kako bi se odredio njihov poredak. Što je veći prioritet neke stvari, to nam je ona „draža“, tj. trebalo bi joj ranije pristupiti.

Četvrta aksioma definiše monotonost prioriteta, dok peta sadrži granične uslove.

Sledi primer PFCSP sistema (Takači & Škrbić, Data Model of FRDB with Different Data Types and PFSQL, 2008), (Takači, Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja, 2006). Neka su X, D, C^f definisani kao u definiciji 2.21, i neka je $\rho : C^f \to [0,\infty)$, $\rho_{\max} = \max\{\rho(R^f) \mid R^f \in C^f\}$ i $\rho_{norm}(R_i^f) = \frac{\rho(R_i^f)}{\rho_{\max}}, i = 1,...,n.$

Za operator agregacije $\oplus_\rho$ uzmimo T_L, a funkciju g definišimo kao $g(\rho(R_i^f), v) = S_P(1 - \rho_{norm}(R_i^f), v)$ (tzv. standardna fazi implikacija). Tada dobijamo PFCSP, a globalni stepen zadovoljenja ograničenja se računa na sledeći način:

$$\alpha_\rho(v_X) = T_L\{S_P(1-\rho_{norm}(R_i^f), \mu_{R_i^f}(v_X)) \mid R_i^f \in C^f\}.$$

Detaljna objašnjenja i dokaz da ovaj sistem zaista jeste PFCSP sistem je dat u (Takači, 2005). Ako se uzmu T_M i S_M takođe se dobija PFCSP sistem (Luo, Lee, Leung, & Jennings, 2003).

Dakle, kako u stvari radi ovaj PFCSP sistem? Normalizacijom vrednosti prioriteta dobijamo situaciju u kojoj svako ograničenje ima vrednost prioriteta iz jediničnog intervala, a ograničenje sa najvećim prioritetom ima vrednost 1. Posle normalizacije koristimo operator $S_P(1-\rho_i, v_i)$, da bi izvršili agregaciju prioriteta i mere zadovoljenja svakog ograničenja ponaosob. Što je veći prioritet ograničenja, to je veća šansa da će agregirana vrednost ostati nepromenjena. Na primer, za ograničenje sa prioritetom 1 dobijamo $S(0, v_i) = v_i$. U slučaju malog prioriteta dobijamo vrednost bližu jedinici, na primer $S(1, v_i) = 1$. Kako za izračunavanje globalnog ograničenja koristimo t-normu T_L, jasno je da će manje vrednosti (ograničenja sa većim prioritetom) imati veći uticaj na rezultat.

U disertaciji (Takači, Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja, 2006) je detaljno predstavljen strožiji koncept prioriteta koji je usvojen i upotrebljen i u ovoj disertaciji. Osim toga, GPFCSP (Generalized Prioritised Fuzzy Constraint Satisfaction Problem) sistemi čija definicija sledi u sledećoj sekciji, su bazirani na ovom strožijem konceptu. On se dobija eliminacijom uslova:

$$g(\rho(R_i^f), \mu_{R_i^f}(v_{\mathrm{var}(R_i^f)})) \le g(\rho(R_j^f), \mu_{R_j^f}(v_{\mathrm{var}(R_j^f)}))$$

iz treće aksiome u definiciji PFCSP-a. Time se dobija nova vrsta PFCSP sistema čije su osobine ispitane u pomenutoj disertaciji. Važna izmena koju donosi brisanje datog uslova iz treće aksiome je da $T_M - S_M$ sistem prestaje da zadovoljava novu definiciju. Sa druge strane, $T_L - S_P$ sistem zadovoljava i novu definiciju, pa se on nadalje i koristi. Dokazi ova dva tvrđenja se takođe mogu naći u (Takači, Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja, 2006).

2.2.3 GPFCSP

Ako fazi ograničenja interpretiramo kao uslove u WHERE klauzuli SQL jezika, kao što je sugerisano u sekciji 2.2.1, onda bi PFCSP sistemi dali formalnu osnovu za uvođenje prioriteta za svaki WHERE uslov. Problem leži u činjenici da PFCSP sistemi dozvoljavaju jedino upotrebu konjunkcije, odnosno t-norme nad ograničenjima, što znači da bi u takvom SQL jeziku mogli dozvoliti upotrebu samo logičkog operatora AND. Očigledno je potrebno generalizovati PFCSP sisteme da bi se omogućila upotreba disjunkcije i negacije. Rezultat ove generalizacije su generalizovani problemi zadovoljenja fazi ograničenja sa prioritetima ili GPFCSP (Takači, Towards Priority Based Logics, 2006). Dakle, tek GPFCSP sistemi daju formalne osnove za fazi SQL jezik sa prioritetima.

Da bi se došlo do GPFCSP sistema, potrebno je proširiti aksiome date u prethodnoj sekciji. Prvo, proširujemo PFCSP uvođenjem disjunkcije i negacije. Za disjunkciju najčešće biramo t-konormu dualnu t-normi izabranoj za konjunkciju. Za negaciju uzimamo isključivo standardnu negaciju: $N(x) = 1 - x$.

Koje izmene na PFCSP aksiomama su potrebne? Aksiome 1, 2 i 5 se odnose samo na konjunkciju ograničenja, tako da na njima nisu potrebne nikakve izmene. Aksioma 3 zahteva izmene u smislu dodavanja disjunkcije. Ovo ne predstavlja problem ukoliko se za disjunkciju izabere dualna konorma izabranoj t-normi kojom je predstavljena konjunkcija. Aksioma 4 takođe mora da se generalizuje u smislu da monotonost mora da važi jedino kada se ne upotrebljava negacija. Imajući ovo u vidu, GPFCSP se definiše u sledećoj definiciji koja se značajno razlikuje od definicije PFCSP-a zbog toga što se globalni stepen zadovoljenja ograničenja mora definisati na drugi način.

Definicija 2.24: Neka su X, D, C^f, ρ i g definisani kao u definiciji 2.23. GPFCSP je torka $(X, D, C^f, \rho, g, \wedge, \vee, \neg)$.

Elementarna formula je uređen par $(x, \rho(R_i^f))$ gde je $R_i^f \in C^f$, a $x \in Dom(R_i^f)$ je stepen zadovoljenja ograničenja R_i^f, dok je $p_i = \rho(R_i^f)$ njegov prioritet.

Formula GPFCSP-a je definisana na sledeći način:

1. Elementarna formula je formula.
2. Ako su f_1 i f_2 formule, onda su i $\wedge(f_1, f_2)$, $\vee(f_1, f_2)$ i $\neg(f_1)$ takođe formule.

Izračunavanje stepena zadovoljenja ograničenja $\alpha_F(v_X)$ za neku valuaciju v_X se izračunava u odnosu na interpretaciju veznika.

Sistem je GPFCSP ako važi:

1. Neka je $F = \wedge_{i\in\{1,\ldots,n\}} f_i$ GPFCSP formula, gde su $f_i, i \in \{1,\ldots,n\}$ elementarne formule i neka je C^F skup ograničenja koji se pojavljuju u formuli. Ako za fazi ograničenje $R^F_{\max}$ važi

   $$\rho_{\max} = \rho_{\max}(R^F_{\max}) = \max\{\rho(R^F \mid R^F \in C^F)\},$$

   tada za svaku formulu F važi i

   $$\mu_{R^F_{\max}}(v_{\mathrm{var}(R^F_{\max})}) = 0 \Rightarrow \alpha_F(v_X) = 0.$$

2. Ako $\exists\rho_0 \in [0,1]$ takvo da $\forall R^f \in C^f$ važi da je $\rho(R^f) = \rho_0$, onda je globalni stepen zadovoljenja ograničenja dat sa:

   $$\alpha_F(v_X) = F_\Lambda(v_X),$$

gde je F_Λ interpretacija logičke formule F u fazi logici $\Lambda(\wedge,\vee,\neg)$.

3. Pretpostavimo da za $R^f_i, R^f_j \in C^f$ važi $\rho(R^f_i) \geq \rho(R^f_j)$, neka je dato $\delta > 0$ i neka su date dve kombinovane valuacije v_X i v'_X, takve da za $\forall R^f \in C^f$ važi:

   a. ako $R^f \neq R^f_i$ i $R^f \neq R^f_j$, onda $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = \mu_{R^f}(v'_{\mathrm{var}(R^f)})$

   b. ako $R^f = R^f_i$, onda $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = \mu_{R^f}(v'_{\mathrm{var}(R^f)}) + \delta$

   c. ako $R^f = R^f_j$, onda $\mu_{R^f}(v_{\mathrm{var}(R^f)}) = \mu_{R^f}(v'_{\mathrm{var}(R^f)}) + \delta$.

Tada za formule $F = \wedge_{k=1,\ldots,n}(x_k, \rho(R^f_k)), x_k \in Dom(R_k)$ i $F = \vee_{k=1,\ldots,n}(x_k, \rho(R^f_k)), x_k \in Dom(R_k)$ važi $\alpha_F(v_X) \geq \alpha_F(v'_X)$.

4. Neka su date dve različite kombinovane valuacije v_X i $v_X^{'}$ sa osobinom da $\forall R^f \in C^f$ važi:

$$\mu_{R^f}(v_{\text{var}(R^f)}) \geq \mu_{R^f}(v^{'}_{\text{var}(R^f)}).$$

Tada, ako formula F ne sadrži negaciju važi $\alpha_F(v_X) \geq \alpha_F(v_X^{'})$.

5. Neka postoji kombinovana valuacija v_X takva da $\forall R^F \in C^F$ važi $\mu_{R^F}(v_{\text{var}(R^F)}) = 1$. Ako je F formula oblika $F = \wedge_{i\in\{1,\ldots,n\}} f_i$ ili $F = \vee_{i\in\{1,\ldots,n\}} f_i$, gde su $f_i, i \in \{1,\ldots,n\}$ elementarne formule, onda sledi $\alpha_F(v_X) = 1$.

Pored rada koji uvodi GPFCSP, (Takači, Towards Priority Based Logics, 2006), neke od primena ovih sistema su prikazane u (Takači & Škrbić, 2008), dok se (Takači, Perović, & Jovanović, 2008) bavi konzistentnošću i odlučivošću GPFCSP sistema. Sledi primer jednog GPFCSP sistema koji se uvodi kao teorema.

Teorema 2.1: Sledeći sistem $(X, D, C^f, \rho, g, \wedge, \vee, \neg, \Diamond)$, gde je $\wedge = T_L$, $\vee = S_L$, $\neg = N_S$ i, konačno, $\Diamond(x_i, p_i) = S_P(x_i, 1-p_i)$ je GPFCSP. Globalni stepen zadovoljenja ograničenja valuacije v_X za formulu F se računa na sledeći način:

$$\alpha_F(v_X) = \Phi\{\Diamond(v_{x_i}) \frac{\rho(R_i^F)}{\rho_{\max}}\} \mid R^F \in C^F\}$$

gde je C^F skup ograničenja formule F, $\rho_{\max} = \max\{\rho(R_i^F), R^F \in C^F\}$, a Φ je interpretacija formule F u GPFCSP.

Dokazom ove teoreme se bavi (Takači, Towards Priority Based Logics, 2006).

Izračunavanje globalnog stepena zadovoljenja ograničenja za datu valuaciju će sada biti prikazano na primeru. Takođe, svrha ovog primera je da se jasnije uoči sličnost između GPFCSP sistema i SELECT upita.

Uzmimo kao primer skup studenata. Neka je naš zadatak da zaposlimo najboljeg od njih kao asistenta, tako što ćemo ih rangirati po tri kriterijuma – starosti, ocene dobijene na lekarskom pregledu i broja poena

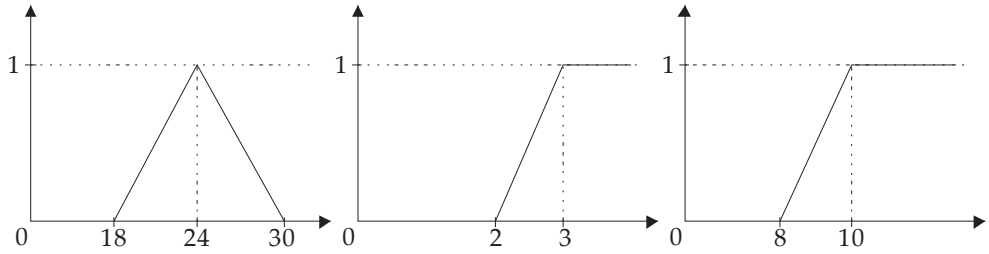
osvojenih na testu. Imali bismo, dakle, sledeći skup promenljivih i njihovih domena:

1. $X_1 =$ godine, $d_1 = [0,150]$,
2. $X_2 =$ ocena na pregledu, $d_3 = [0,5]$,
3. $X_3 =$ poeni na testu, $d_2 = [0,10]$.

Neka skup ograničenja bude sledeći:

1. $R_1^f =$ "oko 24 godine",
2. $R_2^f =$ "dobro zdravlje",
3. $R_3^f =$ "odlični rezultati na testu".

Ograničenja su fazi podskupovi odgovarajućih domena. Modelirajmo prvo ograničenje kao trougaoni fazi broj čiji je centar u broju 24, sa levom i desnom tolerancijom 6. Neka drugo ograničenje bude modelirano kao rastuća fazi veličina koja raste od 2 do 3. Na kraju, neka treće ograničenje bude rastuća fazi veličina koja raste od 8 do 10. Ova tri fazi skupa su prikazana na slici 2.8.



Slika 2.8. Karakteristične funkcije ograničenja R_1^f, R_2^f i R_3^f.

Neka kriterijum za ocenjivanje kandidata (formula F) bude „student mora da bude oko 24 godine i da ima dobro zdravlje ili odlične rezultate na testu" i neka su prioriteti dati sa $\rho(R_1^f) = 0.7$, $\rho(R_2^f) = 0.4$ i $\rho(R_3^f) = 1$. Dakle, najvažniji nam je broj poena na testu, pa zatim godine starosti i na kraju zdravlje. Svaki od četiri kandidata je prikazan valuacijom v_X, koja je data u tabeli 2.1.

student	*godina*	*zdravlje*	*test*
1	25	3	9.8
2	23	4	9.2
3	24	2	9.5
4	28	5	9.4

Tabela 2.1. Testni podaci za GPFCSP.

Prvo je potrebno izračunati stepen zadovoljenja ograničenja za svako ograničenje, za svakog studenta. Ovi stepeni zadovoljenja ograničenja se dobijaju direktno kao vrednosti pojedinih karakterističnih funkcija u odgovarajućoj tački. Rezultati za svakog studenta i za svako ograničenje su dati u tabeli 2.2.

student	*godište*	*zdravlje*	*test*
1	0.833	1	0.9
2	0.833	1	0.6
3	1	0	0.75
4	0.333	1	0.7

Tabela 2.2. Vrednosti karakterističnih funkcija svakog ograničenja za svakog studenta.

Sada izračunavamo vrednost globalnog zadovoljenja ograničenja za svakog studenta ponaosob na sledeći način:

$$\alpha = S_L(T_L(S_P(\mu_{R_1^f}(v),1-\rho(R_1^f)),S_P(\mu_{R_2^f}(v),1-\rho(R_2^f))),S_P(\mu_{R_3^f}(v),1-\rho(R_3^f))).$$

Ako se uvrste vrednosti za prvog studenta, dobija se sledeći izraz:

$$\alpha = S_L(T_L(S_P(0.833,0.3),S_P(1,0.6)),S_P(0.9,0)).$$

Daljim računom se dobija:

$$\alpha = S_L(T_L(0.8831,1),0.9) = S_L(0.8831,0.9) = 1.$$

Vrednosti za sve studente su date u tabeli 2.3. Dakle, po usvojenom kriterijumu, prva tri studenta u potpunosti zadovoljavaju postavljene kriterijume, dok ih četvrti student ne zadovoljava u potpunosti, nego sa pragom zadovoljenja 0.9831.

student	α
1	1
2	1
3	1
4	0.9831

Tabela 2.3. Stepeni globalnog zadovoljenja ograničenja za sve studente.

Primetimo da bi se ovaj primer mogao dovesti u vezu sa sledećim upitom:

```
SELECT * FROM studenti
WHERE starost ="oko 24 godine" PRIORITY 0.7 AND
      zdravlje="dobro zdravlje" PRIORITY 0.4 OR
      test ="odlični rezultati" PRIORITY
```

Poglavlje 3

FAZI BAZE PODATAKA

Nemogućnost modeliranja nejasnih i nepotpunih podataka se u nekim primenama može posmatrati kao nedostatak relacionog modela. Ideja da se upotrebe fazi skupovi i fazi logika u svrhe proširivanja postojećih modela podataka, da bi se ti nedostaci prevazišli nije nova i potiče iz osamdesetih godina prošlog veka. Iako se ova oblast istražuje duže vreme, konkretne implementacije su veoma retke. U literaturi se mogu naći nekoliko modela reprezentacija fazi znanja u relacionom, ali i u drugim modelima podataka.

U ovom poglavlju će biti prezentovana najpre izabrana ranija istraživanja u ovoj oblasti, a zatim i najnoviji rezultati. Potom će biti dat pregled istraživanja vezanih za upotrebu fazi logike u relacionim bazama podataka koja se sprovode na Univerzitetu u Novom Sadu. Najširi pregled različitih pravaca istraživanja u ovoj oblasti je dat u (Galindo, Urrutia, & Piattini, 2006).

3.1 PREGLED ISTRAŽIVANJA U OBLASTI FAZI BAZA PODATAKA

Jedan od najranijih radova na ovu temu, Buckles-Petry model (Buckles & Petry, 1982), uvodi relacije sličnosti (Zadeh, 1971) u relacioni model. U ovom radu je data struktura za predstavljanje nejasnih informacija u relacionim bazama podataka. Struktura se razlikuje od osnovnog relacionog modela u dva aspekta. Prvo, komponente torki ne moraju biti atomarne vrednosti i drugo, relacija sličnosti se zahteva za svaki domen. Tipovi podataka koje ovaj model dozvoljava su:

- konačan skup skalara,
- konačan skup brojeva i

- fazi brojevi.

Prvi pokušaj uvođenja fazi logike u Enitity-Relationship model je dat u (Zvieli & Chen, 1986). Ovaj model uvodi fazi atribute u entitete i poveznike. Definišu se tri nivoa fazifikacije u ER modelu. Na prvom nivou, skupovi entiteta, skupovi poveznika i skupovi atributa mogu da imaju fazi vrednosti, tj. imaju stepen pripadnosti nekom određenom modelu. Drugi nivo je povezan sa fazi pojavama skupova entiteta i skupova poveznika i opažanju koje pojave pripadaju tim skupovima i sa kojim stepenom pripadanja. Na primer, entitet „Mladi_Zaposleni" može da ima pojave – zaposlene, koji mu pripadaju svaki sa svojim stepenom pripadanja. Konačno, treći nivo se bavi fazi vrednostima atributa specijalnih skupova entiteta i poveznika.

Rad (Chaudhry, Moyne, & Rundensteiner, 1994) daje predlog metodologije dizajna fazi relacionih baza podataka koji se oslanja na proširenje ER modela koje su dali Zvieli i Chen. Ova metodologija sadrži proširenja za reprezentaciju nepreciznosti u ER modelu i skup koraka koje je potrebno preduzeti da bi se od takvog ER modela dobila baza podataka. Takođe, obrađuje se mogućnost konvertovanja običnih u fazi baze podataka.

Chen, G. u svojoj knjizi (Chen, 1998) obrađuje osnovne principe modeliranja fazi podataka, načine za reprezentaciju fazi znanja i opisuje tipove fazi ograničenja integriteta. U istom izvoru se može naći predlog vrlo detaljnog fazi ER modela čiji opis izlazi van okvira ove disertacije. Taj model se proširuje u (Chen & Kerre, 1998) i (Kerre & Chen, 2000) uvođenjem fazi ekstenzija proširenog ER modela (EER model). Fazi logika je primenjena na osnovne elemente proširenog ER modela povezane sa pojmom generalizacije, odnosno, potklase i super klase.

GEFRED (Generalized Model of Fuzzy Relational Databases) model (Medina, Pons, & Vila, 1994) je posibilistički model koji uključuje generalizovane fazi domene i raspodelu verovatnoće u domenima. Ovaj fazi relacioni model sadrži mogućnosti za reprezentaciju širokog spektra fazi informacija. Takođe, u okviru njega se definišu fleksibilni mehanizmi za obradu takvih informacija. Ovaj model sadrži i razlikuje koncepte nepoznatog, nedefinisanog i nula vrednosti koji su ranije opisani u (Umano

& Fukami, 1994). GEFRED model je doživeo kasnija proširenja koja su opisana u (Galindo, Medina, & Aranda, 1999) i (Galindo J. , Medina, Cubero, & Garcia, 2001).

Prvi predlog proširenja SQL jezika u pravcu fazi logike su predložili Bosc i Pivert. U seriji radova u kojoj se ističu (Bosc & Pivert, 1994) i (Bosc & Pivert, 1995), oni definišu SQLf, jezik sa mogućnošću postavljanja „nejasnih" upita dobijen fazifikacijom SQL-a. Ovaj jezik je prvobitno namenjen za postavljanje upita nad običnim, relacionim bazama podataka u kojima nema fazi informacija. Međutim, u sledećem stadijumu razvoja je dat i okvir za reprezentaciju nepreciznih i nesigurnih informacija pomoću fazi logike, a SQLf je proširen da bi podržao upite nad ovakvim bazama podataka.

Grupa španskih naučnika na čelu sa dr Jose Galindom se bavi upotrebom fazi logike u bazama podataka duži niz godina. Neki od radova nastalih u okviru ovih istraživanja su već pomenuti gore kod GEFRED modela. U daljim radovima (Galindo J. , Medina, Pons, & Cubero, 1998), (Galindo, Urrutia, & Piattini, 2004) i (Galindo, Urrutia, & Piattini, 2006), je detaljno opisana nova varijanta proširenja EER modela pomoću fazi logike, FuzzyEER model, koja uključuje nove semantičke aspekte. Takođe, opisane su mogućnosti upotrebe ovog modela u modeliranju baze podataka i reprezentaciji fazi informacija.

Nadalje, u okviru ovih istraživanja je dat algoritam za prevođenje FuzzyEER modela u FIRST-2 – model za reprezentaciju fazi informacija u relacionoj bazi podataka. FIRST-2 šema uvodi koncept baze fazi meta podataka (Fuzzy Metaknowledge Base, FMB). Za svaku vrstu atributa, ovaj model definiše način reprezentacije njegovih vrednosti i informacija o tom atributu u bazi fuzzy meta podataka.

Takođe, autori uvode i daju detaljan opis specifikacije i implementacije FSQL-a, proširenja SQL jezika fazi mogućnostima. Ovaj jezik omogućava izvršavanje fleksibilnih upita koristeći sva proširenja definisana u FuzzyEER i FIRST-2 modelima. Implementacija je urađena u Oracle sistemu za upravljanje bazama podataka upotrebom PL/SQL uskladištenih procedura. Na taj način je postojeći moćan SQL interpreter Oracle baze podataka proširen novim mogućnostima.

Može se zaključiti da dostignuti stepen razvoja u ovoj oblasti uključuje zrela fazi proširenja EER modela koja opisuju širok spektar koncepata za modeliranje fazi baza podataka. Ovi konceptualni modeli su podržani robusnim modelima za reprezentaciju fazi informacija u relacionim bazama podataka. Jedan od takvih je svakako pomenuti FIRST-2. Načini za prevođenje konceptualnih modela u one zasnovane na relacionom modelu su takođe detaljno proučeni. Implementacije i primeri upotrebe u praksi su, međutim, retki. FSQL jezik opisan u ovoj sekciji predstavlja praktično prvu sveobuhvatnu implementaciju upitnog jezika za fazi baze podataka koji uključuje najveći broj koncepata fazi logike.

3.2 RAZVOJ IDEJE O UVOĐENJU KONCEPTA PRIORITETA

Ova doktorska disertacija je deo višegodišnjih istraživanja vezanih za mogućnosti upotrebe fazi logike u relacionim bazama podataka koja se sprovode na Univerzitetu u Novom Sadu. U ovoj sekciji se opisuju razvijene ideje i postignuti rezultati u okviru ovih istraživanja.

Osnovna ideja je već nagoveštena, a to je mogućnost iskorišćavanja sličnosti između CSP sistema i SELECT klauzule SQL jezika. Kako su CSP sistemi dobro istraženi, oni predstavljaju formalnu osnovu za razvoj fazi proširenja SQL jezika. Izabrana je specijalna vrsta ovih sistema, PFCSP sistemi, koji su prvi put uvedeni i opisani u (Luo, Lee, Leung, & Jennings, 2003). Dalji razvoj ovih sistema u smeru u kome su oni kasnije upotrebljeni kao osnova za fazi proširenja SQL-a je dat u (Takači, 2005) i u (Pap & Takači, 2005).

Ideja o upotrebi PFCSP sistema u proširivanju SELECT klauzule SQL jezika fazi mehanizmima se prvi put opisuje u (Takači & Škrbić, 2005). Ovde se prikazuju samo pojedini tehnički aspekti implementacije prototipa fazi SQL interpretera koja je tada još bila u toku. Dalja razrada ovih ideja je data u (Takači, Handling Priority Within a Database Scenario, 2006).

U doktorskoj disertaciji (Takači, Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja, 2006) su detaljno opisane trougaone norme i njihova upotreba u PFCSP sistemima. Opisan je i realizovan prototip interpretera prvobitne verzije jezika PFSQL čija se gramatika ovde dalje razrađuje. Za potrebe realizacije ovog

interpretera je konstruisan i opisan jednostavan prototip modela za skladištenje fazi informacija u relacionu bazu podataka. U ovom radu su postavljene smernice za dalji tok istraživanja koje su i realizovane u kasnijim radovima. Treba napomenuti da se ovde radi o pojednostavljenoj verziji PFSQL-a koja sadrži samo minimum proširenja konstrukcije SELECT klauzule. Takođe, GPFCSP sistemi ovde još nisu bili uvedeni, tako da je ova verzija PFSQL jezika bazirana na PFCSP sistemima, što znači da od logičkih veznika dozvoljava samo konjunkciju. Bez obzira na to, ovaj rad daje osnovu za čitav dalji tok istraživanja.

Uvođenje GPFCSP sistema kao proširenja PFCSP sistema je prvi put dato u (Takači, Towards Priority Based Logics, 2006). Razvoj ove nove ideje, opis osobina GPFCSP sistema i njihovo poređenje sa drugim sistemima se dalje mogu pratiti u (Takači & Škrbić, Comparing Priority, Weighted and Queries with Threshold in PFSQL, 2007), (Takači & Škrbić, Measuring the Similarity of Different Types of Fuzzy Sets in FRDB, 2007), (Takači & Škrbić, Short Review of Fuzzy Relational Databases, 2007) i (Takači, Perović, & Jovanović, 2008).

Sva dotadašnja istraživanja i rezultati su sumirani i detaljno opisani u (Takači & Škrbić, Data Model of FRDB with Different Data Types and PFSQL, 2008). Osim opisa svih dotada uvedenih koncepata, u ovom radu su razmatrane i do detalja razrađene mogućnosti za uvođenje poretka fazi skupova u novu, proširenu verziju jezika PFSQL. Uvođenje poretka otvara mogućnosti za značajna proširenja ovog jezika. Ova proširenja su uvedena, detaljno opisana i implementirana u ovoj disertaciji.

Restrukturiranje interpretera PFSQL jezika u smeru njegove kasnije realizacije u ovoj doktorskoj disertaciji, kao i uvođenje ideje o realizaciji fazi JDBC drajvera su dati u (Škrbić & Takači, On Development of Fuzzy Relational Database Applications, 2008) i (Takači & Škrbić, Priority, Weight and Threshold in Fuzzy SQL Systems, 2008). Ovde je opisana osnovna struktura fazi JDBC drajvera koja je dalje razrađena i implementirana u ovoj disertaciji.

Poglavlje 4

SKLADIŠTENJE FAZI PODATAKA

U ovom poglavlju se daje detaljan opis projektovanog modela za skladištenje fazi podataka pomoću relacione baze podataka. Osnovna namena tog modela i razlog njegovog uvođenja je implementacija jezika PFSQL opisana u sledećem poglavlju. Osim opisa modela, daje se i njegovo poređenje sa njegovom prethodnom varijantom detaljno opisanom u (Takači & Škrbić, Data Model of FRDB with Different Data Types and PFSQL, 2008) i sa jednim od najnaprednijih modela danas - FIRST-2 modelom.

4.1 OSOBINE MODELA ZA SKLADIŠTENJE FAZI PODATAKA

Model za skladištenje fazi podataka opisan ovde skladišti precizno zadatu vrednost (*engleski crisp*) bez izmene u odnosu na relacioni model, dok se za fazi vrednosti definišu određena proširenja relacionog modela. Kao što je ranije istaknuto, karakteristična funkcija nekog fazi skupa ga u potpunosti određuje. Dakle, ako želimo da skladištimo neku fazi vrednost, moramo imati načina da skladištimo podatke o njenoj karakterističnoj funkciji. Teoretski, na ovaj način bi se mogla skladištiti bilo koja fazi vrednost. Međutim, ovo bi vodilo ka velikoj složenosti baze podataka već kod samog skladištenja. Zbog toga se uvodi mogućnost upotrebe samo nekih dobro poznatih vrsta fazi skupova. U praksi, ove vrste fazi skupova pokrivaju ogroman procenat praktičnih primena.

Ako se ograničimo samo na neke vrste karakterističnih funkcija, onda je u bazi podataka dovoljno napraviti po jednu tabelu koja omogućava skladištenje parametara za svaki konkretan tip funkcije. Vrste

karakterističnih funkcija koje ovaj model podržava opisuju sledeće vrste fazi skupova:

1. trougaoni fazi brojevi,
2. trapezoidni fazi brojevi,
3. opadajuće i rastuće fazi veličine i
4. intervali.

Osim ovih proširenja, uvodi se još jedna vrsta proširenja vrednosti atributa – lingvističke labele. One služe za zadavanje najčešćih i najšire korišćenih izraza prirodnog jezika, kao što su „visoki ljudi", „male plate" ili „osrednji rezultati". Lingvističke labele su, u stvari, imenovane fazi vrednosti iz nekog domena. Da bismo koristili lingvističku labelu, prvo je moramo definisati kao konkretnu fazi vrednost. Na primer, možemo definisati lingvističku labelu „visoki ljudi" kao rastuću fazi veličinu koja raste od 185 do 200. Dakle, striktno gledano, lingvističke labele nisu novi tip fazi vrednosti, već samo imenovane vrednosti već uvedenih vrsta fazi skupova. Međutim, sa stanovišta baze podataka i one moraju dobiti odgovarajuću reprezentaciju u modelu.

Imajući u vidu ova proširenja, možemo definisati domen fazi atributa na sledeći način:

$$D = D_C \cup F_D \cup L_L$$

Gde je D_C klasičan domen atributa, F_D je skup svih fazi podskupova domena koji se mogu zadati pomoću navedene četiri vrste fazi skupova, dok je L_L skup lingvističkih labela.

Da bismo ovakve podatke skladištili u relacionu bazu podataka, potrebno je da je proširimo fazi meta modelom koji čini centralni deo modela za skladištenje fazi podataka koji se ovde uvodi. Struktura fazi meta modela i načina povezivanja ovog segmenta baze podataka sa ostalim tabelama će biti prikazana na primeru relacionog modela baze podataka prikazanom na slici 4.1. Prvo će biti opisan ovaj relacioni model, a zatim će biti prikazana njegova fazifikacija, odnosno uvođenje fazi atributa i njihovo povezivanje sa fazi meta modelom. Radi se o relacionom modelu baze podataka dela informacionog sistema studentske služba Prirodno-

matematičkog fakulteta u Novom Sadu koji se odnosi na obradu konkursa za upis studenata i polaganje prijemnog ispita.

4.2 POLAZNI RELACIONI MODEL

Centralna tabela u modelu datom na slici 4.1 je *PrijemniGlPodaci* koja sadrži najvažnije podatke o kandidatu. Ovi podaci su dopunjeni dodatnim podacima koji se nalaze u tabeli *PrijemniPrijava*. Pomoćna tabela *StraniJezik* sadrži podatke o stranim jezicima koje je kandidat učio u ranijim školama.

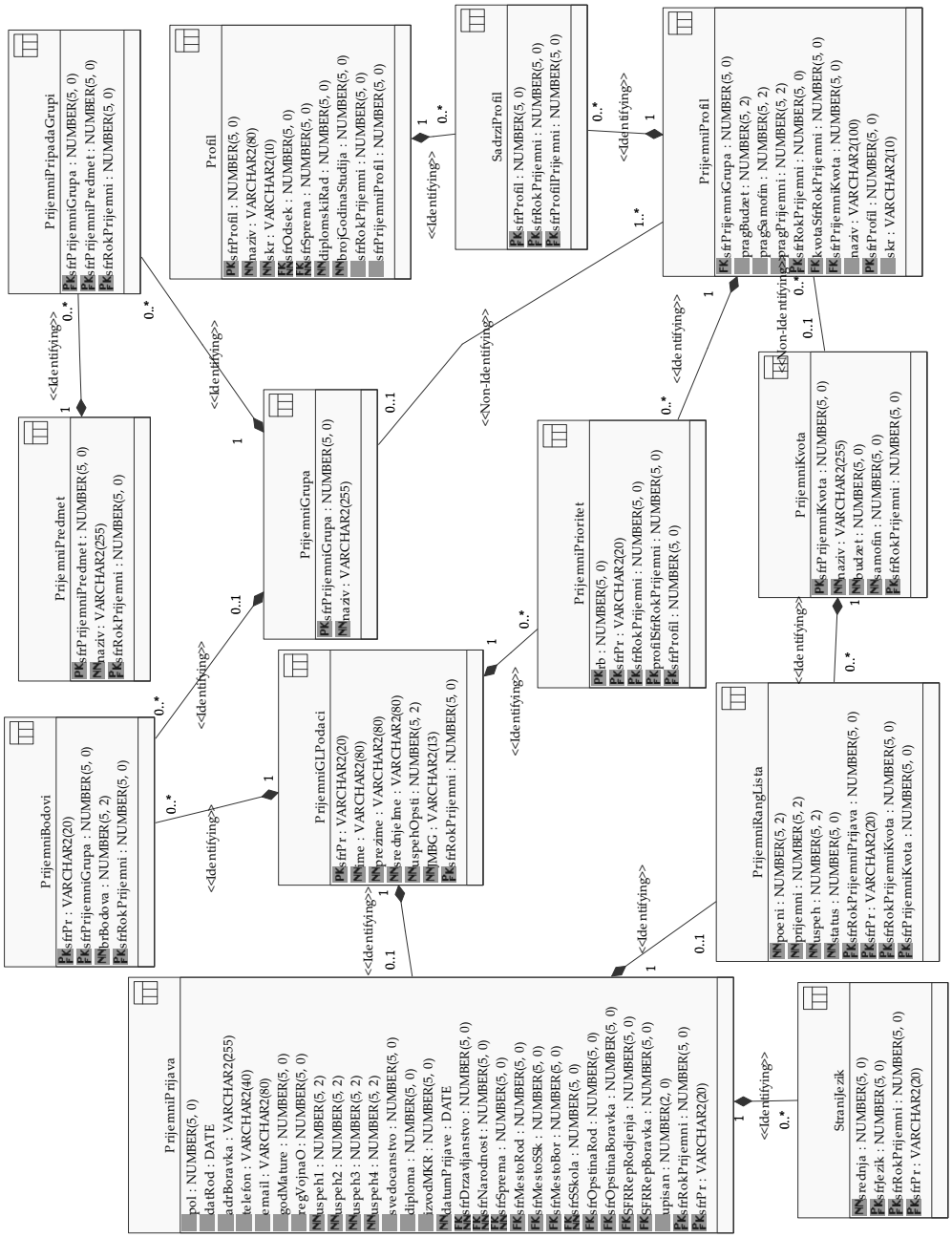
Tabela *PrijemniProfil* sadrži podatke o profilima na koje se kandidati prijavljuju. Ovi profili ne moraju biti jednaki onima koji se kasnije i upisuju, a koji su predstavljeni tabelom *Profil*. Na primer, kandidat može da odabere prijemni-profil „Matematika“, a da se kasnije opredeli da upiše profil „Diplomirani matematičar“ koji je sadržan u prijemni-profilu „Matematika“.

Tabela *SadrziProfil* skladišti podatke o tome koji profil sadrži koje prijemni-profile i obrnuto. Svaki prijemni-profil ima svoju kvotu, odnosno broj mesta za finansiranje iz budžeta i za samofinansiranje. Ovi podaci se nalaze u tabeli *PrijemniKvota*. Tabela *PrijemniPrioritet* sadrži prijemni-profile na koje kandidat konkuriše poslagane po prioritetu.

Za svaki prijemni-profil je definisana grupa predmeta iz kojih student polaže prijemni ispit (tabela *PrijemniGrupa*). Podaci o predmetima koje ove grupe sadrže se skladište pomoću tabela *PrijemniPredmet* i *PrijemniPripadaGrupi*. Tabela *PrijemniBodovi* sadrži podatke o ukupnom broju poena koje je kandidat osvojio za svaku od grupa predmeta koje je polagao.

PrijemniRangLista je tabela koja sadrži podatke o rangiranju kandidata po svakom profilu na koji se studenti prijavljuju u svakom konkursnom roku. Ovi podaci se generišu automatski po unosu svih podataka o prijemnim ispitima i skladište u ovu tabelu za dalju upotrebu.

Prikazani model nije potpun, on sadrži još nekoliko tabela koje predstavljaju šifarnike za mesta, države, škole, konkursne rokove itd. Da bi se dobilo na jasnoći, detalji o ovim tabelama nisu prikazani pošto one ne učestvuju u fazifikaciji.



Slika 4.1. Relacioni model baze podataka – studentska služba PMF.

4.3 FAZIFIKACIJA I FAZI META MODEL

Fazifikacija prikazanog modela uključuje dodavanje tabela fazi meta modela, a zatim i izbor obeležja koja će se fazifikovati i njihovo povezivanje sa fazi meta modelom. Fazi-relacioni model za skladištenje podataka koji se opisuje ovde uključuje jedino mogućnosti za fazifikaciju obeležja čiji je tip podataka broj (ceo ili racionalan). Iako je ovo ograničenje karakteristično i za druge fazi-relacione modele koji se mogu naći, ono može biti prevaziđeno. Dakle, moguća je fazifikacija i drugih tipova podataka, kao što su datumi, pa čak i stringovi. Ovakve varijante modela su u razmatranju i izlaze van okvira ove disertacije.

Kompletan fazi-relacioni model je zbog svog obima podeljen na dve slike. Prva, slika 4.2, prikazuje model sličan čisto relacionom modelu sa slike 4.1 na kome su osenčena obeležja koja postaju fazi skupovi. To su sledeća obeležja:

1. uspeh1, uspeh2, uspeh3, uspeh4 iz tabele *PrijemniPrijava* koja sadrže prosek ocena kandidata za svaki razred srednje škole,
2. uspehOpsti iz tabele *PrijemniGLPodaci*, koje skladišti prosečan uspeh u sva četiri razreda srednje škole,
3. bodovi iz tabele *PrijemniBodovi*, koje sadrži broj bodova koje je kandidat osvojio na prijemnom ispitu iz date grupe predmeta,
4. poeni, prijemni i uspeh iz tabele *PrijemniRangLista*, koja sadrže ukupne poene, kao i poene na prijemnom ispitu i opšti uspeh koji se nalaze na rang listi,
5. budzet i samofin iz tabele *PrijemniKvota*, koja sadrže broj budžetskih i samofinansirajućih mesta za datu kvotu i
6. pragBudzet, pragSamoFin i pragPrijemni iz tabele *PrijemniProfil*, koja sadrže minimalan potreban broj poena za studiranje na budžetu, samofinansiranju i minimalan broj potrebnih poena na prijemnom ispitu, respektivno.

Tip podataka ovih obeležja je izmenjen u NUMBER(10,0) , što je ceo broj sa najviše 10 cifara. Suština je u tome da su ovi brojevi samo ključevi koji povezuju ova obeležja sa fazi meta modelom. Konkretne vrednosti ovih obeležja su opisane fazi meta modelom i ne nalaze se u matičnim tabelama. Detaljniji opis načina reprezentacije fazi skupova i povezivanja

ovih obeležja sa vrednostima u fazi meta modelu se daje na osnovu slike 4.3 koja sadrži fazi meta model.

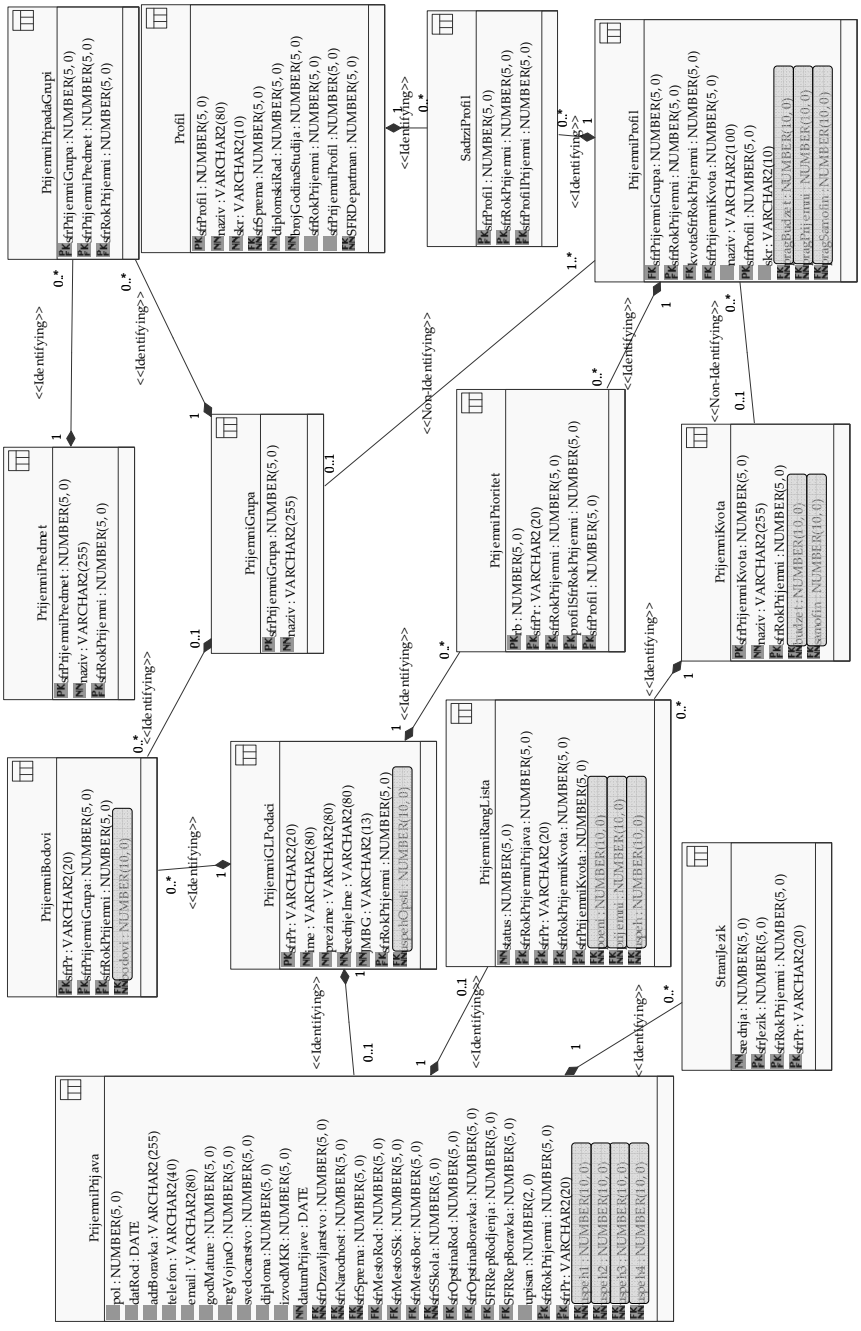
Veze između ova dva modela su ostvarene prevlačenjem primarnog ključa v*alueID* centralne tabele u fazi meta modelu – *FuzzyValue* za svako od obeležja koje postaje fazi. Kardinaliteti ovih veza su 1-1. Dakle, svako osenčeno obeležje na slici 4.2 je strani ključ koji potiče iz tabele *FuzzyValue*. Na taj način je obezbeđeno da svaka pojava nekog fazi obeležja bude povezana sa tačno jednom pojavom u tabeli *FuzzyValue*. Dakle, tabela *FuzzyValue* predstavlja vezu između fazifikovanog relacionog modela i fazi meta modela. U njoj se skladišti po jedan slog za svaku komponentu svakog sloga u bazi podataka koji sadrži vrednosti označene kao fazi.

Obeležje *code* u tabeli *FuzzyValue* je strani ključ koji potiče iz tabele *FuzzyType*. Ova tabela skladišti nazive svih vrsta fazi skupova dozvoljenih u modelu. U skladu sa prethodno iznetim, to su sledeće vrednosti:

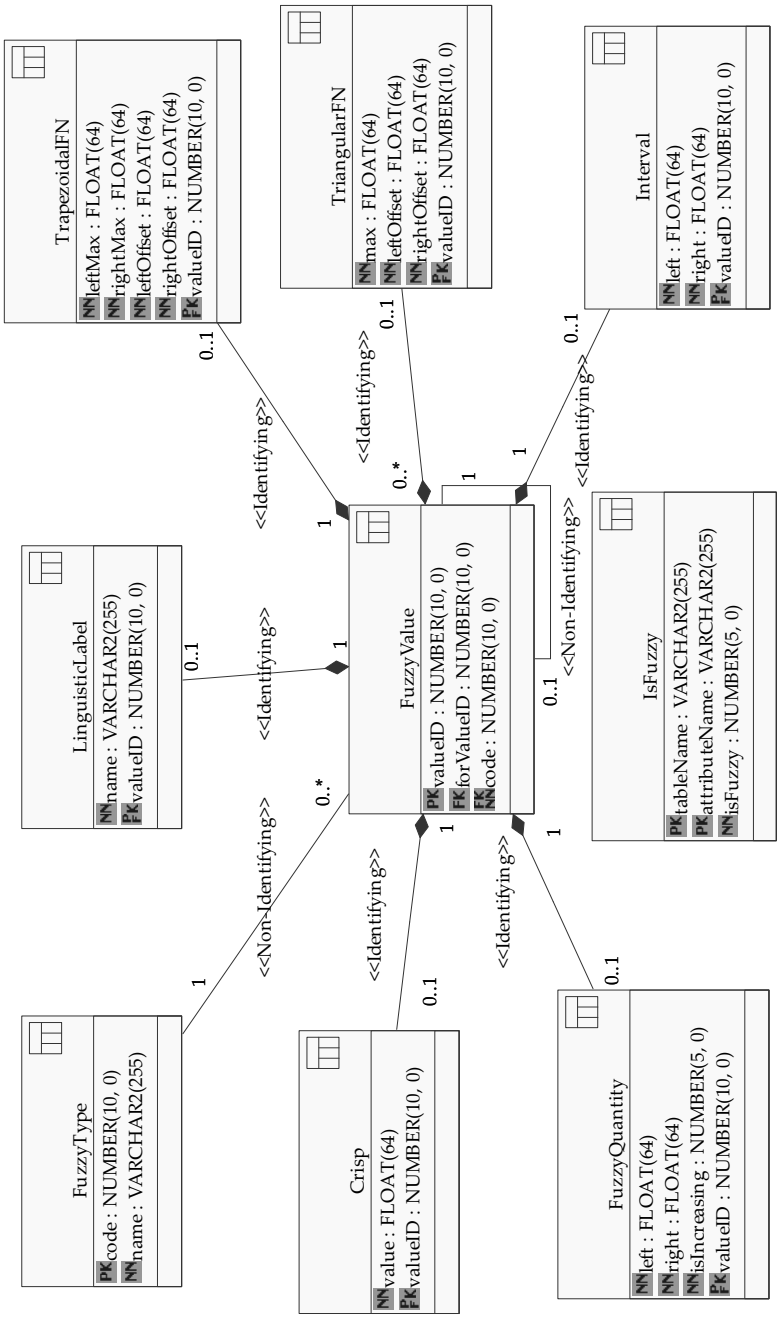
1. „Crisp“, obični brojevi,
2. „TriangularFN“ – trougaoni fazi brojevi,
3. „TrapezoidalFN“ – trapezoidni fazi brojevi,
4. „FuzzyQuantity“ – fazi veličine,
5. „Interval“ – intervali predstavljeni kao fazi skupovi, i
6. „LinguisticLabel“ – lingvističke labele.

Za svaku vrednost u ovoj listi postoji posebna tabela u meta modelu prilagođena skladištenju fazi vrednosti datog tipa. Nazivi tih tabela su jednaki nabrojanim vrednostima u tabeli *FuzzyType*. Svaka od ovih tabela sadrži obeležje *valueID*, strani ključ koji potiče iz tabele *FuzzyValue*. Na ovaj način je omogućeno da se za svaki fazi tip podataka, konkretne vrednosti skladište u tabeli predviđenoj za taj tip.

U fazi meta modelu postoji jedna dodatna tabela, *IsFuzzy*, bez koje ovakav mehanizam ne bi funkcionisao, ili bi imao neprihvatljive performanse. Ova tabela jednostavno skladišti nazive obeležja za svaku tabelu u fazifikovanom relacionom modelu. Za svaku kombinaciju ime tabele-naziv obeležja, ovde je data vrednost atributa *isFuzzy* koja govori da li je atribut fazi ili ne.



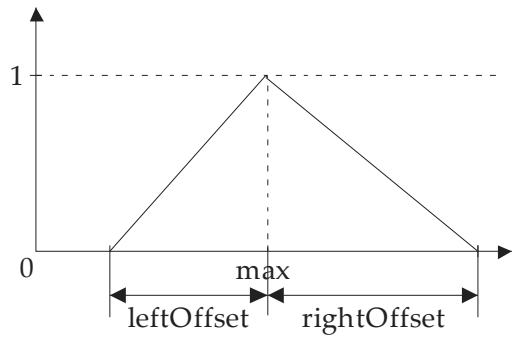
Slika 4.2. Fazifikacija relacionog modela.



Slika 4.3. Fazi meta model.

Posmatrajmo primer obeležja *uspehOpsti* iz tabele *PrijemniGlPodaci*. Neka u nekom konkretnom slogu koji pripada ovoj tabeli za vrednost atributa *uspehOpsti* imamo broj 50. Prvo je potrebno pronaći podatak o tome da li je ovo obeležje fazi ili ne u tabeli *isFuzzy*. U slučaju da je odgovor negativan, vrednost 50 je istovremeno i vrednost tog obeležja u datom slogu. U slučaju da je odgovor pozitivan, potrebno je konsultovati fazi meta model da bi se našla vrednost ovog fazi obeležja. Tada je potrebno naći slog u tabeli *FuzzyType* koji ima vrednost 50 za obeležje *valueID*. Vrednost atributa *code* govori o tipu fazi podatka o kojem se radi, pa se u odnosu na ovu vrednost pretražuje tabela koja skladišti taj tip podataka u potrazi za vrednošću 50 njenog obeležja *valueID*. Pronađeni slog u odgovarajućoj tabeli, sadrži podatke o traženom fazi skupu za taj specifični fazi tip podataka.

Opišimo način skladištenja vrednosti za fazi tipove podataka na primeru trougaonog fazi broja. Kao što je rečeno, u konkretnoj bazi podataka su sve vrednosti fazi tipa trougaoni fazi broj skladištene u tabeli *TriangularFN*. Obeležje *max* ove tabele je vrednost u kome karakteristična funkcija ovog trougaonog fazi broja dostiže maksimum. Obeležja *leftOffset* i *rightOffset* označavaju udaljenost, sa leve i desne strane, tačaka gde karakteristična funkcija dostiže minimum od tačke gde dostiže maksimum. Dakle, za vrednosti manje od *max-leftOffset* i veće od *max+rightOffset*, karakteristična funkcija ima vrednost 0 (slika 4.4).



Slika 4.4. Trougaoni fazi broj.

Vrednost obeležja *valueID*, kao što je ranije opisano, pokazuje na slog tabele *FuzzyValue* koji referencira ovaj slog u tabeli *TriangularFN*.

Tabela 4.1 sadrži nekoliko slogova tabele *TriangularFN*, dok tabela 4.2 sadrži njihove odgovarajuće slogove iz tabele *FuzzyValue*. Vrednost 2 obeležja *code* označava da se radi o trougaonom fazi broju.

max	leftOffset	rightOffset	valueID
4.52	1.97	0.11	10
135.09	109.92	7.31	28569
20.18	6.6	3.22	38851

Tabela 4.1. Vrednosti u tabeli *TriangularFN*.

valueID	forValueID	code
10	null	2
28569	null	2
38851	null	2

Tabela 4.2. Vrednosti u tabeli *FuzzyValue* za trougaone fazi brojeve.

Bitno je uočiti da svaka pojava fazi obeležja u modelu može imati različite tipove fazi vrednosti. Na primer, posmatrajmo obeležje *uspehOpsti* iz tabele *PrijemniGlPodaci*. Za svakog kandidata čiji se osnovni podaci nalaze u ovoj tabeli, vrednost obeležja *uspehOpsti* može biti različitog fazi tipa. Za nekog kandidata opšti uspeh može biti dat trougaonim fazi brojem, za drugog intervalom, dok za trećeg to može biti crisp vrednost.

Sasvim je sličan način skladištenja podataka o vrednostima karakterističnih funkcija za fazi tipove podataka interval, trapezoidni fazi broj i fazi veličine. U slučaju tipa podataka crisp, radi se o situaciji u kojoj fazi obeležje kao svoju vrednost realan broj, pa se baš taj broj skladišti kao vrednost obeležja *value*.

Način skladištenja lingvističkih labela je nešto drugačiji. Ranije je istaknuto da su to samo imenovane fazi vrednosti. Dakle, za lingvističku labelu vezujemo njeno ime, ali i fazi skup koji mora biti unapred definisan i koji može biti bilo koji fazi tip podržan u modelu, sem lingvističke labele. U slučaju da je neka fazi vrednost lingvistička labela, za njenu reprezentaciju se koristi rekurzivna veza tabele *FuzzyValue* sa samom sobom. Strani ključ *forValueID* referencira primarni ključ iste tabele *valueID*. Ako je fazi

vrednost lingvistička labela, onda ovo obeležje ima vrednost različitu od nula vrednosti. Osim toga, obeležje *code*, kao i za ostale vrste fazi tipova ukazuje da se radi o lingvističkoj labeli. Obeležje *forValueID* sadrži vrednost obeležja *valueID* za slog u tabeli *FuzzyValue* koji opisuje konkretnu vrednost koju imenuje lingvistička labela. Dakle, da bi se predstavila lingvistička labela, potrebna su dva sloga koja pripadaju tabeli *FuzzyValue*. Tabela 4.3 prikazuje slogove tabele *FuzzyValue* koji se odnose na lingvističke labele.

valueID	forValueID	code
3856	862	6
862	null	3
96332	58895	6
58895	null	4

Tabela 4.3. Vrednosti u tabeli *FuzzyValue* za lingvističke labele.

Potrebno je uočiti da se vrednosti obeležja *forValueID* u prvom i *valueID* u drugom slogu poklapaju. Isto važi i za treći i četvrti slog. Dakle, prvi i treći slog reprezentuju lingvističke labele, dok drugi i četvrti daju fazi skupove na koje se one odnose. Drugi slog pokazuje na trapezoidni fazi broj, a četvrti na fazi veličinu. Tabele 4.4 i 4.5 prikazuju odgovarajuće slogove u tabelama *TrapezoidalFN* i *FuzzyQuantity*.

leftMax	rightMax	leftOffset	rightOffset	valueID
4.29	4.88	1.21	0.12	862

Tabela 4.4. Vrednost u tabeli *TrapezoidalFN*.

left	right	isIncreasing	valueID
20.32	25.86	1	58895

Tabela 4.5. Vrednost u tabeli *FuzzyQuantity*.

Na ovom mestu je potrebno spomenuti još jednu dobru osobinu modela. Često se dešava da veći broj torki nekog fazi tipa, na primer trapezoidnog fazi broja, ima istu vrednost. Može se desiti da 69 studenata ima opšti uspeh izražen trapezoidnim fazi brojem prikazanim u tabeli 4.4. Da li to znači da za svaku pojavu opšteg uspeha mora postojati po jedna pojava u tabeli *TrapezoidalFN*? Naravno, odgovor je negativan. Dovoljno je

da postoji samo jedna pojava u tabeli *TrapezoidalFN*, onakva kakva je prikazana u tabeli 4.4, a da sve pojave iz modela kojima odgovara takav trapezoidni fazi broj referenciraju ovu pojavu putem vrednosti obeležja *valueID* u tabeli *FuzzyValue*.

Za potrebe testiranja samog modela, kao i PFSQL interpretera opisanog u petom poglavlju, prikazani model je skladišten u bazu podataka i popunjen testnim podacima. Najpre su uzeti podaci iz baze podataka studentske službe, a zatim je napisana posebna aplikacija koja fazifikuje vrednosti atributa označenih kao fazi po principu slučajnog broja. Na primer, za studenta Petra Petrovića je umesto njegovog opšteg uspeha 4.59 stavljen trougaoni fazi broj prikazan u prvoj vrsti tabele 4.1, poštujući opisane mehanizme skladištenja fazi vrednosti. Ovaj primer je izabran zbog toga što se radi o dobro razrađenom modelu baze podataka koji je u upotrebi duži niz godina. Zbog toga u bazi podataka postoji veliki broj pojava, što je pogodno za testiranje.

4.4 MOTIVACIJA I POREĐENJE

Kao što je u uvodu već istaknuto, implementacija interpretera za jezik PFSQL i potreba za skladištenjem fazi podataka u okviru baze podataka su motivacija za projektovanje prikazanog modela. Bez obzira na to, ovakva unapređena verzija modela je sazrela za upotrebu i van konteksta vezanog za PFSQL.

Model za skladištenje fazi podataka prikazan ovde je unapređena varijanta modela-prototipa detaljnije opisanog u (Takači & Škrbić, Data Model of FRDB with Different Data Types and PFSQL, 2008). Najveće unapređenje se odnosi na strukture koje skladište fazi vrednosti. U prethodnom modelu su postojale samo dve tabele u fazi meta modelu – *IsFuzzy* i *FuzzyValue*. Fazi vrednosti su skladištene u okviru tabele *FuzzyValue* kao stringovi sa predefinisanom strukturom za svaki od fazi tipova podataka. Na taj način je dobijena baza podataka čije vrednosti kolona u slogovima nisu atomarne, nego su se morale prvo dekomponovati u odnosu na predefinisana pravila da bi se koristile. Ovde se koristi više tabela, za svaki tip fazi podataka po jedna, pa se na taj način dobijaju atomarne vrednosti kolona u slogovima. Eliminacijom potrebe za

parsiranjem stringova prilikom upotrebe fazi vrednosti su performanse podignute na mnogo viši nivo.

Potrebno je istaći sličnosti i razlike prikazanog modela u odnosu na postojeće. Ovde se iznosi njegovo poređenje sa opštijim FIRST-2 modelom koji se može posmatrati kao najnapredniji model za skladištenje fazi podataka. On je nastao izdvajanjem dobrih osobina postojećih modela uz dodavanje novih (Galindo, Urrutia, & Piattini, 2006). Zbog njegovog obima, ovde se ne daje prikaz FIRST-2 modela, već se čitalac upućuje na literaturu. Poređenjem ova dva modela se došlo do zaključka da postoje značajne sličnosti između njih. Iako su metode za reprezentaciju fazi vrednosti veoma različite, funkcionalno, model prikazan u ovoj disertaciji predstavlja podskup modela FIRST-2.

Fazi atributi tipa 1 iz FIRST-2 modela su crisp vrednosti koji su podržane i ovde. Glavni fazi tipovi podataka koje prikazani model pokriva su podskup fazi atributa tipa 2 i 3 u FIRST-2 modelu. Nula vrednosti, intervali i trapezoidni fazi brojevi su u FIRST-2 modelu predstavljeni posebnim strukturama za svaki od tih tipova. Podskup trougaonih fazi brojeva, jednakostranični trouglovi, su predstavljeni pomoću koncepta približne vrednosti sa eksplicitno zadatom maksimalnom greškom. Sve ostale vrste trougaonih fazi brojeva, kao i fazi veličine se predstavljaju pomoću raspodela verovatnoće sa dve ili četiri vrednosti u FIRST-2 modelu. Nadalje, FIRST-2 model opisuje i širi skup različitih drugih tipova fazi podataka.

Ako sve ove mogućnosti postoje u već razvijenom modelu, postavlja se pitanje opravdanosti kreiranja novog modela. Zbog čega interpreter za PFSQL nije implementiran nad FIRST-2 modelom? Namera je bila da se projektuje što jednostavniji model koji će uključiti mogućnosti za skladištenje fazi tipova podataka koji se najčešće koriste. Takođe, jedan od zahteva je bio i što efikasnije skladištenje i rad sa skladištenim podacima. Uvođenje dodatnih osobina koje se retko koriste, a koje uopštavaju ali i komplikuju model, je opravdano u FIRST-2 modelu koji pretenduje da bude opšti model, ali za ovu namenu se to činilo nepotrebnim i neopravdanim. Možda bi se time dobilo na opštosti, ali bi se osnovna namena modela – implementacija PFSQL interpretera, drastično iskomplikovala.

Zbog potrebe za automatizacijom pojedinih zadataka vezanih za projektovanje fazi-relacione baze podataka po opisanom modelu, razvijen je poseban CASE alat koji je detaljno opisan u šestom poglavlju.

Poglavlje 5

PFSQL I FAZI JDBC DRAJVER

Ovo poglavlje se bavi osobinama PFSQL jezika, kao i implementacijom fazi JDBC drajvera koji sadrži interpreter za taj jezik. Na početku se analiziraju mogućnosti za proširenje SQL jezika fazi konstrukcijama. Na osnovu te analize se definiše PFSQL jezik kroz njegovu EBNF sintaksu. U nastavku se bliže definišu strukture dozvoljene u jeziku PFSQL i opisuju osnovne ideje za implementaciju interpretera za taj jezik. Kao jedna od ideja se navodi i mogućnost da se interpreter implementira u okviru fazi JDBC drajvera.

Pre detaljnijeg opisa same implementacije se daje opis JavaCC kompajler kompajlera, odnosno onih osobina ovog alata koje su bitne za razumevanje preostalog sadržaja poglavlja.

Detalji implementacije su prikazani kroz četiri sekcije. U prvoj sekciji se daje detaljan opis organizacije fazi JDBC drajvera. Druga sekcija sadrži opis implementacije parsera za PFSQL izraze realizovanog pomoću JavaCC kompajler kompajlera. U nastavku su opisani mehanizmi za transformaciju stabla parsiranja PFSQL izraza i izračunavanje fazi stepena zadovoljenja torki u skupu rezultata upita.

Na kraju je dato poređenje PFSQL jezika i opisanog interpretera sa jednim od najnaprednijih jezika te vrste – FSQL-om.

5.1 PFSQL

Kao što je u prethodnim poglavljima već napomenuto, PFSQL je jezik dobijen proširivanjem jezika SQL elementima fazi logike. Pored fazi mogućnosti, ova varijanta jezika u sebe uključuje i mogućnost da se fazi

uslovima dodele prioriteti. Teorijska osnova za izračunavanje vrednosti takvih izraza je GPFCSP predstavljen u drugom poglavlju.

Postavlja se pitanje koji tačno elementi SQL-a mogu biti prošireni i kako to uraditi. Prvo, pošto se radi sa fazi podacima koji su skladišteni u bazi podataka, onda svakako treba omogućiti da promenljive u upitu mogu imati i fazi vrednosti. Zbog toga je, dalje, potrebno omogućiti poređenje među različitim varijantama fazi vrednosti i poređenje fazi vrednosti sa običnim vrednostima. Drugim rečima, PFSQL interpreter mora biti u stanju da izračuna vrednosti izraza kao što je *visina=triangle(180,11,8)*. Gde je *triangle(180,11,8)* trougaoni fazi broj sa maksimumom u 180, levim otklonom 11 i desnim otklonom 8. Nadalje, treba omogućiti izračunavanje izraza koji sadrže relacione operatore, na primer *visina<triangle(180,11,8)*.

Ako se uvede poredak nad fazi skupovima, onda je moguće uvesti i neke skupovne funkcije nad njima. Na primer, potrebno je omogućiti upotrebu funkcija MIN, MAX i COUNT. Nadalje, ako su uvedene skupovne funkcije, može se razmatrati i uvođenje grupisanja sa fazi vrednostima, odnosno GROUP BY klauzula. Ovaj segment nije obrađen u ovoj doktorskoj disertaciji. Naime, nema puno smisla kreirati grupe na osnovu toga što sadrže identičnu vrednost neke fazi promenljive. Fazi skupovi međusobno mogu biti jednaki sa određenim fazi stepenom. Na primer vrednost 0.95 govori da nisu baš sasvim jednaki ali da su vrlo slični ili većim delom jednaki. Zato bi trebalo razmotriti mogućnost da se torke dele u grupe na osnovu njihovog fazi stepena pripadanja određenoj grupi. Mogao bi se zadati i prag, odnosno vrednost koju torka mora da postigne da bi bila član neke grupe. Analiza, modeliranje i implementacija ovakvih mogućnosti ostaje kao jedan od ciljeva u daljem radu na ovoj temi.

Klasični SQL uključuje mogućnosti za kombinovanje uslova pomoću logičkih operatora. Dakle, potrebno je proširiti semantiku ovih operatora tako da oni mogu imati fazi vrednosti kao svoje operande. Osim toga, dodaje se mogućnost da se svakom od uslova dodeli prioritet, pa da se onda takvi uslovi kombinuju pomoću logičkih operatora. Na taj način se dobijaju strukture sasvim slične GPFCSP sistemima, pa se oni otuda mogu prirodno primeniti na izračunavanje rezultata.

Ugnježdeni upiti su još jedan problem na koji se nailazi u pokušaju proširivanja SQL jezika fazi mogućnostima. Takve upite je moguće podeliti u dve kategorije. U prvu kategoriju spadaju oni ugnježdeni upiti koji ne zavise od ostatka SQL upita u koji su ugnježdeni. Ovakvi, „nezavisni" ugnježdeni upiti nisu problematični, oni mogu biti sračunati odvojeno, a rezultujuće vrednosti se mogu iskoristiti u nadređenom SQL upitu kao konstante.

„Zavisni" ugnježdeni upiti, odnosno oni koji zavise od SQL upita u koji su ugnježdeni u smislu da sadrže elemente spoljnog upita mogu predstavljati problem. Mogu se dopustiti upiti koji u izrazima koji povezuju ugnježdeni upit sa nadređenim nema fazi izraza zbog toga što se izračunavanje vrednosti u tom slučaju može prepustiti klasičnom SQL interpreteru. Međutim, ako i u tom segmentu učestvuju fazi vrednosti, onda je nejasno kako izračunati vrednost takvog izraza i koja je uopšte semantika takvog upita. Problem se svodi na „fazi spoj" između tabela, a mogućnost fazi veza između tabela nije razmatrana ni u modelu opisanom u četvrtom poglavlju. Kako mehanizam koji bi omogućio fazifikaciju veza između tabela nije razmatran, ovakve vrste upita se ne mogu dopustiti u PFSQL jeziku.

Imajući u vidu sve gore izneseno, definisana je gramatika PFSQL jezika koja se ovde u celosti navodi u EBNF sintaksi (listing 5.1).

```
Query ::= SelectWithoutOrder ( OrderByClause )? <EOF>

SelectWithoutOrder ::= <SELECT> ( <DISTINCT> )?
   SelectList FromClause ( WhereClause )? (
   GroupByClause )? ( ThresholdClause )?

AggregateFunction ::= ( <AVG> "(" ObjectName ")" |
   <COUNT> "(" ObjectName ")" | <COUNT> "(" <MULT> ")" |
   <MAX> "(" ObjectName ")" | <MIN> "(" ObjectName ")" |
   <SUM> "(" ObjectName ")" )

OrderByClause ::= <ORDER> <BY> SQLSimpleExpression (
   <ASC> | <DESC> )? ( "," SQLSimpleExpression ( <ASC> |
   <DESC> )? )*

SelectList ::= ( <MULT> | ( ObjectName |
   AggregateFunction ) ( "," ( ObjectName |
   AggregateFunction ) )* )
```

```
ObjectName ::= ( <IDENTIFIER> "." <IDENTIFIER> |
   <IDENTIFIER> )

FromClause ::= <FROM> FromItem ( "," FromItem )*

FromItem ::= <IDENTIFIER> ( <IDENTIFIER> )?

WhereClause ::= <WHERE> SQLExpression

SQLExpression ::= SQLAndExpression ( <OR>
   SQLAndExpression )*

SQLAndExpression ::= SQLUnaryLogicalExpression ( <AND>
   SQLUnaryLogicalExpression )*

SQLUnaryLogicalExpression ::= ( ( "(" SQLExpression ")"
   ) | ExistsClause | ( ( <NOT> )?
   SQLRelationalExpression ) )

ExistsClause ::= ( <NOT> )? <EXISTS> "(" SubQuery ")"

SQLRelationalExpression ::= ( "(" SQLSimpleExpression (
   "," SQLSimpleExpression )* ")" | SQLSimpleExpression
   ) ( SQLRelationalOperatorExpression | ( SQLInClause )
   | ( SQLBetweenClause ) | ( SQLLikeClause ) |
   IsNullClause )?

SQLExpressionList ::= SQLSimpleExpression ( ","
   SQLSimpleExpression )*

SQLRelationalOperatorExpression ::= Relop ( ( <ALL> |
   <ANY> )? "(" SubQuery ")" | SQLSimpleExpression )

Relop ::= ( "=" | "!=" | "#" | "<>" | ">" | ">=" | "<"
   | "<=" )

IsNullClause ::= <IS> ( <NOT> )? <NULL>

SQLInClause ::= ( <NOT> )? <IN> "(" ( SQLExpressionList
   | SubQuery ) ")"

SQLBetweenClause ::= ( <NOT> )? <BETWEEN>
   SQLSimpleExpression <AND> SQLSimpleExpression

SQLLikeClause ::= ( <NOT> )? <LIKE> SQLSimpleExpression

SQLSimpleExpression ::= SQLMultiplicativeExpression ( (
   <PLUS> | <MINUS> ) SQLMultiplicativeExpression )* (
   PriorityClause )?

SQLMultiplicativeExpression ::= SQLExponentExpression (
   ( <MULT> | <DIVIDE> ) SQLExponentExpression )*
```

```
SQLExponentExpression ::= SQLUnaryExpression ( <POWER>
   SQLUnaryExpression )*

SQLUnaryExpression ::= ( <PLUS> | <MINUS> )?
   SQLPrimaryExpression

SQLPrimaryExpression ::= ( <NULL> | ObjectName |
   <NUMBER> | <CHAR_LITERAL> | "(" SQLExpression ")" )

GroupByClause ::= <GROUP> <BY> SQLExpressionList (
   <HAVING> HavingExpression )?

HavingExpression ::= ( AggregateFunction Relop
   SQLSimpleExpression | SQLSimpleExpression Relop
   AggregateFunction )

ThresholdClause ::= <THRESHOLD> <NUMBER>

PriorityClause ::= <PRIORITY> <NUMBER>

SubQuery ::= SelectWithoutOrder
```

Listing 5.1. EBNF gramatika jezika PFSQL.

Iako se sve osobine jezika ne mogu predstaviti na sintaksnom nivou, ova gramatika okvirno prikazuje izgled jezika i proširenja koja su u njega uključena. U nastavku će biti prikazani primeri PFSQL upita koji zadovoljavaju ovu gramatiku da bi se jasnije ilustrovale mogućnosti jezika. Kao model baze podataka nad kojom će biti ilustrovani upiti uzet je model opisan u sekciji 4.3, slika 4.2.

Tabela *PrijemniGlPodaci* sadrži fazi obeležje *uspehOpsti*. Dakle, mogao bi se postaviti upit koji vraća ime i prezime svih kandidata čiji je opšti uspeh veći od trougaonog fazi broja *triangle(4,1,0.4)*. Taj PFSQL upit bi glasio ovako:

```
SELECT pgl.ime,pgl.prezime
FROM PrijemniGlPodaci pgl
WHERE (pgl.uspehOpsti>#triangle(4,1,0.4)#)
```

Znak # je izabran da bi se označile fazi konstante. Pod uslovom da imamo definisanu lingvističku labelu osrednjiUspeh čija je vrednost upravo *triangle(4,1,0.4)*, prethodni upit bi mogao da glasi ovako:

```
SELECT pgl.ime,pgl.prezime
FROM PrijemniGlPodaci pgl
WHERE (pgl.uspehOpsti>#ling(osrednjiUspeh)#)
```

Ovakav upit može da se obogati dodatnim uslovima. Sledi upit koji vraća ime i prezime kandidata kojima je opšti uspeh veći od osrednjeg sa prioritetom 0.7, a uspeh na četvrtoj godini veći od 4.50 sa prioritetom 0.4. Upitu može da se doda i prag zadovoljenja (THRESHOLD), tako da on vraća samo one torke čiji je fazi stepen pripadanja skupu rezultata upita veći ili jednak sa 0.2. Ovaj upit mora da uključi i spajanje tabela, pošto se uspesi po godinama nalaze u tabeli PrijemniPrijava.

```
SELECT pgl.ime,pgl.prezime
FROM PrijemniGlPodaci pgl, PrijemniPrijava pp
WHERE (pgl.sfrPr=pp.sfrPr) AND
(pgl.sfrRokPrijemni = pp.sfrRokPrijemni) AND
(pgl.uspehOpsti>#ling(osrednjiUspeh)# PRIORITY 0.7) AND
(pp.uspeh4>4.5 PRIORITY 0.4) THRESHOLD 0.2
```

Kao što je već naglašeno, neke skupovne funkcije mogu imati fazi vrednost kao argument. To su MAX, MIN i COUNT. Funkcije SUM i AVG koje spadaju u standardne ne mogu imati fazi vrednost kao argument zbog toga što sabiranje i druge operacije sa fazi vrednostima nisu podržane iako teorijski postoji mogućnost da se doda i ova osobina. Sledi jednostavan primer upita koji vraća minimalan opšti uspeh svih kandidata.

```
SELECT MIN(pgl.uspehOpsti)
FROM PrijemniGlPodaci pgl
```

Iako je upit jednostavan, njegovo izračunavanje je složeno zbog toga što uključuje poređenje fazi vrednosti. Zato se, na primer, može desiti da kao rezultat dobijemo vrednost *triangle(2.9,0.4,0.7)*.

5.2 IZVRŠAVANJE PFSQL UPITA I FAZI JDBC DRAJVER

5.2.1 Izračunavanje fazi stepena zadovoljenja

Veoma važna razlika između SQL i PFSQL jezika je u načinu obrade slogova u bazi podataka koji se razmatraju u upitu. Klasična relaciona baza podataka izvršava upite tako što torke koje se razmatraju u upitu prihvata u skup rezultata upita ako zadovoljavaju uslove upita. One torke koje ne zadovoljavaju uslove se ne uvrštavaju u skup rezultata. Dakle, za svaku torku koja se razmatra u upitu se određuje da li pripada skupu rezultata ili

ne. Drugim, rečima, svakoj torki koja se razmatra u upitu se pridružuje vrednost tačno ili netačno. Sa druge strane, PFSQL interpreter svakoj od torki koja se razmatra u upitu dodeljuje fazi stepen pripadanja. Dakle, skup rezultata upita sadrži sve torke koje se razmatraju u upitu čiji fazi stepen pripadanja tom skupu prelazi zadati prag. Podrazumevana vrednost praga je 0, pa skup rezultata upita sadrži u stvari sve torke koje se razmatraju u upitu i fazi stepen pripadanja za svaku od njih.

Na koji način se dodeljuju fazi stepeni pripadanja torkama u PFSQL upitu? Neka su u nekom PFSQL upitu zadati uslovi međusobno povezani logičkim operatorima. Nazovimo takve uslove elementarnim uslovima. Potrebno je, dakle, prvo dodeliti istinitosnu vrednost svakom elementarnom uslovu. Jedini način da se to postigne je da se kreira algoritam koji izračunava istinitosne vrednosti za svaku moguću kombinaciju vrednosti u upitu i u bazi podataka. Na primer, ako upit sadrži elementarni uslov koji poredi fazi veličinu sa trougaonim fazi brojem, mora biti na raspolaganju algoritam koji izračunava meru kompatibilnosti ova dva fazi skupa. Slično, mora postojati ovakav algoritam za sve ostale fazi tipove podataka.

Svaki od elementarnih uslova može imati dodeljen prioritet, a takve konstrukcije međusobno mogu biti povezane pomoću logičkih operatora. Ako na opisani način dodelimo fazi stepen pripadanja elementarnim uslovima, postavlja se pitanje kako da izračunamo fazi stepen pripadanja cele torke skupu rezultata upita? Teorijska osnova za ovakva izračunavanja je GPFCSP. U sekciji 2.2.3, u kojoj su GPFCSP sistemi definisani je data teorema 2.1 koja uvodi GPFCSP sistem koji se koristi za ove potrebe. Naime, prioriteti se agregiraju sa fazi stepenom pripadanja elementarnih uslova pomoću trougaone konorme S_P tako što se fazi stepen zadovoljenja elementarnog ograničenja agregiran sa prioritetom računa kao $S_P(x,1-p)$, gde je x fazi stepen zadovoljenja elementarnog uslova, a p prioritet. Kada se na opisani način fazi stepeni pripadanja dodele elementarnim uslovima i njihovim prioritetima, oni se dalje agregiraju zbog toga što su povezani logičkim operatorima. U slučaju operatora AND, koristi se trougaona norma, u slučaju operatora OR, trougaona konorma, dok se operator NOT interpretira kao striktna negacija: $N(x)=1-x$. Kao što GPFCSP sistem dat

u teoremi 2.1 nalaže, kao trougaona norma se koristi T_L, a kao trougaona konorma - S_L.

5.2.2 Proces izvršavanja PFSQL upita

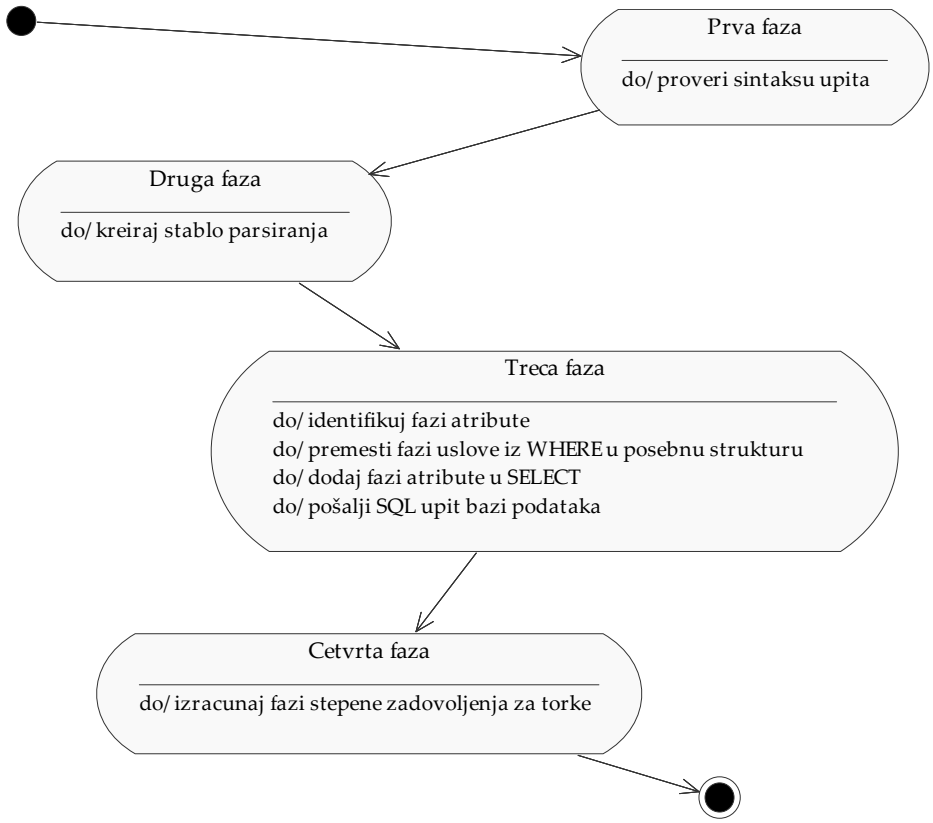
U ovoj sekciji će biti prikazan proces koji omogućava izvršavanja PFSQL upita. Osnovna ideja je da se PFSQL upit prvo transformiše u nešto što klasični SQL interpreter razume. Naime, uslovi koji sadrže fazi vrednosti ili promenljive se izbacuju iz WHERE klauzule. Fazi promenljive izbačene iz WHERE klauzule se premeštaju neposredno posle ključne reči SELECT, da bi se njihove vrednosti ipak dobile iz baze podataka. Na taj način, upit postaje običan SQL upit koji će vratiti i torke koje zadovoljavaju fazi uslove koji su izbačeni sa određenim fazi stepenom zadovoljenja i one koji ih uopšte ne zadovoljavaju. Potom se dobijeni klasični SQL upit šalje bazi podataka, a dobijeni rezultati se interpretiraju koristeći već opisane fazi mehanizme koji svakoj torci u skupu rezultata upita dodeljuju fazi stepen zadovoljenja. Ako je dat prag zadovoljenja, onda se torke čiji je fazi stepen zadovoljenja ispod ovog praga odbacuju.

Proces izvršavanja PFSQL upita je prikazan na dijagramu aktivnosti datom na slici 5.1. On se može podeliti u četiri faze:

1. provera sintakse PFSQL upita,
2. učitavanje upita u memorijsku strukturu,
3. transformacija upita i
4. interpretacija rezultata dobijenih od baze podataka.

Na početku se, dakle, proverava sintaksna ispravnost upita koristeći skener i parser. Ako je upit ispravan, rezultat sintaksne analize koju obavlja parser je memorijska struktura koja predstavlja ovaj upit – stablo parsiranja, ili sintaksno stablo.

Sledeći korak je transformacija ove strukture na već opisani način. Za svaki atribut se proverava da li je fazi ili ne koristeći tabelu *IsFuzzy* (sekcija 4.3, slika 4.3). Kada se fazi atributi identifikuju, uslovi u WHERE klauzuli u kojima oni učestvuju se eliminišu iz stabla parsiranja, a sami atributi se dodaju posle ključne reči SELECT. Rezultujuće stablo parsiranja predstavlja klasični SQL upit, koji se šalje bazi podataka.



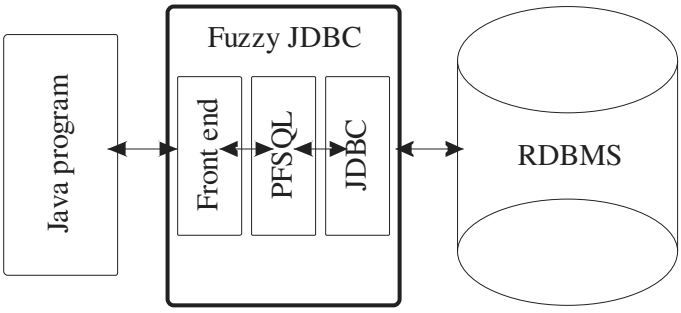
Slika 5.1. Dijagram aktivnosti - izvršavanje PFSQL upita.

Kada se upit izvrši, dobijeni rezultati se dalje procesiraju koristeći već opisane mehanizme fazi logike i fazi uslove koji su u trećoj fazi izbačeni iz WHERE klauzule.

5.2.3 Fazi JDBC drajver

Potreba da se što je moguće više olakša upotreba PFSQL jezika iz Java programa, a da se ipak sačuva nezavisnost od baze podataka, je dovela do ideje da se implementira fazi JDBC drajver. Ovaj drajver je, u stvari, skup klasa koje predstavljaju omotač (wrapper) za JDBC API koji je implementiran u JDBC drajveru za konkretan sistem za upravljanje bazama podataka. JDBC drajver postaje parametar koji se zadaje fazi JDBC drajveru, koga on dalje koristi za pristup konkretnom sistemu za

upravljanje bazama podataka. Takođe, svi mehanizmi za procesiranje PFSQL upita opisani u prethodnim sekcijama su inkorporirani u ovaj fazi JDBC drajver. Arhitektura takvog sistema je prikazana na slici 5.2.



Slika 5.2. Fazi JDBC drajver.

Java program koristi interfejse koje nudi fazi JDBC drajver kao njegovu front end komponentu. Ovi interfejsi uključuju mogućnosti za:

- inicijalizaciju klase - drajvera,
- kreiranje konekcije na bazu podataka,
- kreiranje i izvršavanje PFSQL upita i
- čitanje skupa rezultata upita.

Kada se izvrše, PFSQL upiti se procesiraju na ranije opisan način i šalju bazi podataka kao obični SQL upiti putem običnog JDBC drajvera. Rezultati koji se dobijaju na ovaj način se ponovo procesiraju opisanim mehanizmima i šalju Java programu preko front end interfejsa.

Ovim su definisani zahtevi koji se postavljaju pred PFSQL interpreter, odnosno fazi JDBC drajver. Opisane su i glavne smernice za implementaciju tih sistema. Sledi sekcija u kojoj se opisuje važan alat korišćen u implementaciji – JavaCC, a zatim i opis same implementacije.

5.3 JAVACC

Tehnologija za automatsko generisanje parsera na osnovu specifikacije sintakse postoji već 20 godina. Parser generator je softver koji prihvata sintaksnu specifikaciju kao ulaz, a generiše parser za zadatu sintaksu kao izlaz. JavaCC (Java Compiler Compiler) je open source parser generator (ili kompajler kompajler) poslednje generacije koji generiše

parsere u programskom jeziku Java. Ovaj parser generator je razvijen u kompaniji Sun Microsystems pod imenom Jack 1996. Od tada je promenio nekoliko kompanija koje su ga razvijale, da bi sada završio kao JavaCC, verzija 4, jedan od projekata grupe java.net (https://javacc.dev.java.net/). Ovde će biti ukratko predstavljene njegove mogućnosti upotrebljene u realizaciji fazi JDBC drajvera. Knjiga (Appel, 2002), koja se smatra jednim od najboljih izvora vezanih za ovu temu, je i u ovoj disertaciji korišćena kao osnovna literatura za segment implementacije interpretera za jezik PFSQL.

JavaCC nije i jedini kompajler kompajler koji generiše Java kod. Među značajnijim ovakvim alatima su još i SableCC i ANTLR, čija se poslednja verzija čak smatra i superiornom u odnosu na JavaCC. Međutim, ovde je izabran JavaCC iz više razloga. Prvo, to je besplatan softver, vrlo dobro dokumentovan iz više različitih izvora. Za JavaCC postoje brojni primeri i kompletno urađene gramatike za velikih broj jezika – između ostalih i za SQL (Werner, 2003). Osim toga, JavaCC uključuje i JJTree predprocesor za izgradnju stabala parsiranja koja su bitna za rešavanje problema postavljenih pred ovu disertaciju. I, na kraju, ali ne i najmanje važno, autor je već upoznat sa JavaCC-om sa kojim je radio i u nekim ranijim projektima i o kome je držao uvodna predavanja.

Na listingu 5.2 je data jednostavna gramatika napisana u JavaCC sintaksi. Specificirano je da parser prihvata reči koje se sastoje od sekvence prirodnih brojeva i znakova + između njih.

```
// Specifikacija klase
PARSER_BEGIN(Adder)
package probni;
public class Adder {
      public static void main( String[] args ) throws
ParseException, TokenMgrError {
            Adder parser = new Adder( System.in );
            parser.Start();
      }
}
PARSER_END(Adder)
// leksička specifikacija
SKIP : {
" "
|"\n"
```

```
|"\r"
|"\r\n"
}
TOKEN :
{
      < PLUS : "+" >
|     < NUMBER : (["0"-"9"])+ >
}
// specifikacija gramatike
void Start() :
{}
{
      <NUMBER>
      (<PLUS>    <NUMBER>)*
      <EOF>
}
```

Listing 5.2. Sabiranje celih brojeva u JavaCC sintaksi.

Ova specifikacija sadrži tri celine:

1. **Specifikacija klase.** Ovaj deo se odnosi na specifikaciju osnovne Java klase parsera koji će biti generisan. Ta klasa će imati atribute i metode specificirane na ovom mestu. Osim toga, za svaku produkciju u gramatici se generiše po jedan dodatni metod. U ovom primeru se, pored imena paketa i klase, zadaje i main metod koji kreira instancu ove klase i poziva metod Start() od kojeg počinje izvršavanje parsera.

2. **Leksička specifikacija.** Ovde se opisuju tokeni koje parser prepoznaje pomoću regularnih izraza. Tokeni navedeni u sekciji SKIP su oni koji se automatski preskaču. Sekcija TOKEN u ovom slučaju sadrži opis dve vrste tokena – znaka + i prirodnih brojeva.

3. **Specifikacija gramatike.** Kontekstno slobodna gramatika jezika koji parser prihvata se opisuje u ovom delu pomoću EBNF produkcija. U slučaju prikazanom na listingu 5.2, radi se o parseru koji prihvata reči koje se sastoje od prirodnog broja posle koga sledi sekvenca koja se sastoji od znaka + i prirodnog broja koja se može ponoviti nula ili više puta. Na kraju se nalazi specijalan znak za završetak fajla, <EOF>.

5.3.1 Akcije

Prethodni primer prikazuje upotrebu JavaCC parsera u pokušaju da se ustanovi da li je ulaz saglasan sa opisanom gramatikom. Ako jeste, parser prosto prestaje sa radom, a ako nije, štampa se poruka o greški. Naravno, ovo nije dovoljno, potrebno je omogućiti da parser izvršava i dodatne funkcionalnosti u zavisnosti od ulaza na koji nailazi. Za ove potrebe su u okviru JavaCC-a uvedene akcije – Java kod koji se piše na odgovarajućim mestima u specifikaciji gramatike, a izvršava se kada parser naiđe na to mesto u gramatici dok parsira ulaz. Akcije mogu biti proizvoljno složene, a mogu pozivati i dodatne biblioteke.

Postoje dva tipa akcija, leksičke i akcije parsera. Leksičke akcije se izvršavaju posle uspešnog uparivanja tokena, dok se akcije parsera izvršavaju kada parser prođe određenu tačku u procesu parsiranja. Na listingu 5.3 je data JavaCC specifikacija sa listinga 5.2 sa dodatim akcijama.

```
// Specifikacija klase
PARSER_BEGIN(Adder)
package probni;
public class Adder {
      public static void main( String[] args ) throws
ParseException, TokenMgrError {
            Adder parser = new Adder( System.in );
            int val = parser.Start();
            System.out.println(""+val);
      }
}
PARSER_END(Adder)
// leksička specifikacija
SKIP : {
" "
|"\n"
|"\r"
|"\r\n"
}
TOKEN :
{
      < PLUS : "+" > {System.out.println("Leksička
akcija: pronađen je znak plus ");}
|     < NUMBER : (["0"-"9"])+ >
```

```
}
// specifikacija gramatike
int Start() throws NumberFormatException:
{
      Token t;
      int i;
      int value;}
{
      t=<NUMBER>
      {i = Integer.parseInt( t.image );}
      {value = i;}
      (<PLUS>
            t = <NUMBER>
            {i = Integer.parseInt( t.image );}
            {value += i;}
      )*
      <EOF>
      {return value;}
}
```

Listing 5.3. JavaCC akcije.

Metod *Start()* je bitno izmenjen tako što su dodate akcije parsera. Osim toga, on sada vraća vrednost tipa *int* i potencijalno izaziva *NumberFormatException*. U njemu su deklarisane tri promenljive, promenljiva *t* klase *Token* (automatski generisana klasa kojom je predstavljen token) i dva atributa tipa *int*.

Bitno je uočiti mehanizam koji omogućava da se „uhvati" sadržaj tokena. Prvo se u gramatici token dodeli promenljivoj *t* koja je deklarisana kao *Token* u zaglavlju, a zatim se u akcijama koje slede u vitičastim zagradama ta vrednost i koristi. Atribut *image* klase *Token* sadrži string od kog se token sastoji. Dakle, ako su odgovarajući tokeni prirodni brojevi, ovakav parser će izvršiti njihovo sabiranje.

Specificirana je i jedna leksička akcija koja se može videti u segmentu u kome se definišu tokeni. Ona ispisuje na konzolu odgovarajući tekst svaki put kada parser naiđe na znak plus.

5.3.2 Gledanje unapred i rešavanje neodređenosti

Leksička i sintaksna specifikacija mogu u sebi da sadrže neodređenosti. U takvim situacijama se koriste standardne šeme za rešavanje neodređenosti.

Uzmimo kao primer specifikaciju programskog jezika Java. Šta bi parser trebao da uradi kada naiđe na sekvencu „interface“? Očigledno, trebao bi da je prepozna kao token koji je povezan sa ključnom reči „interface“. Ali mogući su i drugi načini prepoznavanja, možda se radi o imenu „interface“ ili ključnoj reči *int* iza koje sledi sekvenca „erface“.

Rešavanje nedoslednosti u leksičkoj specifikaciji se uglavnom odnosi na pravilo da je najduže uparivanje i najpoželjnije. Kada postoje uparivanja iste dužine, koristi se ono koje se prvo navodi u leksičkoj specifikaciji.

Rešavanje nedoslednosti u sintaksnoj specifikaciji je ozbiljniji problem. Može se desiti da neke produkcije počinju istim tokenom ili nizom tokena, pa nije uvek lako zaključiti o kojoj se produkciji radi. U tom slučaju je potrebno posmatrati sledeće tokene u nizu i zaključiti o ispravnoj produkciji na osnovu tog niza. Posmatrajmo kao primer specifikaciju naredbe import u programskom jeziku Java.

```
void DemandImportDeclaration() :
{}
{
      "import" Name() "." "*" ";"
}
void Name() :
{}
{
      <IDENTIFIER> ("." <IDENTIFIER>)*
}
```

Listing 5.4. Naredba import u programskom jeziku Java.

Ako se na ulazu pojavi „import x.*;“, parser će tu tačku upariti sa onom koja je specificirana u produkciji *Name*. Zbog toga parsiranje neće uspeti, pošto se posle tačke ovde očekuje ime, a ne „*“. Dakle, ovaj inače

ispravan ulaz, neće biti uspešno prepoznat. Zaključak je da ovakva specifikacija nije ispravna.

Da bi se ovaj problem ispravio, potrebno je naznačiti da parser gleda više tokena unapred pre nego što odluči kako da nastavi parsiranje. U ovom primeru je dovoljno da se pogleda tekući token i jedan token unapred – dakle, dva tokena. Ovakvo ponašanje se može specificirati kao globalno ponašanje parsera, ali onda će on za svaku produkciju gledati dva tokena unapred što će izazvati probleme sa performansama. Zbog toga je moguće, koristeći ključnu reč *LOOKAHEAD*, specificirati da parser samo u slučaju ove produkcije na kritičnom mestu pogleda još jedan token unapred (listing 5.5).

```
void DemandImportDeclaration() :
{}
{
      "import" Name() "." "*" ";"
}
void Name() :
{}
{
      <IDENTIFIER> (LOOKAHEAD(2) "." <IDENTIFIER>)*
}
```

Listing 5.5. Naredba import u programskom jeziku Java – tačnija specifikacija.

Postoje i drugi napredniji i kompleksniji načini da se parseru zada pravilo kako treba gledati unapred, ali oni ovde neće biti opisani, tim pre što nisu neophodni za rešavanje problema postavljenih u ovoj disertaciji.

5.3.3 JJTree

JJTree je JavaCC predprocesor koji omogućava unošenje akcija za izgradnju stabla parsiranja na različitim mestima u JavaCC specifikaciji. Dakle, JJTree specifikacija prvo prolazi kroz predprocesor koji od nje kreira JavaCC specifikaciju. Takva specifikacija onda prolazi kroz JavaCC kompajler da bi konačno nastao parser. Iako je izgradnja stabla parsiranja moguća i bez ovog alata, uz upotrebu akcija, JJTree drastično olakšava taj posao.

Ovde su navedene samo osnovne smernice vezane za JJTree bitne sa stanovišta ove disertacije, dok se kompletan opis svih osobina i funkcionalnosti može naći u (Copeland, 2007).

Po podrazumevanim podešavanjima, JJTree generiše kod koji konstruiše čvorove stabla parsiranja za svaki neterminal u jeziku. Ovakvo ponašanje se može modifikovati tako da neki neterminali nemaju odgovarajuće čvorove ili tako da se dodatni čvorovi generišu za proizvoljne celine.

JJTree definiše interfejs *Node* koga implementiraju svi čvorovi stabla parsiranja. Ovaj interfejs sadrži metode za postavljanje roditelja čvora, dodavanje dece, pronalaženje dece itd.

Postoje dva moda u kojima JJTree radi – „simple“ i „multi“. U „simple“ modu svaki čvor je objekat klase *SimpleNode*, dok se u „multi“ modu za svaki čvor generiše odgovarajuća klasa čije ime se izvodi iz imena čvora. Naravno, svaka od ovih klasa implementira interfejs *Node*, kao što je gore naznačeno.

Sa stanovišta ove disertacije, „multi“ mod je naročito interesantan. On omogućava da se za svaki neterminal definiše posebna klasa koja je napisana tako da sadrži one atribute koji su bitni baš za taj neterminal. U JJTree specifikaciji se tim atributima zadaju vrednosti za svaku njihovu konkretnu pojavu. Na primer, posmatrajmo kod dat na listingu 5.6.

```
void ThresholdClause():
{Token t;}{
  <THRESHOLD>t=<NUMBER>{jjtThis.setNumber(t.image);}
}
```

Listing 5.6. Threshold klauzula.

Prikazan je primer klauzule THRESHOLD koja služi za zadavanje praga zadovoljenja ograničenja koji mora da zadovolji svaka torka koja se izlistava u rezultatu. Ovde se pomoću atributa *image* klase *Token* čita vrednost praga i smešta u odgovarajući atribut klase *ThresholdClause*, koja je napisana za ovu klauzulu, pomoću njenog metoda *setNumber* (listing 5.7).

```
package yu.ac.ns.im.is.pfsql.gram;
public @SuppressWarnings("all") class ThresholdClause
extends SimpleNode {
```

```
	public String number = "";
	public String getText(){
		return "THRESHOLD "+number;
	}
	public ThresholdClause(int id) {
		super(id);
	}
	public ThresholdClause(ExampleParser p, int id) {
		super(p, id);
	}
	public String getNumber() {
		return number;
	}
	public void setNumber(String number) {
		this.number = number;
	}
}
```

Listing 5.7. Klasa *ThresholdClause*.

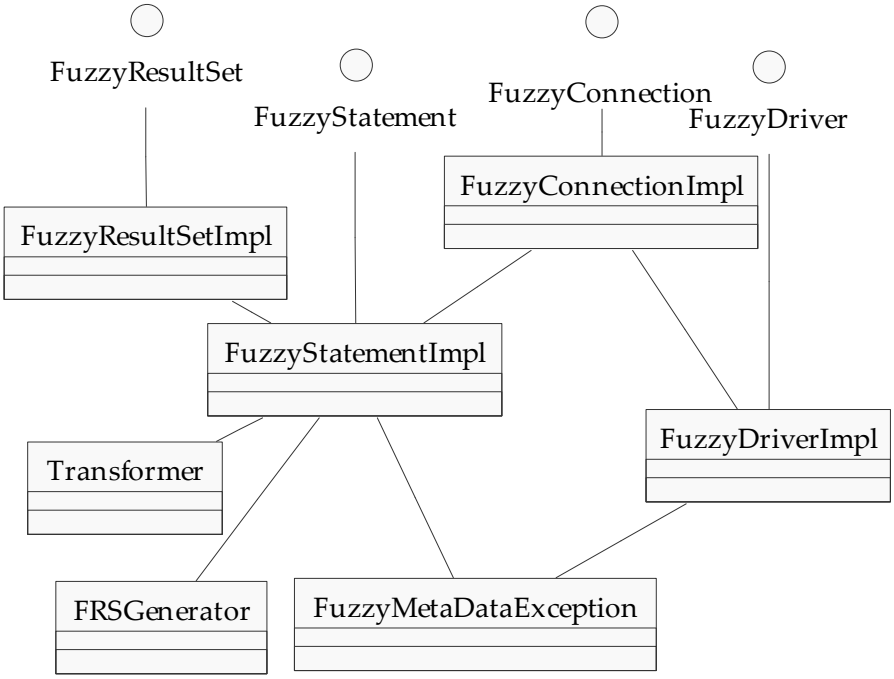
5.4 IMPLEMENTACIJA

U ovoj sekciji će detaljno biti prikazana implementacija fazi JDBC drajvera i mehanizama procesiranja PFSQL upita u programskom jeziku Java. Najpre će biti dat pregled organizacije fazi JDBC drajvera – njegovih klasa i interfejsa. Zatim sledi prikaz implementacije parsera za PFSQL jezik urađene pomoću JavaCC kompajler kompajlera. U okviru ovog dela će biti prikazan i način generisanja stabla parsiranja. Sledeća tema je implementacija transformacije dobijenog stabla u skladu sa već opisanim pravilima, a zatim i postavljanje SQL upita bazi podataka. Poslednji element implementacije je izračunavanje fazi stepena zadovoljenja torki koje su dobijene u prethodnom koraku.

5.4.1 Organizacija fazi JDBC drajvera

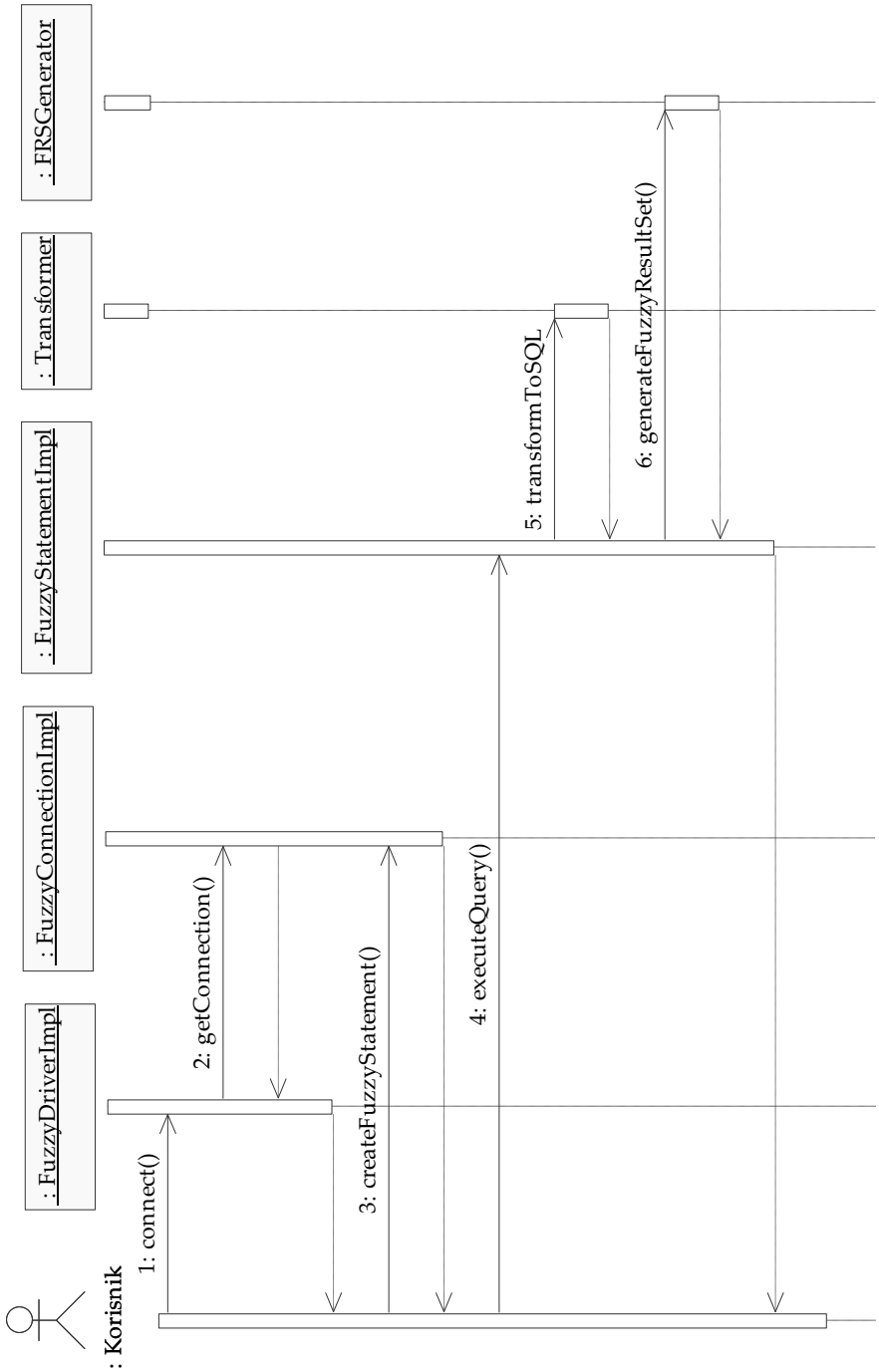
Na slici 5.3 je dat dijagram klasa kojim je ilustrovana struktura fazi JDBC drajvera. Četiri najvažnije klase – *FuzzyDriverImpl, FuzzyConnection-Impl, FuzzyStatementImpl i FuzzyResultSetImpl* predstavljaju omotače (wrapper) za odgovarajuće interfejse JDBC API-ja: *DriverManager*,

Connection, *Statement* i *ResultSet*, respektivno. Ove klase u pozadini rade sa mehanizmima koje nudi JDBC API na koji dodaju nove funkcionalnosti fazi JDBC drajvera. Korisnik nema direktan pristup ovim klasama, već se njima obraća putem odgovarajućih interfejsa prikazanim na slici 5.3. Ovi interfejsi sakrivaju detalje implementacije drajvera i izdvajaju metode značajne za korisnika. Osim toga, tu je i klasa *FuzzyMetaDataException*, koja predstavlja izuzetak koji može da se generiše u slučaju greške. Klase *Transformer* i *FRSGenerator* sadrže implementaciju mehanizama transformacije stabla parsiranja i izračunavanja fazi stepena zadovoljenja torki u skupu rezultata upita, respektivno. Osim ovih, drajver sadrži i neke pomoćne klase koje su izostavljene da bi dijagram bio jasniji.



Slika 5.3. Dijagram klasa fazi JDBC drajvera.

Na slici 5.4 je dat dijagram sekvenci koji okvirno pokazuje kako klase drajvera izvršavaju PFSQL upit. Na slici je namerno prikazana situacija u kojoj korisnik direktno koristi klase fazi JDBC drajvera, iako on to, u stvari, radi preko interfejsa. Interfejsi nisu prikazani da bi dijagram bio jasniji.



Slika 5.4. Dijagram sekvenci – izvršavanje PFSQL upita.

Na zahtev korisnika, pomoću metoda *connect*, klasa *FuzzyDriverImpl* kreira konekciju na bazu podataka i *FuzzyConnectionImpl* objekat koji vraća korisniku. Ovaj objekat, po pozivu njegovog metoda *createFuzzyStatement*, kreira *FuzzyStatementImpl*, odnosno objekat koji predstavlja PFSQL upit. *FuzzyStatementImpl* objekat, sadrži veoma važan metod – *executeQuery*. Ovaj metod kao parametar dobija string – PFSQL upit. Poziv tog metoda znači početak izvršavanja PFSQL upita. Objekat *FuzzyStatementImpl* pozivom metoda *transformToSQL*, klase *Transformer* inicira kreiranje stabla parsiranja i njegovu transformaciju u stablo koje predstavlja klasičan SQL upit. Zatim se takav upit šalje bazi podataka koristeći interfejse i metode klasičnog JDBC API-ja. Rezultat koji se dobija kao klasičan JDBC *ResultSet* objekat, se šalje klasi *FRSGenerator* kao parametar njenog metoda *generateFuzzyResultSet*. Dakle, klasa *FRSGenerator* sadrži implementaciju izračunavanja fazi stepena zadovoljenja torki u skupu rezultata upita. Generisani skup rezultata upita se šalje korisniku kao *FuzzyResultSetImpl* objekat. Kao što je već istaknuto, ovom objektu korisnik pristupa pomoću interfejsa *FuzzyResultSet* da bi pročitao rezultate.

5.4.2 Implementacija PFSQL parsera

Na osnovu EBNF specifikacije jezika PFSQL opisane u sekciji 5.1, konstruisana je JavaCC specifikacija parsera za ovaj jezik. Ovaj zadatak nije naročito težak. Iako je JavaCC specifikacija naizgled potpuno drugačija od EBNF specifikacije, mehanizmi za predstavljanje konstrukcija jezika su identični. Zbog toga je implementacija JavaCC parsera za jezik definisan pomoću EBNF gramatike tehnički jednostavna.

Ovde se, zbog njegovog obima, ne navodi kompletan kod. Umesto toga, na listingu 5.8 je prikazan segment kojim se ilustruje način implementacije.

```
void SelectWithoutOrder():
{}{
  <SELECT> [<DISTINCT>]
  SelectList()
  FromClause()
  [WhereClause()]
  [GroupByClause()]
```

```
  [ThresholdClause()]
}
void SelectList():{}{
  <MULT>  |
(ObjectName()|AggregateFunction())(","(ObjectName()|Agg
regateFunction()))*
}
void FromClause():{}{
  <FROM>FromItem()(","FromItem())*
}
void WhereClause():{}{
  <WHERE>SQLExpression()
}
void SQLExpression():
{}{
  SQLAndExpression()(<OR>SQLAndExpression())*
}
void SQLAndExpression():
{}{
SQLUnaryLogicalExpression()(<AND>SQLUnaryLogicalExpress
ion())*
}
void SQLUnaryLogicalExpression():
{}{
  LOOKAHEAD(2)ExistsClause()
  |([<NOT>]SQLRelationalExpression())
}
```

Listing 5.8. Segment koda JavaCC specifikacije za PFSQL.

Na listingu su prikazani neki elementi definicije naredbe SELECT. U prvom metodu, *SelectWithoutOrder()*, je definisano da se naredba sastoji od ključne reči „SELECT“ iza koje može, a ne mora da sledi ključna reč „DISTINCT“. Zatim sledi izraz kojim se specificira šta se selektuje iz baze podataka, definisan u okviru metoda *SelectList()*. Iza toga sledi FROM klauzula, definisana u okviru metoda *FromClause()*, posle koje ide WHERE klauzula. Ona je ovde prikazana sa više detalja. Metod *WhereClause()* jednostavno specificira da se ova klauzula sastoji iz ključne reči „WHERE“ i konstrukcije nazvane *SQLExpression*. Tom konstrukcijom je predstavljen kompletan izraz koji sledi posle ključne reči „WHERE“, za koga su prikazani neki detalji. Kao što se vidi na listingu, takav izraz se sastoji od

manjih izraza povezanih ključnom reči „OR“. Ti manji izraze se sastoje iz još manjih koji su povezani ključnom reči „AND“ itd.

Međutim, kao što je ranije već nagoveštено, nije dovoljno samo specificirati JavaCC parser. Potrebno je iskoristiti JJTree mehanizme, opisane u sekciji 5.3.3, da bi se dobio kod koji izgrađuje stablo parsiranja i to bez gubitka informacija koje su sadržane u upitu. Dakle, potrebno je:

- izmeniti čvorove stabla koji se generišu automatski iz JJTree specifikacije tako da prime dodatne informacije sadržane u upitu i
- izmeniti JavaCC specifikaciju tako da uključi semantičke akcije koje će informacije iz upita skladištiti u strukturu napravljenu u prvoj tački.

Na listingu 5.9 je dat kod koji je prikazan na listingu 5.8 obogaćen pomenutim semantičkim akcijama.

```
void SelectWithoutOrder():
{Token select;
 Token distinct;}{
  select=<SELECT>{jjtThis.setSelect(select.image);}

[distinct=<DISTINCT>{jjtThis.setDistinct(distinct.image
);}]
  SelectList()
  FromClause()
  [WhereClause(){jjtThis.setHasWhere(true);}]
  [GroupByClause(){jjtThis.setHasGroup(true);}]
  [ThresholdClause(){jjtThis.setHasThreshold(true);}]
}
void SelectList():{}{
  <MULT>{jjtThis.setMult(true);}
  |
(ObjectName()|AggregateFunction())(","(ObjectName()|Agg
regateFunction()){jjtThis.setMultipleHead(true);})*
}
void FromClause():{}{
<FROM>FromItem()(","FromItem(){jjtThis.setMultipleHead(
true);})*
}
void FromItem():
```

```
{Token t1,t2;}{
  t1=<IDENTIFIER>{jjtThis.setIdent1(t1.image);}
  [t2=<IDENTIFIER>{jjtThis.setIdent2(t2.image);}]
}
void WhereClause():{}{
  <WHERE>SQLExpression()
}
void SQLExpression():
{Token t;}
{
SQLAndExpression()(t=<OR>{jjtThis.setOrStr(t.image);}SQ
LAndExpression())*
}
void SQLAndExpression():
{Token t;}
{
SQLUnaryLogicalExpression()(t=<AND>{jjtThis.setAndStr(t
.image);}SQLUnaryLogicalExpression())*
}
void SQLUnaryLogicalExpression():
{Token t;}
{
  LOOKAHEAD(2)ExistsClause()

|([t=<NOT>{jjtThis.setNotStr(t.image);}]SQLRelationalEx
pression(){jjtThis.setExistStatement(false);})
}
```

Listing 5.9. Segment koda JavaCC specifikacije za PFSQL obogaćen JJTree semantičkim akcijama.

Posmatrajmo sam početak specifikacije, kod ključne reči „DISTINCT“. Konstrukcija prikazana na listingu smešta ključnu reč „DISTINCT“ u odgovarajući čvor stabla, ako se prilikom parsiranja na ovu ključnu reč naiđe. To se postiže pozivom metoda *setDistinct()* nad tekućim čvorom u stablu koji se označava sa *jjtThis*. Slično, kod WHERE klauzule se u tekućem čvoru, pozivom metode *setHasWhere()*, specificira da upit ima WHERE klauzulu. Naravno, ove metode se ne generišu automatski. One su specificirane proširivanjem generisanog koda odgovarajućih klasa kojima su opisani čvorovi stabla. Na primer, u ovom slučaju je to klasa *SelectWithoutOrder*, čiji segment je prikazan na listingu 5.10.

```
public @SuppressWarnings("all") class
SelectWithoutOrder extends SimpleNode {
  public String select="";
  public String distinct="";
  public boolean hasWhere = false;
  public boolean hasGroup = false;
  public boolean hasThreshold = false;
  public String getText()  {
        String retStr = select+" "+distinct+"
"+((SelectList)jjtGetChild(0)).getText()+
        " "+((FromClause)jjtGetChild(1)).getText();
        int i=2;
        if(hasWhere){
              retStr+="
"+((WhereClause)jjtGetChild(i)).getText();
              i++;
        }
        if(hasGroup){
              retStr+="
"+((GroupByClause)jjtGetChild(i)).getText();
              i++;
        }
        if(hasThreshold){
              retStr+="
"+((ThresholdClause)jjtGetChild(i)).getText();
        }
        return retStr;
  }

  public SelectWithoutOrder(int id) {
    super(id);
  }

  public SelectWithoutOrder(ExampleParser p, int id) {
    super(p, id);
  }
...
}
```

Listing 5.10. Klasa *SelectWithoutOrder*.

Ostatak implementacije ove klase sadrži set i get metode za sve njene atribute. Među njima su i, dodatno ručno definisani, *distinct*, tipa

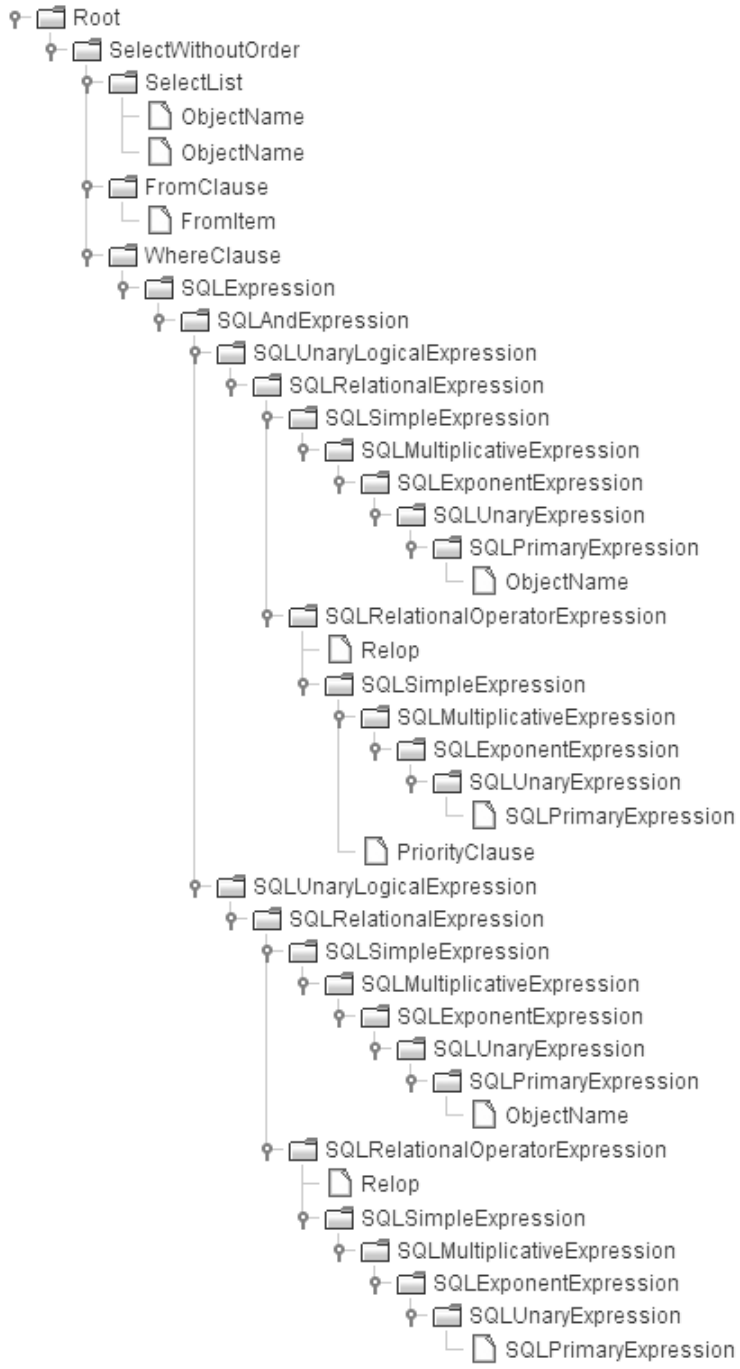
String i *where*, tipa *boolean*. Upravo pozivi set metoda za ove dve promenljive su ilustrovani i opisani u prethodnom paragrafu. Na listingu 5.10 je potrebno uočiti još jednu važnu stvar – metod *getText()*. Radi se o metodu koji služi za pretvaranje stabla parsiranja koje postoji u memoriji nazad u (PF)SQL string. Ovaj važan metod se takođe ne generiše automatski, već ga je potrebno samostalno implementirati. Takav metod je definisan za svaku vrstu čvora u stablu. Iz primera se vidi da se radi o rekurzivnom obilasku stabla – ispis stringa vezano za tekući čvor se oslanja na pozive metoda *getText()* objekata koji se nalaze ispod njega u hijerarhiji.

Stablo parsiranja se, dakle, gradi od različitih vrsta čvorova koji su instance klasa kojima su opisane odgovarajuće konstrukcije JavaCC specifikacije jezika. Osim praznih čvorova različitog tipa, ti čvorovi sadrže i sve dodatne informacije sadržane u upitu koji predstavljaju. Radi ilustracije, navodi se primer stabla parsiranja jednog upita:

```
SELECT pgl.ime,pgl.prezime
FROM PrijemniGlPodaci pgl
WHERE pgl.uspehOpsti>'#triangle(4,1,0.4)#' PRIORITY 0.8
  AND pgl.sfrPr<200
```

Kompletno stablo parsiranja ovog upita je prikazano na slici 5.5.

Implementacijom prikazanom u ovoj sekciji su dobijene metode za kojima je iskazana potreba u sekcijama 5.1 i 5.2. Opisani PFSQL parser uspešno proverava sintaksnu ispravnost unesenog PFSQL upita i, po potrebi, javlja odgovarajuće greške. Zatim, pretvara string upita u stablo parsiranja koje sadrži sve informacije sadržane u upitu. Osim toga, opisana je i važna mogućnost da se stablo parsiranja koje postoji u memoriji pretvori nazad u string upita. Ova osobina je važna zbog samog procesa obrade PFSQL upita koji, kao što je opisano, podrazumeva transformaciju stabla parsiranja i njegovo pretvaranje u string običnog SQL upita koji se šalje relacionoj bazi podataka.



Slika 5.5. Primer stabla parsiranja.

5.4.3 Transformacija stabla parsiranja

Transformacija stabla parsiranja i izračunavanje stepena zadovoljenja fazi ograničenja su najzahtevnije teme sa stanovišta njihove implementacije. One nisu predstavljene na nivou koda i komunikacije konkretnih objekata zbog toga što bi takav prikaz obilovao tehničkim detaljima i ne bi poslužio svrsi – razumevanju načina na koji su implementirana ova dva procesa. Umesto toga, prikazani su dijagrami aktivnosti koji jasno preciziraju koje akcije je potrebno preduzeti da bi se postigao cilj. Detalji implementacije tih akcija se mogu naći u obimno komentarisanom kodu i dokumentaciji koja je iz tih komentara izgenerisana, na sajtu http://www.is.im.ns.ac.yu/fuzzydb.

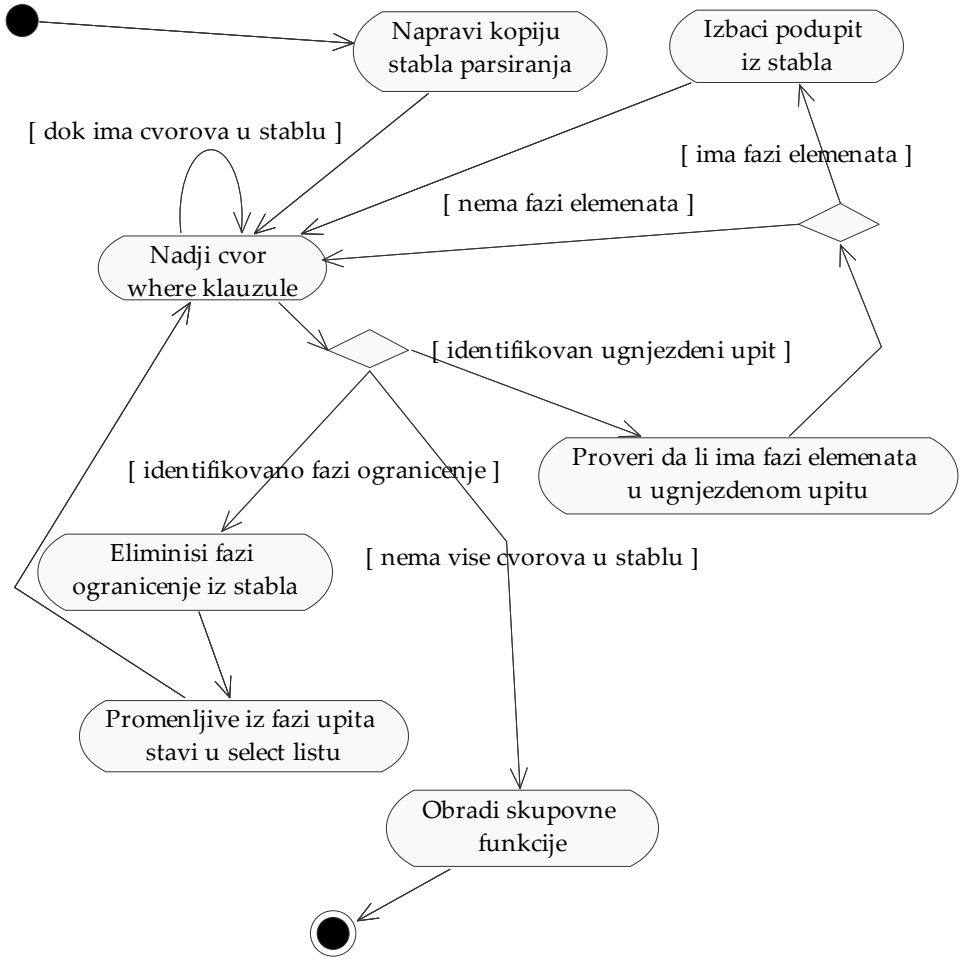
Dijagram aktivnosti koji opisuje proces transformacije PFSQL upita u SQL upit je dat na slici 5.6.

Kako je izvorno stablo parsiranja potrebno u procesu izračunavanja fazi stepena zadovoljenja, ono se ne menja direknto, već se menja njegova kopija. Transformacija se sastoji od rekurzivnog obilaska stabla i izmene određenih elemenata kada se identifikuju potrebe za izmenama.

Posmatra se WHERE klauzula, odnosno odgovarajući čvor u stablu kojim je ona predstavljena. Ukoliko se identifikuje da neki čvor u tom podstablu predstavlja fazi ograničenje, u skladu sa ranije iznetim metodama, on se eliminiše iz stabla. Nije dovoljno samo eliminisati takav čvor iz stabla, već je potrebno ispitati da li on sadrži fazi promenljive. Ako ih sadrži, potrebno ih je dodati u select listu u istom stablu da bi se dobila njihova vrednost iz baze podataka. Fazi ograničenje mora biti tipa vrednost, relacioni operator, vrednost. Dakle, fazi vrednosti se ne smeju kombinovati pomoću računskih operacija, inače se javlja odgovarajuća greška.

Takođe, slučaj na koji treba obratiti pažnju je i pojava ugnježdenog upita. Ako se takav upit identifikuje, onda se ispituje da li on sadrži bilo kakve fazi elemente. Ako ne sadrži, preskače se i ostavlja nepromenjen. Ako sadrži fazi elemente, onda se on izbacuje iz stabla. On će biti izvršen rekurzivno u okviru algoritma za izračunavanje fazi stepena zadovoljenja. Naravno, u skladu sa stavovima iznetim u sekciji 5.1, ugnježdeni upit sa

fazi elementima mora biti nezavisan. U suprotnom se javlja odgovarajuća greška.



Slika 5.6. Dijagram aktivnosti – transformacija stabla parsiranja.

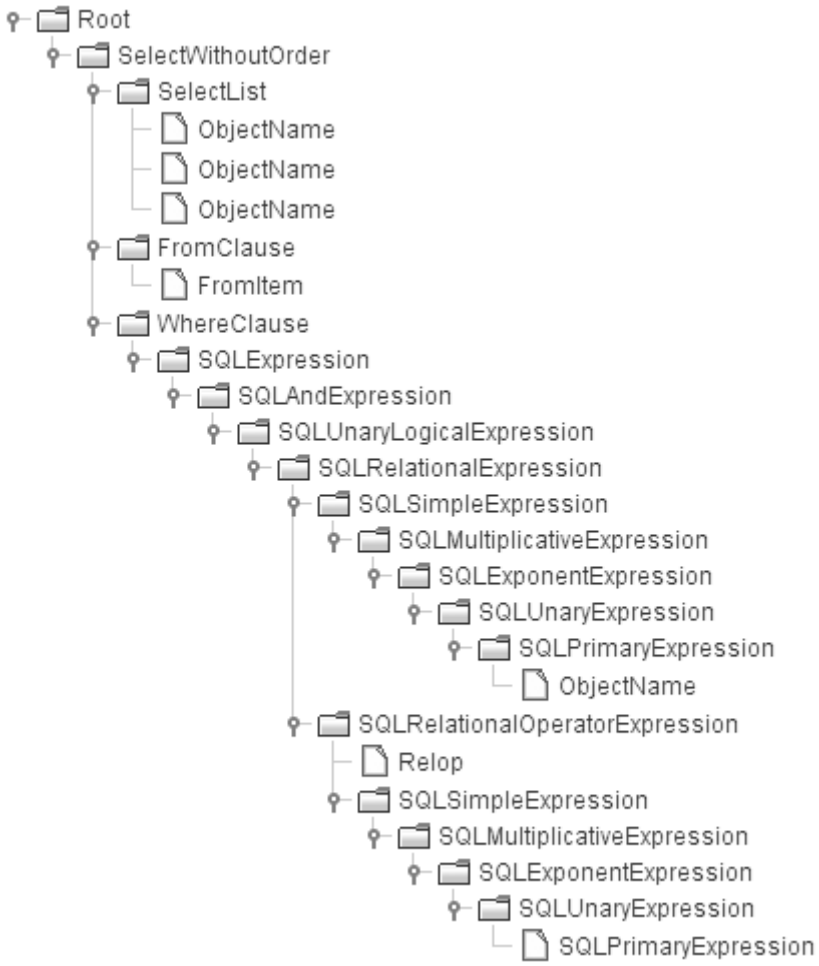
Ukoliko se u čvoru ne naiđe na fazi ograničenje niti na ugnježdeni upit, prelazi se na analizu njegove dece, sve dok se ne obiđe celo stablo.

Na kraju je potrebno obraditi još i pojavu skupovnih funkcija u select listi. Ako se utvrdi da neka od njih sadrži fazi promenljivu kao argument, onda se pojava ove skupovne funkcije u select listi zamenjuje tom fazi promenljivom. Cilj je da se dobiju vrednosti te fazi promenljive, da bi se u sledećem koraku izračunala vrednost skupovne funkcije nad tim

vrednostima. Kao što je već istaknuto, podržan je rad sa funkcijama MIN, MAX i COUNT. Ako se fazi promenljiva pojavi pod nekom drugom skupovnom funkcijom , javlja se greška.

U skladu sa iznetim, upit i stablo parsiranja prikazani u sekciji 5.4.2 i na slici 5.5 bi posle transformacije izgledali ovako:

```
SELECT pgl.ime,pgl.prezime,pgl.uspehOpsti
FROM PrijemniGlPodaci pgl
WHERE pgl.sfrPr<200
```



Slika 5.7. Stablo sa slike 5.5 posle transformacije.

5.4.4 Izračunavanje fazi stepena zadovoljenja

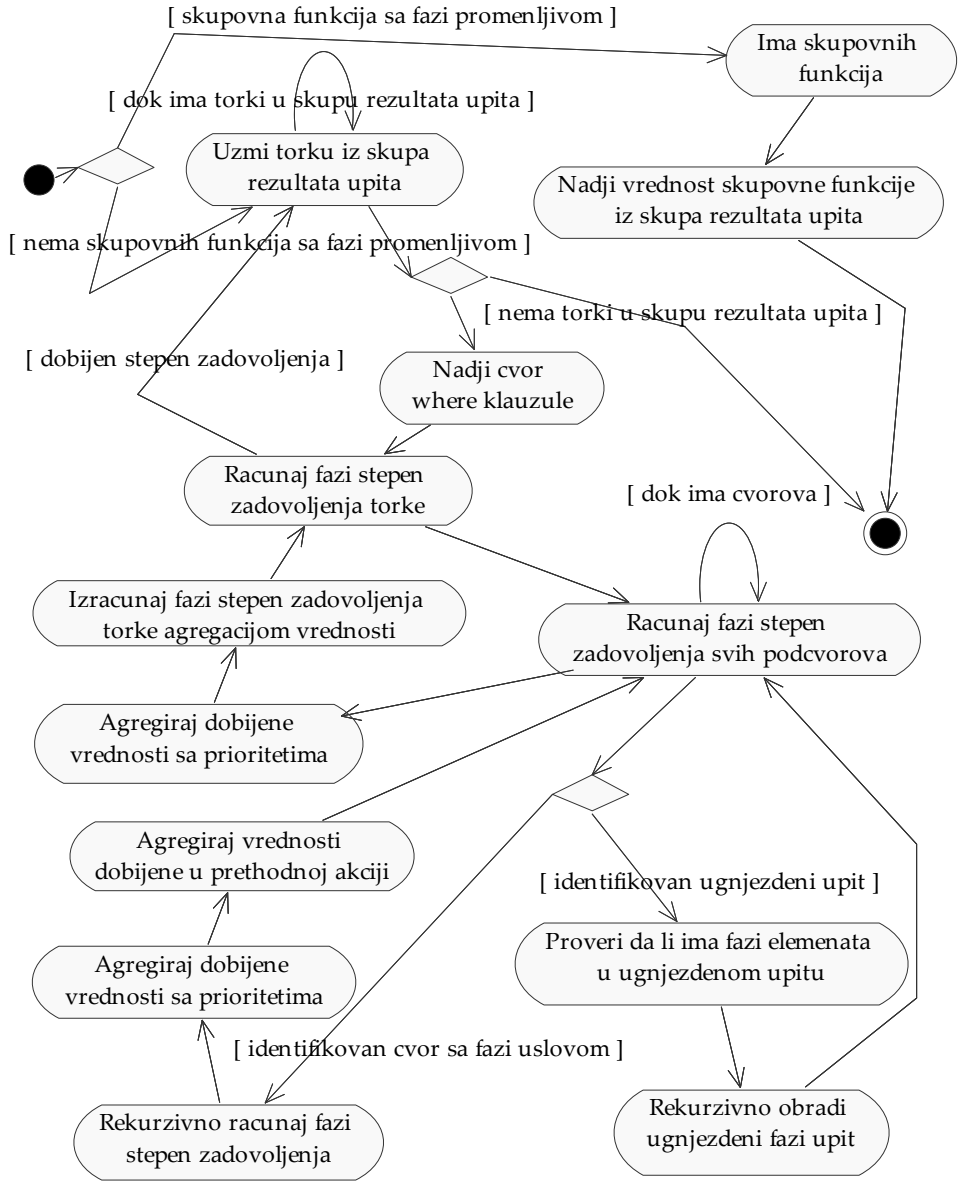
Posle transformacije stabla opisane u prethodnoj sekciji, ono se pretvara u SQL string i šalje bazi podataka na izvršavanje. Baza podataka vraća skup rezultata upita koji se, u opštem slučaju, sastoji od više torki. U tom momentu je potrebno proći kroz svaku pojedinačnu torku i dodeliti joj fazi stepen zadovoljenja upita. U slučaju da se radi o upitu sa skupovnom funkcijom koja ima fazi promenljivu kao argument, potrebno je izračunati vrednost te skupovne funkcije.

Dijagram aktivnosti koji detaljnije opisuje ovaj proces je dat na slici 5.8.

U slučaju da u upitu figuriše skupovna funkcija sa fazi promenljivom kao argumentom, analizira se skup rezultata upita. Ako se radi o skupovnoj funkciji COUNT, onda se prosto prebroji koliko torki ima u skupu. U slučaju da se radi o funkcijama MIN ili MAX, fazi vrednosti se međusobno porede da bi se našao minimum ili maksimum. Taj minimum ili maksimum može da se sastoji i od više vrednosti obzirom na to da algoritam za poređenje opisan u sekciji 2.1.6 nije totalni poredak.

Ako skupovne funkcije nema, ulazi se u drugu granu algoritma koja na osnovu skupa rezultata upita i početnog stabla parsiranja svakoj torki dobijenoj od baze podataka dodeljuje fazi stepen zadovoljenja ograničenja. Na početku se u stablu parsiranja pronalazi čvor koji odgovara WHERE klauzuli, da bi od njega počelo računanje.

Rekurzivno se računa fazi stepen zadovoljenja svih potčvorova. U tom procesu se može naići na čvor sa fazi uslovom i na ugnježđeni upit koji sadrži fazi elemente. Izračunavanje vrednosti fazi uslova je prikazano u nastavku na odvojenom dijagramu aktivnosti. Izračunata vrednost se agregira sa svojim prioritetom, ako je takav dodeljen. U skladu sa rezultatima iznetim u sekciji 2.2.3, za te potrebe se koristi trougaona konorma S_P, tako što se za dobijenu vrednost fazi uslova x računa $S_P(x,1-p)$, gde je p odgovarajući prioritet. Dalje, u zavisnosti od vrste čvora, tako dobijene vrednosti se agregiraju koristeći T_L u slučaju operatora AND i S_L u slučaju operatora OR. Kada se naiđe na negaciju, jednostavno se negirana vrednost oduzme od broja 1: $N(x) = 1 - x$.



Slika 5.8. Dijagram aktivnosti – izračunavanje fazi stepena zadovoljenja torki u skupu rezultata upita.

U slučaju nailaska na ugnježdeni upit sa fazi elementima, on se izvršava nezavisno, rekurzivno, na isti način kao što se izvršava i nadređeni upit. Dobijeni rezultati se potom koriste u daljem računanju

vrednosti čvora u kome se naišlo na takav upit. Naime, po sintaksnoj specifikaciji jezika PFSQL, ispred ugnježdenog upita se mogu pojaviti operatori EXISTS, ALL, ANY i IN. U slučaju operatora EXISTS se prosto proverava da li je skup rezultata ugnježdenog upita prazan ili ne. U slučaju operatora IN je potrebno proveriti da li se, u opštem slučaju, fazi vrednost sa leve strane nalazi u dobijenom skupu rezultata upita. Na kraju, u slučaju operatora ALL i ANY, potrebno je vrednost sa leve strane porediti sa svim vrednostima iz skupa rezultata ugnježdenog upita. Taj zadatak se svodi na izračunavanje vrednosti fazi uslova koje je opisano u nastavku.

Očigledno, za potrebe izračunavanja fazi stepena zadovoljenja je bilo potrebno implementirati metode koje izračunavaju vrednost trougaone norme T_L i konorme S_P koje se koriste. Međutim u slučaju potrebe za korišćenjem drugih trougaonih normi i konormi, dovoljno je napisati metode za njihovo izračunavanje i zameniti postojeće metode tim metodama.

Ostaje još da se razjasni način izračunavanja vrednosti fazi stepena zadovoljenja fazi uslova. Pod fazi uslovom se podrazumeva izraz sa dva operanda i relacionim operatorom između njih. Relacioni operatori definisani gramatikom jezika PFSQL se mogu podeliti u dve grupe:

- grupa znakova koja sadrži operatore =, !=, #, <> i
- grupa znakova koja sadrži operatore <, >, <=, >=.

Operandi mogu biti fazi i crisp promenljive, kao i fazi i crisp konstante. U skladu sa fazi tipovima podataka koje podržava model za skladištenje fazi vrednosti, fazi konstante mogu imati sledeće oblike:

- triangle(max, leftOffset, rightOffset) – trougaoni fazi broj sa maksimumom u max i levim i desnim otklonom leftOffset i rightOffset,
- trapezoid(leftMax, rightMax, leftOffset, rightOffset) – trapezoidni fazi broj sa maksimumom na intervalu [leftMax, rightMax] i levim i desnim otklonom leftOffset i rightOffset,
- interval(left, right) – interval sa levom i desnom granicom left i right,

- fq(left, right, isIncreasing) – fazi veličina koja raste/opada od left do right, a isIncreasing označava da je fazi veličina rastuća ako je veći od nule, odnosno da je opadajuća ako je manji od nule,
- ling(name) – lingvistička labela sa nazivom name.

Dijagram aktivnosti prikazan na slici 5.9 opisuje proces izračunavanja vrednosti fazi stepena zadovoljenja pod iznetim pretpostavkama.

Slika 5.9. Dijagram aktivnosti – izračunavanje vrednosti fazi stepena zadovoljenja fazi uslova.

Na početku procesa se ispituju vrste operanada da bi se utvrdilo da li se radi o fazi ili crisp konstantama ili promenljivama. Na osnovu te analize se u sledećem koraku kreiraju odgovarajući objekti kojima su

predstavljeni operandi. Dakle, za svaku vrstu operanda je kreirana odgovarajuća klasa koja omogućava skladištenje odgovarajućih vrednosti. Na primer, u slučaju trougaonog fazi broja, skladište se vrednosti leftOffset, max i rightOffset. U slučaju konstanti se ovi objekti kreiraju jednostavnim parsiranjem stringa, dok se u slučaju promenljivih vrednosti uzimaju iz skupa rezultata upita. Ako je iz baze podataka tražena vrednost fazi promenljive, skup rezultata upita sadrži samo vrednost valueID iz tabele FuzzyValue (sekcija 4.3), pa je za dobijanje konkretnih vrednosti potrebno konsultovati fazi meta model u bazi podataka.

Kada su odgovarajući objekti vezani za operande kreirani, ispituje se kojoj grupi pripada operator. U slučaju da pripada prvoj grupi (jednako/različito), potrebno je primeniti algoritam izračunavanja kompatibilnosti dva fazi skupa. Ako pripada drugoj grupi, primenjuje se algoritam za poređenje dva fazi broja.

Izračunavanje kompatibilnosti dva fazi skupa (sekcija 2.1.5) je jednostavan koncept, ali je implementacija tehnički zahtevna. Moraju se implementirati pravila za izračunavanje preseka svake dve vrste fazi skupova koje su podržane. Tako, na primer, klasa koja služi za reprezentaciju trougaonih fazi brojeva sadrži metode za izračunavanje kompatibilnosti sa trapezoidnim fazi brojem, intervalom, fazi veličinom i, naravno, crisp vrednosti. Isto važi i za ostale podržane vrste fazi skupova. Rezultat ovakvog izračunavanja je broj iz jediničnog intervala.

U slučaju poređenja dva fazi skupa, izabrani algoritam (sekcija 2.1.6) može da vrati:

- vrednost 1, ako je prvi skup manji ili jednak od drugog,
- vrednost 0, ako prvi skup nije manji ili jednak od drugog,
- grešku, ako dva fazi skupa nisu uporedivi.

Zbog toga se u slučaju neuporedivih fazi skupova vraća vrednost 0.5.

5.5 PFSQL I FSQL

U prethodnom poglavlju je model za skladištenje fazi podataka poređen sa jednim od najnaprednijih poznatih modela - FIRST-2 modelom.

Kao što je model za skladištenje fazi podataka opisan u prethodnom poglavlju osnova za izradu PFSQL interpretera, tako je FIRST-2 osnova nad kojom je implementiran jezik FSQL – jedan od najnaprednijih postojećih fazi SQL jezika (Galindo, Urrutia, & Piattini, 2006). Zbog toga je FSQL uzet kao referenca sa kojom se poredi PFSQL. Situacija je ovde malo drugačija zbog toga što PFSQL nije podskup FSQL jezika, čak ni funkcionalno. PFSQL podrazumeva mogućnosti koje ne postoje u jeziku FSQL, dok, sa druge strane FSQL, sadrži neke svoje specifičnosti koje nisu podržane u PFSQL-u. Naime, FSQL, kao robustan jezik podržava fazi promenljive i konstante, poredak među fazi vrednostima, fazi skupovne funkcije i druge osobine koje podržava i PFSQL. Međutim, PFSQL je jedinstven po konceptu prioriteta po kojem je i dobio ime. Koliko je autoru poznato, ne postoji nijedan sličan jezik koji nudi mogućnost za zadavanje prioriteta fazi uslovima koji se kao takvi kombinuju pomoću logičkih operatora.

Poredeći mehanizam obrade upita predstavljenog PFSQL jezika i FSQL jezika, nailazi se na vrlo značajne razlike. Interpreter za FSQL je implementiran u Oracle PL/SQL jeziku, tako da se on izvršava u okviru sistema za upravljanje bazom podataka u vidu uskladištenih procedura i funkcija. Dakle, FSQL upit se šalje direktno bazi podataka koja ga onda obrađuje unutar sebe, pomoću sopstvenih mehanizama i mehanizama implementiranim u okviru skladištenih procedura i funkcija. Nažalost, detalji implementacije FSQL interpretera se ne mogu naći u literaturi u smislu u kome su ovde dati za jezik PFSQL. To onemogućava dalju analizu sličnosti i razlika u mehanizmima implementacije pojedinih osobina. U slučaju PFSQL jezika, upit se obrađuje van baze podataka. Sam upit se šalje u bazu podataka koristeći fazi JDBC drajver koji u sebi sadrži mehanizme za obradu upita. Dakle, upiti se obrađuju van baze podataka, ona se koristi samo za izvršavanje običnih SQL upita.

Može se zaključiti da je pristup prikazan u ovoj disertaciji opštiji zbog toga što je nezavisan od baze podataka koja se koristi. Osim toga, pošto se radi o programskom jeziku Java, PFSQL interpreter može da se pokrene na svakoj platformi za koju postoji implementacija Java virtuelne mašine. U sedmom poglavlju se čak razmatraju i mogućnosti za upotrebu ovog interpretera na srednjem sloju višeslojnih informacionih sistema. Sa druge strane, žrtvuju se performanse. Izvršavanje upita u okviru baze

podataka koristeći PL/SQL je značajno brži metod, što može imati uticaja na izvršavanje kompleksnih upita koji kao rezultat vraćaju veću količinu podataka. Takođe, mehanizmi koje nudi PL/SQL podrazumevaju bolju kontrolu nad podacima i strukturom baze podataka zbog toga što se kod izvršava unutar baze podataka, a ne izvan nje.

5.6 UPOTREBA FAZI MEHANIZAMA U XML NATIVE BAZAMA PODATAKA

Kao posebno interesantan pravac istraživanja koji se oslanja na rezultate iznete u ovoj disertaciji, otvara se mogućnost upotrebe mehanizama fazi logike u XML native bazama podataka. Ova proširenja mogu da se posmatraju sa dva aspekta. Prvi je proširivanje same strukture XML dokumenata tako da može da prihvati fazi vrednosti. Pokušaji ove vrste već postoje, na primer (Ma, 2005) i (Ma & Yan, 2007), ali njih je potrebno revidirati i posmatrati sa aspekta proširenja relacionog modela fazi strukturama prikazanog u četvrtom poglavlju. Na osnovu ovakvih proširenja se može definisati fazi XML meta model podataka koji bi pratio fazi XML sadržaje i bliže opisivao podatke u njima, dok bi sa druge strane omogućavao izvršavanje upita sa fazi elementima.

Dakle, drugi aspekt upotrebe fazi logike u XML native bazama podataka je fazifikacija upitnog jezika XQuery kao de facto standarda. Iako jezik XQuery ima dosta sličnosti sa SQL-om, između njih postoje značajne razlike, pa je predstavljene ideje za proširenje jezika SQL potrebno u velikoj meri izmeniti i prilagoditi.

Uspešna implementacija interpretera za ovakav jezik se može sprovesti na sličan način kao i kod PFSQL-a. Potrebno je proširiti API za pristup XML native bazama podataka mehanizmima za obradu fazi elemenata u upitu. To proširenje mora da se oslanja na fazi XML meta podatke radi izračunavanja rezultata upita.

Zaključak je da se isti principi koji se koriste u ovoj disertaciji za proširenje relacionog modela podataka i SQL jezika mogu primeniti i u slučaju XML native baza podataka.

Poglavlje 6

CASE ALAT ZA RAD SA FAZI BAZAMA PODATAKA

U prvom delu ovog poglavlja je data specifikacija funkcionalnih zahteva aplikacije za modeliranje fazi-relacionih baza podataka pomoću dijagrama slučajeva korišćenja. Drugi deo sadrži kompletnu specifikaciju klasa i paketa korišćenih za realizaciju tih zahteva. Na kraju su opisani vitalni segmenti dinamičkog modela aplikacije.

6.1 MODELIRANJE

6.1.1 Specifikacija zahteva

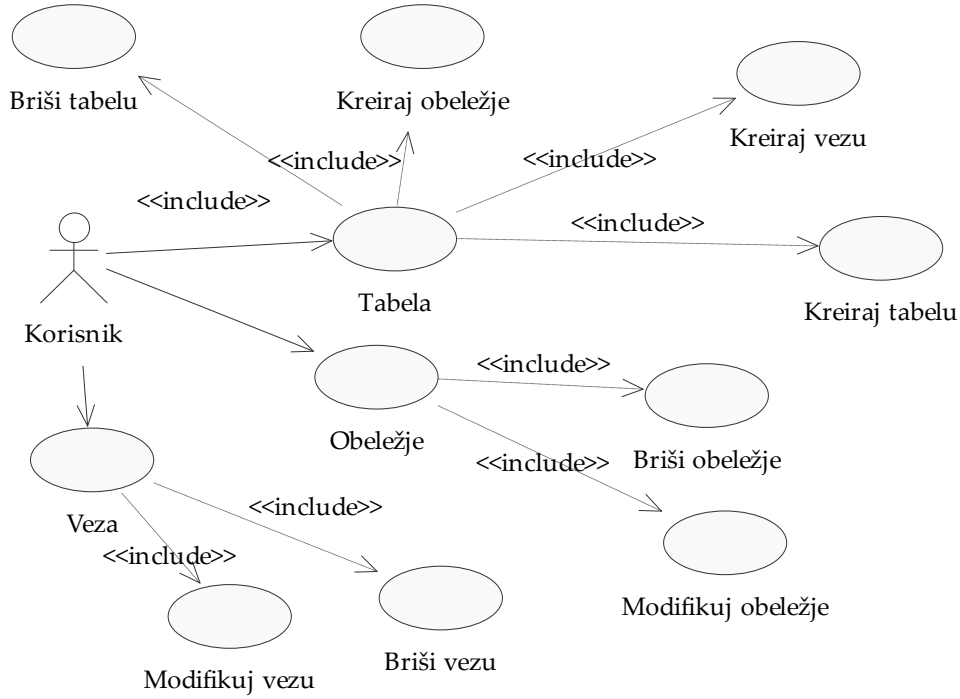
Opis funkcionalnih zahteva aplikacije je prikazan pomoću slučajeva korišćenja, po UML notaciji. Uočeno je ukupno devetnaest slučajeva korišćenja koji su raspoređeni na dva dijagrama. Prvi dijagram opisuje funkcije aplikacije povezane sa samom izradom fazi-relacionog modela, dok se drugi odnosi na rad sa repozitorijumom i generisanje SQL skripta.

6.1.1.1 Funkcije za izradu fazi relacionog modela

Funkcije za izradu fazi-relacionog modela podrazumevaju mogućnost rada sa tabelama, kao i njihovim obeležjima i međusobnim vezama. Dijagram slučajeva korišćenja kojim su opisane ove funkcionalnosti prikazan je na slici 6.1.

Slučajevi korišćenja *Tabela*, *Obeležje* i *Veza* predstavljaju uopštene slučajeve korišćenja kojima su modelirani svi zahtevi vezani za

manipulaciju ovim objektima. Zahtevi povezani sa radom sa tabelama su opisani slučajevima korišćenja koje uključuje slučaj korišćenja *Tabela*: *Kreiraj tabelu*, *Briši tabelu*, *Kreiraj obeležje* i *Kreiraj vezu*. Slučaj korišćenja *Obeležje* sadrži kao svoje podslučajeve *Briši obeležje* i *Modifikuj obeležje*. Slično važi i za slučaj korišćenja *Veza*, njegovi podređeni slučajevi korišćenja su *Briši vezu i Modifikuj vezu*.



Slika 6.1. Funkcije za izradu fazi-relacionog modela.

U okviru slučaja korišćenja *Kreiraj tabelu* je modeliran jednostavan proces kreiranja tabele. Ona se može kreirati u svakom momentu prostim zadavanjem imena. Kreirana tabela se smešta u repozitorijum, a potom se omogućava ispunjavanje njenog sadržaja.

Slučajem korišćenja *Briši tabelu* je modeliran proces brisanja tabele koja postoji u repozitorijumu. Ovaj proces uključuje brisanje same tabele, ali i njenih veza sa drugim tabelama. Prvo se brišu veze ove izabrane tabele sa drugima, što podrazumeva brisanje obeležja drugih tabela koja potiču iz izabrane tabele (stranih ključeva). Potom se briše sama veza, koja postoji

kao objekat u repozitorijumu, a zatim i obeležja, kao i sama tabela. Rezultat izvršavanja ovakve operacije je odsustvo bilo kakvih informacija o obrisanoj tabeli u repozitorijumu.

Izvršavanju slučaja korišćenja *Kreiraj obeležje* prethodi izbor tabele kojoj će buduće obeležje pripadati. U okviru ovog slučaja korišćenja je predviđeno da se zada samo naziv budućeg obeležja, dok se njegove osobine zadaju u okviru slučaja korišćenja *Modifikuj obeležje*. Po zadavanju imena, obeležje se upisuje u podatke o tabeli koja se već nalazi u repozitorijumu.

Izvršavanje slučaja korišćenja *Kreiraj vezu*, kao i kod prethodnog, podrazumeva izbor tabele-roditelja kojoj će pripadati buduća veza. Zadaje se naziv same veze, ali i tabela-dete sa kojom se ova veza ostvaruje. Kreiranje veze podrazumeva prevlačenje primarnog ključa tabele-roditelja u tabelu-dete. Dakle, tabela-dete dobija nova obeležja povezana sa tabelom-roditeljem ograničenjem stranog ključa. Zadavanje ostalih parametara veze je opisano slučajem korišćenja *Modifikuj vezu*.

Slučaj korišćenja *Obeležje* sadrži opis funkcionalnosti vezanih za upravljanje već postojećim obeležjima. Preduslov za izvršavanje slučaja korišćenja *Briši obeležje* je izbor odgovarajućeg obeležja. Posle zadavanja komande, obeležje se briše iz svoje matične tabele u repozitorijumu. Isto tako, brišu se sve pojave tog obeležja koje su prevučene kao strani ključevi u druge tabele. Ako je izbrisano obeležje i poslednje koje neka veza prenosi u drugu tabelu, briše se i svaka takva veza.

U okviru slučaja korišćenja *Modifikuj obeležje* su modelirane funkcionalnosti vezane za zadavanje različitih parametara obeležja. U svakom momentu je moguće promeniti naziv obeležja. Takođe, predviđena je mogućnost da se odredi da li se radi o obeležju koje može da ima nula vrednost. Moguće je zadati ili izmeniti tip obeležja. Rad sa tipovima koji postoje u modelu je opisan u slučaju korišćenja *Unesi tipove* (slika 6.2). Poslednja mogućnost se odnosi na zadavanje opcije koja govori o tome da li se radi o obeležju koje ulazi u primarni ključ, da li nad obeležjem postoji ograničenje stranog ključa, ili je obeležje strani ključ koji ulazi u primarni ključ tabele-deteta. Izmene se vrše nad svim pojavama izabranog obeležja,

kako u matičnoj tabeli, tako i u ostalima u koje je obeležje prevučeno kao strani ključ.

Operacije koje se odnose na brisanje i izmenu osobina veze su modelirane slučajem korišćenja *Veza*. Izvršavanju slučaja korišćenja *Briši vezu* prethodi izbor odgovarajuće veze iz repozitorijuma. Iz repozitorijuma se briše sama veza, ali i sva obeležja iz tabele-deteta koja su prevučena posredstvom te veze.

Modifikuj vezu je slučaj korišćenja kojim je modelirana izmena naziva veze. Veza može biti takva da strani ključevi koje prenosi u tabelu-dete, postaju i deo primarnog ključa te tabele (identifikaciona zavisnost). Ovu osobinu je takođe moguće zadati i u tom slučaju prevučena obeležja se automatski označavaju kao delovi primarnog ključa tabele-deteta.

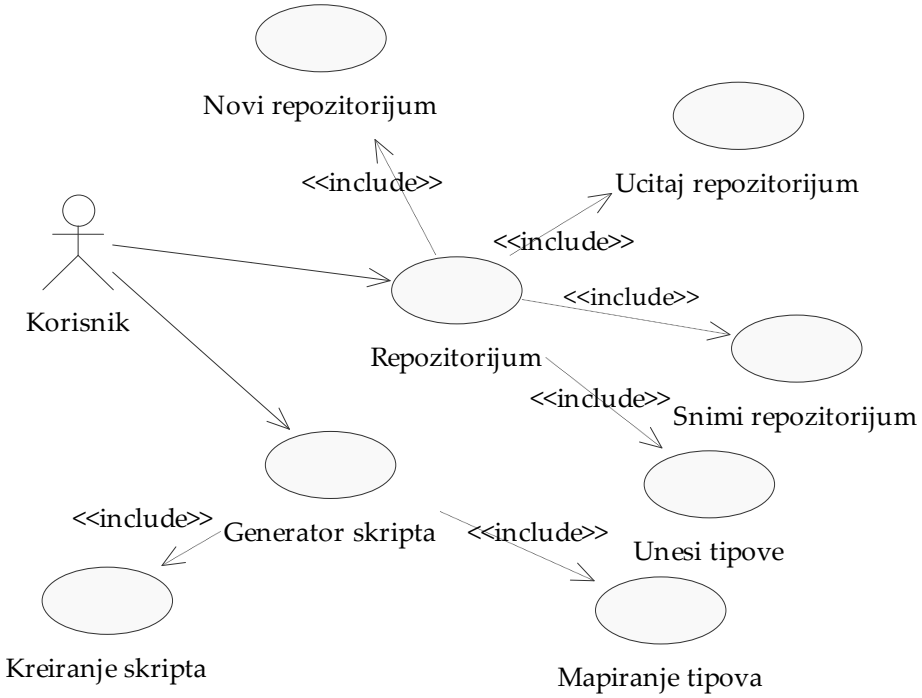
6.1.1.2 Funkcije za rad sa repozitorijumom i generisanje skripta

Funkcije za rad sa repozitorijumom su predstavljene opštim slučajem korišćenja *Repozitorijum* (slika 6.2). Njime su modelirane osnovne operacije sa repozitorijumom koje uključuju njegovo snimanje, učitavanje, kreiranje novog, ali i zadavanje tipova podataka koji postoje u modelu.

Izvršavanje slučaja korišćenja *Novi repozitorijum* rezultuje prostim brisanjem svih postojećih sadržaja.

Slučaj korišćenja *Snimi repozitorijum* se odnosi na snimanje kompletnog sadržaja modela u fajl sistem. *Učitaj repozitorijum* je slučaj korišćenja kojim je modelirano učitavanje prethodno snimljenih sadržaja u memoriju. Ova operacija podrazumeva brisanje postojećih sadržaja iz memorije.

Proces unosa tipova koji postoje u modelu je modeliran slučajem korišćenja *Unesi tipove*. Pri unosu tipa se zadaje njegov naziv, kao i indikator koji određuje da li je taj tip fazi ili ne. U sklopu ovog procesa je predviđena i mogućnost brisanja postojećih tipova iz modela.



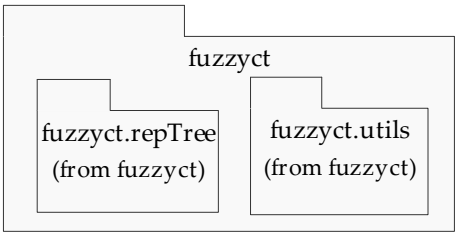
Slika 6.2. Slučajevi korišćenja vezani za repozitorijum i generator SQL skripta.

Generator skripta je opšti slučaj korišćenja kojim je modeliran proces generisanja SQL skripta na osnovu unetog modela. Preduslov za generisanje SQL skripta predstavlja unos informacija o mapiranju tipova. Naime, potrebno je zadati mapiranje tipova koji postoje u modelu na tipove koje podržava sistem za upravljanje bazama podataka koji se koristi. Za svaki tip podataka je potrebno uneti string kojim je taj tip reprezentovan u korišćenom SUBP-u. U okviru tog procesa je predviđena i mogućnost brisanja postojećih mapiranja. Ove funkcionalnosti su modelirane slučajem korišćenja *Mapiranje tipova*.

Kreiranje skripta je slučaj korišćenja kojim je modelirana operacija kreiranja SQL skripta na osnovu postojećeg modela. Kao što je već napomenuto, preduslov za izvršavanje ove operacije je postojanje mapiranja svakog tipa koji postoji u modelu na odgovarajući tip u bazi podataka. Kreirani SQL skript se snima u izabrani fajl u fajl sistemu.

6.1.2 Dijagrami klasa i paketa

Na osnovu opisanih slučajeva korišćenja su uočene klase i njihovi međusobni odnosi potrebni za realizaciju aplikacije. U ovoj sekciji će biti prikazani dijagrami klasa i paketa koji predstavljaju kompletan statički model sistema. Detalji implementacije ovih klasa su dati u narednom poglavlju.



Slika 6.3. Paketi.

Na slici 6.3 su prikazani paketi od kojih se sastoji model aplikacije. Osnovni paket, fuzzyct, sadrži kao svoje potpakete fuzzyct.utils i fuzzyct.repTree koji predstavljaju pomoćne resurse.

6.1.2.1 Paket fuzzyct

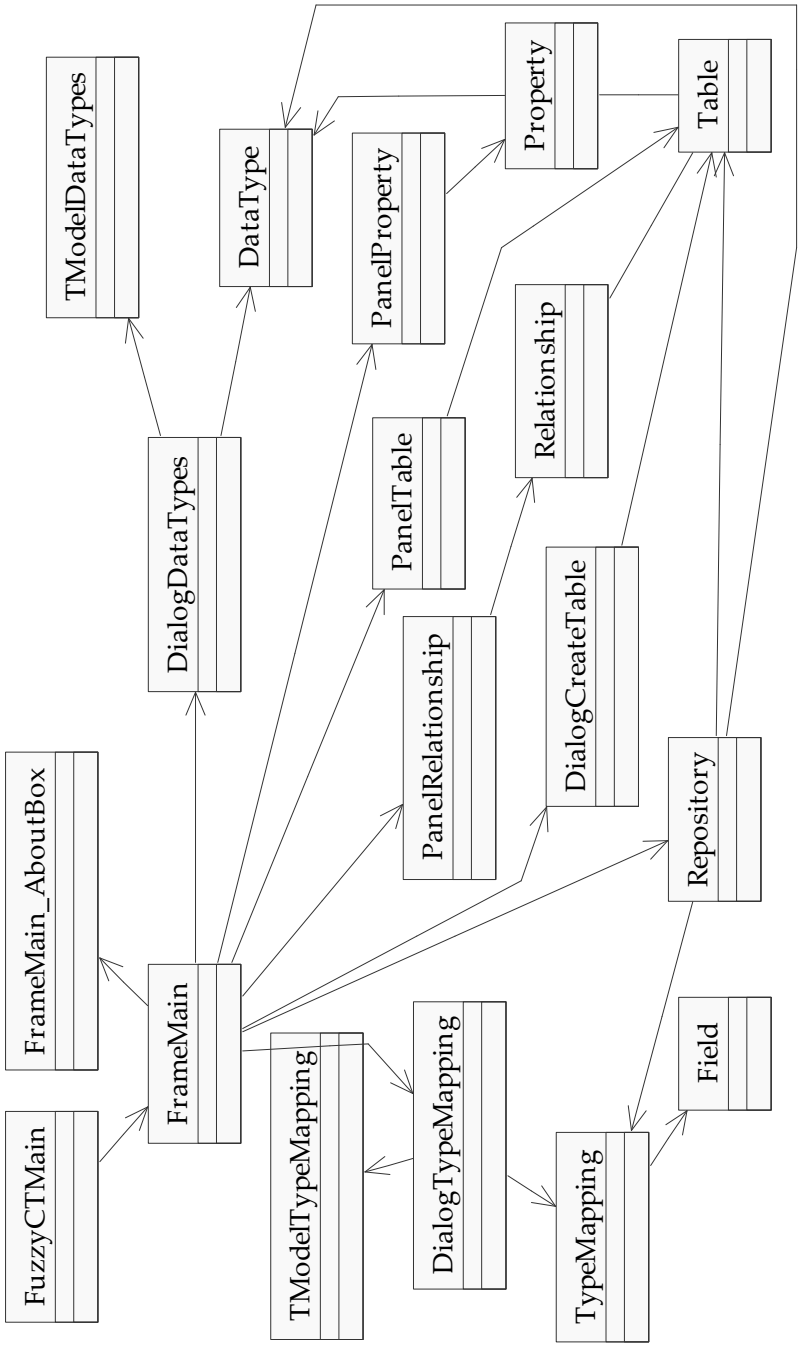
Dijagram klasa paketa fuzzyct, dat je na slici 6.4. Klase su grupisane u četiri celine da bi se dobila bolja preglednost. Prvu celinu čine klase *FrameMain*, *FuzzyCTMain* i *FrameMain_AboutBox.* Prva klasa predstavlja osnovnu klasu u aplikaciji i implementira glavni prozor. Ona kontroliše izgled i ponašanje grafičkog interfejsa kroz implementaciju većeg broja listener-a i handler-a za događaje koji se dešavaju u okviru glavnog prozora. Druga klasa služi za izvršavanje osnovne inicijalizacije i pokretanje glavnog prozora na samom startu aplikacije. Klasa *FrameMain_AboutBox* predstavlja prozor sa osnovnim informacijama o aplikaciji.

Druga celina se sastoji od klasa *DialogDataTypes*, *TModelDataTypes* i *DataType*. Prve dve klase služe za grafički prikaz tipova podataka koji postoje u modelu. Klasa *DialogDataTypes* sadrži i mogućnosti za zadavanje novih i brisanje postojećih tipova korišćenjem grafičkog interfejsa. Klasa

DataType modelira svaki pojedinačni tip podataka koji postoji u modelu. Osim podataka o svakom tipu, ona sadrži i metode koje služe za upravljanje tipovima podataka u modelu.

Klase koje sadrže podatke o modelu su smeštene u treću celinu: *Repository*, *Table*, *Property*, *Relationship*, *PanelTable*, *PanelProperty*, *PanelRelationship* i *DialogCreateTable*. Sam repozitorijum je modeliran klasom *Repository*. Ova klasa sadrži reference na skup tabela koje postoje u modelu (klasa *Table*), zatim, na skup tipova (klasa *DataType*) i na kraju na mapiranje tipova iz modela na tipove baze podataka (klasa *TypeMapping*). Metode ova klase služe za upravljanje elementima repozitorijuma na nivou aplikacije. Klasom *Table* su modelirane tabele koje postoje u modelu. Pored metoda koje služe za upravljanje tabelama, ova klasa sadrži atribut kojim je označen naziv tabele, kao i reference na skup obeležja (klasa *Property*) i veza (klasa *Relationship*) koje pripadaju tabeli. Obeležja koja postoje u fazi-relacionom modelu su predstavljena klasom *Property*. Njenim atributima su modelirani svi parametri koje obeležje može da ima: naziv i tip obeležja, ograničenja primarnog i stranog ključa, kao i oznaku da li vrednost obeležja može ostati prazna. Takođe, ova klasa implementira veći broj metoda koje služe za upravljanje obeležjima. Preostala komponenta modela, veze između tabela, je predstavljena klasom *Relationship*. Ona, pored naziva veze, sadrži reference na tabelu-roditelja i tabelu-dete koje povezuje. Takođe, postoji informacija o tome da li je tabela-dete identifikaciono zavisna od tabele-roditelja. Klase *PanelTable*, *PanelProperty* i *PanelRelationship* predstavljaju vizuelnu reprezentaciju podataka o tabelama, obeležjima i vezama, respektivno. Ovi paneli su vizuelne komponente koje omogućavaju zadavanje, modifikaciju i brisanje parametara svakog od tih objekata. *DialogCreateTable* je vizuelna komponenta pomoću koje korisnik može da kreira novu tabelu u repozitorijumu zadajući njen naziv.

Poslednju, četvrtu celinu čine klase kojima su modelirani podaci i procesi vezani za mapiranje tipova podataka. Ona sadrži četiri klase: *Field*, *TypeMapping*, *DialogTypeMapping* i *TModelTypeMapping*. Klasom *Field* je predstavljen jedan par čija je prva komponenta tip podataka iz modela, a druga komponenta njegov odgovarajući tip u bazi podataka.

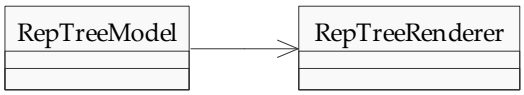


Slika 6.4. Dijagram klasa - paket fuzzyct.

Referencu na skup objekata klase *Field* sadrži klasa *TypeMapping* - osnovna klasa kojom su predstavljene informacije o mapiranju u modelu. Klase *DialogTypeMapping* i *TModelTypeMapping* služe za grafički prikaz i manipulaciju podacima o mapiranju tipova podataka. Klasa *FrameMain*, sem gore opisanih funkcionalnosti, sadrži i važan metod koji, koristeći objekte klasa kojima su predstavljeni repozitorijum i mapiranje tipova, generiše SQL skript koji odgovara unesenom modelu.

6.1.2.2 Paket fuzzyct.repTree

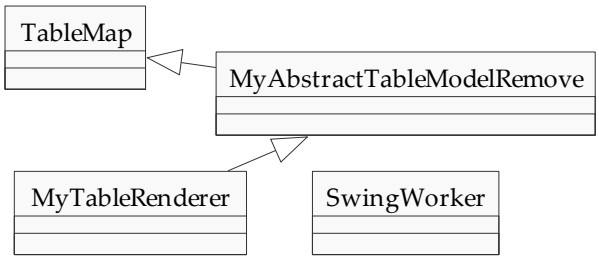
Paket *fuzzyct.repTree* sadrži samo dve klase: *RepTreeModel* i *RepTreeRenderer* (slika 6.5). Ove klase omogućavaju prikaz objekata koji postoje u repozitorijumu pomoću navigacionog stabla na grafičkom interfejsu. Klasa *RepTreeModel* implementira metode nad podacima u repozitorijumu koje su potrebne za iscrtavanje stabla, dok se klasa *RepTreeRenderer* bavi izgledom samog stabla.



Slika 6.5. Dijagram klasa - paket fuzzyct.repTree.

6.1.2.3 Paket fuzzyct.utils

Paket fuzzyct.utils sadrži klase koje nisu usko vezane za logiku aplikacije, ali su potrebne za realizaciju različitih delova aplikacije (slika 6.6). Podešavanje izgleda tabela na grafičkom interfejsu se vrši pomoću prve tri klase: *MyTableRenderer*, *MyAbstractTableModelRemove,* i *TableMap*. *SwingWorker* je javno dostupna klasa koja omogućava kontrolu grafičkog interfejsa paralelno sa nekim procesom čije izvršavanje je u toku.



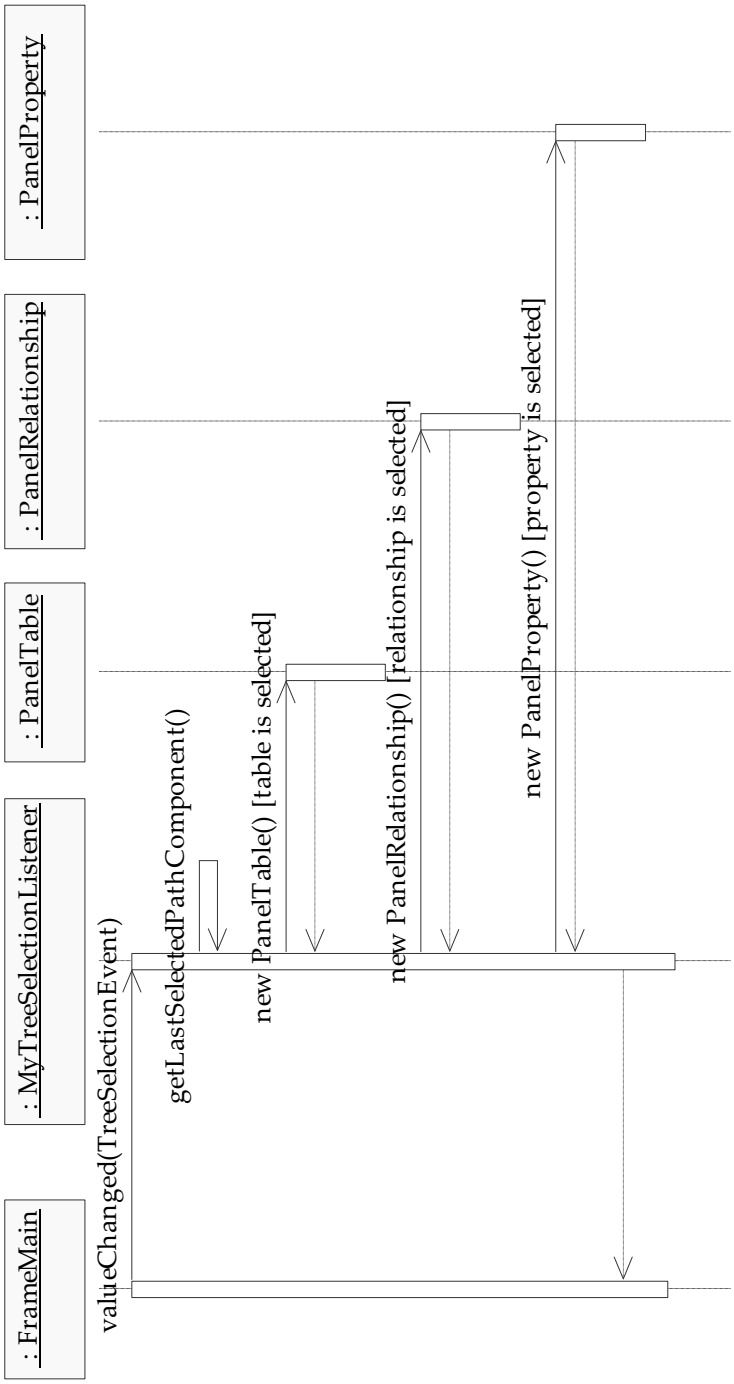
Slika 6.6. Dijagram klasa – paket fuzzyct.utils

6.1.3 Dinamički model osnovnih procesa

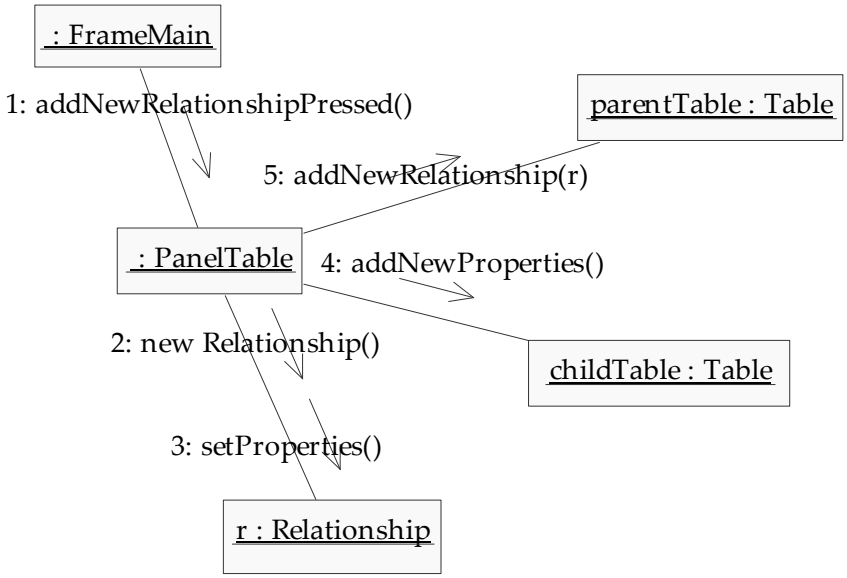
U ovom delu su opisani osnovni elementi dinamike sistema. Dijagramom sekvenci su opisani vitalni procesi vezani za rad grafičkog interfejsa (slika 6.7). Predviđeno je da se elementi modela prikažu hijerarhijski u vidu navigacionog stabla sa leve strane ekrana. U zavisnosti od selektovanog elementa u stablu bi se na desnoj strani prikazivao panel koji omogućava manipulaciju selektovanim elementom. Izgled ovakvog grafičkog interfejsa je prikazan na slici 6.11. *FrameMain* na dijagramu sekvenci predstavlja osnovi prozor, dok su panelima *PanelTable, PanelProperty* i *PanelRelationship* prikazani paneli koji služe za manipulaciju tabelama, vezama i obeležjima, respektivno. *MyTreeSelectionListener* je listener klasa koja „osluškuje" događaje koji se dešavaju na stablu.

Stablo se nalazi na glavnom prozoru (*FrameMain*). Kada se selektuje element prikazan na stablu, ono poziva metod *valueChanged* implementiran u okviru listener klase (*MyTreeSelectionListener*). Listener klasa tada poziva metod *getLastSelectedPathComponent*, koji vraća objekat koji je selektovan na stablu. Tada se, u zavisnosti od vrste objekta koji je selektovan, poziva konstruktor klasa koje predstavljaju panele za manipulaciju objektima u modelu. Po inicijalizaciji jednog od ova tri panela, i njegovom prikazivanju, kontrola toka se vraća glavnom prozoru, odnosno klasi *FrameMain*.

Procesi koji rezultuju izmenama u modelu su prikazani dijagramom slučajeva korišćenja na slici 6.1 i opisani u paragrafu 6.1.1.1. Ovi procesi će biti ilustrovani na primeru kreiranja i brisanja veze između tabela. Slika 6.8 prikazuje dijagram saradnje kojim je opisana interakcija objekata čiji je rezultat kreiranje nove veze između dve tabele. Ovaj proces se pokreće po izdavanju komande na glavnom prozoru. Potom, glavni prozor poziva metod *addNewRelationshipPressed* implementiran u okviru klase *PanelTable*. Druga prikazana poruka je poziv konstruktora klase *Relationship*, a zatim sledi i poziv metoda *setProperties* implementiran u okviru iste klase. Ovaj metod upisuje parametre veze unete na ekranskoj formi u novonastali objekat. Potom se obeležja koja čine primarni ključ tabele – roditelja, predstavljene objektom *parentTable* prenose kao strani ključevi u tabelu – dete, predstavljenu objektom *childTable*. Ovaj proces je predstavljen porukom *addNewProperties*. Na kraju se tabeli – roditelju pridružuje novonastali objekat klase *Relationship*, pozivajući metod *addNewRelationship*.



Slika 6.7. Dijagram sekvenci – rad grafičkog interfejsa.



Slika 6.8. Dijagram saradnje – kreiranje veze.

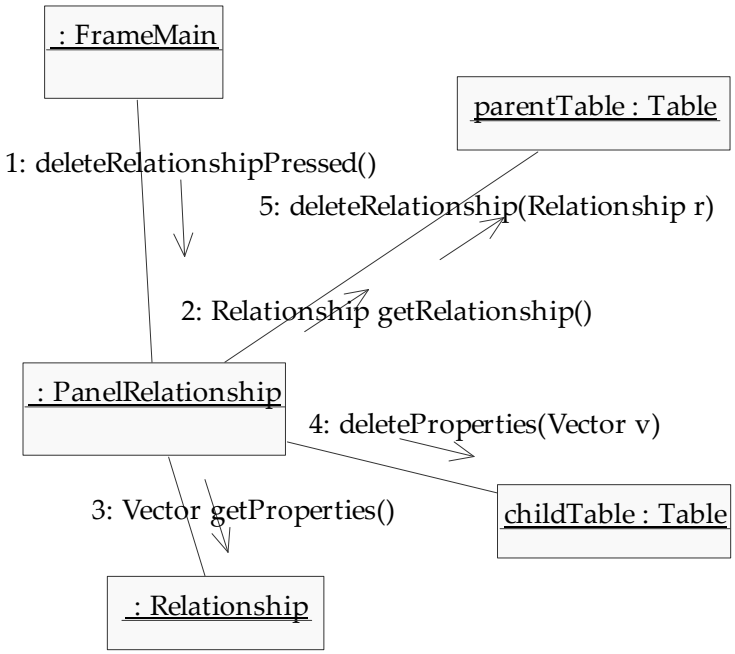
Slika 6.9 prikazuje sličan dijagram saradnje kojim je opisan proces brisanja postojeće veze iz modela. Proces se pokreće kada se izabere veza i izda komanda na glavnom prozoru (*FrameMain*). Potom, glavni prozor poziva metod *deleteRelationshipPressed* implementiran u okviru klase *PanelRelationship*. Zatim se, pozivom metoda *getRelationship*, implemetiranog u okviru klase *Table*, od tabele – roditelja (*parentTable*) preuzima objekat kojim je predstavljena izabrana veza.

Treća poruka predstavlja poziv metoda *getProperties*, implementiranog od strane klase *Relationship*, koji vraća niz obeležja (klasa *java.util.Vector*) koja su posredstvom ove veze prevučena u tabelu – dete (*childTable*). Ta obeležja se brišu iz tabele – deteta pozivom metoda *deleteProperties*. Ostaje još da se sama veza obriše iz tabele – roditelja. Ovaj proces je predstavljen porukom *deleteRelationship*.

Dijagramom aktivnosti, prikazanim na slici 6.10, su modelirane akcije koje je potrebno preduzeti da bi se iz postojećeg modela izgenerisao SQL kod koji služi za kreiranje baze podataka. Detaljan opis ovog postupka, sa opisom načina na koji su tretirana fazi obeležja, je dat u paragrafu 6.2.3. Prva aktivnost koja se odnosi na generisanje koda je vezana

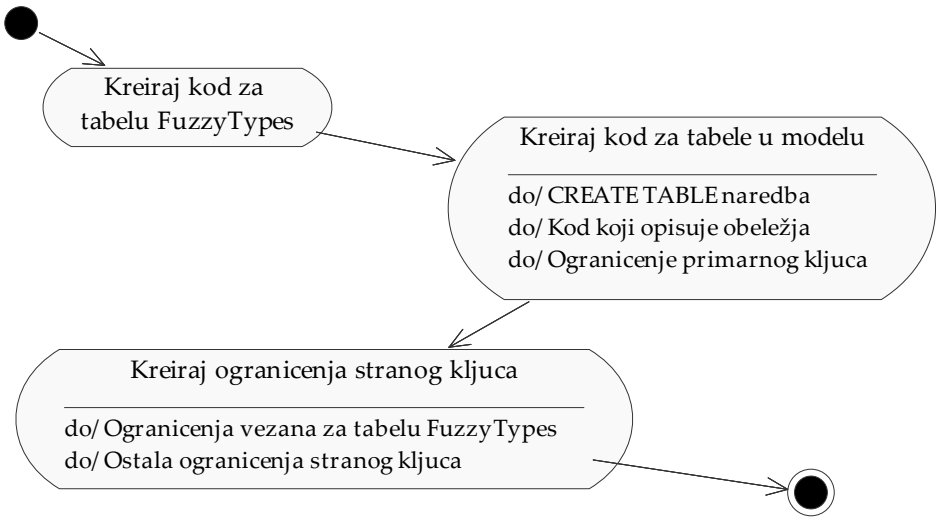
za tabelu FuzzyValue. To je specijalna tabela koja služi za bliži opis svih vrednosti fazi obeležja u modelu.

Potom sledi generisanje koda za tabele koje su unete u model. Ova aktivnost ima tri podaktivnosti. Prvo je, za svaku tabelu, potrebno otvoriti CREATE TABLE klauzulu. Zatim se, na osnovu obeležja koje ta tabela ima, u okviru CREATE TABLE klauzule generiše kod kojim se opisuje svako od tih obeležja. Kod karakterističan za fazi obeležja je opisan i ilustrovan primerom u paragrafu 6.2.3.



Slika 6.9. Dijagram saradnje – brisanje veze.

Kada su sve tabele u modelu kreirane, potrebno je još zadati ograničenja stranog ključa, i time povezati postojeće tabele. Prvo se generiše kod koji opisuje ograničenja vezana za odnos tabela iz modela sa tabelom FuzzyValue. Zatim se zadaju i ostala ograničenja stranog ključa koja postoje u modelu. Rezultat ovog procesa je niz SQL naredbi koje se zapisuju u fajl sistem u obliku tekstualnog fajla (DDL fajl).



Slika 6.10. Dijagram aktivnosti – generisanje SQL koda.

6.2. IMPLEMENTACIJA

Na osnovu specifikacije opisane u prethodnoj sekciji, implementirana je prototip aplikacija za modeliranje fazi-relacionih baza podataka. Aplikacija je implementirana u programskom jeziku Java, a korisnički interfejs, prikazan na slici 6.11, je realizovan uz pomoć platforme za razvoj grafičkog interfejsa Swing.

6.2.1 Glavna ekranska forma

Glavna ekranska forma aplikacije se sastoji od sledećih celina (slika 6.11): meni, toolbar, navigaciono stablo, glavni panel i status bar.

6.2.1.1 Meni i toolbar

Komande koje sadrži glavni meni i njegovi podmeniji su prikazane u tabeli 6.1.

[File]	[Repository]	[Table]	[Database]	[Help]
New	Data types	Create	Create SQL script	About
Open		Delete	Type mapping	
Save				
Exit				

Tabela 6.1. Sadržaj menija.

Osim komandi About i Exit, sve su dostupne i preko prečica koje se nalaze na toolbaru. Komanda About služi za prikazivanje osnovnih informacija o programu, dok Exit služi za izlazak iz programa. Sledi opis tih komandi.

 - brisanje postojećeg i kreiranje novog repozitorijuma (New),

 - učitavanje prethodno snimljenog repozitorijuma iz fajl sistema (Open),

 - snimanje repozitorijuma u fajl sistem (Save),

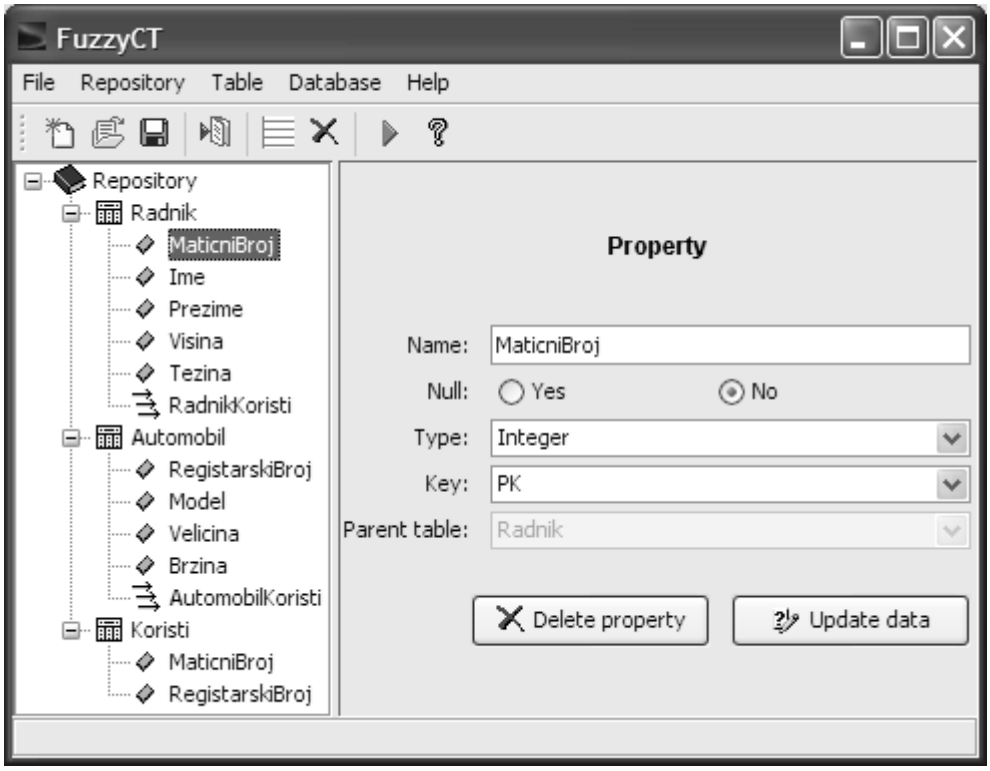
 - ažuriranje informacija o tipovima podataka koji postoje u repozitorijumu (Data types),

 - kreiranje nove tabele (Create),

 - brisanje izabrane tabele (Delete),

 - kreiranje SQL skripta na osnovu modela unetog u repozitorijum (Create SQL script) ,

 - ažuriranje pravila za mapiranje tipova podataka iz repozitorijuma na tipove podataka u konkretnom sistemu za upravljanje bazama podataka (Type mapping).



Slika 6.11. Izgled glavne ekranske forme.

6.2.1.2 Navigaciono stablo

Navigacionim stablom je vizuelno prikazana struktura kompletnog modela baze podataka. Sam repozitorijum je predstavljen sledećom slikom: . Repozitorijum u sebi sadrži skup tabela označenih sa . Tabele mogu da sadrže obeležja ( ) i međusobne veze ( ).

Stablo je implementirano nasleđivanjem klase JTree, koja predstavlja standardnu komponentu platforme Swing. Komponenta (objekat) JTree ne sadrži u sebi podatke, ona samo omogućava njihov prikaz na ekranu. Ova komponenta čita potrebne podatke iz dodeljene memorijske strukture automatski pozivajući metode namenjene za tu svrhu. Dakle, zadatak se svodi na implementaciju metoda koje će omogućiti čitanje podataka iz memorijske strukture kojom je predstavljen repozitorijum. Klase od kojih se sastoji ta memorijska struktura su

prikazane na slici 6.4, u celini broj 3: *Repository*, *Table*, *Property* i *Relationship*. Koristeći ovu strukturu, implementirane su sledeće metode:

- *Object getChild(Object parent, int index)* – metod koji vraća potomka elementa parent koje se nalazi na poziciji index,
- *int getChildCount(Object parent)* – metod koji vraća broj potomaka elementa parent,
- *int getIndexOfChild(Object parent, Object child)* – metod koji vraća redni broj potomka child u okviru elementa parent,
- *int getRoot()* – metod koji vraća koren stabla i
- *boolean isLeaf(Object node)* – metod koji vraća vrednost true ukoliko je element node list, a vrednost false u suprotnom.

Izgled elemenata stabla na ekranu se može podesiti zadavanjem različite slike za svaki od elemenata strukture. Da bi se to postiglo, bilo je potrebno naslediti klasu *DefaultTreeCellRenderer* i implemetirati odgovarajući metod – *getTreeCellRendererComponent()*.

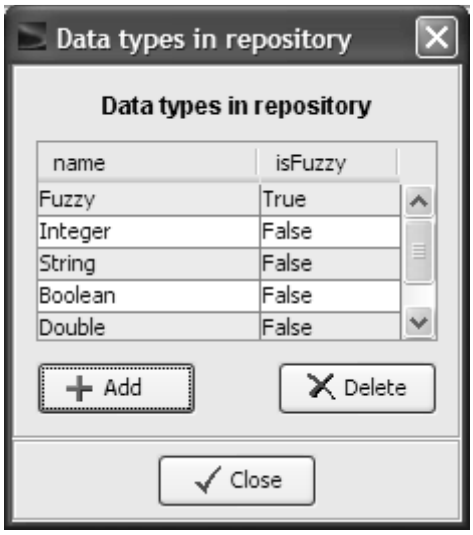
6.2.1.3 Glavni panel i status bar

Glavni panel aplikacije zauzima najveći deo prostora korisničkog interfejsa. Njegov sadržaj zavisi od vrste elementa selektovanog na navigacionom stablu. Za svaki od tih elemenata je definisan panel koji omogućava ažuriranje njegovih osobina. Detaljan prikaz ovih panela je dat u narednoj sekciji.

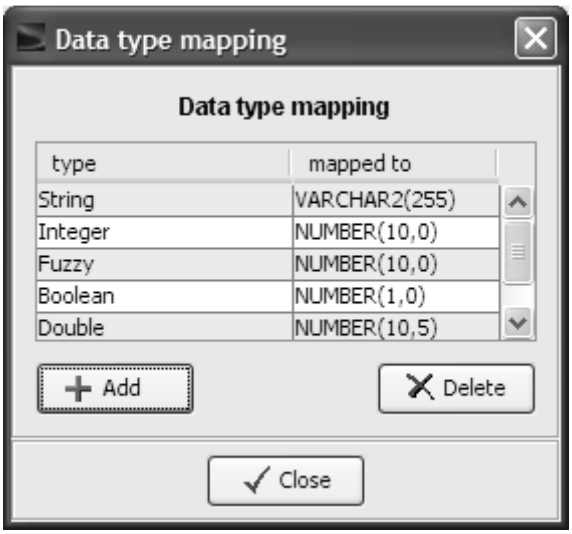
Status bar se nalazi u samom dnu prozora aplikacije i predviđen je za ispis propratnih informacija pri radu.

6.2.2 Unos modela

Direktnom unosu tabela u model prethodi unos tipova podataka koji postoje u repozitorijumu, kao i mapiranja tih tipova na one koje podržava sistem za upravljanje bazama podataka. Tip podataka Fuzzy je inicijalno unet u repozitorijum i njime se obeležavaju sva fazi obeležja. Ostale tipove podataka korisnik unosi proizvoljno koristeći ekransku formu prikazanu na slici 6.12.



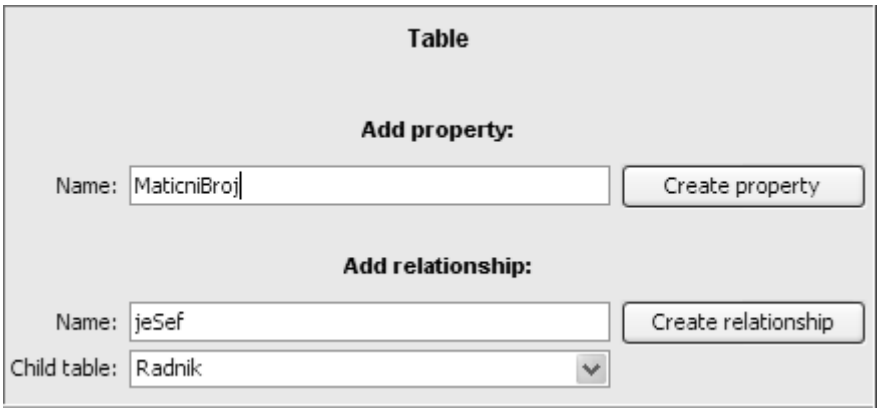
Slika 6.12. Unos tipova podataka u repozitorijum.



Slika 6.13. Unos mapiranja tipova podataka.

Ekranska forma za unos mapiranja tipova podataka je prikazana na slici 6.13. U prvoj koloni tabele se nalaze uneti tipovi podataka, a u drugoj string kojim je taj tip predstavljen u sistemu za upravljanje bazama podataka.

Klikom na komandu za kreiranje tabele se dobija ekranska forma na kojoj se unosi njen naziv. Kako je tabela koja je na taj način uneta u repozitorijum prazna, očekuje se unos obeležja i veza u nju. To se postiže preko panela koji se dobija na glavnom panelu kada se klikne na novo nastalu tabelu na navigacionom stablu. Ovaj panel je prikazan na slici 6.14.



Slika 6.14. Panel za unos podataka o tabeli.

Novo obeležje se zadaje upisivanjem njegovog imena i izdavanjem komande *Create property*. Parametri tog obeležja se unose na panelu za unos podataka o obeležju koji se dobija izborom ovoga obeležja na navigacionom stablu. Ovaj postupak je opisan kasnije u tekstu. Da bi se kreirala veza između tabela, u kojoj je tekuća tabela roditelj, potrebno je uneti naziv buduće veze i tabelu – dete. Posle toga se veza kreira zadavanjem komande *Create relationship*. Automatski se kompletan primarni ključ tabele – roditelja prenosi kao strani ključ u tabelu – dete. Obeležja prenesena na ovaj način imaju iste parametre kao u tabeli roditelju, osim što su označena kao strani ključ. Parametri veze takođe mogu da se menjaju i to na panelu za unos podataka o vezi, koji je opisan kasnije.

Panel za unos podataka o obeležju je prikazan na slici 6.15. On omogućava brisanje obeležja iz tabele, izborom komande *Delete property*. Pri tome se brišu svi podaci o izabranom obeležju iz matične tabele, ali i iz svih tabela u koje je ovo obeležje preneto kao strani ključ. Osim toga, ovaj panel omogućava izmenu naziva obeležja, podatka da li to obeležje može da ima nula vrednost, tipa podataka i ograničenja ključa. Tip podataka se

bira od onih koji su uneti u repozitorijum, dok su za vrednosti ograničenja ključa predviđene sledeće:

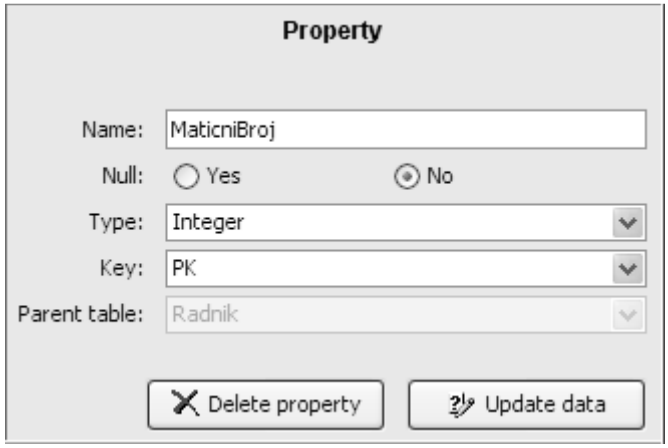
PK - obeležje ulazi u primarni ključ tabele,

FK - obeležje je strani ključ koji potiče iz neke druge tabele,

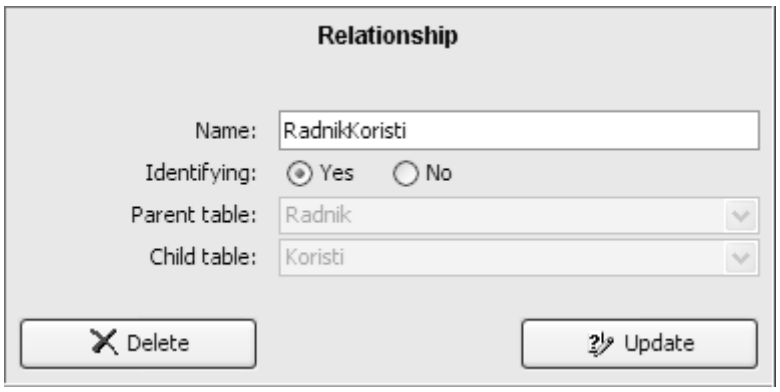
PK/FK - obeležje je strani ključ koji potiče iz neke druge tabele, ali istovremeno ulazi u primarni ključ izabrane tabele (identifikaciona zavisnost) i

- - nad obeležjem ne postoji ograničenje ključa.

Unos načinjenih izmena se postiže izabiranjem komande *Update data*.



Slika 6.15. Panel za unos podataka o obeležju.

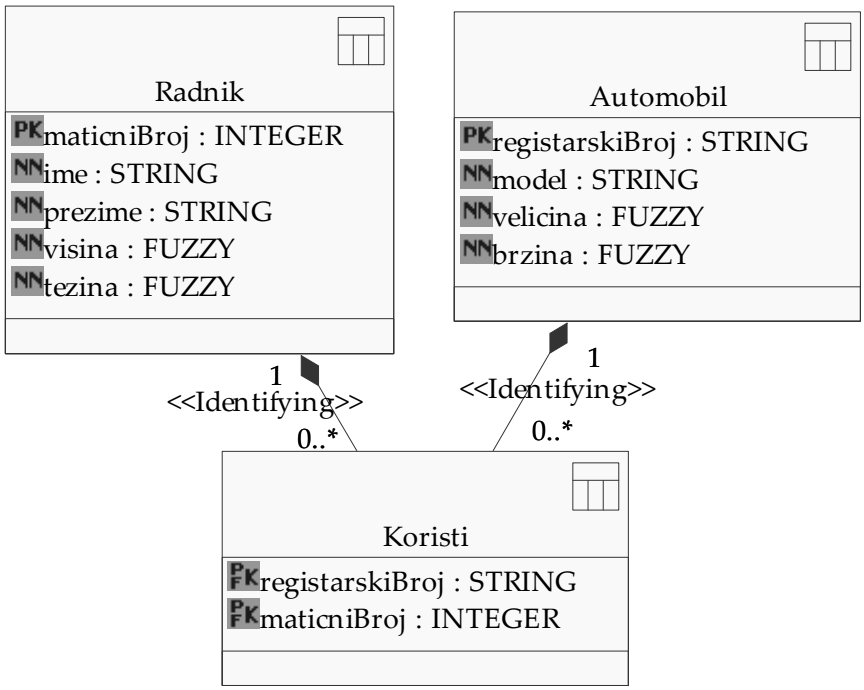


Slika 6.16. Panel za unos podataka o vezi.

Slika 6.16 prikazuje izgled panela za unos podataka o vezi. Brisanjem veze, koje se postiže izborom komande Delete, se briše sama veza, ali i obeležja koja su tom vezom prevučena u tabelu – dete. Moguće je izmeniti naziv veze i osobinu Identifying. Ova osobina se odnosi na identifikacionu zavisnost, odnosno, da li obeležja koja se ovom vezom prevlače ulaze u primarni ključ tabele – deteta. Promena parametara se potvrđuje pritiskom na dugme Update. Promene u smislu izmene identifikacione zavisnosti se, potom, automatski sprovode u modelu.

6.2.3 Kreiranje SQL skripta

Posle unošenja kompletnog modela se izdavanjem komande *Create SQL script* može pokrenuti kreiranje SQL skripta koji služi za kasnije kreiranje baze podataka. Proces prevođenja modela u niz SQL naredbi će biti prikazan nad modelom baze podataka prikazanim na slici 6.17.



Slika 6.17. Primer modela podataka.

Prikazani su radnici i automobili, kao i presečna tabela *Koristi* kojom je realizovana veza više prema više između radnika i automobila. U tabeli *Radnik*, kao i u tabeli *Automobil* postoje obeležja tipa *Fuzzy*.

Bitno je napomenuti da CASE alat nema mogućnost grafičkog prikaza unetog modela. Slika 6.17 potiče iz CASE alata IBM Rational Rose 7. Odluka da se ova osobina preskoči je donesena zbog toga što bi implementacija ovakvih mogućnosti, iako ne previše problematična, bila dugotrajna i obimna, a ne bi mnogo doprinela dostizanju ciljeva postavljenih pred ovu disertaciju.

Rezultat prevođenja ovakvog modela je dat na listingu 6.1.

```
CREATE TABLE FuzzyValue (
  ValueID NUMBER (10,0) NOT NULL,
  ForValueID NUMBER (10,0) NOT NULL,
  Code NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_FuzzyValue27 PRIMARY KEY (ValueID)
)
/
CREATE TABLE FuzzyQuantity (
  Left NUMBER (10,10) NOT NULL,
  Right NUMBER (10,10) NOT NULL,
  IsIncreasing NUMBER (5,0) NOT NULL,
  ValueID NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_FuzzyQuantity42 PRIMARY KEY (ValueID)
)
/
CREATE TABLE LinguisticLabel (
  Name VARCHAR2 ( 255 ) NOT NULL,
  ValueID NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_LinguisticLabel44 PRIMARY KEY (ValueID)
)
/
CREATE TABLE TrapezoidalFN (
  LeftMax NUMBER (10,10) NOT NULL,
  RightMax NUMBER (10,10) NOT NULL,
  LeftOffset NUMBER (10,10) NOT NULL,
  RightOffset NUMBER (10,10) NOT NULL,
  ValueID NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_TrapezoidalFN41 PRIMARY KEY (ValueID)
)
```

```
/
CREATE TABLE IsFuzzy (
  TableName VARCHAR2 ( 255 ) NOT NULL,
  AttributeName VARCHAR2 ( 255 ) NOT NULL,
  IsFuzzy NUMBER (5,0) NOT NULL,
  CONSTRAINT PK_IsFuzzy28 PRIMARY KEY (TableName,
AttributeName)
)
/
CREATE TABLE Interval (
  Left NUMBER (10,10) NOT NULL,
  Right NUMBER (10,10) NOT NULL,
  ValueID NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_Interval39 PRIMARY KEY (ValueID)
)
/
CREATE TABLE TriangularFN (
  Max NUMBER (10,10) NOT NULL,
  LeftOffset NUMBER (10,10) NOT NULL,
  RightOffset NUMBER (10,10) NOT NULL,
  ValueID NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_TriangularFN40 PRIMARY KEY (ValueID)
)
/
CREATE TABLE FuzzyType (
  Code NUMBER (10,0) NOT NULL,
  Name VARCHAR2 ( 255 ) NOT NULL,
  CONSTRAINT PK_FuzzyType29 PRIMARY KEY (Code)
)
/
CREATE TABLE Crisp (
  Value NUMBER (10,10) NOT NULL,
  ValueID NUMBER (10,0) NOT NULL,
  CONSTRAINT PK_Crisp43 PRIMARY KEY (ValueID)
)
/
ALTER TABLE LinguisticLabel ADD ( CONSTRAINT
FK_LinguisticLabel46 FOREIGN KEY (ValueID) REFERENCES
FuzzyValue (ValueID))
/
ALTER TABLE Interval ADD ( CONSTRAINT FK_Interval40
FOREIGN KEY (ValueID) REFERENCES FuzzyValue (ValueID))
/
```

```
ALTER TABLE FuzzyValue ADD ( CONSTRAINT FK_FuzzyValue34
FOREIGN KEY (ForValueID) REFERENCES FuzzyValue
(ValueID))
/
ALTER TABLE FuzzyValue ADD ( CONSTRAINT FK_FuzzyValue37
FOREIGN KEY (Code) REFERENCES FuzzyType (Code))
/
ALTER TABLE FuzzyQuantity ADD ( CONSTRAINT
FK_FuzzyQuantity43 FOREIGN KEY (ValueID) REFERENCES
FuzzyValue (ValueID))
/
ALTER TABLE TriangularFN ADD ( CONSTRAINT
FK_TriangularFN41 FOREIGN KEY (ValueID) REFERENCES
FuzzyValue (ValueID))
/
ALTER TABLE TrapezoidalFN ADD ( CONSTRAINT
FK_TrapezoidalFN42 FOREIGN KEY (ValueID) REFERENCES
FuzzyValue (ValueID))
/
ALTER TABLE Crisp ADD ( CONSTRAINT FK_Crisp43 FOREIGN
KEY (ValueID) REFERENCES FuzzyValue (ValueID))
/
INSERT INTO FuzzyType (Code,Name) VALUES (1,'Crisp')
/
INSERT INTO FuzzyType (Code,Name) VALUES
(2,'TriangularFN')
/
INSERT INTO FuzzyType (Code,Name) VALUES
(3,'TrapezoidalFN')
/
INSERT INTO FuzzyType (Code,Name) VALUES
(4,'FuzzyQuantity')
/
INSERT INTO FuzzyType (Code,Name) VALUES (5,'Interval')
/
INSERT INTO FuzzyType (Code,Name) VALUES
(6,'LinguisticLabel')
/
CREATE TABLE Radnik(
  MaticniBroj NUMBER(10,0) NOT NULL,
  Ime VARCHAR2(255) NOT NULL,
  Prezime VARCHAR2(255) NOT NULL,
  Visina NUMBER(10,0),
```

```
  Tezina NUMBER(10,0),
  CONSTRAINT PK_Radnik2 PRIMARY KEY (MaticniBroj)
)
/
CREATE TABLE Automobil(
  RegistarskiBroj VARCHAR2(255) NOT NULL,
  Model VARCHAR2(255) NOT NULL,
  Velicina NUMBER(10,0),
  Brzina NUMBER(10,0),
  CONSTRAINT PK_Automobil3 PRIMARY KEY
(RegistarskiBroj)
)
/
CREATE TABLE Koristi(
  MaticniBroj NUMBER(10,0) NOT NULL,
  RegistarskiBroj VARCHAR2(255) NOT NULL,
  CONSTRAINT PK_Koristi4 PRIMARY KEY (MaticniBroj,
RegistarskiBroj)
)
/
INSERT INTO IsFuzzy VALUES ('Radnik','MaticniBroj',0)
/
INSERT INTO IsFuzzy VALUES ('Radnik','Ime',0)
/
INSERT INTO IsFuzzy VALUES ('Radnik','Prezime',0)
/
ALTER TABLE Radnik ADD (CONSTRAINT FK_Radnik2 FOREIGN
KEY (Visina) REFERENCES FuzzyValue (ValueID))
/
INSERT INTO IsFuzzy VALUES ('Radnik','Visina',1)
/
ALTER TABLE Radnik ADD (CONSTRAINT FK_Radnik3 FOREIGN
KEY (Tezina) REFERENCES FuzzyValue (ValueID))
/
INSERT INTO IsFuzzy VALUES ('Radnik','Tezina',1)
/
ALTER TABLE Koristi ADD (CONSTRAINT FK_Koristi4 FOREIGN
KEY (MaticniBroj) REFERENCES Radnik (MaticniBroj))
/
INSERT INTO IsFuzzy VALUES
('Automobil','RegistarskiBroj',0)
/
INSERT INTO IsFuzzy VALUES ('Automobil','Model',0)
```

```
/
ALTER TABLE Automobil ADD (CONSTRAINT FK_Automobil5
FOREIGN KEY (Velicina) REFERENCES FuzzyValue (ValueID))
/
INSERT INTO IsFuzzy VALUES ('Automobil','Velicina',1)
/
ALTER TABLE Automobil ADD (CONSTRAINT FK_Automobil6
FOREIGN KEY (Brzina) REFERENCES FuzzyValue (ValueID))
/
INSERT INTO IsFuzzy VALUES ('Automobil','Brzina',1)
/
ALTER TABLE Koristi ADD (CONSTRAINT FK_Koristi7 FOREIGN
KEY (RegistarskiBroj) REFERENCES Automobil
(RegistarskiBroj))
/
INSERT INTO IsFuzzy VALUES ('Koristi','MaticniBroj',0)
/
INSERT INTO IsFuzzy VALUES
('Koristi','RegistarskiBroj',0)
/
```

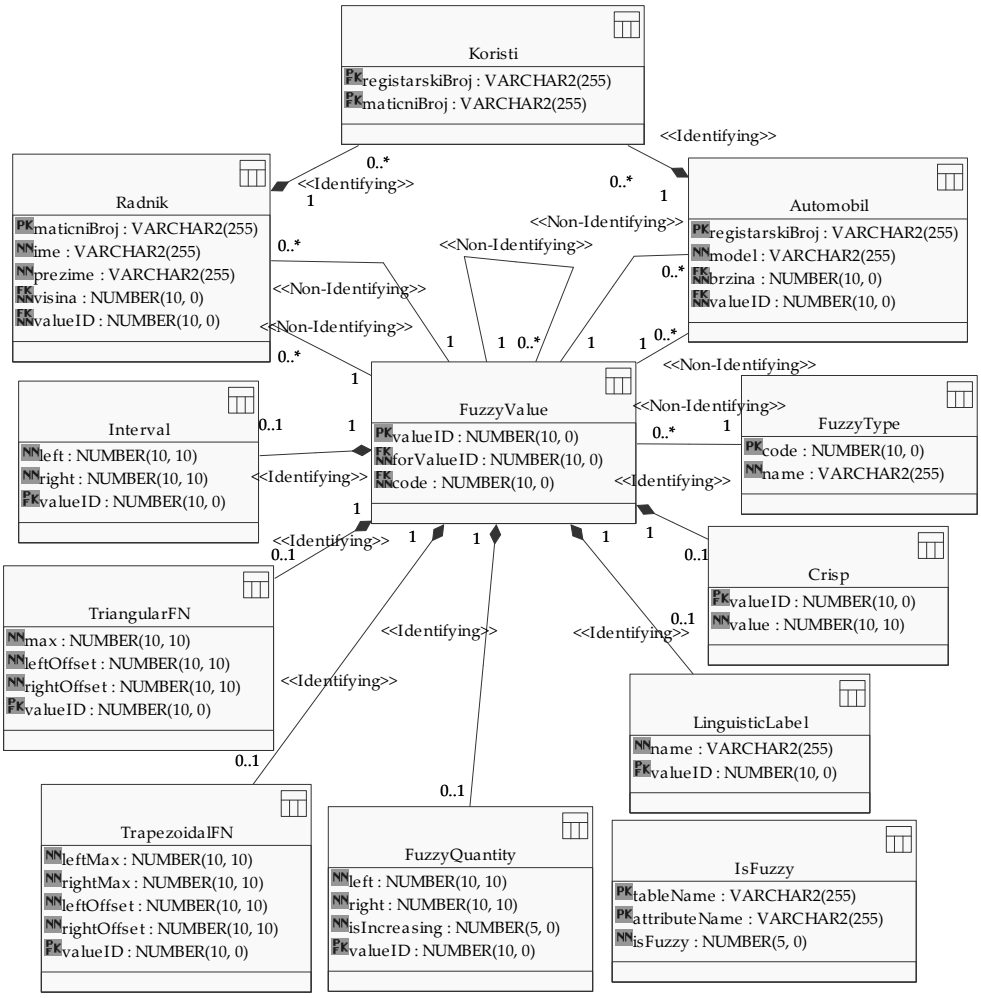
Listing 6.1. SQL skript.

Konačan izgled tabela u bazi podataka koji nastaje kada se ovaj skript izvrši je prikazan na slici 6.18. Ova struktura je, naravno, potpuno prilagođena modelu za skladištenje fazi podataka opisanom u četvrtom poglavlju. Ovde se ukratko opisuje generisani model sa kratkim osvrtom na činjenice istaknute u pomenutom poglavlju.

Obeležja tipa Fuzzy u tabelama Radnik i Automobil su prevedena u tip podataka NUMBER(10,0). Ona predstavljaju strani ključ šifre te fazi vrednosti (*valueID*) u tabeli *FuzzyValue*. Tabela *FuzzyValue*, dakle, predstavlja opis fazi obeležja. Ona sadrži obeležje *code* kojim je predstavljena vrsta fazi obeležja i koje predstavlja ključ za dešifrovanje sadržaja fazi vrednosti iz tabela pomoću tabele *FuzzyType*.

Same vrednosti fazi obeležja se nalaze u odgovarajućim tabelama, u zavisnosti od tipa. Na primer, ako se radi o trougaonom fazi broju, onda bi se vrednosti za taj fazi broj našle u tabeli *TriangularFN* i to u onoj torci koja ima vrednost obeležja *valueID* jednaku vrednosti ovog obeležja u tabeli *FuzzyValue* za taj fazi broj.

Obeležje *forValueID* potiče od veze tabele *FuzzyValue* sa samom sobom i služi za reprezentaciju lingvističkih labela. U slučaju da se ne radi o lingvističkoj labeli, vrednost ovog obeležja je null. Torka koja predstavlja lingvističku labelu, za vrednost obeležja code ima šifru lingvističke labele kao fazi tipa u tabeli *FuzzyType*. Vrednost obeležja *value* je šifra pomoću koje se u tabeli *LinguisticLabel* nalazi naziv ove lingvističke labele.



Slika 6.18. Tabele u bazi podataka.

Na kraju, vrednost obeležja *forValueID* predstavlja *valueID* sloga u tabeli *FuzzyValue* kojim je opisana ova lingvistička labela. Na primer, ako je

to trougaoni fazi broj, onda se slog koji je opisuje ne bi razlikovao od bilo kojeg drugog koji opisuje trougaoni fazi broj. Naravno, svakom od fazi obeležja se može pridružiti i crisp vrednost. Takva situacija bi se označila tako što se za vrednost obeležja *code* stavlja šifra kojom su označene crisp vrednosti u tabeli *FuzzyType*, a vrednost se upisuje u tabelu *Crisp*, po već usvojenom mehanizmu.

Poglavlje 7

UPOTREBA JEZIKA PFSQL U VIŠESLOJNIM SISTEMIMA

Ubrzani razvoj telekomunikacija i interneta, kao i rastuće potrebe za velikim količinama informacija su izazvale potrebu za razvojem velikih, enterprise informacionih sistema, odnosno višeslojnih distribuiranih aplikacija. Višeslojna arhitektura i moderna programska okruženja koja omogućavaju njenu implementaciju su osnovna pretpostavka razvoja modernih informacionih sistema.

Fazi JDBC drajver opisan u petom poglavlju je alat koji omogućava razvoj jednostavnih dvoslojnih aplikacija koje komuniciraju sa bazom podataka i koriste jezik PFSQL. U ovom poglavlju se razmatraju mogućnosti njegove upotrebe u sistemima zasnovanim na višeslojnoj arhitekturi.

Kao tehnologija za razvoj višeslojnih aplikacija, a u skladu sa orijentacijom prema open source rešenjima, izabrana je J2EE (Java 2 Enterprise Edition) platforma. U okviru te platforme centralno mesto zauzima Enterprise JavaBeans 3.0 (EJB 3.0) tehnologija koja služi za razvoj srednjeg sloja. Upravo srednji sloj je mesto na kome je potrebno ponuditi mogućnost upotrebe PFSQL jezika u radu sa fazi podacima koji se nalaze u relacionoj bazi podataka.

Pred ovu disertaciju je kao cilj postavljena samo diskusija i razmatranje različitih mogućnosti upotrebe fazi JDBC drajvera u višeslojnim aplikacijama. Implementacija konačnog rešenja izlazi van njenih okvira. Bez obzira na to, jedno od mogućih rešenja je opisano i implementirano. To rešenje nije i najbolje moguće. Zbog toga su istaknuti

osnovni pravci u kojima treba nastaviti istraživanje ovog segmenta upotrebe da bi se došlo do konačnog rešenja.

U prvom delu poglavlja je dat kratak osvrt na EJB 3.0 tehnologiju, dok se u drugom razmatraju tri različite mogućnosti upotrebe PFSQL-a na EJB 3.0 srednjem sloju.

7.1 EJB 3.0

U ovoj sekciji je dat kratak uvod u EJB 3.0 tehnologiju. Ideja autora je da pokuša da približi koncepte EJB 3.0 tehnologije čitaocu koji već ima iskustva u razvoju srednjeg sloja zbog toga što se radi o novoj i još nedovoljno raširenoj tehnologiji. Izostavljeni su brojni detalji kojima ova specifikacija obiluje, pa čak i neke osnovne teme koje nisu značajne za ovu disertaciju. Sveobuhvatan prikaz EJB 3.0 tehnologije se može naći u (Panda, Rahman, & Lane, 2007), (Keith & Schincariol, 2006) i (Monson-Haefel & Burke, 2006).

7.1.1 Uvodna razmatranja

Java 2 Enterprise Edition (J2EE) platforma je poznata po svojoj širokoj podršci i fleksibilnosti u razvoju višeslojnih enterprise aplikacija. Enterprise JavaBeans (EJB) je ključna komponenta J2EE specifikacije.

J2EE verzija 1.4 i njoj pripadajuća EJB 2.1 specifikacija su se pokazale kao previše složene. Da bi se proizvela upotrebljiva aplikacija, potrebno je proučiti EJB model, API-je, XML deployment descriptor-e i različite dizajn paterne. Složeni programski model je jedan od najvažnijih faktora koji je usporavao usvajanje J2EE platforme.

U pokušaju rešavanja ovih problema, EJB 3.0 ekspertska grupa, koju sačinjavaju najveće kompanije iz ove oblasti (Sun, Oracle, IBM, Red Hat – JBoss…), je razvila novi programski model za novu generaciju višeslojnih enterprise Java aplikacija. Na razvoj ove specifikacije su snažno uticali uspešni open source projekti kao što su Hibernate i XDoclet.

EJB 3.0 je, dakle, Java 2 Enterprise Edition (J2EE) programski interfejs za razvoj srednjeg sloja i novi prilaz razvoju višeslojnih enterprise

aplikacija. Ona je deo J2EE 5 specifkacije. Ideja koja prožima celu specifikaciju je upotreba POJO (Plain Old Java Object) objekata, Java anotacija i dependency injection dizajn paterna.

Anotacije su nova osobina programskog jezika Java uvedene u JDK 5 specifikaciji. One omogućavaju povezivanje atributa koji se zadaju u kodu (meta podataka) i opcija za konfigurisanje direktno u Java klasama. Na primer, u kodu neke Java klase se može zadati anotacija koja označava da je ta klasa dostupna za daljinsko pozivanje (remote invocation).

Anotacije mogu gotovo u potpunosti da zamene XML descriptor-e široko korišćene u J2EE 1.4 specifikaciji. Imajući u vidu da su se baš XML descriptor-i pokazali kao najlošija komponenta prethodne specifikacije, anotacije predstavljaju značajan korak napred.

Novi EJB objekti su obični Java objekti (POJO) kojima upravlja EJB kontejner. Oko tih objekata, odnosno njihovih klasa, se mogu kreirati proizvoljne strukture nasleđivanja. Postavlja se pitanje kako kontejner upravlja tim objektima kada nema njihov model (koji je ranije specificiran pomoću XML descriptor-a)? EJB 3.0 specifikacija predviđa da se interakcija između POJO objekata/klasa i kontejnera deklarativno specificira pomoću Java anotacija. Dakle, EJB 3.0 klasa je sada obična Java klasa koja sadrži anotacije.

Upravljanje EJB objektima (u ovom kontekstu se nazivaju još i servisnim objektima) je osnovni servis koji nudi svaki J2EE aplikativni server. U poređenju sa upravljanjem standardnim Java objektima, bez aplikativnog servera, ovaj način upravljanja objektima ima značajne prednosti.

Kao prvo, omogućeno je „labavo povezivanje“ (loose coupling). Naime, kako klijentska aplikacija ne instancira direktno EJB objekte, ona ne mora da zna naziv klase u kojoj je implementirano ponašanje objekta. Potrebno je da pozna samo interfejs prema toj klasi. Isto to važi i za pozive između komponenata na srednjem sloju. Ovakvo ponašanje omogućava serveru da podrži alternativne implementacije EJB objekta bez izmena klijentskog koda.

Nadalje, ovakav način upravljanja nudi nezavisnost od mreže i implementacije veze između klijenta i servera. Servisi koje nudi server se mogu ponuditi preko skoro svake vrste mrežnog transporta. Što se klijenta tiče, on poziva metode na serveru kao da se one nalaze na lokalnom računaru, iako server fizički može da se nalazi na drugom kraju sveta. Takođe, o implementaciji mrežnih protokola i transportu podataka programer ne mora da brine.

Kontejner instancira objekte, upravlja njihovim životnim ciklusima, kreira i upravlja pool-ovima objekata i pruža servise za pristup tim objektima iz drugih delova aplikacije. Osim ovoga, kontejner nudi i podršku za transakcije, perzistentnost, odnosno vezu sa bazom podataka, sigurnosne mehanizme itd.

EJB 3.0, kao i prethodna specifikacija, predviđa tri vrste EJB objekata: session bean, entity bean i message driven bean. Message driven bean-ovi ovde neće biti opisani obzirom na to da nisu značajni za temu koja se obrađuje u ovom poglavlju.

7.1.2 Session bean

Session bean-ovi su komponente koje su namenjene implementaciji poslovne logike sistema. Funkcionalnost session bean-a je definisana u njegovom interfejsu koji je, u duhu EJB 3.0 specifikacije, običan Java interfejs. Koristeći naziv interfejsa klijent koji poziva session bean dobija stub objekat od JNDI (Java Naming and Directory Interface) baze podataka koja postoji na serveru. Stub objekat implementira interfejs session bean-a. Njegov zadatak je prosto da prosledi poziv instanci toga session bean-a koja sadrži implementaciju poslovne logike. Stub objekte automatski generiše kontejner.

Postoje dve vrste session bean-ova: stateful i stateless. U slučaju stateless session bean-a, stub objekat rutira poziv bilo kojoj instanci session bean klase koja je na raspolaganju u pool-u objekata. Sledeći poziv preko istog stub objekta može biti upućen nekoj drugoj instanci. Dakle, pozivi su međusobno sasvim nezavisni. Za razliku od stateless session bean-a, stateful session bean održava svoja interna stanja. Ako klijent pozove metod preko istog stub objekta, poziv će uvek biti rutiran na istu instancu

session bean-a u kontejneru, a ta instanca može da sadrži podatke kojima je pristupano ili koji su kreirani u prethodnom pozivu.

Session bean, bio on stateless ili stateful, se jednostavno kreira. Listinzi 7.1 i 7.2 sadrže primer jednostavnog session bean-a koji računa n! za zadato n. Na prvom listingu je dat interfejs, a na drugom sama implementacija session bean-a. Anotacijom @Stateless je specificirano da se radi o stateless session bean-u.

```
public interface Fact(){
      public int calculate(int n);
}
```

Listing 7.1. Interfejs session bean-a.

```
import javax.ejb.*;
@Stateless
public class StatelessCalc implements Fact{
      public int calculate(int n){
            int res = 1;
            for(int k=2;k<=n;k++){
                  res = res*k;
            }
            return res;
      }
}
```

Listing 7.2. Session bean klasa.

Session bean može da implementira više različitih interfejsa za različite vrste klijenata. Po default-u, interfejs je namenjen lokalnim klijentima, odnosno onima koji se izvršavaju u istoj virtuelnoj mašini kao i kontejner. Ovaj interfejs se naziva lokalnim interfejsom. Kada lokalni klijent zatraži stub objekat preko lokalnog interfejsa, kontejner vraća običnu Java referencu na session bean objekat.

Druga vrsta interfejsa je udaljeni (remote) interfejs koji je namenjen udaljenim klijentima. Kada udaljeni klijent zatraži stub objekat preko udaljenog interfejsa, kontejner vraća serijalizovani stub objekat koji implementira taj interfejs. Ovakav stub objekat „zna“ kako da uputi udaljene pozive procedura (Remote Procedure Call, RPC) serveru. Naravno, ovakav način rada je daleko sporiji od direktnog poziva preko

Java reference. Jedan od načina da se za neki interfejs zada da je on udaljeni interfejs je prosto zadavanje anotacije @Remote.

U ovom kratkom pregledu mogućnosti koje nude session bean-ovi je značajno napomenuti još da postoje i anotacije koje omogućavaju upravljanje životnim ciklusom session bean objekta. Na primer, moguće je zadati kod koji se izvršava pre kreiranja objekta od strane kontejnera, posle njegovog uništavanja itd.

7.1.3 Entity bean i rad sa bazom podataka

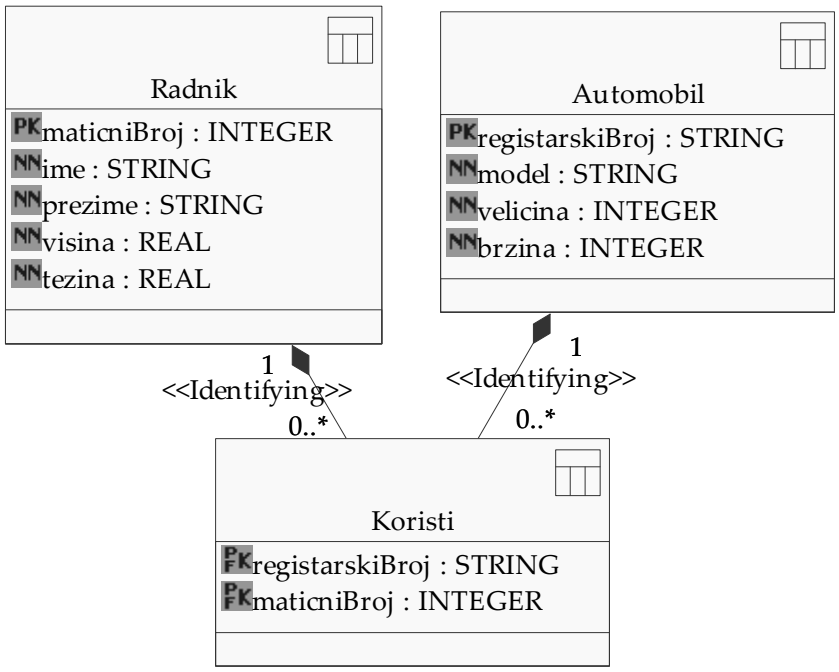
Da bi se razvila aplikacija koja se oslanja na bazu podataka potrebno je modelirati podatke u dve različite strukture. Unutar aplikacije, podaci su modelirani kao Java objekti. Kompletna aplikacija se izrađuje tako da radi sa tim objektima. Međutim kada se podaci smeštaju u bazu podataka, oni moraju biti modelirani kao relacione tabele. Manipulacija Java objektima i relacionim tabelama zahteva upotrebu dva sasvim različita programska jezika – Java i SQL. Dakle, isti podaci se dva puta modeliraju, a kroz celu aplikaciju se smenjuje upotreba dva programska jezika. Ovakav način rada je neefikasan i podložan greškama.

Entity bean-ovi su dizajnirani da bi se prevazišao jaz između objektne i relacione reprezentacije podataka. Sa tačke gledišta programera, to su POJO objekti sa anotacijama kojima se specificira način skladištenja u bazu podataka. Objektno-relaciono mapiranje se izvršava automatski od strane kontejnera. Na taj način je programer oslobođen brige o detaljima šeme baze podataka, upravljanju konekcijama i API-ju za pristup bazi podataka.

U nastavku je dat prikaz osnovnih mogućnosti za rad sa bazom podataka koje nudi EJB 3.0 tehnologija.

7.1.3.1 Objektno-relaciono mapiranje

Na slici 7.1 je dat primer jednostavne šeme baze podataka. On podseća na primer dat u šestom poglavlju, ali u ovom slučaju ne postoje fuzzy obeležja u modelu – radi se o običnoj relacionoj bazi podataka.



Slika 7.1. Primer šeme baze podataka.

Ovde će ukratko biti prikazan način implementacije entity bean-ova koji odgovaraju ovakvoj bazi podataka. Potrebno je napomenuti da objektno-relaciono mapiranje u EJB 3.0 tehnologiji nije trivijalna tema. Zbog toga se ono ovde prikazuje samo u najosnovnijim crtama. Mnogi detalji i napredne osobine nisu ni pomenuti.

Na listingu 7.3 je data entity bean klasa koja odgovara tabeli Radnik.

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="radnik")
public class Radnik implements Serializable {
     private static final long serialVersionUID =
8591965677143806509L;
     private int maticnibroj;
```

```
      private String ime;
      private String prezime;
      private double visina;
      private double tezina;

      public Radnik () { }
      public Radnik(String ime, int maticnibroj, String
prezime, double tezina,
                  double visina) {
            this.ime = ime;
            this.maticnibroj = maticnibroj;
            this.prezime = prezime;
            this.tezina = tezina;
            this.visina = visina;
      }

      @Id
      public int getMaticnibroj() {
            return maticnibroj;
      }
      public void setMaticnibroj(int maticnibroj) {
            this.maticnibroj = maticnibroj;
      }
      public String getIme() {
            return ime;
      }
      public void setIme(String ime) {
            this.ime = ime;
      }
      public String getPrezime() {
            return prezime;
      }
      public void setPrezime(String prezime) {
            this.prezime = prezime;
      }
      public double getVisina() {
            return visina;
      }
      public void setVisina(double visina) {
            this.visina = visina;
      }
      public double getTezina() {
            return tezina;
```

```
      }
      public void setTezina(double tezina) {
            this.tezina = tezina;
      }
}
```

Listing 7.3. Entity bean klasa koja odgovara tabeli Radnik.

Klasa se proglašava entity bean-om pomoću anotacije @Entity. Tipično, jedan entity bean mapira jednu tabelu u bazi podataka. Ta tabela se zadaje pomoću anotacija @Table, kao što se vidi na primeru. Entity bean sadrži atribute koji odgovaraju obeležjima mapirane tabele po nazivu i tipu. Osim toga, zadaju se set i get metode za svaki od ovih atributa, kao i default konstruktor.

U primeru je dat i konstruktor sa parametrima koji služi za kreiranje entity objekta popunjenog podacima. Takav objekat odgovara jednom slogu u bazi podataka. Anotacijom @Id iznad get metode nekog atributa se taj atribut markira kao primarni ključ tabele. U ovom slučaju je to atribut maticnibroj.

Predstavljeni entity bean je najosnovniji primer. Podrška za mapiranje baze podataka u EJB 3.0 tehnologiji je daleko šira. Postoje mogućnosti za mapiranje složenih ključeva, veza različitih kardinaliteta između tabela, nasleđivanje, odnosno IS-A hijerarhije itd. Na listingu je dat segment koda entity bean-a koji mapira tabelu Koristi kojim se ilustruje mapiranje veza između tabela.

```
@Entity
@Table(name="koristi")
public class Koristi implements Serializable{
…
      private Radnik radnik;
      private Automobil automobil;
      @ManyToOne(optional=false)
      @JoinColumn(name="maticnibroj")
      public Radnik getRadnik() {
            return radnik;
      }
      public void setRadnik(Radnik radnik) {
            this.radnik = radnik;
```

```
        }
        @ManyToOne(optional=false)
        @JoinColumn(name="registarskibroj")
        public Automobil getAutomobil() {
              return automobil;
        }
        public void setAutomobil(Automobil automobil) {
              this.automobil = automobil;
        }
…
}
```

Listing 7.4. Mapiranje veza između tabela.

Posmatrajmo primer veze između tabele Koristi i Radnik. Entity bean Radnik se pojavljuje kao atribut entity bean-a Koristi. On ima svoje set i get metode. Detalji veze između ove dve tabele se specificiraju anotacijama pridruženim get metodi. Anotacijom @ManyToOne je specificirano da se radi o vezi kardinaliteta jedan-prema-više. Anotacija @JoinColumn služi za specificiranje obeležja koja su prevučena iz tabele Radnik u tabelu Koristi kao strani ključevi. U ovom slučaju je to obeležje maticnibroj.

7.1.3.2 Upravljanje entity bean-ovima

Pošto su entity bean-ovi POJO objekti, oni se mogu instancirati koristeći ključnu reč new. Međutim, da bi se podaci skladištili u bazu podataka ili pročitali iz nje, potreban je poseban API koji se naziva EntityManager.

EntityManager objekat se može pozvati u session bean-u pomoću @PersistenceContext anotacije. Dakle, nije potrebno eksplicitno kreirati ili pozvati objekat preko lookup mehanizma, već se to može uraditi koristeći mehanizam injektovanja (dependency injection). Na listingu 7.5 je dat segment session bean-a u koji je injektovan EntityManager.

```
import javax.persistence.PersistenceContext;
@Stateless
@Remote (StatelessCalcRemote.class)
public class StatelessCalc {
      @PersistenceContext (unitName="ProbaEJB")
```

```
        protected EntityManager em;
…
}
```

Listing 7.5. Injektovanje EntityManager-a u session bean.

Da bi se skladištio jedan slog neke tabele u bazu podataka, potrebno je prvo kreirati instancu odgovarajućeg entity bean-a, a zatim pozvati metod persist() koji nudi EntityManager. Primer kreiranja novog sloga u tabeli Radnik je dat na listingu 7.6.

```
Radnik ra = new
      Radnik("Milan",5,"Marković",89.56,184.57);
em.persist(ra);
```

Listing 7.6. Kreiranje novog sloga u tabeli radnik.

Čitanje podataka iz baze preko EntityManagera je daleko složenija tema. Najjednostavniji način da se podaci pročitaju je direktno pronalaženje određenog sloga u bazi podataka preko njegovog primarnog ključa pomoću metode find(). Listing 7.7 prikazuje segment koda koji pronalazi radnika sa matičnim brojem 5 i ispisuje njegovo ime i prezime.

```
Radnik ra = em.find(Radnik.class, Integer.valueOf(5));
System.out.println(ra.getIme()+" "+ra.getPrezime());
```

Listing 7.7. Osnovno pretraživanje pomoću EntityManager-a.

Dakle, pretraživanje entity bean-ova po primarnom ključu je jednostavno, ali problem je što ne znamo uvek primarni ključ svih slogova koje želimo da pročitamo. Naravno, EJB 3.0 tehnologija nudi mogućnosti da se baza podataka pretražuje na drugi, mnogo fleksibilniji način, pomoću jezika EJB QL (EJB Query Language) specijalno dizajniranog za te svrhe. Ovo je jezik koji liči na SQL i služi za pretraživanje baze podataka, a rezultate pretrage vraća u vidu entity bean objekata, što je mnogo pogodnije u objektno-orijentisanom okruženju.

Osnovni primer upotrebe ovog jezika je čitanje svih slogova iz neke tabele. Na listingu 7.8 je dat segment koda koji čita sve podatke iz tabele Radnik, a rezultate vraćene u vidu kolekcije entity objekata Radnik ispisuje na konzolu.

```
Collection <Radnik> col =
```

```
      em.createQuery("from Radnik r").getResultList();
Iterator <Radnik> it = col.iterator();
Radnik r = null;
while(it.hasNext()){
      r = it.next();
      System.out.println(r.getIme()+" "+r.getPrezime());
}
```

Listing 7.9. Čitanje svih slogova iz tabele Radnik.

Dakle, izraz *from Radnik r* je EJB QL upit koji vraća sve podatke iz tabele Radnik. EJB QL nudi različite mogućnosti za filtraciju podataka. Sledi primer upita koji vraća samo one radnike čija je visina veća od 180, a težina manja od 90. Kao što se vidi u kodu na listingu 7.10, EJB QL izrazi se mogu parametrizovati na sličan način kao kod JDBC API-ja.

```
Collection <Radnik> col =
      em.createQuery("select r from Radnik r where "+
+"r.visina > :vis and r.tezina < :tez")
            .setParameter ("vis", new Double (180))
            .setParameter ("tez", new Double (90))
            .getResultList();
Iterator <Radnik> it = col.iterator();
Radnik r = null;
while(it.hasNext()){
      r = it.next();
      System.out.println(r.getIme()+" "+r.getPrezime()+
" "+r.getVisina()+" "+r.getTezina());
}
```

Listing 7.10. Čitanje radnika viših od 180 cm i lakših od 90 kg.

Rezultujući EJB QL upit je gotovo identičan sa odgovarajućim SQL upitom: *select r from Radnik r where r.visina > 180 and r.tezina < 90*.

Pored prikazanih mogućnosti, u segmentu rada sa bazom podataka EJB 3.0 nudi dodatne, napredne osobine jezika EJB QL, različite mogućnosti modifikacije i sinhronizacije strukture objekata sa bazom podataka, veoma široku podršku za transakcije kroz anotacije, upravljanje životnim ciklusom enitity bean objekata itd.

7.1.4 Interceptor

Za kraj ostaje još jedna tema kojoj ne bi bilo mesta u ovako kratkom pregledu EJB 3.0 tehnologije da nije značajna za problem koji se obrađuje u ovom poglavlju. Runtime servisi, kao što su podrška za transakcionu obradu podataka i bezbednosni mehanizmi se primenjuju na bean objekte u vreme izvršavanja njihovih metoda. Interno, ti servisi su implementirani pomoću interceptor metoda čijim izvršavanjem upravlja kontejner. Ove metode mogu da se izvrše pre izvršavanja metoda za koji su vezane.

EJB 3.0 dopušta programerima da pišu proizvoljne interceptor metode za EJB metode. Ta mogućnost, u suštini, predstavlja mehanizam za proširenje ili izmenu funkcionalnosti kontejnera. Koristeći interceptor metode, kontejneru se mogu dodati nove funkcionalnosti, a postojeće se mogu redefinisati.

Jedan način da se definiše interceptor metod je da se on smesti u samu bean klasu. Takav metod se označava anotacijom @AroundInvoke, a kontejner ga poziva i izvršava neposredno pre izvršavanja bilo kojeg drugog metoda. Takođe, kontejner prosleđuje kontekst u kome se poziv izvršava u vidu interfejsa InvocationContext (listing 7.11). Iz takvog objekta se može saznati sve o pozivu metoda koji je presretnut – njegov naziv, parametri itd. Poziv metoda ctx.proceed() služi za nastavak izvršavanja. U tom slučaju se izvršava sledeći interceptor metod, a ako takvoga nema, onda i sam glavni metod. Na listingu 7.11 je dat session bean koji izračunava n! sa listinga 7.2 izmenjen pomoću interceptor metoda tako da izračunava (n+1)!.

```
import javax.ejb.*;
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;
@Stateless
public class StatelessCalc implements Fact{
      @AroundInvoke
      public Object increaseN(InvocationContext ctx)
      throws Exception {
            Object[] p = ctx.getParameters();
            int n = ((Integer)p[0]).intValue();
            n++;
            p[0] = new Integer(n);
```

```
            ctx.setParameters(p);
            return ctx.proceed();
      }
      public int calculate(int n){
            int res = 1;
            for(int k=2;k<=n;k++){
                  res = res*k;
            }
            return res;
      }
}
```

Listing 7.11. Interceptor metod.

Drugi način realizacije interceptor metoda je njihovo izdvajanje u posebnu klasu. U tom slučaju je potrebno dodati anotaciju @Interceptor u ciljnu bean klasu. Na listingu 7.12 je dat primer ovakve implementacije interceptor metoda. Funkcionalnost je ista kao i u prethodnom primeru. Anotacija koju bi trebalo dodati u ciljnu bean klasu izgleda ovako:

```
@Interceptors (Tracer.class)
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;
public class Tracer {
      @AroundInvoke
      public Object increaseN(InvocationContext ctx)
      throws Exception {
            Object[] p = ctx.getParameters();
            int n = ((Integer)p[0]).intValue();
            n++;
            p[0] = new Integer(n);
            ctx.setParameters(p);
            return ctx.proceed();
      }
}
```

Listing 7.12. Interceptor metod izdvojen u posebnu klasu.

Ovde je prikazana mogućnost zadavanja interceptor metoda na nivou klase. Ove metode se izvršavaju pre poziva bilo kog metoda te klase. Osim ove mogućnosti, postoji i mogućnost da se, na sličan način, zada interceptor metod za svaki pojedinačan metod neke bean klase. Ovakve

interceptor metode se izvršavaju neposredno pre izvršavanja metoda za koji su vezane.

Potrebno je skrenuti pažnju na redosled pozivanja interceptor metoda u slučaju da je za neki bean vezano više njih. Interceptor metode se pozivaju onim redosledom kojim su navedene u nizu atributa anotacije @Interceptor. Kada se sve ove interceptor metode izvrše, poziva se još i ona koja je implementirana unutar bean klase, ako takva postoji. Na kraju se poziva i sam glavni metod čije izvršavanje je tražio klijent.

7.2 PFSQL I EJB 3.0

Postavlja se pitanje kakve su mogućnosti za upotrebu fazi JDBC drajvera opisanog u petom poglavlju na EJB 3.0 srednjem sloju? Ili, šire od toga, kakve su mogućnosti za rad sa fazi podacima struktuiranim po modelu opisnom u četvrtom poglavlju i PFSQL jezikom opisanom u petom poglavlju na srednjem sloju višeslojne aplikacije bazirane na EJB 3.0 tehnologiji? U ovoj sekciji se razmatraju tri mogućnosti da se to uradi.

7.2.1 Prva mogućnost

Najjednostavniji, ali i najmanje dobar pristup je prosta upotreba fazi JDBC drajvera takvog kakav je opisan u petom poglavlju na srednjem sloju. Klase i interfejsi od kojih je drajver sastavljen mogu da se pokrenu takve kakve jesu. Prosto, to će biti obične Java klase i interfejsi, a ne neka vrsta EJB 3.0 bean-ova. Takvim klasama i interfejsima se može pristupiti iz EJB bean objekata.

Problem u ovom pristupu je u konekciji na bazu podataka. Naime, ako je JDBC drajver za konkretnu bazu podataka dostupan, konekcija na nju će biti bez problema ostvarena sa srednjeg sloja.

Međutim, aplikativni server, odnosno EJB kontejner, je taj koji treba da upravlja konekcijama na baze podataka koje su zadate u njegovim konfiguracionim fajlovima. On uspostavlja te konekcije, kreira pool-ove konekcija i vodi računa o njihovom zatvaranju. U ovom slučaju se, međutim, pravi direktna konekcija na bazu podataka na koju se kontejner već konektovao preko svojih mehanizama. Dakle, takva konekcija je

neprirodna i suvišna. Osim toga, u tom slučaju nema pristupa mehanizmima upravljanja pool-ovima konekcija koje kontejner inače nudi. Sa druge strane, dobra osobina ovog pristupa je svakako to što nisu potrebne apsolutno nikakve izmene u kodu fazi JDBC drajvera.

7.2.2 Druga mogućnost

Modifikacijom pristupa opisanog gore se može doći do boljeg rešenja. Problem sa zaobilaženjem serverskih mehanizama za upravljanje konekcijama opisan u prvom poglavlju može biti prevaziđen. EJB 3.0 tehnologija, pored mogućnosti za rad sa bazom podataka opisanim u sekciji 7.1.3, nudi i mogućnost da se iz nekog session bean-a direktno zatraži konekcija na bazu podataka. Ovom konekcijom upravlja kontejner, on ju je uspostavio, i može da je izdvoji iz pool-a i prosledi na korišćenje nekom session bean objektu. Sledi segment koda session bean klase na kome se ova mogućnost ilustruje.

```
…
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import javax.annotation.Resource;
import javax.ejb.*;
import javax.sql.DataSource;
@Stateless
@Remote({StatelessQueryRemote.class})
public class StatelessQuery implements
  Serializable,StatelessQueryRemote{
  @Resource(mappedName="java:/maxdbisfuzzy")
    DataSource dataSource;
  public List getAZR(int maticniBroj) {
    Connection conn = null;
    ArrayList res = new ArrayList();
    PreparedStatement ps = null;
```

```
    ResultSet rs = null;
    try {
      conn = dataSource.getConnection();
      String query = "select a.registarskibroj,a.model,
        a.velicina,a.brzina from automobil a, radnik r,
        koristi k where (r.maticnibroj="+maticniBroj+")
        and (r.maticnibroj=k.maticnibroj) and
        (k.registarskibroj=a.registarskibroj)";
      ps = conn.prepareStatement(query);
      rs = ps.executeQuery();
      ArrayList row = null;
      while (rs.next()){
        row = new ArrayList();
        row.add(rs.getString(1));
        row.add(rs.getString(2));
        row.add(Integer.valueOf(rs.getInt(3)));
        row.add(Integer.valueOf(rs.getInt(4)));
        res.add(row);
      }
    } catch (SQLException e) {e.printStackTrace();}
    finally {
      try{
        if (rs != null) { rs.close(); }
        if (ps != null) { ps.close(); }
        if (conn != null) { conn.close(); }
      } catch (Exception e) {e.printStackTrace();}
    }
    return res;
  }
…
```

Listing 7.13. Eksplicitno zatražena konekcija na bazu podataka.

Na ovom listingu je prikazan segment session bean-a koji sadrži metod getAZR(). Ovaj metod, koristeći konekciju na bazu podataka dobijenu od kontejnera i običan JDBC interfejs, postavlja standardni SQL upit bazi podataka. Upit vraća sve automobile koje koristi radnik sa zadatim matičnim brojem.

Najbitniji deo koda počinje anotacijom @Resource. Ovom anotacijom se u session bean injektuje konekcija, predstavljena interfejsom DataSource, koja se dobija od kontejnera iz njegovog pool-a konekcija.

Pomoću JNDI imena se specificira na koju tačno bazu podataka želimo da se konektujemo. Naravno, ona mora biti već definisana u konfiguracionim fajlovima servera.

Takođe, bitan deo koda je dobijanje Connection interfejsa od injektovanog DataSource interfejsa pomoću poziva dataSource.getConnection(). Ostatak koda je standardno postavljanje upita i čitanje rezultata upita preko JDBC interfejsa.

Ostaje pitanje koje tačno izmene fazi JDBC interfejsa treba napraviti da bi on mogao da se koristi na gore opisani način?

Prvo, samo konektovanje na fazi bazu podataka se izvršava preko FuzzyDriver interfejsa (sekcija 5.4.1), odnosno njenog metoda connect(). Ovom metodu se prosleđuju parametri baze podataka. U tom segmentu je potrebna izmena zbog toga što su parametri baze podataka dati u konfiguracionim fajlovima servera. Umesto njih, ovom metodu je potrebno proslediti samo JNDI naziv željene baze podataka.

Nadalje, metod connect(), koristeći prosleđeni JNDI naziv, treba da zatraži DataSource interfejs od kontejnera. Međutim, ako bi se on izvršavao u okviru session bean-a u koji je ovaj interfejs injektovan pomoću anotacije, kao što je prikazano na listingu 7.13, onda ne bi mogla da se specificira željena baza podataka u vreme izvršavanja. U ovom slučaju bi ona morala biti zadata u kodu. Zbog toga se ovde koristi malo drugačija varijanta koda datog na listingu 7.13, naime, injekcija pomoću anotacije se zamenjuje JNDI lookup-om. Na taj način se JNDI ime prosleđeno u vreme izvršavanja traži u JNDI bazi podataka i dobija DataSource interfejs.

Na listingu 7.14 je prikazana nova varijanta metode connect() implementirane na opisani način. DataSource interfejs dobijen na ovaj način je jednak onome dobijenom u primeru datom na listingu 7.13, ali JNDI ime može da se prosledi u vreme izvršavanja kao parametar metoda connect().

```
public FuzzyConnection connect(String DBJNDI)
  throws InstantiationException,
    IllegalAccessException,
    ClassNotFoundException,
    SQLException,
```

```
    FuzzyMetaDataException{
  Connection conn = null;
  FuzzyConnection fC = null;
  try {
    InitialContext ctx = new InitialContext();
    DataSource dataSource =
      (DataSource)ctx.lookup(DBJNDI);
    conn = dataSource.getConnection();
    fC = new FuzzyConnection();
    fC.setConn(conn);
    if(!isFDBMetaDataOK(conn)){
      throw new FuzzyMetaDataException("Illegal format"+
        "of fuzzy meta data");
    }
  } catch (InstantiationException e) {
    throw e;
  } catch (IllegalAccessException e) {
    throw e;
  } catch (ClassNotFoundException e) {
    throw e;
  } catch (SQLException e) {
    throw e;
  } catch (FuzzyMetaDataException e) {
    throw e;
  }
  return fC;
}
```

Listing 7.14. Nova varijanta metode connect() klase FuzzyDriver.

Kada se dobije DataSource interfejs i od njega zatraži Connection interfejs, onda se jednostavno od njega napravi FuzzyConnection objekat koji čuva konekciju. Ostatak koda drajvera može da ostane nepromenjen zbog toga što nadalje radi sa standardnom JDBC konekcijom.

Opisane izmene u kodu drajvera su i učinjene, te je na taj način dobijena nova varijanta fazi JDBC drajvera koja je sasvim upotrebljiva na EJB 3.0 srednjem sloju aplikacije.

7.2.3 Treća mogućnost

U prethodnoj sekciji je opisana varijanta fazi JDBC drajvera koja omogućava upotrebu jezika PFSQL sa EJB 3.0 srednjeg sloja, ali takav način rada sa bazom podataka ima brojne nedostatke.

Ovakav pokušaj je jednak pokušaju da se razvije EJB 3.0 srednji sloj neke aplikacije tako što se bazi podataka pristupa direktno preko SQL jezika. Iako ova tehnologija nudi različite mogućnosti za to, ovakav pristup svakako nije preporučljiv. Problemi koji nastaju na taj način su istaknuti u sekciji 7.1.3. Da bi se ti nedostaci prevazišli, u okviru EJB 3.0 specifikacije je definisan Java Persistence API (JPA), odnosno, širok spektar mogućnosti koje nude entity bean-ovi i prateći mehanizmi koji se na ovaj način potpuno zaobilaze.

Dakle, potrebno je obezbediti rad sa fazi podacima u bazi podataka i fazi proširenjima jezika za pristup tim podacima ali pod pretpostavkom da se za rad sa bazom podataka koristi JPA. Drugim rečima, potrebno je, po uzoru na PFSQL definisati fazi proširenja jezika EJB QL. Iako ovaj jezik veoma podseća na SQL, između njih postoje značajne razlike, pa je potrebno ideje za proširenje SQL jezika prikazane u petom poglavlju prilagoditi tim razlikama.

Da bi se uspešno implementirao interpreter za ovakav jezik, potrebno je obezbediti i objektno-relaciono mapiranje fazi meta modela pomoću entity bean-ova.

Na kraju, postavlja se pitanje kako uopšte na EJB 3.0 srednjem sloju realizovati ova proširenja EJB QL jezika? U sekciji 7.1.4 su opisane interceptor metode, odnosno mehanizmi presretanja poziva metoda. Na istom mestu je konstatovano da se taj mehanizam može iskoristiti za proširenje funkcionalnosti EJB 3.0 kontejnera. Dakle, koristeći ove mehanizme se mogu presresti upiti u proširenom i fazifikovanom EJB QL jeziku. Sledeći ideju opisanu u ovoj disertaciji, ovakvi upiti se nadalje mogu transformisati u EJB QL upite koje izvršava kontejner. Odgovor korisniku bi se takođe morao presresti da bi se izvršilo izračunavanje fazi stepena zadovoljenja objekata u skupu rezultata upita, na sličan način kao kod PFSQL-a.

Ova veoma obimna tema predstavlja još jedan mogući pravac daljeg razvoja istraživanja prikazanog u ovoj disertaciji.

ZAKLJUČAK

Fazi logika i fazi skupovi su se pokazali kao prirodno rešenje za modeliranje neodređenosti u relacionim bazama podataka. Realizacija skupa alata koji omogućava upotrebu ovih mehanizama u razvoju aplikacija koje se oslanjaju na bazu podataka predstavlja najznačajniji doprinos ove disertacije.

Proširenjem relacionog modela podataka elementima fazi logike je realizovan novi model za skladištenje fazi informacija. Ovaj model proširuje relacioni model podataka mogućnostima za skladištenje fazi vrednosti i predstavlja osnovu za rad interpretera za jezik PFSQL. Poređenjem sa FIRST-2 modelom se došlo do zaključka da je predstavljeni model funkcionalno podskup FIRST-2 modela, ali i da je to model prilagođen efikasnom izvršavanju PFSQL upita i da su mehanizmi za skladištenje podataka bitno drugačiji.

Uz prikazani model je implementiran i prototip CASE alata za projektovanje modela baza podataka po tom modelu. Time je otvorena mogućnost za projektovanje fazi baze podataka bez poznavanja detalja interne realizacije proširenja relacionog modela. Ovaj CASE alat predstavlja i prvu aplikaciju te vrste.

Detaljnom analizom mogućnosti za proširenje SQL-a fazi elementima se došlo do jezika PFSQL. U okviru njega je omogućeno postavljanje upita sa prioritetom gde je uslov proizvoljan logički izraz. Taj jezik ujedno predstavlja i prvo proširenje SQL-a elementima fazi logike koje može da realizuje upite sa uslovima različitog prioriteta.

Kao teorijska osnova za izračunavanje rezultata PFSQL upita su uvedeni GPFCSP sistemi. Opisana je kompletna gradacija koja nastaje u pokušaju fazifikacije problema zadovoljenja ograničenja prioritetnom fazi logikom.

Ideja da se proces obrade upita proširi i da se realizuje u okviru fazi JDBC drajvera je prvi put opisana i realizovana u ovoj doktorskoj disertaciji. Taj mehanizam za obradu upita je potpuno nezavisan od korišćenog sistema za upravljanje bazama podataka. Za konekciju na bazu podataka se koristi odgovarajući JDBC drajver, a upiti se obrađuju van baze podataka.

U poslednjem poglavlju disertacije je data diskusija o mogućnostima upotrebe opisanih mehanizama na srednjem sloju višeslojne aplikacije. Ova ideja se takođe ne može sresti nigde u postojećoj literaturi.

Kompletan kod, dokumentacija i finalne verzije svih opisanih aplikacija i interfejsa se mogu naći na sledećem web sajtu: http://www.is.im.ns.ac.yu/fuzzydb

Kao bitan rezultat ove doktorske disertacije se može navesti i značajan broj problema i novih pravaca istraživanja koje prezentovani rezultati otvaraju. Proširivanje PFSQL jezika tako da on prihvata što širu klasu upita predstavlja stalni zadatak. U petom poglavlju je pomenuto da bi uvođenje fazi GROUP BY klauzule i razmatranje fazifikacije veza između tabela mogli biti sledeći ciljevi koje treba postići. U istom poglavlju je pomenuto da skupovne funkcije SUM i AVG, koje spadaju u standardne, ne mogu imati fazi vrednost kao argument zbog toga što sabiranje i druge operacije sa fazi vrednostima nisu podržane. Kako teorijski postoji mogućnost da se i ove osobine dodaju, mogao bi se razmotriti način da se to uradi u sledećoj verziji jezika PFSQL i interpretera za njega.

Takođe, postoje brojne mogućnosti za proširenje modela za skladištenje fazi informacija. U četvrtom poglavlju je spomenuta mogućnost uvođenja mehanizma koji bi omogućio fazifikaciju veza između tabela.

Upotreba elemenata fazi logike u XML native bazama podataka, razmatrana u petom poglavlju, se nameće kao veoma interesantna i aktuelna tema za razmišljanje i istraživanje.

Ipak, najširi zadatak, prvi te vrste, bi predstavljao pokušaj da se opisani mehanizmi koriste na srednjem sloju višeslojne aplikacije i da se realizuju proširenja EJB QL jezika elementima fazi logike.

LITERATURA

1. Appel, A. W. (2002). *Modern Compiler Implementation in Java* (2nd ed.). New York: Cambridge University Press.
2. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., & Fargier, H. (1999). Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. *Constraints , 4* (3), 199-240.
3. Bodenhofer, U. (2003). Representations and constructions of similarity-based fuzzy orderings. *Fuzzy Sets and Systems , 137* (1), 113-136.
4. Bosc, P., & Pivert, O. (1994). Fuzzy Queries and Relational Databases. *Proceedings of the 1994 ACM Symposium on Applied Computing*, (pp. 170-174). Phoenix,AZ.
5. Bosc, P., & Pivert, O. (1995). SQLf: A Relational Database Language for Fuzzy Querying. *IEEE Transactions on Fuzzy Systems , 3* (1), 1-17.
6. Buckles, B., & Petry, F. (1982). A Fuzzy Representation of Data for Relational Databases. *Fuzzy Sets and Systems , 7* (3), 213-226.
7. Chaudhry, N., Moyne, J., & Rundensteiner, E. (1994). A Design Methodology for Databases with Uncertain Data. *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database Management* (pp. 32-41). Charlottesville, VA: IEEE Computer Society.
8. Chen, G. (1998). *Fuzzy Logic in Data Modeling, Semantics, Constraints and Database Design* (The Kluwer International Series on Advances in Database Systems ed.). Springer.
9. Chen, G., & Kerre, E. (1998). Extending ER/EER Concepts Towards Fuzzy Conceptual Data Modeling. *Proceedings of the IEEE International*

Conference on Fuzzy Systems (pp. 1320-1325). Beijing: IEEE Computer Society.

10. Copeland, T. (2007). *Generating Parsers with JavaCC.* Alexandria, VA: Centennial Books.
11. Dubois, D., & Fortemps, P. (1999). Computing Improved Optimal Solutions to max-min Flexible Constraint Satisfaction Problems. *European Journal of Operational Research , 118*, 95-126.
12. Dubois, D., & Prade, H. (1980). *Fuzzy Sets and Systems: Theory and Applications* (Mathematics in Science and Engineering ed., Vol. 144). Academic Press.
13. Fortemps, P., & Pirlot, M. (2004). Conjoint Axiomatization of Min, DiscriMin and LexiMin. *Fuzzy Sets and Systems , 148* (2), 211-229.
14. Galindo, J., Medina, J., & Aranda, M. (1999). Querying Fuzzy Relational Databases Through Fuzzy Domain Calculus. *International Journal of Intelligent Systems , 14* (4), 375-411.
15. Galindo, J., Medina, J., Cubero, J., & Garcia, M. (2001). Realxing the Universal Quantifier of the Division in Fuzzy Relational Databases. *International Journal of Intelligent Systems , 16* (6), 713-742.
16. Galindo, J., Medina, M., Pons, O., & Cubero, J. (1998). A Server for Fuzzy SQL Queries. In T. Andreasen, H. Christiansen, & H. Larsen (Eds.), *Lecture Notes in Artificial Intelligence* (Vol. 1495, pp. 164-174). Springer.
17. Galindo, J., Urrutia, A., & Piattini, M. (2006). *Fuzzy Databases: Modeling Design and Implementation.* Hershey:PA: IDEA Group.
18. Galindo, J., Urrutia, A., & Piattini, M. (2004). Representation of fuzzy knowledge in relational databases. *Proceedings of the 15th International Workshop on Database and Expert Systems Applications*, (pp. 917-921). Saragosa, Spain.
19. Keith, M., & Schincariol, M. (2006). *Pro EJB 3: Java Persistence API* (1st ed.). Berkeley, CA: Apress.

20. Kerre, E., & Chen, G. (2000). Fuzzy Data Modeling at a Conceptual Level: Extending ER/EER Concepts. In O. Pons (Ed.), *Knowledge Management in Fuzzy Databases* (pp. 3-11). Heidelberg: Physica Verlag.

21. Klement, E., Mesiar, R., & Pap, E. (2000). Triangular Norms. In *Series: Trends in Logic* (Vol. 8). Dordrecht: Kluwer Academic Publishers.

22. Lukasiewicz, J. (1920). O logice trojwartosciowej. *Ruch filozoficzny , 5*, 170-171.

23. Luo, X., Lee, J., Leung, H., & Jennings, N. (2003). Prioritised Fuzzy Constraint Satisfaction Problems: Axioms, Instantiation and Validation. *Fuzzy Sets and Systems , 136* (2), 151-188.

24. Ma, Z. (2005). *Fuzzy Database Modelling With XML* (1st ed.). New York: Springer.

25. Ma, Z., & Yan, L. (2007). Fuzzy XML data modeling with the UML and relational data models. *Data & Knowledge Engineering , 63* (3), 972-996.

26. Medina, J., Pons, O., & Vila, M. (1994). GEFRED: A Generalized Model for Fuzzy Relational Databases. *Information Sciences , 76* (1), 87-109.

27. Monson-Haefel, R., & Burke, B. (2006). *Enterprise JavaBeans 3.0* (5th ed.). Sebastopol, CA: O'Reilly Media, Inc.

28. Panda, D., Rahman, R., & Lane, D. (2007). *EJB 3 in Action* (1st ed.). Greenwich, CT: Manning Publications.

29. Pap, E., & Takači, A. (2005). An Application of Schur-concave t-norms in PFCSP. *Proceedings of the 4th Conference of the EUSFLAT*, (pp. 380-384). Barcelona, Spain.

30. Škrbić, S. (2004). *Upravljanje bibliografskim zapisima u XML tehnologiji.* (Magistarska teza) Novi Sad: PMF.

31. Škrbić, S., & Surla, D. (2008). Bibliographic records editor in XML native environment. *Software-Practice and Experience , 38* (5), 471-491.

32. Škrbić, S., & Takači, A. (2008). On Development of Fuzzy Relational Database Applications. *Proceedings of the 12th International Conference on*

Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2008), (pp. 268-273). Malaga, Spain.

33. Takači, A. (2006). Handling Priority Within a Database Scenario. *ETF Journal of Electrical Engineering , 17*, 130-134.
34. Takači, A. (2005). Schur-concave Triangular Norms: Characterization and Application in PFCSP. *Fuzzy Sets and Systems, 155* (1), 50-64.
35. Takači, A. (2006). Towards Priority Based Logics. *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2006)*, (pp. 651-657). Paris, France.
36. Takači, A. (2006). *Trougaone norme prioriteta i njihova primena na modeliranje ispunjenja fazi ograničenja* (Doktorska disertacija). Novi Sad: Prirodno-matematički fakultet.
37. Takači, A., & Škrbić, S. (2007). Comparing Priority, Weighted and Queries with Threshold in PFSQL. *Proceedings of the 5th Serbian-Hungarian Joint Symposium on Intelligent Systems and Informatics*, (pp. 77-80). Subotica.
38. Takači, A., & Škrbić, S. (2008). Data Model of FRDB with Different Data Types and PFSQL. In J. Galindo (Ed.), *Handbook of Research on Fuzzy Information Processing in Databases* (pp. 403-430). Hershey, PA: Information Science Reference.
39. Takači, A., & Škrbić, S. (2005). How to Implement FSQL and Priority Queries. *Proceedings of the 3rd Serbian-Hungarian Joint Symposium on Intelligent Systems*, (pp. 261-267). Subotica.
40. Takači, A., & Škrbić, S. (2007). Measuring the Similarity of Different Types of Fuzzy Sets in FRDB. *Proceedings of the 5th Conference of the EUSFLAT*, (pp. 247-252). Ostrava, Czech Republic.
41. Takači, A., & Škrbić, S. (2008). Priority, Weight and Threshold in Fuzzy SQL Systems. *Acta Polytechnica Hungarica , 5* (1), 59-68.

42. Takači, A., & Škrbić, S. (2007). Short Review of Fuzzy Relational Databases. *Proceedings of the 51st ETRAN* (u štampi). Herceg Novi: Elektrotehnički fakultet, Beograd.

43. Takači, A., Perović, A., & Jovanović, A. (2008). Measuring with Priority Based Logic. *Proceedings of the 12th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2008)*, (pp. 1490-1496). Malaga, Spain.

44. Umano, M., & Fukami, S. (1994). Fuzzy Relational Algebra for Possibility-Distribution-Fuzzy-Relational Model of Fuzzy Data. *Journal of Intelligent Information Systems , 3* (1), 7-27.

45. Werner, M. (2003). A Parser Project in a Programming Languages Course. *Journal of Computing Sciences in Colleges , 18* (5), 184-192.

46. Zadeh, L. (1965). Fuzzy Sets. *Information and Control , 8* (3), 338-353.

47. Zadeh, L. (1971). Similarity Relations and Fuzzy Orderings. *Information Sciences , 3*, 177-200.

48. Zvieli, A., & Chen, P. (1986). ER Modeling and Fuzzy Databases. *Proceedings of the Second International Conference on Data Engineering* (pp. 320-327). Los Angeles: IEEE Computer Society.

SKRAĆENICE

API - Application Programming Interface
CASE - Computer Aided Software Engineering
CSP - Constraint Satisfaction Problem
DDL - Data Definition Language
EBNF - Extended Backus-Naur Form
EER - Extended Entity Relationship
EJB - Enterprise JavaBeans
EJB QL - Enterprise JavaBeans Query Language
ER - Entity Relationship
FCSP - Fuzzy Constraint Satisfaction Problem
FMB - Fuzzy Metaknowledge Base
FSQL - Fuzzy Structured Query Language
GEFRED - Generalized Model of Fuzzy Relational Databases
GPFCSP - Generalized Priority Fuzzy Constraint Satisfaction Problem
J2EE - Java 2 Enterprise Edition
JDBC - Java Database Connectivity
JNDI - Java Naming and Directory Interface
JPA - Java Persistence API
PFCSP - Priority Fuzzy Constraint Satisfaction Problem
PFSQL - Priority Fuzzy Structured Query Language
POJO - Plain Old Java Object
RPC - Remote Procedure Call
SQL - Structured Query Language
UML - Unified Modeling Language
XML - eXstensible Markup Language

BIOGRAFIJA

Srđan Škrbić je rođen 17. februara 1978. godine u Somboru. Na Prirodno-matematički fakultet Univerziteta u Novom Sadu, Odsek za matematiku, Smer diplomirani informatičar upisao se školske 1996/97. godine. U periodu 1996-2001. godine je položio sve ispite predviđene planom i programom sa prosečnom ocenom 9.48. Diplomski rad je odbranio u novembru 2001. godine sa ocenom 10.

U decembru 2001. godine je upisao poslediplomske studije, smer Informatika. U narednom periodu je radio kao istraživač stipendista Ministarstva za nauku, tehnologije i razvoj Republike Srbije. U januaru 2003. godine je zasnovao radni odnos na Prirodno-matematičkom fakultetu u Novom Sadu, kao asistent-pripravnik.

Kao stipendista, i kasnije kao asistent-pripravnik, od 2002 – 2005. godine učestvuje na projektu „XML tehnologije i kooperativni informacioni sistemi“, koji finansira Ministarstvo nauke i zaštite životne sredine Republike Srbije. Od 2002. godine radi na projektu razvoja informacionog sistema Prirodno-matematičkog fakulteta u Novom Sadu.

Na poslediplomskim studijama položio je sve ispite predviđene planom i programom sa ocenom 10. U decembru 2004. je odbranio magistarsku tezu pod naslovom “Upravljanje bibliografskim zapisima u XML tehnologiji” i stekao akademski naziv magistra računarskih nauka. U periodu od septembra 2004. do juna 2005. je bio na odsluženju vojnog roka.

U oktobru 2005. godine je izabran za asistenta na Prirodno-matematičkom fakultetu u Novom Sadu. od januara 2006. godine učestvuje

na projektu „Apstraktni modeli i primene u računarskim naukama“ koji finansira Ministarstvo nauke Republike Srbije. U februaru 2006. je primio nagradu “Mileva Marić-Ajnštajn“ za najbolju magistarsku tezu.

Držao je nastavu u okviru desetak akademskih kurseva koji pripadaju oblasti informacionih sistema. Objavio je 20 naučnih radova.

Oženjen je i ima sina starog godinu dana. Od stranih jezika govori engleski.

Novi Sad, 20.8.2008. Srđan Škrbić

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj: **RBR**	
Identifikacioni broj: **IBR**	
Tip dokumentacije: **TD**	Monografska dokumentacija
Tip zapisa: **TZ**	Tekstualni štampani materijal
Vrsta rada: **VR**	Doktorska disertacija
Autor: **AU**	mr Srđan Škrbić
Mentor: **MN**	dr Miloš Racković, redovni profesor
Naslov rada: **NR**	Upotreba fazi logike u relacionim bazama podataka
Jezik publikacije: **JP**	srpski (latinica)
Jezik izvoda: **JI**	srpski/engleski
Zemlja publikovanja: **ZP**	Srbija
Uže geografsko područje: **UGP**	Vojvodina
Godina: **GO**	2008
Izdavač: **IZ**	Autorski reprint
Mesto i adresa: **MA**	Prirodno-matematički fakultet, Trg Dositeja Obradovića 4, Novi Sad
Fizički opis rada: **FO**	7/160/40/0/48/0
Naučna oblast: **NO**	Informatika
Naučna disciplina: **ND**	Informacioni sistemi

Predmetna odrednica/ ključne reči: **PO** **UDK**	Fazi - relacione baze podataka, Fazi SQL jezik, Fazi problem zadovoljenja ograničenja
Čuva se: **ČU**	Biblioteka Departmana za matematiku i informatiku PMF-a u Novom Sadu
Važna napomena: **VN**	Nema
Izvod: **IZ**	Doktorska disertacija pripada oblasti informacionih sistema, odnosno podoblasti koja se bavi upravljanjem skladištenjem i pretraživanjem informacija. Osnovni cilj disertacije je modeliranje i implementacija skupa alata koji omogućavaju upotrebu fazi logike u radu sa relacionim bazama podataka. Da bi se do tog skupa alata došlo, najpre je relacioni model podataka proširen elementima teorije fazi skupova, a zatim je definisano fazi proširenje upitnog jezika SQL – PFSQL. Interpreter za taj jezik je implementiran u okviru fazi JDBC drajvera koji, osim implementacije interpretera, sadrži i elemente koji omogućavaju jednostavnu upotrebu ovih mehanizama iz programskog jezika Java. Skup alata je zaokružen implementacijom CASE alata za razvoj fazi-relacionog modela baze podataka. Osim toga, razmatrane su i mogućnosti za upotrebu PFSQL jezika u višeslojnim aplikacijama.
Datum prihvatanja teme od NN veća: **DP**	24.11.2006.
Datum odbrane: **DO**	
Članovi komisije: **KO**	

Predsednik:	dr Dušan Surla, red. prof., PMF, Novi Sad
član:	dr Miloš Racković, red. prof., PMF, Novi Sad, mentor
član:	dr Ivan Luković, red. prof., FTN, Novi Sad
član:	dr Vladan Devedžić, red. prof., FON, Beograd
član:	dr Aleksandar Takači, docent, TF, Novi Sad

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCE AND MATHEMATICS
KEY WORDS DOCUMENTATION

Accession number: **ANO**	
Identification number: **INO**	
Document type: **DT**	Monograph publication
Type of record: **TR**	Textual printed material
Content code: **CC**	Doctoral dissertation
Author: **AU**	Srđan Škrbić
Mentor/comentor: **MN**	Miloš Racković, Ph. D., full professor
Title: **TI**	Fuzzy logic usage in relational databases
Language of text: **LT**	Serbian (Latin)
Language of abstract: **LA**	Serbian/English
Country of publication: **CP**	Serbia
Locality of publication: **LP**	Vojvodina
Publication year: **PY**	2004
Publisher: **PU**	Author's reprint
Publication place: **PP**	Faculty of Science and Mathematics, Trg Dositeja Obradovića 4, Novi Sad
Physical description: ***PD***	7/160/40/0/48/0
Scientific field: **SF**	Informatics
Scientific discipline: **SD**	Information Systems

Subject/ Key words: Fuzzy-relational databases, Fuzzy SQL language, Fuzzy constraint satisfaction problem
SKW
UC

Holding data: Library of Department of Mathematics and Informatics, Trg Dositeja Obradovića 4
HD

Note: None
N

Abstract:
AB

This doctoral dissertation belongs to the field of information systems, subfield information storage and retrieval management. The main subject of the dissertation is modeling and implementation of a set of tools that allow usage of fuzzy logic in relational database applications

In order to achieve that goal, at first, the relational data model is extended with elements of fuzzy set theory. After that, a fuzzy extension of the SQL query language, called PFSQL, is defined. An interpreter for that language is implemented as a part of the fuzzy JDBC driver. Beside the implementation of the interpreter, this fuzzy JDBC driver contains elements that allow simple usage of offered mechanisms from Java programming language. The set of tools is concluded with the implementation of the CASE tool for the development of fuzzy-relational data models. In addition, possibilities to use PFSQL language on the middle tier of multi tier systems are discussed.

Accepted by the Scientific Board:
ASB

Defended on:
DE

Thesis defend board:
DB

President:	Dušan Surla, Ph.D., full prof., Faculty of Science and Mathematics, Novi Sad
Member:	Miloš Racković, Ph.D., full prof., Faculty of Science and Mathematics, Novi Sad - menthor
Member:	Ivan Luković, Ph.D., full prof., Faculty of Engineering, Novi Sad.
Member:	Vladan Devedžić, Ph.D., full prof., FON – School of Business Administration, Novi Sad
Member:	Aleksandar Takači, Ph.D., assist. prof., Faculty of Technology, Novi Sad