



ELEKTRONSKI FAKULTET NIŠ
STRUČNA PRAKSA

Izveštaj o realizaciji stručne prakse

Katarina Mladenović 16765

Niš, Oktobar 2021.

Sadržaj

Uvod	3
Izveštaj	4
Zaključak	13

Uvod

Praksa je održana u laboratorijama Elektronskog fakulteta u Nišu, i to u periodu od 13.09.2021. do 24.09.2021. Na praksi sam se bavila tehnikama particionisanja u PostgreSQL bazi podataka.

PostgreSQL predstavlja objektno-relacioni sistem za upravljanje bazama podataka (objektno-relacioni DBMS ili ORDBMS), proizveden na osnovu Berklijevog sistema za upravljanje bazama podataka Postgres. PostgreSQL sadrži moćan objektno-relacioni model podataka, bogat izbor vrsta podataka, laku nadogradivost, kao i nadograđeni set naredbi SQL jezika.

Entity Framework Core predstavlja moderan objektno-relacioni mapper za .NET. Podržava LINQ upite, praćenje promena, ažuriranja i migracije šema. EF Core radi sa mnogim bazama podataka, uključujući SQLite, MySQL, PostgreSQL i Azure Cosmos DB.

Za izradu .NET Core aplikacija potrebno je instalirati razvojno okruženje Microsoft Visual Studio. Microsoft Visual Studio je integrisano programsko okruženje, programirano od strane kompanije Microsoft. Visual Studio se koristi za programiranje računarskih igara, programa (Metro UI, desktop), veb-sajtova, veb-servisa i veb-aplikacija na Microsoft Windows-u.

Kod se nalazi na sledećem repozitorijumu: https://github.com/vale-decem/strucna_praksa

Izveštaj

Dan 1 – Upoznavanje sa konceptom particionisanja u PostgreSQL

Particionisanje se odnosi na razdvajanje onoga što je logički jedna velika tabela na što manje fizičke delove. Particionisanje može pružiti nekoliko prednosti:

- Performanse upita se mogu dramatično poboljšati u određenim situacijama, posebno kada je većina redova tabele sa velikim pristupom na jednoj particiji ili malom broju particija.
- Kada upiti ili ažuriranja pristupaju velikom procentu jedne particije, performanse se mogu poboljšati korišćenjem prednosti uzastopnog skeniranja te particije, umesto korišćenja indeksa i čitanja sa nasumičnim pristupom rasutih po celoj tabeli.
- Grupno učitavanje i brisanje se može postići dodavanjem ili uklanjanjem particija, ukoliko je taj zahtev planiran u dizajnu particionisanja.
- Podaci koji se retko koriste mogu se migrirati na jeftinije i sporije medije za skladištenje.

Benefiti će obično biti vredni onda kada bi tabela bila veoma velika. Tačan momenat kada će tabela imati benefite od particionisanja zavisi od same aplikacije, iako je opšte pravilo da veličina tabele treba da premaši fizičku memoriju servera baze podataka.

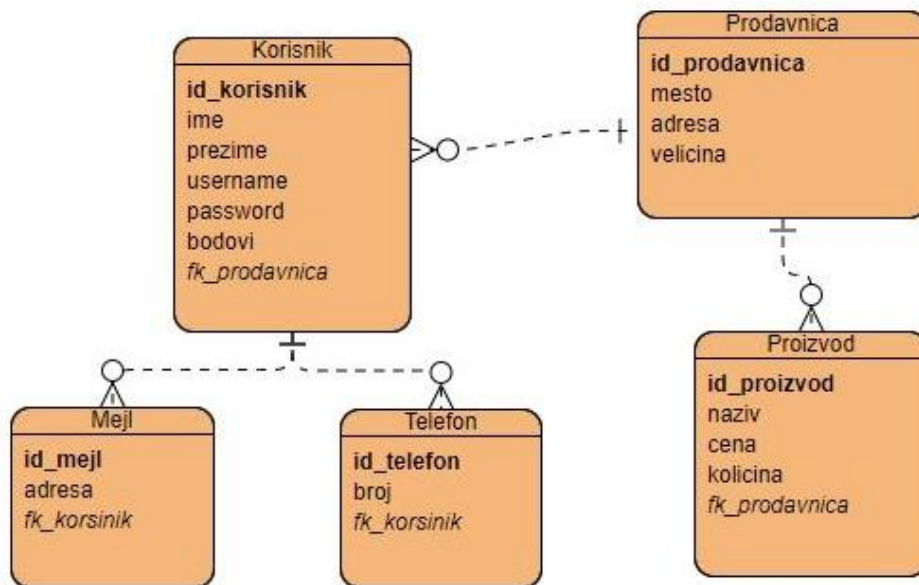
PostgreSQL nudi built-in podršku za sledeće oblike deklarativnog particionisanja:

- **Range particionisanje** – tabela je podeljena na „opsege“ definisane ključnom kolonom ili skupom kolona, bez preklapanja između opsega vrednosti dodeljenih različitim particijama,
- **List particionisanje** – tabela je podeljena eksplicitnim navođenjem koje vrednosti ključa se pojavljuju u svakoj particiji,
- **Hash particionisanje** – tabela je podeljena specificiranjem modula i ostatka za svaku particiju.

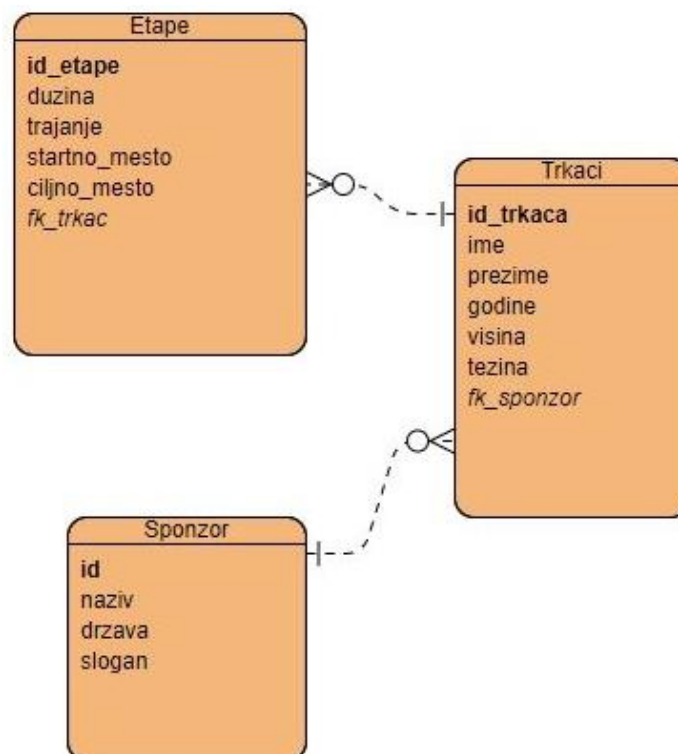
Iako je deklarativno particionisanje pogodno za najčešće slučajeve upotrebe, postoje neke okolnosti u kojima bi fleksibilniji pristup bio pogodan. Particionisanje se može implementirati pomoću nasleđivanja tabele, što dozvoljava nekoliko funkcionalnosti koje nisu podržane deklarativnim particionisanjem, kao što su mogućnost dodatnih kolona child tabele u odnosu na parent tabelu, mogućnost višestrukog nasleđivanja, mogućnost deljenja podataka na način koji korisnik izabere.

Dan 2 – Kreiranje modela baza podataka

U procesu kreiranja modela baza podataka, fokus je bio na tome da se obezbedi bar jedna tabela koja će referencirati particionisanu tabelu, kako bi se adekvatno prikazao mehanizam particionisanja tabela, kao i referenciranje particionisanih tabela u PostgreSQL-u.



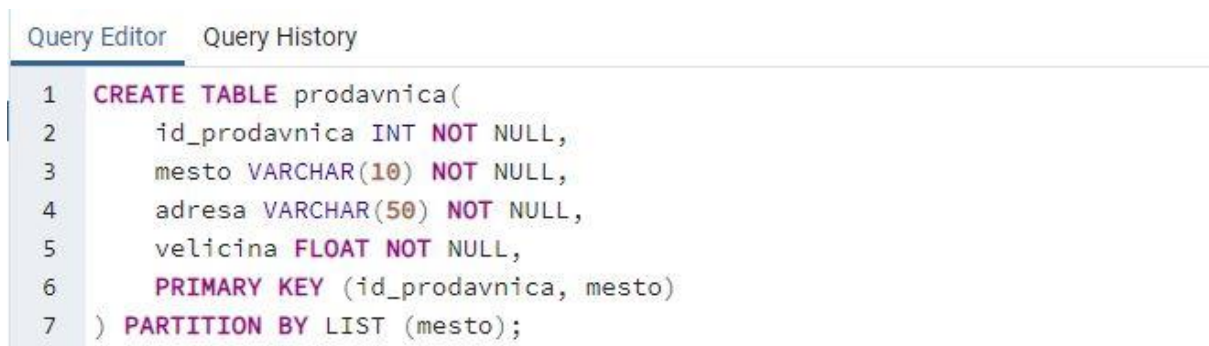
slika 1.0 - EER dijagram baze „Prodavnica“



slika 1.1 - EER dijagram baze „Trkači“

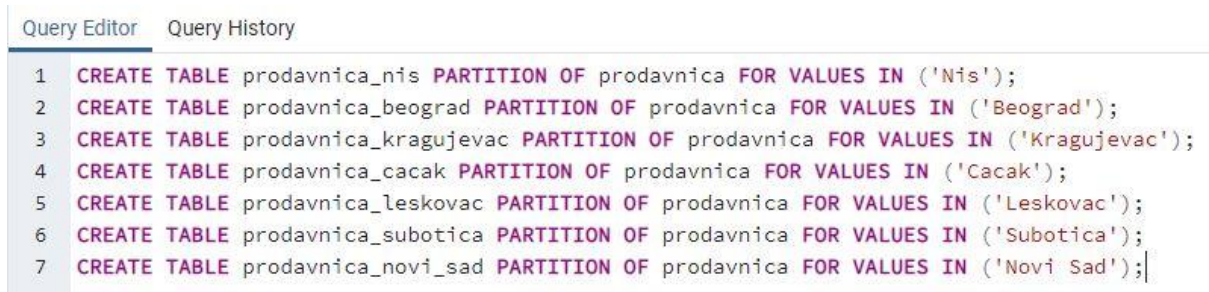
Dan 3 – Kreiranje baza podataka u pgAdmin-u

U okviru CREATE TABLE naredbi neophodno je bilo definisati način particionisanja, kao i kolonu/kolone po kojima će se particionisanje tabele vršiti. Zatim se vrši kreiranje particija, takođe korišćenjem CREATE TABLE naredbe, pri čemu je bilo potrebno navesti naziv tabele, čije su one particije, kao i opseg vrednosti kolone particionisanja, koji se može naći u tabeli.



```
Query Editor  Query History
1 CREATE TABLE prodavnica(
2     id_prodavnica INT NOT NULL,
3     mesto VARCHAR(10) NOT NULL,
4     adresa VARCHAR(50) NOT NULL,
5     velicina FLOAT NOT NULL,
6     PRIMARY KEY (id_prodavnica, mesto)
7 ) PARTITION BY LIST (mesto);
```

slika 1.2 - Kreiranje tabele



```
Query Editor  Query History
1 CREATE TABLE prodavnica_nis PARTITION OF prodavnica FOR VALUES IN ('Nis');
2 CREATE TABLE prodavnica_beograd PARTITION OF prodavnica FOR VALUES IN ('Beograd');
3 CREATE TABLE prodavnica_kragujevac PARTITION OF prodavnica FOR VALUES IN ('Kragujevac');
4 CREATE TABLE prodavnica_cacak PARTITION OF prodavnica FOR VALUES IN ('Cacak');
5 CREATE TABLE prodavnica_leskovac PARTITION OF prodavnica FOR VALUES IN ('Leskovac');
6 CREATE TABLE prodavnica_subotica PARTITION OF prodavnica FOR VALUES IN ('Subotica');
7 CREATE TABLE prodavnica_novi_sad PARTITION OF prodavnica FOR VALUES IN ('Novi Sad');
```

slika 1.3 - Kreiranje particija

Za ostala dva tipa deklarativnog particionisanja, postupak je identičan.

Dan 4 – Ubacivanje podataka u baze

Postupak ubacivanja podataka u baze podataka se realizuje INSERT INTO naredbom. Ova naredba se koristi i za ubacivanje podataka u particionisanu tabelu, pri čemu se u naredbi navodi ime tabele, a sam postupak raspoređivanja podatka u adekvatnu particiju vrši PostgreSQL mehanizam.

```
1 insert into sponzor (id, naziv, drzava, slogan) values (1, 'Photojam', 'Czech Republic', 'Hatity');
2 insert into sponzor (id, naziv, drzava, slogan) values (2, 'Vipe', 'Bulgaria', 'Matsoft');
3 insert into sponzor (id, naziv, drzava, slogan) values (3, 'Mydeo', 'Germany', 'Kanlam');
4 insert into sponzor (id, naziv, drzava, slogan) values (4, 'Mynte', 'Great Britain', 'Latlux');
5 insert into sponzor (id, naziv, drzava, slogan) values (5, 'Skajo', 'France', 'Fintone');
6 insert into sponzor (id, naziv, drzava, slogan) values (6, 'Cogibox', 'Switzerland', 'Opela');
7 insert into sponzor (id, naziv, drzava, slogan) values (7, 'Innotype', 'Norway', 'Trippledex');
8 insert into sponzor (id, naziv, drzava, slogan) values (8, 'Lajo', 'Romania', 'Fintone');
9 insert into sponzor (id, naziv, drzava, slogan) values (9, 'Talan', 'Italy', 'Otcom');
10 insert into sponzor (id, naziv, drzava, slogan) values (10, 'Abata', 'Greece', 'Viva');
```

slika 1.4 - Insert into naredbe

```
1 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (1, 'Novi Sad', '87 Lien Terrace', 273.9);
2 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (2, 'Subotica', '2237 Merrick Street', 832.09);
3 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (3, 'Beograd', '7 Bartelt Place', 437.32);
4 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (4, 'Novi Sad', '501 Judy Terrace', 526.13);
5 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (5, 'Beograd', '45985 Vahlen Way', 648.68);
6 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (6, 'Beograd', '312 Oriole Park', 914.71);
7 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (7, 'Novi Sad', '93374 Londonderry Hill', 459.0);
8 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (8, 'Novi Sad', '8585 Oxford Circle', 460.43);
9 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (9, 'Beograd', '55 Luster Junction', 630.09);
10 insert into prodavnica (id_prodavnic, mesto, adresa, velicina) values (10, 'Nis', '26 Butterfield Park', 631.03);
```

slika 1.5 - Insert into naredbe particionisane tabele

Dan 5 – Povezivanje sa PostgreSQL bazom podataka

Za rad sa PostgreSQL bazom podataka u okviru .NET Core-a, korišćen je Entity Framework Core objektno-relacioni mapper. Kreira se Context klasa, koja nasleđuje DbContext klasu, gde se navode sve tabele baze podataka kao property-ji tipa DbSet<NazivTabele>. Za povezivanje na postojeću bazu podataka koristi se metoda OnConfiguring, koja za argument uzima string, koji sadrži neophodne podatke o bazi. Nakon toga se kreiraju modeli svih tabela baza podataka, gde je potrebno voditi računa o poklapanju naziva kolona sa nazivima u PostgreSQL bazama podataka. U okviru modela tabele kreiraju se property-ji za svaku kolonu tabele i dodaju im se atributi (npr. Key, ForeignKey) gde je to neophodno.

```
namespace ListPartitioning
{
    2 references
    public class ListContext : DbContext
    {
        5 references
        public DbSet<Prodavnica> prodavnica { get; set; }
        1 reference
        public DbSet<Proizvod> proizvod { get; set; }
        1 reference
        public DbSet<Korisnik> korisnik { get; set; }
        0 references
        public DbSet<Mejl> mejl { get; set; }
        0 references
        public DbSet<Telefon> telefon { get; set; }

        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
            => optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=prodavnica_list;" +
                "User Id=postgres;Password=root");
    }
}
```

slika 1.6 - Context klasa

```
namespace ListPartitioning.Models
{
    7 references
    public class Prodavnica
    {
        [Key]
        4 references
        public int id_prodavnica { get; set; }
        5 references
        public string mesto { get; set; }
        2 references
        public string adresa { get; set; }
        2 references
        public double velicina { get; set; }
    }
}
```

slika 1.7 - Model tabele

Kako tabela Prodavnica ima kompozitni primarni ključ, bilo je potrebno dodati i ovo ograničenje, i to korišćenjem metode OnModelCreating.

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Prodavnica>()
        .HasKey(c => new { c.id_prodavnica, c.mesto });
}
```

slika 1.8 - Dodatna ograničenja

Dan 6 – *CRUD operacije*

Entity Framework Core omogućava jednostavan i intuitivan rad sa PostgreSQL bazom podataka, koristeći EF Core metode, koje za nas u pozadini izvršavaju SQL upite nad bazom podataka.

Za dodavanje novog podatka u bazu, potrebno je kreirati novi objekat sa željenim podacima i pozvati Add metodu nad tabelom podataka u koju želimo da se podatak unese. Kako bi ove promene bile vidljive u bazi, treba pozvati metodu SaveChanges nad kontekstom.

```
0 references
private void btnCreate_Click(object sender, EventArgs e)
{
    Prodavnica pr = new Prodavnica();
    pr.id_prodavnica = 151;
    pr.mesto = "Leskovac";
    pr.adresa = "Bulevar oslobodjenja 57";
    pr.velicina = 910;

    context.prodavnica.Add(pr);
    context.SaveChanges();
}
```

slika 1.9 - Dodavanje podatka u bazu

Za čitanje podatka potrebno je pronaći podatak sa odgovarajućim id-jem koristeći metodu Find, zatim i prikazati taj podatak.

```
1 reference
private void btnRead_Click(object sender, EventArgs e)
{
    Prodavnica pr = context.prodavnica.Find(2, "Subotica");
    MessageBox.Show(pr.mesto + " " + pr.adresa);

    context.SaveChanges();
}
```

slika 2.0 - Čitanje podatka iz baze

Za ažuriranje podatka potrebno je prvo pronaći taj podatak u bazi, izvršiti željene promene nad podatkom i usnimiti promene u bazu korišćenjem metode SaveChanges.

```
1 reference
private void btnUpdate_Click(object sender, EventArgs e)
{
    Prodavnica pr = context.prodavnica.Find(2, "Subotica");
    pr.velicina = 570;
    context.prodavnica.Update(pr);

    context.SaveChanges();
}
```

slika 2.1 - Ažuriranje podatka u bazi

Za brisanje podataka potrebno je korišćenjem metode Find naći podatak u bazi, zatim se metodom Remove briše podatak iz baze podataka.

```
1 reference
private void btnDelete_Click(object sender, EventArgs e)
{
    Prodavnica prod = context.prodavnica.Find(151, "Leskovac");

    var sviKorisnici = context.korisnik.Where(
        (korisnik => korisnik.fk_prodavnica == prod.id_prodavnica &&
        korisnik.fk_mesto == prod.mesto)).ToList();

    foreach (Korisnik k in sviKorisnici)
        context.Remove(k);

    var sviProizvodi = context.proizvod.Where(
        (proizvod => proizvod.fk_prodavnica == prod.id_prodavnica &&
        proizvod.fk_mesto == prod.mesto)).ToList();

    foreach (Proizvod p in sviProizvodi)
        context.Remove(p);

    context.Remove(prod);

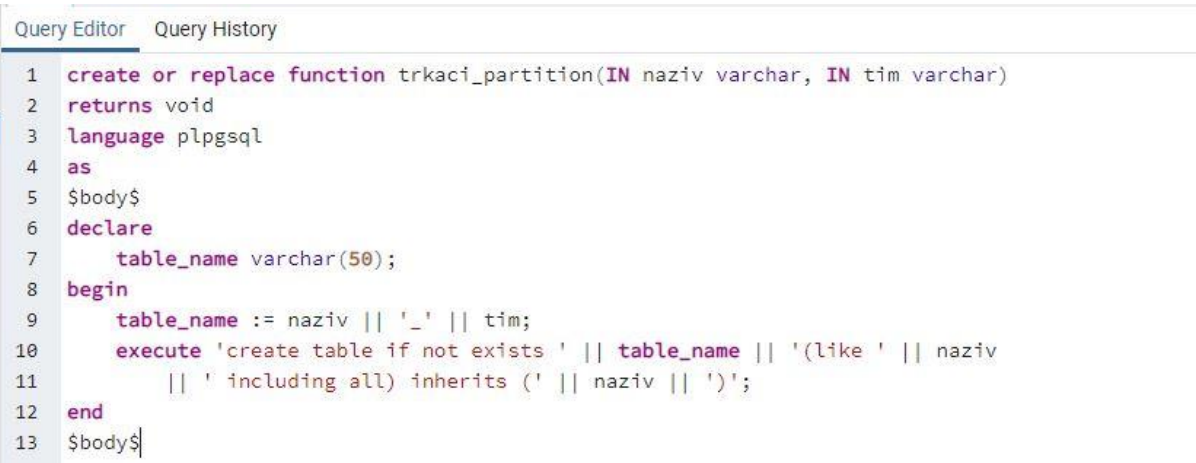
    context.SaveChanges();
}
```

slika 2.2 - Brisanje podatka iz baze podataka

Dan 7 – Partitionisanje nasleđivanjem

Za razliku od deklarativnog partitionisanja, za demonstraciju partitionisanja nasleđivanjem se vrši partitionisanje nad tabelom, koja je već popunjena podacima. Bilo je potrebno rešiti problem brzog porasta baze podataka, gde se javlja pogoršanje performansi. U ovom slučaju bi partitionisanje podataka moglo da ubrza upite, samim tim i poboljša performanse. Potrebno je bilo smisliti način na koji će se podaci smestiti u odgovarajuće particije, a ukoliko particija ne postoji, onda obezbediti njeno kreiranje.

Kreiranje particije obezbeđuje funkcija `trkaci_partition(naziv, tim)`, čiji argument “naziv” predstavlja naziv roditeljske tabele, a “tim” predstavlja novu particiju. Ovu funkciju poziva funkcija `trg_insert_trkaci()`.



```
Query Editor  Query History
1  create or replace function trkaci_partition(IN naziv varchar, IN tim varchar)
2  returns void
3  language plpgsql
4  as
5  $body$
6  declare
7      table_name varchar(50);
8  begin
9      table_name := naziv || '_' || tim;
10     execute 'create table if not exists ' || table_name || '(like ' || naziv
11         || ' including all) inherits (' || naziv || ')';
12 end
13 $body$
```

slika 2.3 - Implementacija funkcije `trkaci_partition(naziv, tim)`

Kako bismo obezbedili proveravanje postojanja particije svaki put kada se pokuša ubacivanje novog podatka u bazu, i samo kreiranje particije ukoliko ona ne postoji, kreirali smo funkciju `trg_insert_trkaci()`. Ova funkcija pokušava ubacivanje podatka u odgovarajuću particiju, a ukoliko particija ne postoji, poziva se funkcija `trkaci_partition(naziv, tim)`, zatim se podatak ubacuje u novokreiranu particiju.

```

1 create or replace function trg_insert_trkaci()
2 returns trigger
3 language plpgsql
4 as
5 $body$
6 declare
7     tim trkaci.naziv_tima%type := null;
8 begin
9     perform naziv_tima into tim
10    from trkaci;
11    execute 'insert into trkaci_' || new.naziv_tima || ' values ( $1.* )' using new;
12    return null;
13 exception
14 when undefined_table then
15     perform pg_advisory_xact_lock('trkaci'::regclass::oid::integer);
16     perform trkaci_partition('trkaci', new.naziv_tima);
17
18     execute 'insert into trkaci_' || new.naziv_tima || ' values ( $1.* )' using new;
19     return null;
20 end
21 $body$

```

slika 2.4 - Implementacija funkcije trg_insert_trkaci()

Na kraju treba kreirati i trigger row_trg_insert_trkaci, koji će se okidati pre svake INSERT naredbe i pozivati funkciju trg_insert_trkaci().

```

1 create trigger row_trg_insert_trkaci before insert
2   on trkaci for each row
3   execute procedure trg_insert_trkaci();

```

slika 2.5 - Implementacija trigger-a row_trg_insert_trkaci

Nakon ovog koraka, obezbeđeno je razvrstavanje novih podataka u adekvatne particije, a funkcija LoopForExisting() prolazi kroz već postojeće podatke tabele roditelja i raspoređuje ih u odgovarajuće particije.

```

1 CREATE OR REPLACE FUNCTION LoopForExisting()
2 RETURNS VOID
3 AS
4 $$
5 DECLARE
6     t_row trkaci%rowtype;
7 BEGIN
8     FOR t_row in SELECT * FROM trkaci LOOP
9         IF t_row.naziv_tima != 'UAE' and t_row.id_trkaca != 77 and t_row.id_trkaca != 78 then
10             perform trkaci_partition('trkaci', t_row.naziv_tima);
11             execute 'insert into trkaci_' || t_row.naziv_tima || ' values ( $1.* )' using t_row;
12         END IF;
13     END LOOP;
14 END;
15 $$
16 LANGUAGE plpgsql;

```

slika 2.6 - Implementacija funkcije LoopForExisting()

Zaključak

Praktičnom primenom postojećeg znanja, kao i novostečenog znanja, stekla sam iskustvo u oblasti particionisanja tabela baze podataka. Rad je bio bez pritiska i smešten u pogodan vremenski interval za obavljanje svih pojedinačnih zadataka. Mentor je uvek bio dostupan za sva dodatna pitanja i nejasnoće. Uz njegovu pomoć i veliku podršku, stekla sam nova znanja i proširila postojeća.