SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Implementacija binarnog stabla u PostgreSQL-u — poopćene baze podataka

Projektna dokumentacija

Domagoj Šitum
br. indeksa: 42571\13-R

Mentor:
Doc. dr. sc. Markus Schatten

Varaždin, 25. kolovoza 2014.

Sadržaj

Poglavlje 1

Uvod

PostgreSQL, kao sustav za upravljanje bazama podataka, ima široku funkcionalnost. Tako je u njemu, između ostaloga, moguće i dodavanje novih tipova podataka, funkcija, operatora i sl. Ovaj rad će opisati implementaciju i korištenje binarnih stabala u PostgreSQL sustavu za upravljanje bazama podataka. Čitava implementacija je napisana u programskom jeziku C. Vidjet ćemo na koji način PostgreSQL prihvaća unos od korisnika i kako ga pohranjuje i ispisuje u bazu podataka. Osim toga, definirat ćemo sve funkcije potrebne za rad s binarnim stablom.

Poglavlje 2

Priprema razvojnog okruženja

Prije početka rada, potrebno je pripremiti razvojno okruženje za rad. Najprije je potrebno instalirati PostgreSQL. Bitno je da je verzija veća od 8.x, zato jer programski kod nije kompatibilan s verzijama PostgreSQL-a koje koriste staru *"Version 0"* konvenciju. No, više o tome u nastavku.

Budući da ne programiramo samoizvršni program, nego *plugin* (ekstenziju) za PostgreSQL, sam programski kod (pisan u C-u) nećemo moći kompilirati služeći se samo naredbom *gcc*. Umjesto toga je potrebno stvoriti *Makefile* datoteku koja će pomoći u kompiliranju. Sadržaj te datoteke je prikazan ispod:

```
MODULES = stablo
PGXS := $(shell pg_config --pgxs)
include $(PGXS)
```

Iz priloženog koda možemo vidjeti da se plugin (ekstenzija, modul) koji stvaramo zove stablo. Zbog toga razloga je potrebno napraviti i novu C datoteku naziva *stablo.c*, koju će *Makefile* skripta koristiti pri kompiliranju. Druge dvije linije koda predstavljaju drugu *Makefile* skriptu koja se koristi pri kompiliranju. Ta druga skripta je nativna skripta sustava PostgreSQL koja služi isključivo za stvaranje novih pluginova. Lokacija do te skripte je definirana naredbom *pg_config --pgxs*. Međutim, da bi mogli koristiti tu naredbu, moramo instalirati dodatak za PostgreSQL pod nazivom *postgresql-server-dev-X.Y*, gdje je X.Y verzija PostgreSQL sustava.

Tip podataka koji smo napisali u *stablo.c* datoteci možemo kompilirati sa naredbom ***make***, a dodati ga u PostgreSQL koristeći naredbu ***make install***.

Poglavlje 3

Unos i ispis stabla u PostgreSQL-u

Svaki tip podataka u PostgreSQL-u mora omogućiti najmanje dvije operacije - unos podataka i ispis podataka. Te dvije operacije su predstavljene s dvije funkcije (PostgreSQL, 2014d).

Funkcija za unos podataka se koristi kod operacija umetanja i ažuriranja (INSERT i UPDATE) dok se funkcija za ispis podataka koristi kod naredbe za dohvaćanje podataka iz baze (SELECT). Jasno je da funkcija za unos stabla pretvara korisnički unos u objekt binarnog stabla, dok je funkcija za ispis stabla inverzna funkciji za unos – pretvara objekt binarnog stabla u tekstualnu reprezentaciju stabla. Međutim, sada se postavlja pitanje kako pomoću ekstenzije *stablo* unijeti binarno stablo u PostgreSQL. Postoji mnogo načina da se to izvede, a način koji je odabran u ovom radu ne mora biti najbolji niti najjednostavniji. Najjednostavnije bi bilo binarno stablo definirati grafički. Međutim, zbog nemogućnosti grafičke definicije, bilo se potrebno dosjetiti alternativnih načina. U ovom *plugin*-u, binarno se stablo unosi u čvorovima. Svaki čvor je definiran uređenom četvorkom na sljedeći način:

```
(ID,VRIJEDNOST_CVORA,ID_LIJEVOG_DJETETA,ID_DESNOG_DJETETA)
```

Pri tome sve četiri definirane vrijednosti, koje čine jedan čvor, moraju biti cjelobrojne. Ukoliko se u stablo želi istovremeno unijeti više od jednog čvora, tada čvorovi moraju međusobno biti odvojeni znakom **;**.

Komponente čvora su definirane na sljedeći način: *ID* predstavlja identifikacijski broj čvora. Svaki čvor mora imati jedinstveni identifikacijski broj koji ga razlikuje od drugih čvorova. *ID* čvora poprima vrijednosti veće ili jednake nuli. *VRIJEDNOST_CVORA* predstavlja vrijednost koja se upisuje u sam čvor. Ta vrijednost je, radi jednostavnosti, također cijeli broj. Komponente *ID_LIJEVOG_DJETETA* i *ID_DESNOG_DJETETA* mogu poprimiti identifikacijske oznake stvarnih čvorova ili -1, ukoliko lijevi, odnosno desni, čvor ne postoji.

Dakle, ako je stablo definirano sa primjerice *(5,45,-1,6);(6,99,-1,-1)*, tada korijen stabla ima identifikacijsku oznaku 5 i vrijednost 45. Korijen nema lijevo dijete, a njegovo desno dijete predstavlja čvor sa identfikacijskom oznakom 6. Desno dijete korijena ima vrijednost 99 i nema djece.

3.1 Vrste implementacija binarnog stabla

Postoje dvije temeljne implementacije binarnog stabla, a to su implementacija uz pomoć polja i implementacija uz pomoć pokazivača (Orehovački, 2011). Implementacija uz pomoć polja alocira odjednom unaprijed određeni memorijski prostor koji će se koristiti za sve čvorove toga stabla. Za razliku od toga, implementacija uz pomoć pokazivača svaki čvor alocira na zasebnu memorijsku lokaciju.

Oba pristupa imaju svoje prednosti i nedostatke. Prednost implementacije binarnog stabla pomoću pokazivača je ta što ne moramo unaprijed znati koliko će memorijskog prostora biti potrebno za alokaciju, budući da čvorove stabla alociramo po potrebi. Međutim, glavni nedostatak te implementacije je taj što takvo stablo teško možemo pohraniti na sekundarnu memoriju. Razlog zbog kojeg je to tako je taj što su adrese lijevog i desnog djeteta svakog čvora apsolutne memorijske adrese. Budući da nam ništa ne može garantirati to da će se program sljedeći puta pokrenuti na istoj memorijskoj lokaciji, ne možemo koristiti implementaciju binarnog stabla uz pomoć pokazivača. Osim toga, kada bi PostgreSQL-u proslijedili korijen stabla, da ga pohrani na disk, on bi morao imati ugrađeni mehanizam koji bi pohranio i sve ostale čvorove toga stabla, što je nemoguće, budući da PostgreSQL nije implementiran da funkcionira na taj način.

Dakle, zbog navedenih nedostataka implementacije binarnog stabla uz pomoć pokazivača, u ovom projektu će se koristiti **implementacija binarnog stabla uz pomoć polja**. Prednost takve implementacije je mogućnost pohranjivanja binarnog stabla u sekundarnu memoriju. Za potrebe ovog seminara, je korišteno binarno stablo dubine 8, odnosno, binarno stablo od ukupno 511 raspoloživih čvorova.

3.2 Funkcija za unos binarnog stabla u bazu

Već je ranije spomenuto kako binarno stablo mora imati funkcije za unos i ispis binarnog stabla. U ovom poglavlju će se opisati funkcija za unos binarnog stabla u bazu podataka. Ta funkcija pretvara tekstualni zapis u objekt binarnog stabla te ga prosljeđuje PostgreSQL-u, koji ga pohranjuje u bazi podataka. Pogledajmo kako to funkcionira u programskom kodu koji slijedi:

```
PG_FUNCTION_INFO_V1(stablo_in);

/*
Priprema ulazni niz znakova za pohranu u bazu podataka
 */
Datum stablo_in(PG_FUNCTION_ARGS) {
        char *unos = PG_GETARG_CSTRING(0);
        bstablo *rezultat;

        rezultat = pretvoriUStablo(unos);

        PG_RETURN_POINTER(rezultat);
}
```

Prije definicije same funkcije vidimo makro naredbu *PG_FUNCTION_INFO_V1(ime_funkcije)*. Ta makro naredba simbolizira korištenje već spomenute *"Version 1"* konvencije (PostgreSQL, 2014a). Ta konvencija omogućava jednostavno pisanje funkcija koje će se koristiti u PostgreSQL-u. Sve takve funkcije imaju tip *Datum*. Taj tip predstavlja interni tip podataka u PostgreSQL-u i mijenja svaki drugi nepoznati tip podataka koji se u C jezik dohvaća iz PostgreSQL-a. Nova konvencija zahtijeva i korištenje makro naredbe *PG_FUNCTION_ARGS*, umjesto popisivanja svih argumenata koje funkcija prima. Nakon što C jezik obradi podatke, njegova funkcija vraća vrijednost PostgreSQL-u pomoću makro naredbe *PG_RETURN_X(ime_varijable)*, pri čemu *X* može predstavljati *CSTRING* za niz znakova, *INT32* za cjelobrojni tip podataka, *BOOL* za boolean tip podataka, *DOUBLE* za brojčani

tip podataka s pomičnim zarezom (veličine 8 bajta), *POINTER* za bilo koji dinamički alocirani tip podataka.

Budući da argumenti funkcije nisu definirani (nego se umjesto njih koristi makro naredba), potrebno ih je dohvatiti uz pomoć makro naredbe *PG_GETARG_X(i)*, pri čemu je *i* redni broj argumenta (počevši od nule), a X tip podatka koji želimo dohvatiti (opisan u prethodnom paragrafu). Nakon što se dohvate svi argumenti, na opisani način, moguće je izvršavati naredbe koristeći standardni C jezik i biblioteke koje su na raspolaganju. PostgreSQL zahtijeva pisanje C-funkcija u C90 standardu, što znači da je na početku svakog bloka potrebno definirati sve varijable.

Nakon što učita stablo u obliku tekstualnog niza, funkcija za unos binarnog stabla parsira stablo i pretvara ga u memorijski objekt. Parsiranje i pretvorba teksta u objekt binarnog stabla je prepuštena funkciji *pretvoriUStablo(char *znakovniNiz)* koja poziva niz drugih funkcija. Sve te funkcije ćemo opisati u nastavku.

3.3 Funkcija za ispis binarnog stabla na zaslon

Kada korisnik SELECT naredbom u bazi podataka zatraži ispis binarnog stabla ili nekog njegovog podstabla, tada je nakon izvršene operacije to stablo potrebno pretvoriti iz binarnog zapisa natrag u tekstualni. U tu svrhu nam koristi sljedeća funkcija:

```
PG_FUNCTION_INFO_V1(stablo_out);

/*
Pretvara stablo iz baze podataka u izlazni niz znakova
 */
Datum stablo_out(PG_FUNCTION_ARGS) {
        bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
        char *izlaz = (char *) palloc(10000);

        // alociranoj varijabli izlaz za prvi znak
        // postavljamo terminator znak
        izlaz[0] = '\0';
        preorder(stablo, INDEKS_KORIJENA_STABLA, izlaz);

        PG_RETURN_CSTRING(izlaz);
}
```

Nakon što funkcija učita svoj prvi i jedini argument, binarno stablo, tada najprije alocira polje *izlaz* u koje će se pohraniti tekstualni zapis binarnog stabla. Nakon toga se stablo u tekstualnom obliku pohranjuje u varijablu *izlaz* načinom *preorder* koji ćemo, skupa s *inorder* i *postorder*, objasniti u nekom od sljedećih poglavlja.

Poglavlje 4

Implementacija binarnog stabla pomoću polja

U ovom poglavlju ćemo definirati sve funkcije koje su korištene za izradu apstraktnog tipa podataka – binarnog stabla. Funkcije koje su korištene su sljedeće:

- **roditeljB** – dohvaća indeks roditelja čvora
- **lijevoDijeteB** – dohvaća indeks lijevog djeteta čvora
- **desnoDijeteB** – dohvaća indeks desnog djeteta čvora
- **vrijednostB** – vraća vrijednost čvora
- **promijeniVrijednostB** – mijenja vrijednost čvora
- **korijenB** – odgovara na pitanje je li stablo prazno
- **stvoriLijevoB** – dodaje čvoru lijevo dijete
- **stvoriDesnoB** – dodaje čvoru desno dijete
- **obrisiCvorB** – briše čvor
- **jednakiB** – provjerava jesu li dva binarna stabla jednaka
- **kopirajStabloB** – obavlja *deep* kopiju binarnog stabla
- **inicijalizirajB** – inicijalizira binarno stablo

Napomenimo da funkcije, koje bi trebale dohvaćati određeni čvor, dohvaćaju zapravo indeks toga čvora. To je iz razloga što se radi o implementaciji binarnog stabla uz pomoć polja. Sve funkcije su strukturirane tako da rade s indeskima, a ne s čvorovima, upravo iz tog razloga. Ukoliko znamo indeks nekog čvora u polju binarnog stabla, tada s tim čvorom možemo učiniti što god želimo.

Temeljna struktura koja opisuje svaki čvor je sljedeća:

```
struct bstablo {
        int iskoristen;
        int v;
};
```

Čvor se sastoji od dvije varijable. Varijabla *v* predstavlja vrijednost čvora, dok varijabla *iskoristen* daje informaciju o tome je li čvor iskorišten ili ne. Čvor je iskorišten ako je inicijaliziran.

Funkcija koja dohvaća **indeks čvora roditelja** u polju binarnog stabla je prikazana u kodu ispod:

```
/**
 * Vraca indeks polja roditelja čvora
 * @param  pozicija pozicija u polju od čvora kojem želimo
 * naći indeks roditelja
 * @param  T        binarno stablo
 * @return          indeks polja roditelja stabla
 */
int roditeljB(int pozicija, bstablo T[]) {
        if ((int)(pozicija/2)) return pozicija/2;
        else return NEPOSTOJECI_CVOR;
}
```

U implementaciji binarnog stabla pomoću polja možemo vrlo jednostavno odrediti indekse povezanih čvorova. Naime, ako je indeks čvora djeteta *i*, tada je indeks čvora roditelja cjelobrojna vrijednost količnika $\frac{i}{2}$. Karakteristika implementacije binarnog stabla uz pomoć polja je ta što korijen ima indeks 1, a ne 0. Ukoliko se pokuša pronaći roditelj od korijena stabla, funkcija će vratiti vrijednost *NEPOSTOJECI_CVOR* koja je jednaka **nuli**.

Lijevo dijete čvora s indeksom *i* ima indeks *2i*. **Indeks čvora lijevog djeteta** u polju binarnog stabla se dohvaća sljedećom funkcijom:

```
/**
 * Vraća indeks lijevog djeteta čvora
 * @param  pozicijaRoditelja indeks polja roditelja čvora čije
 * lijevo dijete želimo naći
 * @param  T                 binarno stablo
 * @return                   indeks lijevog djeteta čvora
 */
int lijevoDijeteB(int pozicijaRoditelja, bstablo T[]) {
        if (pozicijaRoditelja*2 >= MAX_BROJ_CVOROVA) {
                return NEPOSTOJECI_CVOR;
        } else if (T[pozicijaRoditelja*2].iskoristen) {
                return pozicijaRoditelja*2;
        } else {
                return NEPOSTOJECI_CVOR;
        }
}
```

Ako se pokuša dohvatiti indeks čvora lijevog djeteta koji bi prelazio granice alociranog stabla, tada se vraća već spomenuta konstanta NEPOSTOJECI_CVOR. To znači da čvor s tim indeksom nema lijevog djeteta.

Slična je implementacija za dohvaćanje indeksa čvora desnog djeteta. Desno dijete čvora s indeksom *i* ima indeks $2i + 1$. **Indeks čvora desnog djeteta** u polju binarnog stabla se dohvaća sljedećom funkcijom:

```
/**
 * Vraća indeks desnog djeteta čvora
 * @param  pozicijaRoditelja indeks polja roditelja čvora čije
 * desno dijete želimo naći
 * @param  T                 binarno stablo
 * @return                   indeks desnog djeteta čvora
 */
int desnoDijeteB(int pozicijaRoditelja, bstablo T[]) {
        if (pozicijaRoditelja*2+1 >= MAX_BROJ_CVOROVA) {
                return NEPOSTOJECI_CVOR;
        } else if (T[pozicijaRoditelja*2+1].iskoristen) {
                return pozicijaRoditelja*2+1;
        } else {
                return NEPOSTOJECI_CVOR;
        }
}
```

Vrijednost čvora se dohvaća sljedećom funkcijom. Čvor može imati vrijednost samo ako je iskorišten.

```
/**
 * Vraća vrijednost čvora
 * @param  pozicija indeks čvora u polju
 * @param  T        binarno stablo
 * @return          vrijednost čvora
 */
int vrijednostB(int pozicija, bstablo T[]) {
        if (T[pozicija].iskoristen==1) return T[pozicija].v;
        return NEPOSTOJECI_CVOR;
}
```

Funkcija za promjenu vrijednosti čvora je prilično jednostavna, te ju ne treba objašnjavati:

```
/**
 * Mijenja vrijednost čvora binarnog stabla
 * @param v        nova vrijednost čvora
 * @param pozicija indeks čvora u polju
 * @param T        binarno stablo
 */
void promijeniVrijednostB(int v, int pozicija, bstablo T[]) {
        T[pozicija].v=v;
}
```

Funkcija *korijenB* vraća indeks korijena stabla i odgovara na pitanje **je li stablo prazno**:

```
/**
 * Daje odgovor na pitanje je li stablo prazno
```

```
 * @param  T binarno stablo
 * @return   Odgovor na pitanje je li stablo prazno
 */
int korijenB(bstablo T[]) {  //je li stablo prazno ?
        if (T[1].iskoristen) return 1;
        else return 0;
}
```

Lijevo dijete se stvara pomoću funkcije navedene ispod. Funkcija će inicijalizirati novi čvor samo u slučaju ako čvor na toj poziciji već nije iskorišten (u uporabi). U suprotnom će izjaviti grješku sadržaja: "Taj cvor vec ima lijevo dijete". Ta će se grješka izjaviti kroz *ereport* funkciju, koja je dio PostqreSQL API-ja. Tekst koji proslijedimo toj funkciji će se prikazati kada se unutar PostgreSQL-a dogodi pogrješka. Okidanje pogrješke će također prouzrokovati i terminiranje funkcije.

```
/**
 * Stvara lijevo dijete čvora
 * @param v                  vrijednost novostvorenog čvora
 * @param pozicijaRoditelja indeks polja roditelja stabla čije
 * lijevo dijete želimo stvoriti
 * @param T                  binarno stablo
 */
void stvoriLijevoB(int v, int pozicijaRoditelja, bstablo T[]) {
        int lijevoDijete = pozicijaRoditelja*2;

        if (T[lijevoDijete].iskoristen) {
                ereport(ERROR,
                        (errcode(ERRCODE_RAISE_EXCEPTION),
                        errmsg("Taj cvor vec ima lijevo dijete!\n"))
                );
        } else {
                T[lijevoDijete].v=v;
                T[lijevoDijete].iskoristen=1;
        }
}
```

Desno dijete se stvara slično kao i lijevo:

```
/**
 * Stvara desno dijete čvora
 * @param v                  vrijednost novostvorenog čvora
 * @param pozicijaRoditelja indeks polja roditelja stabla čije
 * desno dijete želimo stvoriti
 * @param T                  binarno stablo
 */
void stvoriDesnoB(int v, int pozicijaRoditelja, bstablo T[]) {
        int desnoDijete = pozicijaRoditelja*2+1;
```

```
        if (T[desnoDijete].iskoristen) {
                ereport(ERROR,
                        (errcode(ERRCODE_RAISE_EXCEPTION),
                        errmsg("Taj cvor vec ima desno dijete!\n"))
                );
        } else {
                T[desnoDijete].v=v;
                T[desnoDijete].iskoristen=1;
        }
}
```

Funkcija prikazana ispod, koja **briše čvor**, je rekurzivna iz razloga što se brisanjem pojedinog čvora moraju obrisati i sva njegova djeca s njegovog lijevog i desnog podstabla. Rekurzija se zaustavlja kada se dođe do čvora koji nije iskorišten, budući da takve nema potrebe brisati. Brisanje čvora se svodi na postavljanje varijable *iskoristen* na nulu.

```
/**
 * Briše čvor i sve njegove potomke
 * @param pozicija indeks polja čvora koji želimo obrisati
 * iz binarnog stabla
 * @param T        binarno stablo
 */
void obrisiCvorB(int pozicija, bstablo T[]) {
        if (pozicija < MAX_BROJ_CVOROVA && T[pozicija].iskoristen == 1)
        {
                T[pozicija].iskoristen=0;
                obrisiCvorB(lijevoDijeteB(pozicija, T), T);
                obrisiCvorB(desnoDijeteB(pozicija, T), T);
        }
}
```

Sljedeća funkcija provjerava jesu li **dva stabla jednaka**. Provjera se obavlja tako da se uspoređuje svaki pojedini čvor kod oba stabla. Kada se jednom naiđe na određenu nejednakost, vraća se informacija da čvorovi nisu jednaki.

```
/**
 * Provjerava jesu li dva binarna stabla jednaka
 * @param T1  prvo binarno stablo
 * @param T2  drugo binarno stablo
 * @return    ukoliko su binarna stabla jednaka, vraća 1.
 * U suprotnom vraća 0
 */
int jednakiB(bstablo T1[], bstablo T2[]) {
        int i, jednaki = 1;

        for (i = INDEKS_KORIJENA_STABLA; i < MAX_BROJ_CVOROVA; i++) {
                if (T1[i].iskoristen != T2[i].iskoristen ||
```

```
                        T1[i].v != T2[i].v) {

                        jednaki = 0;
                        break;
                }
        }

        return jednaki;
}
```

Ponekad je potrebno izraditi novu kopiju binarnog stabla jednaku postojećoj. Sljedeća funkcija izrađuje *deep* kopiju binarnog stabla, što znači da ova funkcija kopira sve čvorove postojećeg binarnog stabla na nove memorijske lokacije. Ova funkcija je korisna kada je potrebno napraviti privremene promjene na binarnom stablu (brisanje, dodavanje novih čvorova i sl.)

```
/**
 * Obavlja duboko kopiranje stabla i vraća kopiju
 * @param  T Binarno stablo koje treba kopirati
 * @return   Duboka kopija binarnog stabla
 */
bstablo *kopirajStabloB(bstablo *T) {
  bstablo *T2 = palloc(MAX_BROJ_CVOROVA * sizeof(struct bstablo));
  memcpy(T2, T, MAX_BROJ_CVOROVA * sizeof(struct bstablo));
  return T2;
}
```

Posljednja funkcija vezana uz tip podataka binarno stablo je funkcija za inicijalizaciju binarnog stabla, prikazana sljedećim kodom:

```
/**
 * Alocira binarno stablo u memoriji i inicijalizira njegov prvi čvor
 * @param  v  vrijednost korijena stabla
 * @return    Pokazivač na binarno stablo
 */
bstablo *inicijalizirajB(int v) {
        int i;

        bstablo *T = palloc(MAX_BROJ_CVOROVA * sizeof(struct bstablo));

        // postavljamo sve čvorove stabla na neiskorištene
        for (i = 0; i < MAX_BROJ_CVOROVA; i++) {
                T[i].iskoristen=0;
        }

        // inicijaliziramo korijen stabla
        T[INDEKS_KORIJENA_STABLA].v=v;
        T[INDEKS_KORIJENA_STABLA].iskoristen=1;
```

```
        return T;
}
```

Inicijalizacija se obavlja tako da se najprije alocira čitavo stablo sa unaprijed zadanim maksimalnim brojem čvorova (koji je u ovom slučaju 512). Nakon toga se svi čvorovi postavljaju na neiskorištene, a inicijalizira se samo korijen stabla. Može se zamijetiti da se za alokaciju binarnog stabla ne koristi ključna riječ *malloc*, nego **palloc** (PostgreSQL, 2014b). *palloc* je PostgreSQL-ov wrapper za funkciju *malloc*.

Poglavlje 5

Konverzija tekstualnog oblika binarnog stabla

Već je spomenuto da funkcija koja pretvara tekstualnu reprezentaciju binarnog stabla u objekt binarnog stabla koristi funkciju *pretvoriUStablo(char *znakovniNiz)*. Spomenuto je i da ta funkcija poziva niz drugih funkcija, koje provode operaciju pretvorbe. Sve te funkcije su smještene u datoteku *parsiranje.h*, čiji ćemo sadržaj sada opisati.

5.1 Privremeni čvorovi stabla

Već smo upoznati s činjenicom da postoje tekstualni i binarni oblik binarnog stabla. Tekstualni oblik je čitljiv ljudskom oku, dok je binarni oblik čitljiv računalu. Kako bi stoga učinili binarno stablo pogodnim za različite operacije, moramo ga iz tekstualnog oblika pretvoriti u binarni. Međutim, ta pretvorba se sastoji od niza operacija. Jedna od operacija predstavlja i pretvorbu tekstualnih čvorova stabla u privremene čvorove stabla. Privremeni čvor je definiran sljedećom strukturom:

```
typedef struct tmp_cvor {
  int id, v, lijevo, desno;
} tmp_cvor;
```

Privremeni čvor je dobiven direktnom konverzijom iz tekstualnog čvora, što možemo primijetiti promatrajući parametre kojima je definiran:

- **id** – ID čvora
- **v** – vrijednost čvora
- **lijevo** – ID lijevog djeteta čvora
- **desno** – ID desnog djeteta čvora

Njegova je uloga da pohrani sve vrijednosti iz tekstualnih čvorova u polje kako se oni ne bi morali uvijek iznova parsirati, odnosno kako bi se olakšao sam proces pretvorbe.

5.2 Pretvorba u binarni oblik stabla

Funkcionalnost koja ostvaruje pretvorbu u binarni oblik stabla je definirana funkcijom *pretvoriUStablo*. Ta funkcija poziva sve ostale potrebne funkcije, a prikazana je u priloženom kodu:

```
/**
 * Ova funkcija pretvara tekstualnu reprezentaciju stabla
 *   u binarno stablo
 * @param  stablo_tekst tekstualna reprezentacija stabla
 * @return              binarno stablo u memoriji
 */
bstablo *pretvoriUStablo(char *stablo_tekst) {
        int idKorijenaStabla = 0;

        int brojCvorova = brojCvorovaUStablu(stablo_tekst);
        tmp_cvor *privremeniCvorovi =
          popuniPrivremeneCvorove(stablo_tekst, brojCvorova);
        bstablo *stablo =
          pronadjiKorijenStabla(privremeniCvorovi, brojCvorova,
                &idKorijenaStabla); // inicijalizira stablo
        popuniStablo(stablo, INDEKS_KORIJENA_STABLA,
                idKorijenaStabla, privremeniCvorovi, brojCvorova);
        provjeriNeiskoristeneCvorove(privremeniCvorovi, brojCvorova);

        pfree(privremeniCvorovi);
        return stablo;
}
```

Najprije se pobrojavaju svi čvorovi binarnog stabla. Nakon toga slijedi alokacija i inicijalizacija privremenih čvorova, čija je uloga opisana u prethodnom poglavlju. Potom se inicijalizira stablo i pronalazi korijen stabla (u tekstualnoj reprezentaciji binarnog stabla korijen ne mora uvijek biti prvi čvor). Inicijalizirano se stablo popunjava, a na kraju se provjerava je li ostao koji čvor viška, koji se nije uvrstio u objekt binarnog stabla. Postojanje takvog čvora je znak da je tekstualni oblik stabla krivo definiran od strane korisnika.

5.2.1 Prebrojavanje čvorova

Ukupni broj čvorova binarnog stabla je jednostavno pronaći ako je stablo u tekstualnom obliku. Poznato je da su čvorovi u tekstualnom obliku međusobno odvojeni znakom **;**, stoga je potrebno pronaći broj pojavljivanja znaka **;** te ga uvećati za jedan. Funkcija koja izvršava taj posao je prikazana ispod:

```
/**
 * Vraća broj čvorova u tekstualnoj reprezentaciji stabla
 * @param  stablo Tekstualna reprezentacija stabla
 * @return        Broj čvorova
 */
int brojCvorovaUStablu(char *stablo) {
```

```
        int brojCvorova = 1, i;
        for(i = 0; i < strlen(stablo); i++) {
                if (stablo[i] == ';') {
                        brojCvorova++;
                }
        }
        return brojCvorova;
}
```

5.2.2 Stvaranje privremenih čvorova

Budući da bi stalno kretanje po tekstualnom obliku stabla bilo naporno i za programera koji implementira, a i za računalo koje bi stalno moralo izvršavati parsiranje, tekstualni čvorovi koje unese korisnik se najprije parsiraju i pretvaraju u privremene čvorove. Kao što je već spomenuto, privremeni čvorovi nisu ništa drugo, nego tekstualni čvorovi pretvoreni u polje. Razlika između privremenog (kao i tekstualnog) čvora i binarnog čvora (koji je pogodan računalu) je u strukturi. Naime, privremeni čvor osim vrijednosti sadrži ID samog čvora i ID-eve djeteta toga čvora. Binarnom čvoru ID-evi nisu potrebni jer je koncept binarnog stabla uz pomoć polja takav da su jedini potrebni podaci za svaki čvor vrijednost toga čvora te informacija o tome je li taj čvor iskorišten ili ne. Programski kod kojim je definirana funkcija za pretvorbu tekstualnih čvorova se nalazi ispod. Funkcija je prilično jednostavna, ali dugačka zbog svih provjera koje je potrebno obaviti nad svakim od čvorova.

```
/**
 * Stvara privremene čvorove iz tekstualne reprezentacije stabla.
 * Takvi privremeni čvorovi služe kako bi olakšali
 *  stvaranje stvarnog stabla
 * @param  stablo      Tekstualna reprezentacija stabla
 * @param  brojCvorova Broj čvorova u tekstualnoj reprezentaciji stabla
 * @return             Polje privremenih čvorova
 */
tmp_cvor* popuniPrivremeneCvorove(char *stablo, int brojCvorova) {
        int i;
        char *cvor;

        // najprije stvaramo polje privremenih čvorova
        tmp_cvor *privremeniCvorovi =
          (tmp_cvor *) palloc(brojCvorova * sizeof(struct tmp_cvor));

        // potom to polje popunjavamo s podacima iz
        // tekstualne reprezentacije stabla
        cvor = strtok(stablo, ";");
        for (i = 0; i < brojCvorova; i++) {
                int id, v, l, d;
                if (sscanf(cvor, "(%d, %d, %d, %d)",
                  &id, &v, &l, &d) != 4) {
```

```
            ereport(ERROR,
              (errcode(ERRCODE_RAISE_EXCEPTION),
                errmsg("Pogresna sintaksa cvora \"%s\"",cvor))
            );
          }


          // popunjavanje privremenog čvora
          // (uz javljanje pogrješki ukoliko dođe do njih)
          if (id >= 0) {
            privremeniCvorovi[i].id = id;
          } else {
            ereport(ERROR,
              (errcode(ERRCODE_RAISE_EXCEPTION),
                errmsg("ID cvora \"%s\" mora biti "
                  "nenegativan broj\n", cvor))
              );
          }

          privremeniCvorovi[i].v = v;

          if (l >= -1) {
            privremeniCvorovi[i].lijevo = l;
          } else {
            ereport(ERROR,
              (errcode(ERRCODE_RAISE_EXCEPTION),
                errmsg("Lijevo dijete cvora \"%s\" mora biti"
                  "nenegativan broj ili -1\n", cvor))
              );
          }

          if (d >= -1) {
            privremeniCvorovi[i].desno = d;
          } else {
            ereport(ERROR,
              (errcode(ERRCODE_RAISE_EXCEPTION),
                errmsg("Desno dijete cvora \"%s\" mora biti"
                  "nenegativan broj ili -1\n", cvor))
              );
          }


          cvor = strtok(NULL, ";");
        }
```

```
        return privremeniCvorovi;
}
```

Budući da je već poznato koliko čvorova je korisnik unio, može se petljom proći kroz sve tekstualne čvorove i pretvoriti ih u privremene. Budući da svi tekstualni čvorovi moraju imati jednaku strukturu i budući da svi moraju biti međusobno odvojeni znakom **;**, parsiranje je prilično jednostavno. Funkcijom *strtok* dohvaćamo čvor po čvor i potom ga parsiramo. Ukoliko čvor ne odgovara standardnoj strukturi, dolazi do pogrješke "Pogrešna sintaksa čvora". Kao što je već navedeno, ID svakog čvora (uključujući i čvorove djece) mora biti veći od nule ili -1. U suprotnom dolazi do pogrješke. Radi jednostavnosti, vrijednost čvora također mora biti cjelobrojna.

Tako parsirani čvorovi se pohranjuju u polje *privremeniCvorovi*. Kada se parsiranje završi, potrebno je privremene čvorove pretvoriti u binarno stablo koje će biti servirano PostgreSQL sustavu.

5.2.3 Pronalaženje korijena stabla

Iako se može činiti trivijalnim ili nebitnim, ovaj korak je vrlo važan i ne smije se preskočiti. Za očekivati je da će korijen binarnog stabla biti prvi čvor koji se navede. Međutim, to ne mora biti tako. U ovoj implementaciji je dopušteno čvorove pisati bilo kojim redoslijedom. Stoga isto mora biti dopušteno i za korijen stabla. Korijen stabla je najteže pronaći, dok je su svi ostali čvorovi lakši za pronaći, budući da njihovi roditelji pokazuju na njih.

Kako naći korijen stabla? Možemo se poslužiti činjenicom da na korijen stabla nema roditelja, odnosno da na njega ne pokazuje niti jedan drugi čvor. To je zapravo jedina činjenica kojom se možemo poslužiti, zbog čega je potrebno svaki element binarnog stabla usporediti sa svakim drugi elementom. Zbog toga je složenost ovoga algoritma nešto veća, i iznosi *O(n)*. Algoritam pronalaska korijena stabla je prikazan u nastavku.

```
/**
 * Iz polja struktura privremenih čvorova pronalazi korijen stabla.
 * Korijen stabla je onaj čvor koji nema roditelja, odnosno
 * kojega niti jedan drugi čvor nema za lijevo (ili desno) dijete
 * @param  privremeniCvorovi polje struktura privremenih čvorova
 * @param  brojCvorova       broj privremenih čvorova u strukturi
 *   privremenih čvorova (broj čvorova koje je korisnik unio)
 * @param  idKorijenaStabla  u ovu varijablu će se upisati
 *   ID pronađenog korijena stabla
 * @return                   memorijska reprezentacija binarnog stabla
 */
bstablo *pronadjiKorijenStabla(tmp_cvor *privremeniCvorovi,
  int brojCvorova, int *idKorijenaStabla) {
    int i,j, korijenPronadjen = 1, vrijednostKorijena;
    bstablo *stablo = NULL;

    for (i = 0; i < brojCvorova; i++) {
      for (j = 0; j < brojCvorova; j++) {
        if (privremeniCvorovi[j].lijevo == privremeniCvorovi[i].id ||
          privremeniCvorovi[j].desno == privremeniCvorovi[i].id) {
```

```
            korijenPronadjen = 0;
            break;
        }
      }

      // ako je korijen stabla pronađen,
      // stvaramo stvarno stablo i vraćamo ga
      // također zapisujemo vrijednost u varijablu idKorijenaStabla
      if (korijenPronadjen == 1) {
        vrijednostKorijena = privremeniCvorovi[i].v;
        stablo = inicijalizirajB(vrijednostKorijena);
        *idKorijenaStabla = privremeniCvorovi[i].id;
        return stablo;
      }

      // u drugom slučaju, postavljamo da je korijen pronađen
      // i promatramo hoće li se u sljedećoj iteraciji
      // ugniježđene for petlje to promijeniti
      korijenPronadjen = 1;
    }

    free(privremeniCvorovi);
    // ako korijen stabla nije pronađen
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Uneseni oblik stabla nema korijena!"))
    );

    return NULL;  // da se izbjegne warning pri kompiliranju
}
```

Korijen se traži nad privremenim čvorovima stabla, koji su prikazani uz pomoć polja. Kada na red za provjeru dođe čvor *n*, provjerava se postoji li bilo koji drugi čvor u polju koji bi za lijevo ili desno dijete upravo imao čvor *n*. Kada se pronađe čvor na koji ne pokazuje niti jedan drugi, za njega se pretpostavi da je korijen. Iz privremenog stabla se uzima vrijednost toga čvora te se inicijalizira novo binarno stablo.

Ukoliko se korijen binarnog stabla ne pronađe, to znači da je stablo definirano rekurzivno, zbog čega dolazi do pogrješke.

5.2.4 Popunjavanje čvorova binarnog stabla

U ovom trenutku izvođenja programa je binarno stablo već inicijalizirano s početnim čvorom. Sada ga je potrebno popuniti. Popunjavanje binarnog stabla se mora obavljati rekurzivno.

```
/**
 * Popunjava stablo s ostalim čvorovima i postupno dealocira
```

```
 *   privremene čvorove
 * @param  stablo              Pokazivač na binarno stablo
 * @param  indeksPocetnogCvora Indeks čvora od kojeg kreće
 *   popunjavanje stabla
 * @param  idPocetnogCvora     ID čvora od kojeg kreće
 *   popunjavanje stabla
 * @param  privremeniCvorovi   Polje privremenih čvorova
 *   iz kojeg će se dohvaćati čvorovi
 * @param  brojCvorova         Broj čvorova u polju
 *   privremenih čvorova
 */
void popuniStablo(bstablo *stablo, int indeksPocetnogCvora,
  int idPocetnogCvora, tmp_cvor *privremeniCvorovi, int brojCvorova) {
    int i;
    int idLijevog = 0, idDesnog = 0, indeksRoditelja = 0;
    int vrijednostLijevog = 0, vrijednostDesnog = 0;

    // najprije pronalazimo id lijevog i desnog djeteta početnog čvora
    for (i = 0; i < brojCvorova; i++) {
      if (privremeniCvorovi[i].id == idPocetnogCvora) {
        indeksRoditelja = i;
        idLijevog = privremeniCvorovi[i].lijevo;
        idDesnog = privremeniCvorovi[i].desno;
        break;
      }
    }

    // potom pronalazimo vrijednosti lijevog
    // i desnog djeteta početnog čvora
    for (i = 0; i < brojCvorova; i++) {
      if (idLijevog != -1 && privremeniCvorovi[i].id == idLijevog)
        vrijednostLijevog = privremeniCvorovi[i].v;
      if (idDesnog != -1 && privremeniCvorovi[i].id == idDesnog)
        vrijednostDesnog = privremeniCvorovi[i].v;
    }

    // zatim stvaramo lijevi i desni čvor
    if (idLijevog != -1)
      stvoriLijevoB(vrijednostLijevog, indeksPocetnogCvora, stablo);
    if (idDesnog != -1)
      stvoriDesnoB(vrijednostDesnog, indeksPocetnogCvora, stablo);


    // postavljamo ID trenutnog čvora u polju privremenih čvorova na -1
    // to činimo kako bi kasnije mogli napraviti provjeru
```

```
    // naime, ukoliko postoje privremeni čvorovi koji nisu iskorišteni,
    // tada ćemo okinuti pogrješku
    privremeniCvorovi[indeksRoditelja].id = CVOR_ISKORISTEN;

    // na kraju nastavljamo dalje puniti stablo za lijevi i desni čvor
    if (idLijevog != -1)
      popuniStablo(stablo, lijevoDijeteB(indeksPocetnogCvora, stablo),
        idLijevog, privremeniCvorovi, brojCvorova);
    if (idDesnog != -1)
      popuniStablo(stablo, desnoDijeteB(indeksPocetnogCvora, stablo),
        idDesnog, privremeniCvorovi, brojCvorova);
}
```

U svakoj iteraciji ove rekurzivne funkcije se za dani ID čvora pronalaze lijevi i desni čvorovi. Budući da se pretraga odvija nad privremenim stablom, jasno je da će se dobiti informacije samo o ID-u lijevog, odnosno desnog stabla. Međutim, da bi dodali novi čvor u pravo binarno stablo, potrebne su vrijednosti tih čvorova, a ne njihovi ID-evi. Stoga se još jednom u privremenom stablu moraju potražiti vrijednosti lijevog i desnog djeteta stabla, s obzirom na ID-eve tih čvorova. Ukoliko su ID lijevog djeteta ili ID desnog djeteta različiti od -1 (odnosno, ako su definirani), stvara se lijevo i/ili desno dijete binarnog stabla.

Posljednji korak je obilježavaje onih čvorova privremenog stabla koji su iskorišteni, kako bi se naknadno moglo provjeriti je li sve u redu s definiranim stablom. Na posljetku funkcije se rekurzija nastavlja na lijevu i desnu stranu stabla ukoliko su čvorovi na tim stranama definirani.

5.2.5 Provjera neiskorištenih čvorova

U ovom, posljednjem, ali ne i najmanje bitnom, koraku je binarno stablo već stvoreno. Ako je do sada sve prošlo u redu, to ne mora značiti da zbilja i jest sve u redu. Može se dogoditi da primjerice nisu svi čvorovi iz privremenog stabla pretvoreni u binarno stablo. Ukoliko se dogodi takva situacija, to znači da stablo koje je korisnik unio nije dobro definirano. Ova funkcija će dati odgovor na to pitanje i prekinuti bilo kakav daljnji unos ili obradu binarnog stabla ukoliko u unosu postoji grješka.

```
void provjeriNeiskoristeneCvorove(tmp_cvor *privremeniCvorovi,
  int brojCvorova) {
    int i;
    for (i = 0; i < brojCvorova; i++) {
      if (privremeniCvorovi[i].id != CVOR_ISKORISTEN) {
        pfree(privremeniCvorovi);

        ereport(ERROR,
          (errcode(ERRCODE_RAISE_EXCEPTION),
            errmsg("Neispravan unos stabla\n"))
        );

        return;
      }
```

```
    }
}
```

Koncept provjere je vrlo jednostavan. Funkcija koja je pretvarala pomoćno stablo u binarno, bilježila je svaki iskorišteni čvor. Zadatak ove funkcije je da provjeri samo jesu li svi čvorovi pomoćnog stabla iskorišteni. Ako nisu, dolazi do pogrješke. Međutim, ako jesu, mogu se nastaviti daljnje operacije sa stablom.

Poglavlje 6

Funkcije binarnog stabla

Od binarnog stabla ne bi imali puno koristi ako bi ga mogli samo unositi u bazu podataka i ispisivati na zaslon. Stablo mora omogućiti pretragu, različite vrste ispisa, dodavanje čvorova i podstabala, orezivanje te brojne druge mogućnosti. Izvorni kod svih tih funkcija se nalazi unutar datoteke *stablo.c* .

6.1 Broj čvorova stabla

Vjerojatno najjednostavnija funkcija je ona koja vraća ukupni broj čvorova binarnog stabla, čiji je kod prikazan ispod.

```
Datum stablo_broj_cvorova(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  int brojCvorova = 0, i;

  for (i = INDEKS_KORIJENA_STABLA; i < MAX_BROJ_CVOROVA; i++) {
    if (stablo[i].iskoristen) {
      brojCvorova++;
    }
  }

  PG_RETURN_INT32(brojCvorova);
}
```

Funkcija kao argument uzima binarno stablo i broji sve njegove čvorove koji su iskorišteni.

6.2 Broj čvorova podstabla

Ova funkcija je složenija od prethodne funkcije, jer se kroz stablo više nije moguće kretati linearno kao kroz polje, nego se potrebno kretati rekurzivno. No prije nego li se krenu prebrojavati čvorovi, mora se označiti početni čvor od kojeg se počinje brojati. On se označava pomoću niza znakova 'L' i 'D'. Ukoliko je znakovni niz primjerice *LDLL*, tada će se iz korijena stabla ući u lijevu granu, pa u desnu. Nakon toga će se ući u lijevu granu i iz tog čvora će se ponovno ući u lijevu granu. Tek će se iz te pozicije početi prebrojavati preostali čvorovi.

Ova funkcija dakle prebrojava sve čvorove od zadanog čvora naniže, uključujući i taj zadani čvor. Ukoliko je putanja do zadanog čvora prazan *string*, tada je rezultat ove funkcije jednak ukupnom broju svih čvorova u stablu.

```
Datum stablo_broj_cvorova_podstabla(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *putanja = (char *) PG_GETARG_CSTRING(1);
  int indeksCvora, brojCvorovaLijevo, brojCvorovaDesno;

  indeksCvora = pronadjiIndeksCvora(stablo, putanja);

  if (indeksCvora == NEPOSTOJECI_CVOR) {
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Taj cvor jos nije alociran!\n"))
    );
  }

  brojCvorovaLijevo =
    brojCvorovaPodstabla(lijevoDijeteB(indeksCvora, stablo), stablo);
  brojCvorovaDesno =
    brojCvorovaPodstabla(desnoDijeteB(indeksCvora, stablo), stablo);

  return brojCvorovaLijevo + brojCvorovaDesno + 1;
}
```

Može se uočiti da funkcija prima dva argumenta - binarno stablo i putanju (znakovni niz). Potom najprije pronalazi indeks čvora koji se nalazi na zadanoj putanji. Ukoliko je putanjom definiran čvor koji još ne postoji u stablu, dolazi do grješke. U suprotnom, broji se broj čvorova lijevo od zadanog čvora i desno od zadanog čvora. Budući da i odabrani čvor ulazi u zbroj, dobivenim se rezultatima dodaje 1.

Funkcija koja pronalazi indeks čvora je definirana ispod.

```
int pronadjiIndeksCvora(bstablo *stablo, char *putanja) {
  int indeksCvora = INDEKS_KORIJENA_STABLA;
  int i;

  for (i = 0; i < strlen(putanja); i++) {
    if (putanja[i] == 'L') {
      indeksCvora = lijevoDijeteB(indeksCvora, stablo);
    } else {
      indeksCvora = desnoDijeteB(indeksCvora, stablo);
    }
  }

  return indeksCvora;
}
```

Ova funkcija je prilično jednostavna. Ona troši znakovni niz putanje znak po znak. Za svaki znak iz znakovnog niza ona pronalazi lijevo, odnosno desno, dijete i njegov indeks pohranjuje u varijablu. Kada potroši sve znakove, vraća indeks posljednjeg čvora s putanje.

Funkcija koja broji čvorove podstabla prikazana je sljedećim kodom.

```
int brojCvorovaPodstabla(int indeksPocetnogCvora, bstablo *stablo) {
  int sumaCvorova = 0, indeksLijevogDjeteta, indeksDesnogDjeteta;

  if (indeksPocetnogCvora == NEPOSTOJECI_CVOR) {
    return 0;
  }

  indeksLijevogDjeteta = lijevoDijeteB(indeksPocetnogCvora, stablo);
  indeksDesnogDjeteta = desnoDijeteB(indeksPocetnogCvora, stablo);

  if (indeksLijevogDjeteta == NEPOSTOJECI_CVOR &&
    indeksDesnogDjeteta == NEPOSTOJECI_CVOR) {
      return 1;
  } else {
      sumaCvorova += brojCvorovaPodstabla(indeksLijevogDjeteta, stablo);
      sumaCvorova += brojCvorovaPodstabla(indeksDesnogDjeteta, stablo);
      return sumaCvorova + 1;
  }
}
```

Ova funkcija je klasični primjer rekurzivnog prebrojavanja elemenata. Na početku je definiran uvjet za izlazak iz rekurzije. Svaka iteracija rekurzije vraća broj čvorova lijevog podstabla i broj čvorova desnog podstabla, čijem je zbroju dodan 1 (trenutni čvor).

6.3 Provjera postojanja čvora

Ova jednostavna funkcija provjerava postoji je li čvor na zadanoj putanji inicijaliziran.

```
Datum stablo_postoji_cvor(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *putanja = (char *) PG_GETARG_CSTRING(1);
  int indeksCvora;

  indeksCvora = pronadjiIndeksCvora(stablo, putanja);

  if (indeksCvora == NEPOSTOJECI_CVOR) {
    PG_RETURN_BOOL(FALSE);
  } else {
    PG_RETURN_BOOL(TRUE);
  }
}
```

6.4 Vrijednost čvora

Postoje dvije funkcije koje se odnose na vrijednost čvora. Jedna od tih funkcija čita vrijednost čvora, a druga ga postavlja. Fukncija koja čita i vraća vrijednost čvora na zadanoj putanji je definirana ispod.

```
Datum stablo_vrijednost(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *putanja = (char *) PG_GETARG_CSTRING(1);
  int indeksCvora, v = 0;

  indeksCvora = pronadjiIndeksCvora(stablo, putanja);

  if (indeksCvora == NEPOSTOJECI_CVOR) {
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Taj cvor nije jos definiran!\n"))
    );
  } else {
    v = vrijednostB(indeksCvora, stablo);
  }

  PG_RETURN_INT32(v);
}
```

Ukoliko čvor na zadanoj putanji nije definiran, tada funkcija vraća grešku. U suprotnom vraća vrijednost čvora.

Funkcija koja mijenja vrijednost čvora na određenoj putanji obavlja sličan posao.

```
Datum stablo_promijeni_vrijednost(PG_FUNCTION_ARGS) {
  bstablo *s = (bstablo *) PG_GETARG_POINTER(0);
  char *putanja = (char *) PG_GETARG_CSTRING(1);
  int v = (int) PG_GETARG_INT32(2);
  bstablo *stablo = kopirajStabloB(s);
  int indeksCvora;

  indeksCvora = pronadjiIndeksCvora(stablo, putanja);

  if (indeksCvora == NEPOSTOJECI_CVOR) {
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Taj cvor nije jos definiran!\n"))
    );
  } else {
    promijeniVrijednostB(v, indeksCvora, stablo);
  }

  PG_RETURN_POINTER(stablo);
```

```
}
```

Razlika je u tome što funkcija za promjenu vrijednosti čvora vraća čitavo stablo. Naime, sve funkcije koje mijenjaju stablo moraju vratiti pokazivač na stablo, kako bi, u slučaju da se nađu unutar klauzule UPDATE, mogle ažurirati postojeće stablo u bazi podataka. Tako je slučaj i s ovom funkcijom. Međutim, prije bilo kakve modifikacije binarnog stabla, potrebno je napraviti njegovu duboku kopiju (funkcija *kopirajStabloB*). U suprotnom bi ova funkcija ažurirala bazu podataka i u samoj SELECT klauzuli.

6.5 Dodavanje djeteta čvoru

Kada se nekom čvoru dodaje dijete, mora se specificirati nekoliko stvari - samo binarno stablo, putanja do roditelja, želi li se dodati lijevo ili desno stablo te vrijednost čvora koji se želi dodati. Pojednostavljeno, potrebno je znati lokaciju na koju se želi dodati čvor te vrijednost za novi čvor.

U programskom kodu navedenom ispod se putanja do novog čvora specificira direktno, bez specificiranja putanje do roditelja. Dakle, ako je roditelj na poziciji *DLD* (u odnosu na korijen stabla), tada čvoru kojeg želimo dodati možemo specificirati dvije putanje: *DLDL* ili *DLDD*, u ovisnosti o tome želimo li da to bude lijevo ili desno dijete čvora na poziciji *DLD*.

```
Datum stablo_dodaj_cvor(PG_FUNCTION_ARGS) {
  bstablo *s = (bstablo *) PG_GETARG_POINTER(0);
  char *putanjaDjeteta = (char *) PG_GETARG_CSTRING(1);
  int vrijednostDjeteta = (int) PG_GETARG_INT32(2);
  int indeksCvoraRoditelja;

  bstablo *stablo = kopirajStabloB(s);

  char *putanjaRoditelja = (char *) palloc(strlen(putanjaDjeteta));
  char pozicijaDjeteta;

  // u putanju roditelja kopiramo putanju djeteta bez posljednjeg znaka
  strncpy(putanjaRoditelja, putanjaDjeteta, strlen(putanjaDjeteta) - 1);
  // postavljamo null-character na kraj putanje roditelja
  putanjaRoditelja[strlen(putanjaDjeteta) - 1] = '\0';
  // u poziciju djeteta stavljamo vrijednost
  // posljednjeg znaka putanje djeteta
  // pozicija djeteta je relativna s obzirom na putanju roditelja
  pozicijaDjeteta = putanjaDjeteta[strlen(putanjaDjeteta) - 1];

  // pronalazimo najprije indeks čvora roditelja
  indeksCvoraRoditelja = pronadjiIndeksCvora(stablo, putanjaRoditelja);

  pfree(putanjaRoditelja);

  // ako roditelj ne postoji, izbacujemo grešku
  if (indeksCvoraRoditelja == NEPOSTOJECI_CVOR) {
```

```
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Cvor roditelja jos nije definiran!\n"))
    );
  }

  // ako roditelj postoji, provjeravamo je li zauzeto njegovo
  // lijevo ili desno dijete, ovisno o tome na koju stranu
  // želimo alocirati novo dijete
  if (pozicijaDjeteta == 'L') {

    if (lijevoDijeteB(indeksCvoraRoditelja, stablo) ==
      NEPOSTOJECI_CVOR) {
        stvoriLijevoB(vrijednostDjeteta, indeksCvoraRoditelja, stablo);
    } else {
        ereport(ERROR,
          (errcode(ERRCODE_RAISE_EXCEPTION),
            errmsg("Lijevo dijete cvora roditelja je vec zauzeto!\n"))
        );
    }

  } else {

    if (desnoDijeteB(indeksCvoraRoditelja, stablo) ==
      NEPOSTOJECI_CVOR) {
        stvoriDesnoB(vrijednostDjeteta, indeksCvoraRoditelja, stablo);
    } else {
        ereport(ERROR,
          (errcode(ERRCODE_RAISE_EXCEPTION),
            errmsg("Desno dijete cvora roditelja je vec zauzeto!\n"))
        );
    }
  }

  PG_RETURN_POINTER(stablo);
}
```

Kao što se može vidjeti, putanja do novog djeteta čvora se najprije rastavlja na dvije putanje - apsolutnu putanju do roditelja i putanju do djeteta u odnosu na roditelja (znak L ili D). Potom se pokušava pronaći indeks roditelja i provjerava se postoji li njegovo lijevo, odnosno desno, dijete (u ovisnosti o tome koje dijete se želi dodati). Ukoliko na toj poziciji već postoji dijete, okida se greška. Međutim, ukoliko sve prođe u redu, dodaje se novo dijete.

6.6 Orezivanje stabla

Orezivanje stabla je brži način uklanjanja čvorova od pojedinačnog brisanja. Potrebno je zadati granu koja će se orezati. Primjerice, ako se za putanju do djeteta zada niz 'D', tada će se obrisati čitava desna strana stabla, uključujući i desno dijete korijena stabla. Međutim, valja napomenuti da **korijen stabla ne može biti obrisan!** Ukoliko se stoga ne specificira grana, obrisat će se svi potomci korijena stabla, a korijen će ostati netaknut. Ukoliko se želi obrisati čitavo stablo, umijesto nekog njegovog dijela, potrebno je koristiti *DELETE* SQL naredbu.

```
Datum stablo_orezi(PG_FUNCTION_ARGS) {
  bstablo *s = (bstablo *) PG_GETARG_POINTER(0);
  char *putanja = (char *) PG_GETARG_CSTRING(1);
  bstablo *stablo = kopirajStabloB(s);

  int indeksCvora = pronadjiIndeksCvora(stablo, putanja);

  if (indeksCvora == NEPOSTOJECI_CVOR) {
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Stablo na toj putanji ne postoji!\n"))
    );
  }

  // ako je čvor koji se želi obrisati korijen stabla,
  // tada će se obrisati lijeva i desna strana korijena
  // a u suprotnom će se obrisati navedeni čvor i sva njegova djeca
  if (indeksCvora == INDEKS_KORIJENA_STABLA) {
    obrisiCvorB(lijevoDijeteB(indeksCvora, stablo), stablo);
    obrisiCvorB(desnoDijeteB(indeksCvora, stablo), stablo);
  } else {
    obrisiCvorB(indeksCvora, stablo);
  }

  PG_RETURN_POINTER(stablo);
}
```

6.7 Ispis podstabla

Iako funkciju za ispis podstabla možemo koristiti za prikaz podstabla na zaslonu, uporaba ove funkcije može biti i šira. Ona se može primjerice koristiti za umetanje novih stabala u bazu, ažuriranje postojećih i razne slične operacije. Pogledajmo najprije njezinu implementaciju.

```
Datum stablo_podstablo(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *putanja = (char *) PG_GETARG_CSTRING(1);
```

```
  // najprije pronalazimo indeks čvora čije podstablo želimo pronaći
  int indeksCvora = pronadjiIndeksCvora(stablo, putanja);
  bstablo *podstablo;

  if (indeksCvora == NEPOSTOJECI_CVOR) {
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
        errmsg("Stablo na toj putanji ne postoji!\n"))
    );
  }

  // potom najprije alociramo podstablo i inicijaliziramo
  // njegov korijen na vrijednosti čvora u originalnom stablu
  podstablo = inicijalizirajB(vrijednostB(indeksCvora, stablo));

  // potom koristimo pomoćnu rekurzivnu funkciju koja će nam pomoći
  // popuniti alocirano podstablo sa čvorovima originalnog stabla
  popuniStablo(podstablo, stablo, indeksCvora);

  PG_RETURN_POINTER(podstablo);
}
```

Kako ova funkcija vraća novo podstablo, potrebno je najprije alocirati memorijski prostor za to podstablo. Potom se pronalazi indeks čvora originalnog stabla čije podstablo želimo pronaći. I dalje je jednostavno - potrebno je samo prekopirati dio originalnog stabla u novo stablo koristeći rekurziju. Funkcija koja obavlja kopiranje je prikazana sljedećim kodom.

```
void popuni_stablo(bstablo *podstablo, bstablo *originalnoStablo,
  int indeksCvora, int indeksCvoraPodstabla) {
    int indeksLijevogDjeteta, indeksDesnogDjeteta;

    if (indeksCvora == NEPOSTOJECI_CVOR ||
      indeksCvoraPodstabla == NEPOSTOJECI_CVOR) {
        return;
    }

    podstablo[indeksCvoraPodstabla].v =
      originalnoStablo[indeksCvora].v;
    podstablo[indeksCvoraPodstabla].iskoristen = 1;

    indeksLijevogDjeteta =
      lijevoDijeteB(indeksCvora, originalnoStablo);
    indeksDesnogDjeteta =
      desnoDijeteB(indeksCvora, originalnoStablo);

    popuni_stablo(podstablo, originalnoStablo,
```

```
      indeksLijevogDjeteta, indeksCvoraPodstabla*2);
    popuni_stablo(podstablo, originalnoStablo,
      indeksDesnogDjeteta, indeksCvoraPodstabla*2+1);
}
```

Funkcija za kopiranje čvorova iz jednog stabla u drugo mora imati dvije različite varijable pokazivača, kako bi mogla istovremeno voditi računa o trenutnoj lokaciji u oba stabla.

6.8 Dodavanje podstabla

Za razliku od funkcije za dodavanje pojedinačnih čvorova stablu, funkcija za dodavanje podstabla je mnogo složenija. Međutim, budući da su već implementirane sve značajnije funkcije, ova ih funkcija samo mora moći iskoristiti. Programski kod je prikazan u nastavku.

```
Datum stablo_dodaj_podstablo(PG_FUNCTION_ARGS) {
  bstablo *s = (bstablo *) PG_GETARG_POINTER(0);
  char *putanjaPodstabla = (char *) PG_GETARG_CSTRING(1);
  bstablo *podstablo = pretvoriUStablo((char *) PG_GETARG_POINTER(2));
  bstablo *stablo = kopirajStabloB(s);
  int indeksCvoraRoditelja;

  char *putanjaRoditelja = (char *) palloc(strlen(putanjaPodstabla));
  char pozicijaPodstabla; //lijevo ili desno u odnosu na čvor roditelja

  // u putanju čvora roditelja kopiramo putanju djeteta
  // bez posljednjeg znaka
  strncpy(putanjaRoditelja, putanjaPodstabla,
    strlen(putanjaPodstabla) - 1);
  // postavljamo null-character na kraj putanje čvora roditelja
  putanjaRoditelja[strlen(putanjaPodstabla) - 1] = '\0';
  // u poziciju djeteta stavljamo
  // vrijednost posljednjeg znaka putanje djeteta
  // pozicija djeteta je relativna s obzirom na
  // putanju do čvora roditelja
  pozicijaPodstabla = putanjaPodstabla[strlen(putanjaPodstabla) - 1];

  // pronalazimo najprije indeks čvora roditelja
  indeksCvoraRoditelja = pronadjiIndeksCvora(stablo, putanjaRoditelja);

  pfree(putanjaRoditelja);


  // ako roditelj ne postoji, izbacujemo grešku
  if (indeksCvoraRoditelja == NEPOSTOJECI_CVOR) {
    ereport(ERROR,
      (errcode(ERRCODE_RAISE_EXCEPTION),
```

```
        errmsg("Cvor roditelja jos nije definiran!\n"))
    );
  }

  // ako roditelj postoji, provjeravamo je li zauzeto njegovo
  // lijevo ili desno dijete, ovisno o tome na koju stranu
  // želimo dodati podstablo
  if (pozicijaPodstabla == 'L') {

    if (lijevoDijeteB(indeksCvoraRoditelja, stablo) ==
      NEPOSTOJECI_CVOR) {
        int indeksLijevogDjeteta = indeksCvoraRoditelja*2;
        popuni_stablo(stablo, podstablo, INDEKS_KORIJENA_STABLA,
          indeksLijevogDjeteta);
    } else {
        ereport(ERROR,
          (errcode(ERRCODE_RAISE_EXCEPTION),
            errmsg("Odabrana putanja vec ima lijevo dijete!\n"))
        );
    }

  } else {

    if (desnoDijeteB(indeksCvoraRoditelja, stablo) ==
      NEPOSTOJECI_CVOR) {
        int indeksDesnogDjeteta = indeksCvoraRoditelja*2+1;
        popuni_stablo(stablo, podstablo, INDEKS_KORIJENA_STABLA,
          indeksDesnogDjeteta);
    } else {
        ereport(ERROR,
          (errcode(ERRCODE_RAISE_EXCEPTION),
            errmsg("Odabrana putanja vec ima desno dijete!\n"))
        );
    }
  }

  PG_RETURN_POINTER(stablo);
}
```

Kao i kod funkcije za dodavanje novog čvora, putanju do podstabla, koje se treba umentuti, se raščlanjuje na putanju do roditelja i poziciju podstabla u odnosu na roditelja (lijevo ili desno). Nakon toga se provjerava je li pozicija, na koju se treba umetnuti podstablo, slobodna. Ako jest, podstablo se umeće u originalno stablo korištenjem već opisane funkcije *popuni_stablo*.

Fukncija *popuni_stablo* (odnosno njezina varijanta *popuniStablo*) je originalno korištena kako bi popunila prazno stablo. Međutim, tu funkciju možemo koristiti da kopiramo bilo koje stablo ili podstablo

u bilo koju poziciju nekog drugog stabla. *popuniStablo* je obična makro naredba definirana na sljedeći način:

```
#define popuniStablo(a, b, c)
  popuni_stablo(a, b, c, INDEKS_KORIJENA_STABLA)
```

gdje je *INDEKS_KORIJENA_STABLA* konstanta koja ima vrijednost 1.

6.9 Načini ispisa stabla

Postoje tri načina ispisa stabla na zaslon: **preorder**, **inorder** i **postorder**. Za ispis se po defaultu koristi *preorder* način, međutim, može se eksplicitno specificirati jedan od druga dva načina.

6.9.1 Preorder način ispisa

Preorder način ispisa je definiran sljedećim programskim kodom.

```
Datum stablo_preorder(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *izlaz = (char *) palloc(10000);

  // alociranoj varijabli izlaz za prvi znak
  // postavljamo terminator znak
  izlaz[0] = '\0';
  preorder(stablo, INDEKS_KORIJENA_STABLA, izlaz);

  PG_RETURN_CSTRING(izlaz);
}

void preorder(bstablo *stablo, int pocetniIndeksCvora, char *izlaz) {
  int indeksLijevogDjeteta, indeksDesnogDjeteta;
  int idLijevo, idDesno;

  if (pocetniIndeksCvora == NEPOSTOJECI_CVOR) return;

  indeksLijevogDjeteta = lijevoDijeteB(pocetniIndeksCvora, stablo);
  indeksDesnogDjeteta = desnoDijeteB(pocetniIndeksCvora, stablo);

  idLijevo = (indeksLijevogDjeteta == NEPOSTOJECI_CVOR) ?
    -1 : indeksLijevogDjeteta;
  idDesno = (indeksDesnogDjeteta == NEPOSTOJECI_CVOR) ?
    -1 : indeksDesnogDjeteta;

  // čvor se ispisuje u varijablu izlaz
  ispisiCvor(pocetniIndeksCvora,
    vrijednostB(pocetniIndeksCvora, stablo), idLijevo, idDesno, izlaz);
```

```
  // granamo se lijevo i desno
  preorder(stablo, indeksLijevogDjeteta, izlaz);
  preorder(stablo, indeksDesnogDjeteta, izlaz);
}
```

Vidimo da funkcija *stablo_preorder* poziva rekurzivnu funkciju *preorder*. Zadatak *preorder* funkcije je da se ispiše čvorove čim prije. Ona će, dakle, najprije ispisati čvor, te se pomaknuti na lijevo dijete. Nakon toga će ponovno ispisati čvor te se ponovno pomaknuti na lijevu stranu. Ukoliko se ne uspije pomaknuti lijevo, pokušat će preći na desno dijete, ispisati ga i ponovno se pomicati lijevo.

6.9.2 Inorder način ispisa

Za razliku od preorder način, *inorder* način ispisa će se najprije pokušati pomicati lijevo u binarnom stablu, koliko god može. Neposredno prije nego se pokuša pomaknuti desno, inorder način će ispisati čvor na kojem se trenutno nalazi. Programski kod je prikazan ispod.

```
Datum stablo_inorder(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *izlaz = (char *) palloc(10000);

  // alociranoj varijabli izlaz
  // za prvi znak postavljamo terminator znak
  izlaz[0] = '\0';
  inorder(stablo, INDEKS_KORIJENA_STABLA, izlaz);

  PG_RETURN_CSTRING(izlaz);
}

void inorder(bstablo *stablo, int pocetniIndeksCvora, char *izlaz) {
  int indeksLijevogDjeteta, indeksDesnogDjeteta;
  int idLijevo, idDesno;

  if (pocetniIndeksCvora == NEPOSTOJECI_CVOR) return;

  indeksLijevogDjeteta = lijevoDijeteB(pocetniIndeksCvora, stablo);
  indeksDesnogDjeteta = desnoDijeteB(pocetniIndeksCvora, stablo);

  idLijevo = (indeksLijevogDjeteta == NEPOSTOJECI_CVOR) ?
    -1 : indeksLijevogDjeteta;
  idDesno = (indeksDesnogDjeteta == NEPOSTOJECI_CVOR) ?
    -1 : indeksDesnogDjeteta;

  inorder(stablo, indeksLijevogDjeteta, izlaz);
  ispisiCvor(pocetniIndeksCvora,
    vrijednostB(pocetniIndeksCvora, stablo), idLijevo, idDesno, izlaz);
  inorder(stablo, indeksDesnogDjeteta, izlaz);
```

```
}
```

Možemo vidjeti da je funkcija ispisiCvor između dva poziva funkcije *inorder*. Funkciju *ispisiCvor* nećemo ovdje detaljno objašnjavati. Samo ćemo reći da ta funkcija pretvara jedan čvor binarnog stabla u čvor pogodan za ispis, te ga dodaje na kraj string-a *izlaz*.

6.9.3 Postorder način ispisa

Postorder pokušava obići stablo prije nego što započne s ispisivanjem čvorova.

```
Datum stablo_postorder(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  char *izlaz = (char *) palloc(10000);

  // alociranoj varijabli izlaz
  // za prvi znak postavljamo terminator znak
  izlaz[0] = '\0';
  postorder(stablo, INDEKS_KORIJENA_STABLA, izlaz);

  PG_RETURN_CSTRING(izlaz);
}

void postorder(bstablo *stablo, int pocetniIndeksCvora, char *izlaz) {
  int indeksLijevogDjeteta, indeksDesnogDjeteta;
  int idLijevo, idDesno;

  if (pocetniIndeksCvora == NEPOSTOJECI_CVOR) return;

  indeksLijevogDjeteta = lijevoDijeteB(pocetniIndeksCvora, stablo);
  indeksDesnogDjeteta = desnoDijeteB(pocetniIndeksCvora, stablo);

  idLijevo = (indeksLijevogDjeteta == NEPOSTOJECI_CVOR) ?
    -1 : indeksLijevogDjeteta;
  idDesno = (indeksDesnogDjeteta == NEPOSTOJECI_CVOR) ?
    -1 : indeksDesnogDjeteta;

  postorder(stablo, indeksLijevogDjeteta, izlaz);
  postorder(stablo, indeksDesnogDjeteta, izlaz);
  ispisiCvor(pocetniIndeksCvora,
    vrijednostB(pocetniIndeksCvora, stablo), idLijevo, idDesno, izlaz);
}
```

6.10 Pretraživanje stabla

Jedna od vjerojatno najkorisnijih mogućnosti binarnog stabla je pretraživanje po vrijednosti. Pravilno implementirana funkcija pretraživanja vratit će sve instance čvorova koji odgovaraju zadanoj vrijednosti.

```
Datum stablo_pretrazi(PG_FUNCTION_ARGS) {
  bstablo *stablo = (bstablo *) PG_GETARG_POINTER(0);
  int vrijednost = (int) PG_GETARG_INT32(1);

  int i;
  // ovdje ćemo bilježiti sve putanje do čvorova
  char *putanje = (char *) palloc(1000);
  // ovdje ćemo bilježiti putanju do pojedinog čvora
  char *putanja;

  // null-terminiramo varijablu putanje
  putanje[0] = '\0';


  // tražimo čvorove koji se poklapaju sa zadanom vrijednošću
  for (i = INDEKS_KORIJENA_STABLA; i < MAX_BROJ_CVOROVA; i++) {
    // kada smo našli čvor koji se podudara
    if (stablo[i].iskoristen == 1 && stablo[i].v == vrijednost) {
      // tražimo njegovu putanju
      putanja = pronadjiPutanjuCvora(stablo, i);

      // ako je putanja prazna, to znači da se radi o korijenu
      if (strlen(putanja) == 0) {
        strcpy(putanja, "K");
      }

      // dodajemo dobivenu putanju u listu putanja
      // u ovisnosti o tome je li lista inicijalizirana ili ne
      if (strlen(putanje) == 0) {
        strcpy(putanje, putanja);
      } else {
        strcat(putanje, ", ");
        strcat(putanje, putanja);
      }

      pfree(putanja);
    }
  }

  // na kraju, ako nije pronađena niti jedna vrijednost,
  // u rezultat umjesto putanje upisujemo poruku o tome
  // da niti jedan čvor s tom vrijednošću nije pronađen
  if (strlen(putanje) == 0) {
    strcpy(putanje, "Ne postoji cvor s tom vrijednoscu!");
```

```
  }

  PG_RETURN_CSTRING(putanje);
}
```

Pretraživanje se vrši slijedno (a ne rekurzivno). To je moguće, zato jer je stablo zapravo implementirano pomoću polja. Kada se pronađe čvor čija vrijednost odgovara traženoj, traži se njegova putanja korištenjem već poznate funkcije *pronadjiPutanjuCvora*. Pretraga se nastavlja sve dok se ne pretraži čitavo stablo.

6.11 Uspoređivanje jednakosti stabla

U ovoj implementaciji je također moguće uspoređivati i jednakost dvaju stabala. U tu svrhu se mogu koristiti dva operatora, = i <>. C funkcije koje implementiraju tu funkcionalnost su navedene ispod.

```
Datum stablo_eq(PG_FUNCTION_ARGS) {
  bstablo *stablo1 = (bstablo *) PG_GETARG_POINTER(0);
  bstablo *stablo2 = (bstablo *) PG_GETARG_POINTER(1);

  if (jednakiB(stablo1, stablo2))
    PG_RETURN_BOOL(TRUE);
  else
    PG_RETURN_BOOL(FALSE);
}

Datum stablo_neq(PG_FUNCTION_ARGS) {
  bstablo *stablo1 = (bstablo *) PG_GETARG_POINTER(0);
  bstablo *stablo2 = (bstablo *) PG_GETARG_POINTER(1);

  if (! jednakiB(stablo1, stablo2))
    PG_RETURN_BOOL(TRUE);
  else
    PG_RETURN_BOOL(FALSE);
}
```

Vrlo se lako može primijetiti da se ove dvije funkcije razlikuju samo u operatoru nejednakosti u jednoj liniji koda. Obje za usporedbu koriste funkciju *jednakiB*, koja je sastavni dio implementacije binarnog stabla.

Poglavlje 7

Binarno stablo u PostgreSQL-u

Nakon što se *plugin* za binarno stablo kompilira pomoću naredbe *make* i kopira u PostgreSQL *lib* direktorij pomoću naredbe *make install*, potrebno ga je podržati u PostgreSQL-u, o čemu se nešto više govori u sljedećem odlomku.

7.1 Omogućavanje binarnog stabla u PostgreSQL-u

Binarno stablo je izrađeno kao tip podataka. Stoga je u postgreSQL-u potrebno dodati novi tip podataka. Naziv tipa mora biti jednak nazivu implementacije koji je specificiran u Makefile datoteci. U PostgreSQL-u je potrebno izvršiti sljedeću naredbu:

```
CREATE TYPE stablo;
```

Ta će naredba kreirati opći tip podataka. Da bi ga bolje specificirali, potrebno je i podržati dvije funkcije za unos i ispis binarnog stabla:

```
CREATE OR REPLACE FUNCTION stablo_in(cstring) RETURNS stablo AS
  'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_out(stablo) RETURNS cstring AS
  'stablo' LANGUAGE C IMMUTABLE STRICT;
```

Imena funkcija u PostgreSQL-u moraju biti jednaka imenima funkcija koje su definirane u C jeziku. Ukratko, prva naredba govori o sljedećem: "Dodaj ili zamijeni funkciju *stablo_in* koja prima jedan argument tipa *cstring* i vraća tip podataka *stablo*. Kao izvor podataka koristi dodatak *stablo* koji je napisan u jeziku C. Funkcija sama po sebi ne može mijenjati sadržaj baze podataka (immutable) i za NULL ulaz vraća NULL izlaz (strict)."

Budući da sada postoje funkcije za ulaz i izlaz podataka, moguće je proširiti binarno stablo redefinicijom.

```
CREATE TYPE stablo (internallength = 4096, input = stablo_in,
  output = stablo_out, alignment = int);
```

Ovom redefinicijom zadajemo fiksnu veličinu jednog objekta binarnog stabla koji zauzima 4096 bajta (čime poprima maksimalnu dubinu od 9 razina). Također je moguće dodati i nove funkcije koje su implementirane u C jeziku.

```
CREATE OR REPLACE FUNCTION stablo_broj_cvorova(stablo) RETURNS INT AS
  'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_broj_cvorova_podstabla(stablo,
  cstring) RETURNS INT AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_postoji_cvor(stablo, cstring)
  RETURNS BOOL AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_vrijednost(stablo, cstring)
  RETURNS INT AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_promijeni_vrijednost(stablo,
  cstring, int) RETURNS stablo AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_dodaj_cvor(stablo, cstring, int)
  RETURNS stablo AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_orezi(stablo, cstring)
  RETURNS stablo AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_podstablo(stablo, cstring)
  RETURNS stablo AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_dodaj_podstablo(stablo, cstring,
  cstring) RETURNS stablo AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_preorder(stablo) RETURNS cstring
  AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_inorder(stablo) RETURNS cstring
  AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_postorder(stablo) RETURNS cstring
  AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_pretrazi(stablo, int) RETURNS cstring
  AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
```

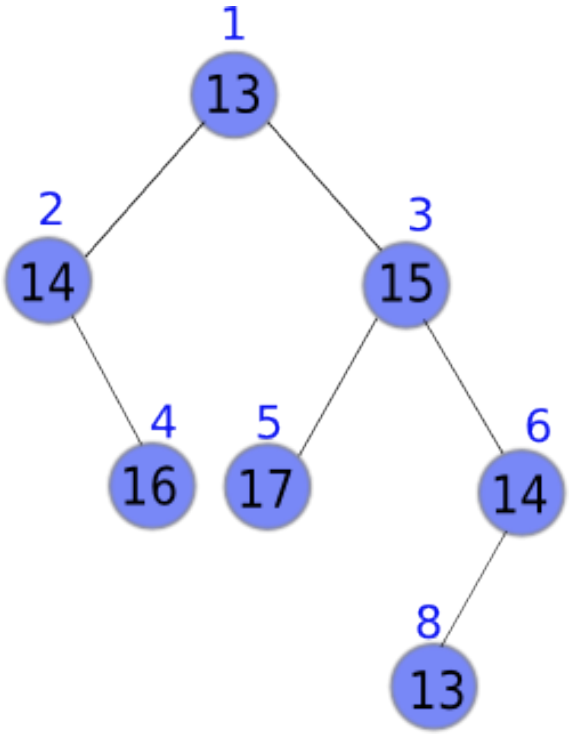
Na kraju je potrebno još definirati operatore jednakosti. Operatori u PostgreSQL-u zapravo u pozadini koriste funkcije, tako da je osim dva operatora potrebno dodati i dvije funkcije.

```
CREATE OR REPLACE FUNCTION stablo_eq(stablo, stablo)
  RETURNS BOOL AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OR REPLACE FUNCTION stablo_neq(stablo, stablo)
  RETURNS BOOL AS 'stablo' LANGUAGE C IMMUTABLE STRICT;
CREATE OPERATOR = (leftarg = stablo, rightarg = stablo,
  procedure = stablo_eq, commutator = '=', negator = '<>');
CREATE OPERATOR <> (leftarg = stablo, rightarg = stablo,
  procedure = stablo_neq, commutator = '<>', negator = '=');
```

7.2 Primjeri korištenja binarnog stabla

Kreirajmo najprije jednostupčanu tablicu u koju će se umetnuti binarno stablo prikazano na slici.

```
CREATE TABLE s(s stablo);
```



Slika 7.1: Binarno stablo korišteno u primjerima

Sada možemo u tablicu umetnuti binarno stablo na slici 7.1. Brojevi unutar kružića prikazuju vrijednosti, dok brojevi iznad kružića prikazuju identifikacijske oznake čvorova.

```
INSERT INTO s VALUES ('(1,13,2,3);(2,14,-1,4);(3,15,5,6);(4,16,-1,-1);
(5,17,-1,-1);(6,14,8,-1);(8,13,-1,-1)');
```

Unos se u bazu odvija na sljedeći način. Svakom čvoru moramo dati njegov ID i vrijednost. Prvi navedeni čvor (korijen stabla) ima ID 1, vrijednost 13. Njegovo lijevo dijete ima ID 2, a desno 3. Ako sada pogledamo primjerice čvor s identifikacijskom oznakom 6, primijetit ćemo da ono ima samo lijevo dijete, dok je njegovo desno dijete označeno s -1, što znači da ono zapravo ne postoji.

Stablo se neće promijeniti, ako se čvorovi napišu u nekom drugom redosljedu, zato jer se tekstualni oblik u kojem korisnik unosi stablo svakako pretvara u binarni oblik koji ima vlastitu strukturu u memoriji.

Upitom u bazu podataka, dobivamo sljedeće:

```
SELECT * FROM s;

                              s
-------------------------------------------------------------------
(1,13,2,3);(2,14,-1,5);(5,16,-1,-1);(3,15,6,7);
  (6,17,-1,-1);(7,14,14,-1);(14,13,-1,-1)
```

Možemo primijetiti da izlaz nije potpuno jednak ulazu. To je zato jer se binarno stablo iz memorije (s diska) moralo pretvoriti u tekstualnu reprezentaciju. Kada se u bazu podataka unosilo binarno stablo, nisu se pohranjivali neki nebitni podaci kao što su ID-evi stabla i slično. Stoga su se te vrijednosti morale rekreirati pri ispisu, zbog čega su one poprimile druge vrijednosti. Međutim, ukoliko pokušamo izjednačiti uneseno stablo sa onim koje je dobiveno pomoću upita,

```
SELECT '(1,13,2,3);(2,14,-1,4);(3,15,5,6);(4,16,-1,-1);(5,17,-1,-1);
(6,14,8,-1);(8,13,-1,-1)'::stablo =
'(1,13,2,3);(2,14,-1,5);(5,16,-1,-1);(3,15,6,7);(6,17,-1,-1);
(7,14,14,-1);(14,13,-1,-1)'::stablo;



 ?column?
----------
 t
```

dobit ćemo da su stabla jednaka.

Ako bismo željeli prebrojiti elemente podstabla, koje počinje na putanji desno-desno (u odnosu na korijen stabla), dobili bismo sljedeći rezultat.

```
SELECT stablo_broj_cvorova_podstabla(s, 'DD') AS broj FROM s;

 broj
------
    2
```

Dobili bismo broj 2. Ukoliko bismo željeli vidjeti koji su to čvorovi, postavili bismo sljedeći upit.

```
SELECT stablo_podstablo(s, 'DD') AS podstablo FROM s;

         podstablo
---------------------------
 (1,14,2,-1);(2,13,-1,-1)
```

Ispostavilo se da su to dva krajnje desna čvora, s vrijednostima 14 i 13, što možemo provjeriti i na priloženoj slici.

Broj čvorova na putanji *DD* smo mogli dobiti i na nešto složeniji način:

```
SELECT stablo_broj_cvorova(stablo_podstablo(s, 'DD')) AS broj FROM s;

 broj
------
    2
```

Međutim, nije preporučano koristiti ovaj način prebrojavanja, zato jer se troši puno više računalnih resursa (zbog dvije funkcije koje se pozivaju, umjesto jedne). Ovaj primjer je napravljen samo kao demonstracija mogućnosti plugin-a.

Ukoliko bismo željeli orezati cijelu desnu granu, ostao bi samo korijen stabla i njegova lijeva grana:

```
SELECT stablo_orezi(s, 'D') AS orezano FROM s;

                orezano
---------------------------------------
```

```
(1,13,2,-1);(2,14,-1,5);(5,16,-1,-1)
```

Orezanom stablu iz prethodnog primjera možemo na desnu stranu dodati novo podstablo.

```
SELECT stablo_dodaj_podstablo(stablo_orezi(s, 'D'), 'D',
  '(1,2,-1,4);(4,8,-1,-1)') FROM s;

                    stablo_dodaj_podstablo
-----------------------------------------------------------------
(1,13,2,3);(2,14,-1,5);(5,16,-1,-1);(3,2,-1,7);(7,8,-1,-1)
```

Možemo pronaći gdje se u stablu nalazi čvor koji ima vrijednost 17.

```
SELECT stablo_pretrazi(s, 17) AS putanja FROM s;

 putanja
---------
 DL
```

Također možemo provjeriti ima li taj čvor desno dijete:

```
SELECT stablo_postoji_cvor(s, 'DLD') FROM s;

 stablo_postoji_cvor
---------------------
 f
```

Originalno stablo možemo ispisati načinom *postorder*.

```
SELECT stablo_postorder(s) AS postorder FROM s;

                          postorder
-----------------------------------------------------------
(5,16,-1,-1);(2,14,-1,5);(6,17,-1,-1);(14,13,-1,-1);
(7,14,14,-1);(3,15,6,7);(1,13,2,3)
```

Poglavlje 8

Zaključak

Iako je ovo bio školski primjer binarnog stabla, s mnogo ograničenja, iz njega je jasno da se u PostgreSQL sustavu za upravljanje bazama podataka može napraviti tip podataka za bilo kakve potrebe. Na samom kraju ovog projeka bih želio napomenuti da u PostgreSQL-u postoje i takozvani TOAST tipovi podataka (PostgreSQL, 2014c), koji su skalabilni u smislu da veličina pohranjenih podataka može s vremenom varirati. TOAST koncept je idealan za tipove podataka koji mogu sadržavati varijabilan broj elemenata, kao što je recimo binarno stablo.

Bibliografija

Orehovački, T., 2011. *ATP Binarno stablo*. Dostupno na: `http://elfarchive.foi.hr/11_12/mod/resource/view.php?id=6740`.

PostgreSQL, 2014a. *C-Language Functions*. Dostupno na: `http://www.postgresql.org/docs/9.3/interactive/xfunc-c.html`.

PostgreSQL, 2014b. *SPI palloc*. Dostupno na: `http://www.postgresql.org/docs/9.3/interactive/spi-spi-palloc.html`.

PostgreSQL, 2014c. *TOAST*. Dostupno na: `http://www.postgresql.org/docs/9.3/static/storage-toast.html`.

PostgreSQL, 2014d. *User-defined Types*. Dostupno na: `http://www.postgresql.org/docs/9.3/static/xtypes.html`.