

ELEKTRONSKI FAKULTET NIŠ STRUČNA PRAKSA

**Izveštaj o realizaciji stručne**

**prakse**

Katarina Mladenović 16765

Niš, Oktobar 2021.

Sadržaj

[Uvod 3](#_Toc84342638)

[Izveštaj 4](#_Toc84342639)

[Zaključak 13](#_Toc84342640)

# 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 maper 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.

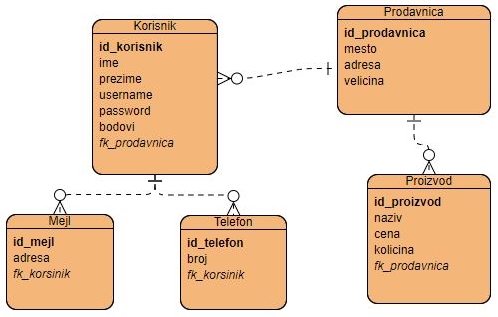
PostgreSQL nudi built-in podršku za sledeće oblike deklarativnog particionisanja:

* ***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.

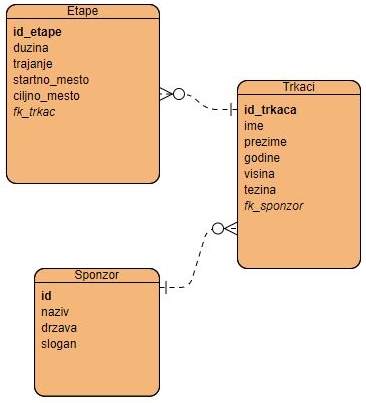
Iako je deklarativno particionisanje pogodno za najčešće slučajeve upotrebe, postoje neke okolnosti u kojima bi fleksibilniji pristup bio pogodan. Particionisanje se može implementirati pomoću nasleđivanja tabela, što dozvoljava nekoliko funkcionalnosti koje nisu podržane deklarativnim particionisanjem, 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.



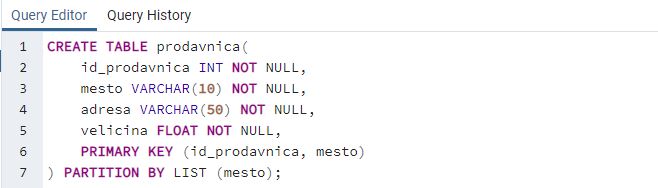
slika 1.0 - EER dijagram baze ,,Prodavnica''



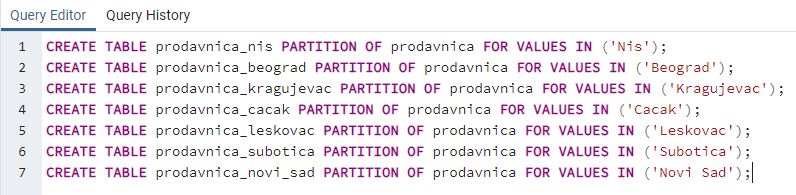
slika 1.1 - EER dijagram baze „Trkači’’

**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.



slika 1.2 - Kreiranje tabele

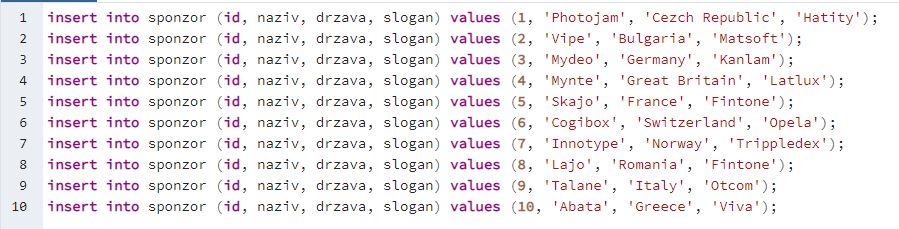


slika 1.3 - Kreiranje particija

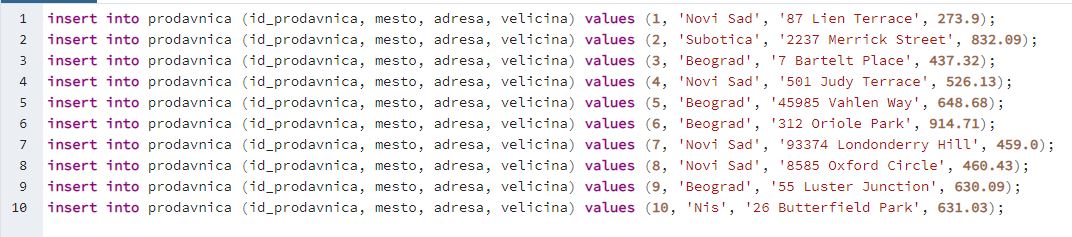
Za ostala dva tipa deklarativnog particionisanja, postupak je identičan.

**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.



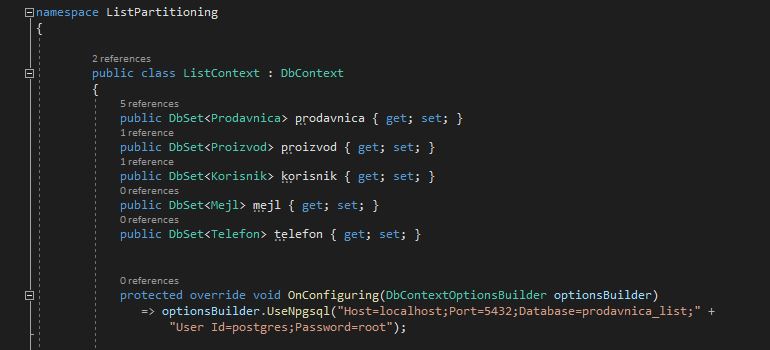
slika 1.4 - Insert into naredbe



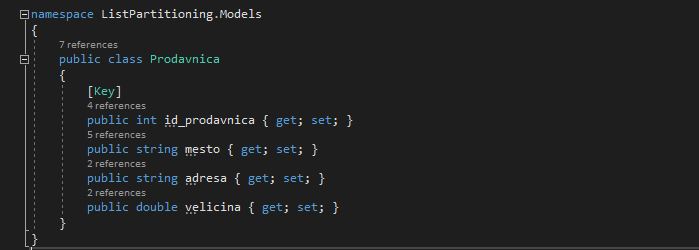
slika 1.5 - Insert into naredbe particionisane tabele

**Dan 5 – *Povezivanje sa PosgreSQL bazom podataka***

Za rad sa PostgreSQL bazom podataka u okviru .NET Core-a, korišćen je Entity Framework Core objektno-relacioni maper. 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.

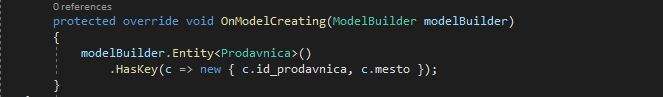


slika 1.6 - Context klasa



slika 1.7 - Model tabele

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

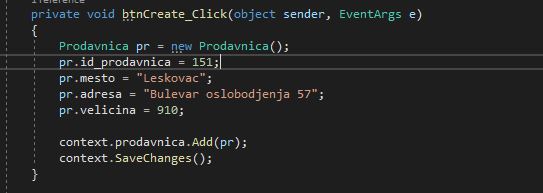


slika 1.8 - Dodatna ograničenja

**Dan 6 – *CRUD operacije***

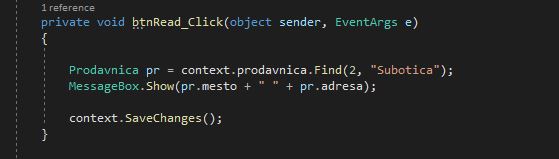
Entity Framework Core omogućava jednostavan i intuitivan rad sa PostgreSQL bazom podataka, koristeći EF Core metode, koje za nas u pozadini izvršavaju SQL upite nad bazom podataka.

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.



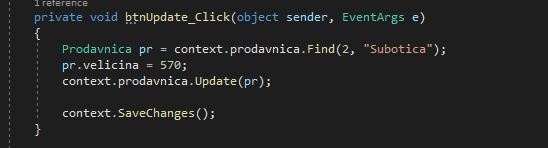
slika 1.9 - Dodavanje podatka u bazu

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



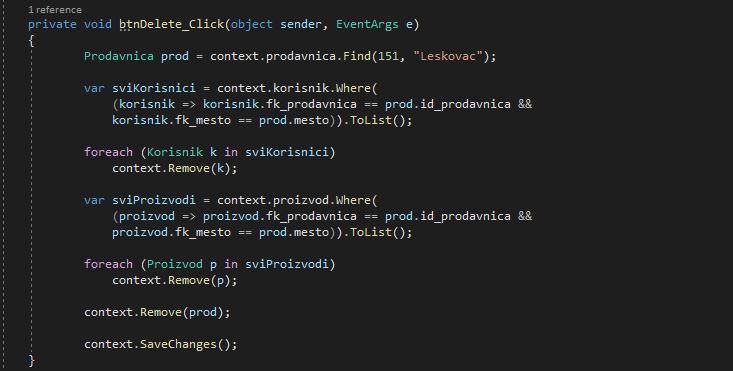
slika 2.0 - Čitanje podatka iz baze

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



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.

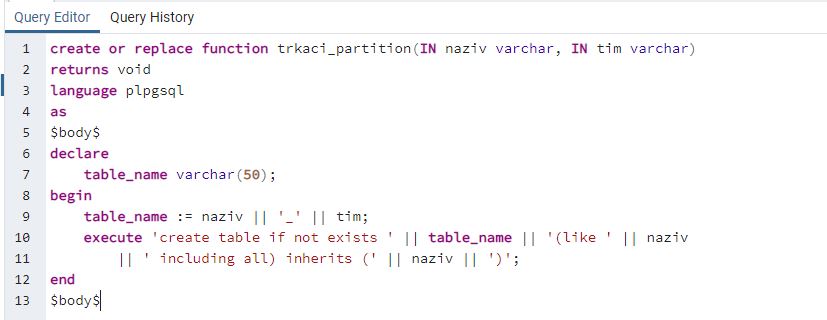


slika 2.2 - Brisanje podatka iz baze podataka

**Dan 7 – *Particionisanje nasleđivanjem***

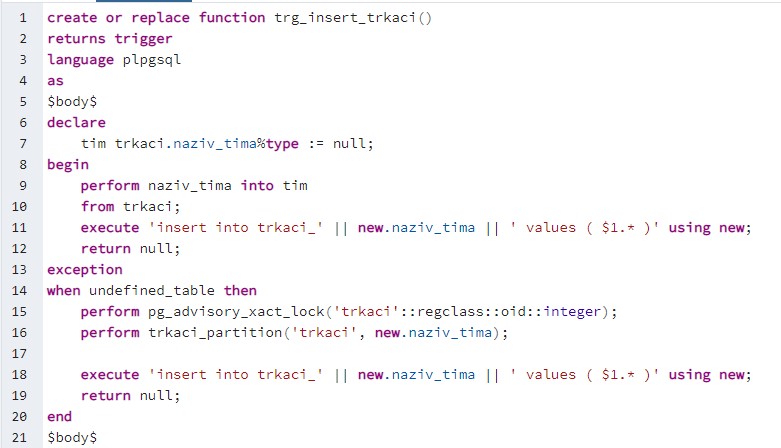
Za razliku od deklarativnog particionisanja, za demonstraciju particionisanja nasleđivanjem se vrši particionisanje nad tabelom, koja je već popunjena podacima. Bilo je potrebno rešiti problem brzog porasta baze podataka, gde se javlja pogoršanje performansi. U ovom slučaju bi particionisanje podataka 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().



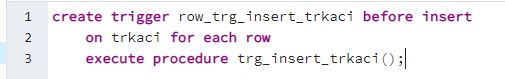
slika 2.3 - Implementacija funkcije trkaci\_partition(naziv, tim)

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.



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().



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.



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.