|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NIŠU  ELEKTRONSKI FAKULTET |  |

Miljan Denić

**OPTIMIZACIJA UPITA KOD POSTGRESQL BAZE PODATAKA**

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

|  |
| --- |
| Kandidat: |
|  |
| Miljan Denić, br. ind. 1517 |

Niš, april 2023. godina

SADRžaj

[1. uvod 3](#_Toc132092633)

[2. Šta je i čemu služi Plan upita 4](#_Toc132092634)

[2.1. Struktura plana upita 4](#_Toc132092635)

[2.2. Primeri primene EXPLAIN komande za analizu upita 6](#_Toc132092636)

[2.3. ANALYZE komanda 8](#_Toc132092637)

[2.4. VACUUM komanda 10](#_Toc132092638)

[3. Genetski optimizator upita 10](#_Toc132092639)

[3.1. Genetski algoritam 11](#_Toc132092640)

[3.2. Način rada GEQO u PosgreSQL bazi podataka 12](#_Toc132092641)

[3.3. Podešavanje parametara genetskog optimizatora upita 12](#_Toc132092642)

[3.4. Primer korišćenja genetskog optimizatora upita 13](#_Toc132092643)

[4. Optimizacija upisa u bazu podataka 15](#_Toc132092644)

[4.1. Isključivanje AUTOCOMMIT opcije 15](#_Toc132092645)

[4.2. Korišćenje COPY i PREPARE opcije 15](#_Toc132092646)

[4.3. Uklanjanje indeksa 16](#_Toc132092647)

[4.4. Brisanje stranih ključeva 16](#_Toc132092648)

[4.5. Povećanje maintenence\_wal\_size promenljive 16](#_Toc132092649)

[4.6. Povećanje max\_wal\_size promenljive 17](#_Toc132092650)

[4.7. Isključenje WAL arhiviranja i strimovanja aplikacije 17](#_Toc132092651)

[4.8. Pokretanje ANALYZE komande nakon upisa 18](#_Toc132092652)

[5. Zaključak 19](#_Toc132092653)

[6. Literatura 20](#_Toc132092654)

# uvod

PostgreSQL je open-source sistem za upravljanje bazama podataka koji se koristi za razvoj i upravljanje bazama podataka. Prva verzija je objavljena 1989. godine i od tada je postao jedna od najpopularnijih baza podataka na svetu. PostgreSQL podržava širok spektar tipova podataka polazeći od onih osnovnih pa do složenih kao što su nizovi, point, JSON i slični. Jedna od glavnih funkcionalnosti jeste i proširivost. PostgreSQL omogućava širok spektar alata za kreiranje različitih funkcija i operatora koji mogu biti iskorišćeni za kreiranje različitih tipova podataka kao i za manipulaciju podacima na različitim načinima. PostgreSQL je poznat po svojoj stabilnosti i pouzdanosti, kao i po mogućnostima u upravljanju velikim skupom podataka. Pruža podršku za kreiranje bekapova i oporavak baze, replikaciju, klasterizaciju što garantuje visoku dostupnost i skalabilnost aplikacije. Podržava rad na različitim operativnim sistemima uključujući Linux, macOS i Windows.

Optimizacija upita je proces koji uključuje modifikovanje strukture upita kako bi se unapredila efikasnost i brzina. Glavni cilj optimizacije upita je da se smanji vreme izvršenja upita koje ima veliki uticaj na performanse aplikacije ili sistema. Jedan od glavnih faktora koji mogu da utiču na performanse izvršenja upita jeste i količina podataka koja se povlači. Optimizacija upita uključuje pronalaženje najefikasnijeg načina za povlačenje podataka iz baze selektovanjem najboljih indeksa, najoptimalnijim organizovanjem tabela kao i biranjem najefikasnijih algoritama za obradu podataka. Optimizacija upita u PostgreSQL bazi podataka predstavlja važnu temu u oblasti baza podataka, posebno u kontekstu velike količine podataka i potrebe za brzim odgovorom na upite korisnika.

Cilj ovog rada je da pruži pregled različitih strategija za optimizaciju upita u PostgreSQL-u, kao i najboljih praksi za postizanje optimalnih performansi. U radu će se razmotriti kako optimizacija upita utiče na performanse PostgreSQL baze podataka i kako se može primeniti u praksi. Posebno će se istaći uloga indeksa i njihov uticaj na optimizaciju upita, kao i značaj konfiguracionih parametara za postizanje optimalnih performansi.

Rad će se sastojati od pet delova. Nakon uvoda, sledi analiza plana upita u PostgreSQL-u, gde će se detaljno objasniti šta je to plan upita i zašto je važan. U trećem delu, diskutovaće se o genetskom optimizatoru upita, kao i o načinu njegovog rada. U četvrtom delu, opisaće se najbolje prakse za postizanje optimalnih performansi kod upisa podataka i kako se one mogu primeniti u praksi. U završnom delu, dati će se zaključak i predlozi za dalja istraživanja.

Ovaj rad će pomoći čitaocu da bolje razume proces optimizacije upita u PostgreSQL-u i da stekne uvid u različite strategije i najbolje prakse koje se mogu primeniti za postizanje optimalnih performansi. Za izvršavanje upita će se koristiti baza na PostgreSQL serveru verzije 15.2.

# Šta je i čemu služi Plan upita

U PostgreSQL bazama plan upita predstavlja detaljno objašnjenje kako će baza podataka izvršiti određeni SQL upit. Plan upita opisuje različite korake koje će baza podataka primeniti da bi izvršila upit i dobila željene rezultate.

Plan upita je generisan od strane PostgreSQL planera upita koji je odgovoran za analizu upita i određivanje najefikasnijeg načina za povlačenje zahtevanih podataka. Planer upita uzima u obzir razne faktore prilikom generisanja plana upita, uključujući strukturu upita, podatke koji se zahtevaju, postojeće indekse i druge opcije za optimizaciju.

Plan upita se može videti koristeći EXPLAIN komandu u PostgreSQL-u. EXPLAIN komanda dozvoljava da se vide koraci koje će baza podataka primeniti kako bi izvršila upit, uključujući redosled u kojem će tabele biti skenirane, koji tipovi join operacija će biti korišćeni kao i bilo koja operacija sortiranja ili filtriranja koja će biti izvršena.

Razumevanje plana upita je bitno za optimizaciju upita jer omogućava da identifikujemo potencijalna uska grla i neefikasnosti u procesu izvršenja upita. Analizom plana upita i menjanjem upita ili šeme baze podataka mogu se poboljšati performanse upita i smanjiti količina vremena neophodna za povlačenje podataka iz baze podataka.[1]

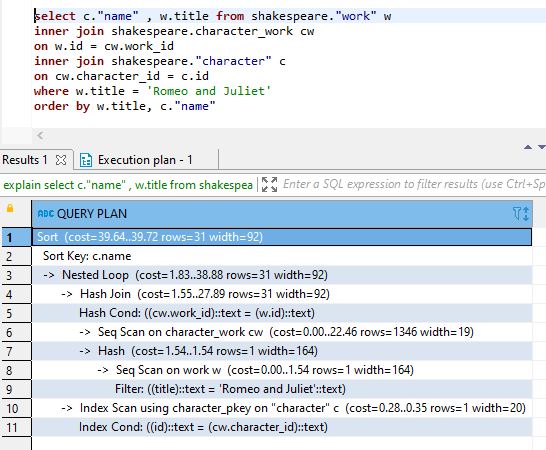
## Struktura plana upita

Struktura plana upita je stablo čvorova izvršenja. Čvorovi na najnižem novou stabla su čvorovi skeniranja koji vraćaju neobrađene redove iz tabele. Postoji više različitih vrsta skenirajućih čvorova za različite načine pristupa tabelama.

* Sekvencijalno skeniranje
* Indeksno skeniranje
* Bitmap indeksno skeniranje

Ako upit zahteva spajanje tabela, agregaciju, sortiranje ili neku drugu operaciju nad redovima iz tabele onda će postojati dodatni čvorovi iznad skenirajućih čvorova koji će predstavljati ove operacije. Svakako postoji više načina da se izvrše ove operacije tako da se različite vrste čvorova mogu ovde pojaviti.

Povratna vrednost EXPLAIN komande ima jednu liniju za svaki čvor u stablu plana koja prikazuje osnovni tip čvora sa procenjenim troškovima koje je planer napravio za izvršenje tog plana. Uvučene dodatne linije se mogu pojaviti iznad linije koja prikazuje sumirane podatke o čvoru i te linije prikazuju dodatne osobine čvora. Prva od svih linija ima procenjene ukupne troškove izvršenja plana. Ove vrednosti planer pokušava da umanji.



Slika 1. Primer izgleda plana upita

Na slici 1. imamo prikazan primer plana izvršenja jednog upita. Ovaj plan izvršenja će biti iskorišćen za objašnjenje prikazanih vrednosti svakog reda u zagradama (s leva na desno).

* **Procenjena početna cena**: Vreme potrošeno pre početka izlazne faze(npr. vreme potrošeno na sortiranje u čvoru za sortiranje)
* **Procenjena ukupna cena**: Prestavlja cenu pod pretpostavkom da je čvor plana izvršen do samog kraja, odnosno da su preuzeti svi dostupni redovi. U praksi to može značiti da je roditeljski čvor prestao da čita sve dostupne redove (primer komande LIMIT).
* **Procenjeni broj redova koji vraća čvor plana upita**: Takođe i kod ove vrednosti se pretpostavlja da je čvor izvršen do samog kraja.
* **Procenjena prosečna širina redova koje vraća čvor plana upita**: izražava se u bajtovima

Vrednosti su izražene u proizvoljnim jedinicama koje su određene parametrima troškova planera. Praksa je da se merenje troškova izražava u jedinicama preuzimanja stranica na disku(„seq\_page\_cost“) čija je vrednost postavljena na 1.0 i ostali parametri troškova su postavljeni u odnosu na ovu vrednost. Primeri koji će biti ovde prikazani će biti izraženi sa podrazumevanim vrednostima. Bitno napomenuti jeste da trošak čvora uključuje i trošak svih čvorova potomka. Takođe, trošak ne uključuje vreme potrebno za prenošenje rezultata upita sa servera na klijentsku stranu.

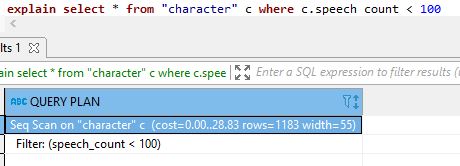
## Primeri primene EXPLAIN komande za analizu upita

**Primer sekvencijalnog skeniranja:**

Sekvencijalno skeniranje je jedan od načina pristupa podacima u PostgreSQL bazi podataka. Ova tehnika se obično koristi kada je potrebno da se pristupi većem broju redova unutar tabele, a nema potrebe za pretragom korišćenjem uslova ili sortiranjem. Sekvencijalno skeniranje podrazumeva prolazak kroz sve redove tabele, jedan po jedan, u određenom redosledu.

Princip rada sekvencijalnog skeniranja je relativno jednostavan. Kada se izvršava SQL upit koji zahteva sekvencijalno skeniranje, PostgreSQL baza podataka čita podatke sa diska i smešta ih u memoriju za keširanje. Nakon toga, baza podataka prolazi kroz sve redove tabele, počevši od prvog reda, i preuzima podatke iz svakog reda. Ovi podaci se potom obrađuju prema uslovima navedenim u SQL upitu.

Sekvencijalno skeniranje može biti vrlo efikasno za velike tabele koje ne zahtevaju složenu obradu podataka. Međutim, ovaj pristup nije preporučljiv za tabele koje imaju mnogo redova, jer bi to moglo da dovede do preopterećenja baze podataka i da značajno uspori izvršavanje upita.



Slika 2. Primer plana upita sa sekvencijalnim skeniranjem

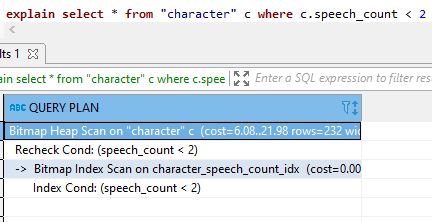
Na slici 2 imamo upit koji pretražuje tabelu karaktera i iz nje povlači sve redove gde je broj govora manji od 100. Samim tim što će ovaj upit obuhvatiti veći deo podataka iz tabele planer primenjuje sekvencijalno skeniranje jer je neophodno obići svaki red i proveriti da li se vrednosti broja govora uklapa u navedeni filter. Ovakav upit će čak imati veću cenu izvršenja od upita gde ne bi postojao nikakav uslov jer pored toga što planer mora da skenira svaki red potrošiće se dodatno procesorsko vreme na proveravanje WHERE uslova.

**Primer bitmap indeksnog skeniranja:**

Bitmap indeksno skeniranje predstavlja jednu od najefikasnijih tehnika pretraživanja velikih skupova podataka u PostgreSQL bazi podataka. Ovaj pristup se koristi kada su pretrage nad velikim skupovima podataka izrazito skupe i kada je potrebno poboljšati performanse upita.

Bitmap indeksno skeniranje koristi nekoliko različitih bitmapa kako bi se omogućilo brzo i efikasno pretraživanje velikog broja podataka. Svaki bit u bitmapi predstavlja jedan od mogućih rezultata pretrage i zavisno od toga da li je taj bit postavljen na 1 ili 0, pretraživanje se nastavlja dalje ili se prekida.

Prednost bitmap indeksnog skeniranja je što se može koristiti za pretraživanje bilo kog tipa podataka, uključujući i velike skupove sličnih vrednosti. Osim toga, ova tehnika omogućava izvođenje različitih operacija nad podacima, kao što su join i agregacije i zbog toga je korisna u različitim vrstama upita. Važno je napomenuti da bitmap indeksno skeniranje ima određene nedostatke. Na primer, kada su podaci vrlo retki, bitmap indeksi mogu biti vrlo veliki, što može negativno uticati na performanse upita.



Slika 3. Primer plana upita sa bitmap indeksnim skeniranjem

Na slici 3 imamo upit koji pretražuje tabelu karaktera i iz nje povlači sve redove gde je broj govora manji od dva. U ovom slučaju upit će obuhvatiti manji deo podataka iz tabele gde je kolona broj govora indeksirana i planer primenjuje bitmap indeksno skeniranje. Izršenje upita će se izvršiti iz dva koraka. U prvom koraku čvor potomak obilazi koristi indeks da pronađe lokacije redova koji zadovoljavaju uslov, a u drugom koraku čvor iznad zapravo povlači redove iz tabele. Povlačenje redova pojedinačno je skuplje nego čitanje sekvencijalno ali iz razloga zato što nema potrebe za obilaženjem svih stranica tabele i dalje je jeftinije nego sekvencijalno skeniranje.

Razlog korišćenja izvršenja upita u dva koraka je to što viši čvorovi planiranja sortiraju lokacije redova identifikovane po indeksu u fizički sortirani redosled pre čitanja što smanjuje trošak pojedinačnih čitanja. Bitmap mehanizam zapravo je zadužen za taj posao.

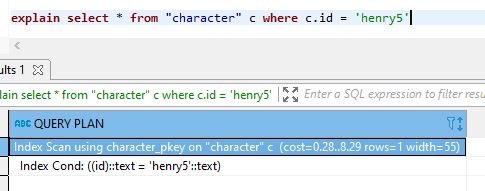
**Primer indeksnog skeniranja:**

Indeksno skeniranje u PostgreSQL bazi podataka predstavlja jedan od načina na koji se efikasno mogu pretraživati velike količine podataka u bazi. Indeks se koristi za brzo pronalaženje redova u tabeli koji zadovoljavaju uslove upita, umesto da se cela tabela prođe sekvencijalno. Indeksi se mogu kreirati na jednoj ili više kolona, a svaki indeks ima jedinstveno ime.

Indeksi se obično grade na kolonama koje se često koriste u WHERE uslovima i JOIN operacijama. Kada se indeks kreira, PostgreSQL koristi B-stabla (B-tree) ili GIN (Generalized Inverted Index) strukturu podataka. B-stabla se koriste za pretraživanje po vrednosti jedne ili više kolona, dok se GIN koristi za pretraživanje po skupovima vrednosti u jednoj koloni, kao što su nizovi ili JSON objekti.

Kada se upit prosledi bazi podataka, PostgreSQL prvo proverava da li postoji odgovarajući indeks koji bi mogao da se koristi. Ako postoji, baza podataka će koristiti indeks umesto sekvencijalnog skeniranja tabele, što može značajno poboljšati performanse upita. Međutim, indeksiranje ima svoje troškove, kao što su povećanje veličine baze podataka, smanjenje performansi prilikom upisa novih redova u tabelu i troškovi održavanja indeksa. Kako možemo smanjiti neke od ovih troškova će biti obrađeno u nastavku rada.

Kao i kod drugih vrsta skeniranja, indeksno skeniranje takođe može biti optimizovano korišćenjem genetskog optimizatora upita koji će takođe biti obrađen u nastavku rada.



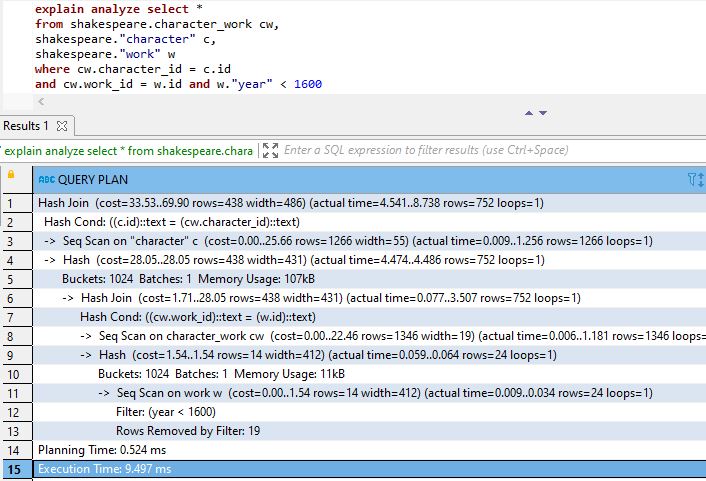
Slika 4. Primer plana upita sa indeksnim skeniranjem

Na slici 4 imamo upit koji pretražuje tabelu karaktera i iz nje povlači sve redove gde id ima vrednost „henry5“. U ovom slučaju planer će odabrati obično indeksno skeniranje. Kod ovog tipa skeniranja redovi se povlače iz tabele indeksno sortirani što ih čini još skupljim za čitanje ali postoji samo jedan red tako da je trošak zanemarljiv. Ovaj način skeniranja planer će obično odabrati za upite gde se povlači samo jedan red. Takođe, često je korišćen za upite koji imaju ORDER BY naredbu koja se poklapa sa indeksim redosledom, zato što onda nema potrebe za dodatnim korakom sortiranja koji bi zadovoljio ORDER BY uslov. U ovom primeru dodavanjem ORDER BY id bi se i dalje koristio isti plan zato što indeks već implicitno obezbeđuje zadato sortiranje.

## ANALYZE komanda

Analiza u PostgreSQL-u se koristi za izračunavanje statistika o tabelama, a rezultati se skladište u PostgreSQL-ovoj tabeli koja će biti korišćena od strane planera upita za bolje planiranje izvršenja upita. Kada se upit pošalje, on se prvo transformiše u PostgreSQL format. Zatim planer upita generiše plan izvršenja upita. [2]

Planer koristi statistike tabela kako bi formirao plan izvršenja. Međutim, tokom vremena, statistike postaju zastarele i plan izvršenja može biti neprecizan ili neefikasan. Zato je važno obezbediti da se statistike uvek ažuriraju. Upravo u ovom kontekstu ANALYZE komanda ima svoj značaj. Kada se analiza pokrene, komanda ponovo računa statistike za datu tabelu i ažurira ih u bazi podataka. Te statistike će biti korišćene za planiranje izvršenja upita u budućnosti.



Slika 5. Primer upita sa ANALYZE komandom

Važno je napomenuti da su vrednosti **actual time** izražene u milisekundama stvarnog vremena, dok su procene troškova izražene u proizvoljnim jedinicama, i zbog toga je mala verovatnoća podudaranja. Bitno je da se procenjene vrednosti broja redova približe stvarnosti. U ovom primeru, procene su bile tačne, ali to se ne dešava tako često u praksi. U nekim planovima upita, moguće je da se podplan čvor izvrši više od jednom. U takvim slučajevima, vrednost petlje prikazuje ukupan broj izvršavanja čvora, a prikazane stvarne vrednosti vremena i broja redova su prosečne vrednosti po izvršenju. Ovo je urađeno da bi se brojevi mogli uporediti sa prikazom procene troškova. U nekim slučajevima, EXPLAIN ANALYZE prikazuje dodatne statistike izvršavanja osim vremena izvršavanja plana čvorova i broja redova. Na primer, broj eliminisanih redova na osnovu filter uslova. Ove statistike su posebno korisne za filter uslove primenjene na čvorovima spajanja (join). Linija **Rows Removed** se pojavljuje samo kada se barem jedan skenirani red eliminiše filter uslovom.

EXPLAIN komanda se može koristiti sa BUFFERS opcijom kako bi se dobila još detaljnija statistika. Broj koji BUFFERS vrati pomaže da se utvrdi koji delovi upita su najviše ulazno-izlazno zahtevni.

**Vreme planiranja** koje se prikazuje predstavlja vreme koje je utrošeno na generisanje i optimizaciju plana upita iz parsiranog upita i ne uključuje vreme potrebno za parsiranje upita.

**Vreme izvršavanja** koje se prikazuje uključuje vreme pokretanja i isključivanja egzekutora, kao i vreme potrebno za pokretanje bilo kojih okidača koji se aktiviraju tokom izvršavanja upita. Vreme utrošeno za izvršenje BEFORE okidača uključeno je u vreme za pripadajući čvor za unos, a vreme utrošeno za izvršavanje AFTER okidača nije zabeleženo, jer se okidači AFTER aktiviraju nakon završetka celog plana. Ukupno vreme provedeno u svakom okidaču (BEFORE ili AFTER) prikazano je zasebno.

Važno je napomenuti da EXPLAIN ANALYZE zapravo pokreće upit i da će upit stvarno izvršiti, bez obzira na to što će se rezultati upita neće prikazati zbog ispisa EXPLAIN komande. Ako je potrebno analizirati upit bez menjanja tabela moguće je odraditi ROLLBACK na kraju upita.

## VACUUM komanda

U PostgreSQL-u, kada se vrši ažuriranje ili brisanje reda ili tuple-a, zapisi se zapravo ne brišu ili ne menjaju fizički. Ovo ostavlja zastarele zapise na disku koji zauzimaju prostor i negativno utiču na performanse upita. Kako bi se to poboljšalo, PostgreSQL pruža mogućnost čišćenja zastarelih zapisa sa diska korišćenjem komande VACUUM koja omogućava jednostavno uklanjanje zastarelih zapisa sa diska i oslobađanje prostora, poboljšavajući performanse upita.

Komanda VACUUM dolazi sa mnogo opcija. Na primer, može se koristiti komanda VACUUM ANALYZE kako bi se koristeći VACUUM prvo fizički izbrisali zastareli zapisi, a zatim komandom ANALYZE osvežila statistika tabela za planer upita kako bi bolje planirao buduće upite na toj tabeli.

Druga opcija koja se može iskoristiti je VACUUM FULL komanda. Osnovna VACUUM komanda će izbrisati zapise iz tabele i zadržati novi oslobođeni prostor na disku sa samom tabelom za buduću upotrebu kada tabela bude rasla. Ovo neće osloboditi prostor na disku za operativni sistem. Međutim, upotreba opcije FULL sa komandom će zapravo prepisati kompletnu tabelu u novu datoteku na disku; oslobođeni prostor na disku će biti vraćen operativnom sistemu za korišćenje od strane drugih procesa. Zbog toga se VACUUM FULL ne može koristiti paralelno sa bilo kojom drugom operacijom čitanja ili pisanja na tabeli. Samo VACUUM komanda može se pokrenuti paralelno.

S obzirom na važnost procesa VACUUM, PostgreSQL dolazi sa automatskom verzijom, koja se naziva autovacuum. Autovacuum proces se sastoji od različitih modula koji rade zajedno kako bi automatski pokrenuli VACUUM komandu u redovnim intervalima kako bi se osiguralo da su svi zastareli zapisi izbrisani iz tabela i da se upiti izvršavaju bolje.

Kao i sa bilo kojim drugim komponentama PostgreSQL-a, autovacuum proces se može konfigurisati prema poslovnim potrebama. Na ovaj način, izvršenje se može zakazati u vreme kada ima najmanje operacija nad tabelama kako paralelna izvršenja ne bi imala uticaj na tabele.

Administratori baza podataka mogu pokrenuti VACUUM komandu sa bilo kojom opcijom u bilo koje vreme ad hoc na određenim tabelama, a takođe i na celoj bazi podataka.

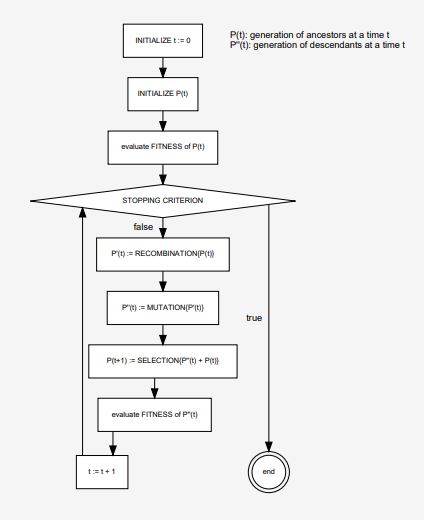
# Genetski optimizator upita

Genetski optimizator upita (GEQO – the genetic query optimizer) je algoritam koji vrši planiranje upita koristeći heurističku pretragu. To značajno smanjuje planirano vreme za složene upite tj. one upite koji imaju puno pridruženih tabela ali na račun izrade planova koji su ponekad inferiorniji u odnosu na one koji su pronađeni korišćenjem standardnog algoritma za pretraživanje.

Među svim relacijskim operatorima, najteži za obradu i optimizaciju je join operator. Broj mogućih planova upita eksponencijalno raste sa brojem joinova u upitu. Dodatni napor za optimizaciju uzrokuje podrška različitim metodama join operatora (npr. ugnežđena petlja, hash join, merge join u PostgreSQL-u) za obradu pojedinačnih joinova i raznolikost indeksa (npr. B-stablo, hash, GiST i GIN u PostgreSQL-u) kao putova pristupa relacijama. Normalni PostgreSQL optimizator upita proizvodi gotovo optimalan redosled joinova, ali može da zahteva ogromnu količinu vremena i memorijskog prostora kada broj joinova u upitu postane veliki. To čini obični PostgreSQL optimizator upita neoptimalnim za upite koji spajaju veliki broj tabela i zato je u tim situacijama praktična primena genetskog algoritma.

## Genetski algoritam

Genetski algoritam (GA) je heuristička optimizaciona metoda koja funkcioniše kroz nasumičnu pretragu. Skup mogućih rešenja za optimizacioni problem se posmatra kao populacija jedinki. Stepen prilagođenosti jedinke svom okruženju se definiše njenom adekvatnošću (eng. fitness). Koordinate jedinke u prostoru pretrage se predstavljaju hromozomima, u suštini skupom karakterističnih nizova. Gen predstavlja podsekciju hromozoma koja kodira vrednost jednog parametra koji se optimizuje. Tipična kodiranja za gen mogu biti binarna ili celobrojna. Kroz simulaciju evolutivnih operacija rekombinacije, mutacije i selekcije pronalaze se nove generacije tačaka pretrage koje pokazuju veću prosečnu adekvatnost od svojih prethodnika.



Slika 6. Genetski algoritam

## Način rada GEQO u PosgreSQL bazi podataka

Genetski otimizator upita pristupa problemu optimizacije upita po principu problema putujućeg prodavca (TSP). Mogući planovi se kodiraju kao celobrojni nizovi. Svaki niz predstavlja redosled spajanja jedne veze upita na sledeću. Na primer (slika 7), stablo spajanja kodira se celobrojnim nizom '4-1-3-2', što znači da se prvo spaja veza '4' i '1', zatim '3' i na kraju '2', pri čemu su 1, 2, 3, 4 identifikatori veza u okviru optimizatora PostgreSQL-a.



Slika 7. Primer stabla spajanja

Genetski otimizator upita koristi standardni kod planera za generisanje planova za skeniranje pojedinačnih relacija. Nakon toga, planovi spajanja se razvijaju upotrebom genetskog pristupa. U početnoj fazi, genetski otimizator upita generiše nekoliko mogućih nizova spajanja nasumično. Za svaki niz spajanja, standardni kod planera se poziva da proceni trošak izvršavanja upita upotrebom tog niza spajanja. Nizovi spajanja sa nižim procenjenim troškom smatraju se "prikladnijim" od onih sa višim troškom. Genetski algoritam odbacuje najneoptimalnije kandidate. Zatim se novi kandidati generišu kombinovanjem gena prikladnijih kandidata, odnosno upotrebom nasumično odabranih delova nizova sa nižim troškom spajanja radi stvaranja novih nizova za razmatranje. Ovaj proces se ponavlja sve dok se ne razmotri unapred postavljena količina nizova spajanja, a zatim se najbolji pronađeni niz koristi za generisanje konačnog plana.

Kako bi se izbegle neočekivane promene izabranog plana, svako pokretanje genetskog optimizatora upita ponovo pokreće svoj generator nasumičnih brojeva sa trenutnim podešavanjem parametra geqo\_seed. Sve dok se geqo\_seed i ostali parametri GEQO drže fiksnim, za dati upit (i druge ulaze planera poput statistike) generisaće se isti plan. Kako bi se eksperimentisalo sa različitim putevima pretrage, može se pokušati promeniti geqo\_seed.

## Podešavanje parametara genetskog optimizatora upita

Promenljiva **geqo** se koristi za uključivanje ili isključivanje GEQO algoritma. Po defaultu je ova opcija uključena.

GEQO se automatski aktivira kada je broj tabela koje se povezuju u upitu veći od određenog praga, koji se može konfigurisati kroz parametre konfiguracije baze podataka. Podrazumevana vrednost ovog praga je 12. To znači da će GEQO biti aktiviran samo ako se u upitu koristi 13 ili više tabela. Međutim, ova vrednost može biti promenjena konfigurisanjem parametra **geqo\_threshold**

Parametar **geqo\_effort** kontroliše odnos između vremena potrebnog za planiranje upita i kvaliteta plana upita koji se generiše GEQO algoritmom. Vrednosti ovog parametra kreću se u opsegu od 1 do 10, a podrazumevana vrednost je pet. Veće vrednosti ovog parametra povećavaju vreme planiranja upita, ali istovremeno povećavaju verovatnoću izbora efikasnog plana upita.

Parametri **geqo\_pool\_size** i **geqo\_generations** kontrolišu veličinu populacije i broj iteracija algoritma. geqo\_pool\_size određuje broj jedinki u populaciji i mora biti veći od dva, a korisne vrednosti se obično kreću u opsegu od 100 do 1000. Ako je postavljena na nulu (podrazumevana vrednost), odgovarajuća vrednost se bira na osnovu parametra geqo\_effort i broja tabela u upitu. geqo\_generations određuje broj iteracija algoritma i mora biti veći od jedan. Korisne vrednosti su u istom opsegu kao i za geqo\_pool\_size. Ako je postavljena na nulu, odgovarajuća vrednost se bira na osnovu parametra "geqo\_pool\_size".

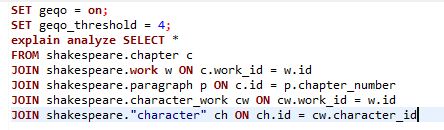
Parametar **geqo\_selection\_bias** kontroliše selektivni pritisak unutar populacije. Vrednosti mogu biti u opsegu od 1,50 do 2,00, a podrazumevana vrednost je 2,00.

Parametar **geqo\_seed** kontroliše početnu vrednost generatora slučajnih brojeva koji se koristi za odabir slučajnih puteva kroz prost

## Primer korišćenja genetskog optimizatora upita

U daljem delu će biti prikazan primer plana upita generisanog primenom GEQO optimizatora i primer plana istog upita gde je GEQO isključen. Baza podataka koja se koristi u ovom radu ima ukupno 6 tabela pa nije moguće napisati upit koji bi pokazao kako GEQO vrši optimizaciju upita. Iz tog razloga će biti prikazano kako implicitno uključivanje GEQO za rad sa manjim brojem JOIN operacija utiče na performanse upita.

Na slici 7 imamo prikazan upit koji će se koristiti za navedenu analizu.



Slika 8. Upit za analizu rada GEQO

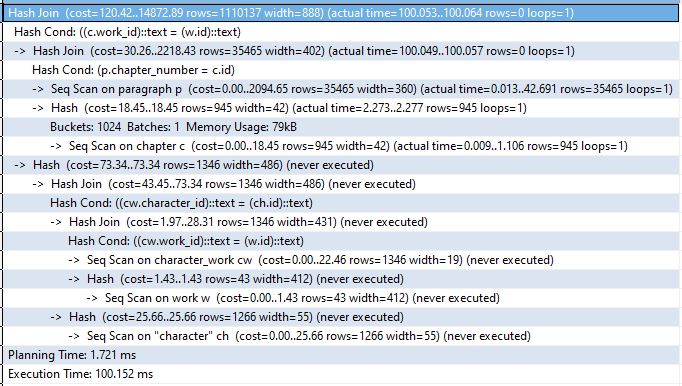
Na slici 8 je prikazan plan upita sa uključenim GEQO i sa smanjenim geqo\_threshold na 4 što znači da se GEQO primenjuje u ovom slučaju. Nakon toga je geqo parametar postavljen na vrenost „off“ što znači da je isljučen GEQO i plan takvog upita se može videti na slici 9. Primećuje se da je plan upita efikasniji u slučaju kada je GEQO isključen što je i bilo očekivano.

To se zaključuje posmatranjem cost parametra koji je znatno veći kod upita gde je GEQO uključen. Viša cena se opravdava time što PostgreSQL pokušava da primenom genetskog algoritma pronađe najefikasniji plan izvršavanja. Taj proces zahteva dodatno vreme izvršenja i to dovodi do povećanja ukupne cene upita. Takođe i vreme planiranja je znatno veće, a izabrani plan je u tom slučaju neefikasniji što se može videti na osnovu vremena planiranja i vremena izvršenja.

Na osnovu ovog primera može se zaključiti kao što je ranije i rečeno da za mali broj JOIN operatora nije efikasna primena genetskog optimizatora upita i da je u tom slučaju standardni PostgreSQL optimizator efikasniji. Vrednost genetskog optimizatora upita dolazi do izražaja u dosta složenijim upitima gde imamo dosta veći broj JOIN operacija.



Slika 9. Plan upita sa uključenim GEQO



Slika 10. Plan upita sa isključenim GEQO

# Optimizacija upisa u bazu podataka

Ponekad je potrebno uneti velike količine podataka u bazu podataka, npr. prilikom prvog popunjavanja baze podataka. Optimizacija u ovom slučaju može znatno uticati na performanse i brzinu izvršenja upita. U nastavku će biti predstavljeni neki od načina kako možemo ubrzati i optimizovati proces upisa podataka u bazu podataka.

## Isključivanje AUTOCOMMIT opcije

PostgreSQL baza podataka ima mogućnost automatskog potvrđivanja transakcija nakon svake pojedinačne SQL naredbe. Ova opcija se naziva autocommit i po inicijalno je uključena kod PostgreSQL baze. Iako autocommit opcija olakšava rad s bazom podataka ona takođe može dovesti do sporijeg upisa podataka.

Jedan od načina za optimizaciju upisa podataka u PostgreSQL bazu podataka jeste isključivanje autocommit opcije. Ovim postupkom transakcije se neće potvrđivati nakon svake pojedinačne SQL naredbe, već će se potvrđivati tek nakon završetka grupe povezanih SQL naredbi. Ova metoda optimizacije ima smisla u situacijama gde se velike količine podataka unose u bazu podataka, kao što je uvoz velikih datoteka ili migracija podataka iz drugih sistema. Isključivanjem autocommit opcije može se smanjiti broj potvrda transakcija i ubrzati proces upisa podataka.

Međutim, važno je napomenuti da se isključivanjem autocommit opcije takođe može povećati rizik od grešaka u transakcijama. Takođe, može se doći u situaciju da ukoliko se neka od INSERT naredbi nije uspešno izvršila da imamo delimično unete podatke. Iz tog razloga je preporučljivo isključiti autocommit opciju samo u situacijama kada je to apsolutno potrebno i proveriti transakcije pre potvrđivanja.

## Korišćenje COPY i PREPARE opcije

Kada je u pitanju optimizacija upisa podataka u PostgreSQL bazu podataka, jedna od opcija koja se često koristi je korišćenje COPY komande umesto serije INSERT naredbi. COPY komanda je efikasnija opcija za ubrzavanje upisa velikih količina podataka u bazu. Pošto je COPY jedna komanda nema potrebe za isključivanje autocommit opcije ako koristimo COPY komandu za upis podataka.

COPY komanda se koristi za brzo i efikasno ubacivanje podataka u tabelu iz eksternog izvora(npr. CSV fajl). Ova komanda preuzima fajl sa podacima i ubacuje ih u tabelu u bazi podataka. COPY komanda zahteva manje procesorske snage i manje resursa od serije INSERT naredbi, jer se podaci ubacuju u blokovima umesto redom.

Ako nije moguće koristiti COPY komandu optimizacija se može izvršiti i PREPARE komandom kako bi se kreirala pripremljena INSERT komanda i onda korišćenjem EXECUTE komande izvršila koliko god puta je potrebno. Na ovaj način smanjujemo troškove konstantnog parsiranja i planiranja INSERT naredbe. Treba napomenuti da je COPY komanda svakako brža od korišćenja kombinacije PREPARE i INSERT komandi.

## Uklanjanje indeksa

Ukoliko treba uneti podatke u tek kreiranu tabelu, najbrži metod jeste prvo kreirati tabelu, zatim upotrebiti COPY komandu za brzo učitavanje podataka u tabelu, a zatim kreirati sve indekse neophodne za tu tabelu. Kreiranje indeksa na postojećim podacima je brže nego ažuriranje podataka inkrementalno, pri svakom unosu novog reda.

Ukoliko se unosi velika količina podataka u postojeću tabelu, može biti korisno ukloniti indekse, učitati podatke, a zatim ponovo kreirati indekse. Naravno, tokom perioda u kojem indeksi nisu prisutni, performanse baze podataka mogu se smanjiti za druge korisnike. Takođe, treba dobro razmisliti pre brisanja UNIQUE indeksa, jer će provera grešaka koju pruža UNIQUE indeks biti onemogućena dok indeks ne bude ponovo kreiran.

## Brisanje stranih ključeva

Kao i sa indeksima, strani ključevi se proveravaju efikasnije kod bulk metoda nego kod unosa podataka red po red. Iz tog razloga može biti korisno ukloniti spoljne ključeve, učitati podatke i ponovo ih kreirati. Međutim, postoji loša strana ovog pristupa, a to je gubitak mogućnosti provere grešaka usled nedostatka stranih ključeva.

Takođe, kada učitavate podatke u tabelu sa postojanjem stranih ključeva, svaki novi red zahteva unos u listu čekanja okidača servera (jer okidanje okidača proverava ograničenje stranog ključa).

Učitavanje miliona redova može izazvati prekoračenje liste događaja okidača servera i dovesti do nemogućnosti zamene memorije ili čak do greške u izvršenju komande. Iz tog razloga može biti neophodno, a ne samo poželjno, ukloniti i ponovo primeniti strane ključeve prilikom učitavanja velikih količina podataka. Ako privremeno uklanjanje ograničenja nije prihvatljivo, jedino drugo rešenje može biti da se operacija učitavanja podeli na manje transakcije.

## Povećanje maintenence\_wal\_size promenljive

Maintenance\_work\_mem varijabla predstavlja konfiguracionu promenljivu u PostgreSQL bazi podataka koja se koristi za određivanje koliko memorije može biti upotrebljeno u toku rada različitih administrativnih operacija nad bazom podataka. Ova promenljiva utiče na performanse nekih važnih operacija, poput kreiranja indeksa i dodavanja stranih ključeva u postojeće tabele.

Ukoliko se u bazu podataka dodaju velike količine podataka, povećanje vrednosti maintenance\_work\_mem konfiguracione promenljive može znatno poboljšati performanse rada baze podataka. Konkretno, ovaj korak pomaže da se ubrzaju operacije kreiranja indeksa i dodavanja stranih ključeva, što se naročito odnosi na komande CREATE INDEX i ALTER TABLE ADD FOREIGN KEY. Međutim, ovo neće biti od koristi za operacije poput COPY komande koja se koristi za ubacivanje podataka u bazu.

Sa druge strane, treba biti oprezan prilikom povećanja vrednosti maintenance\_work\_mem konfiguracione promenljive, jer to može dovesti do povećanog zauzeća memorije na serveru. Ovo može negativno uticati na performanse drugih operacija koje se odvijaju na serveru, tako da se preporučuje pažljivo praćenje korišćenja ove promenljive u cilju postizanja optimalnih performansi u radu sa bazom podataka.

## Povećanje max\_wal\_size promenljive

Varijabla max\_wal\_size u PostgreSQL bazi podataka predstavlja maksimalnu veličinu Write-Ahead Log (WAL) fajlova koji se koriste za obezbeđivanje konzistentnosti podataka i oporavka od kvarova. Write-Ahead Log je mehanizam koji se koristi za čuvanje informacija o svim promenama koje se dešavaju u bazi podataka pre nego što se te promene trajno zapišu u tabelu na disku. Ovaj mehanizam obezbeđuje da se podaci ne izgube u slučaju pada sistema, a takođe omogućava i ponovnu primenu promena u slučaju neuspešne transakcije.

Kada se u PostgreSQL bazi podataka vrši veliko učitavanje podataka, to može da dovede do češćeg izvršavanja checkpoint operacija nego što je to uobičajeno. Ove operacije služe za snimanje svih nesnimljenih promena na disk i oslobađanje memorije koja se koristi za čuvanje promena. Ova operacija može da uspori proces učitavanja velikih količina podataka. Upravo zbog toga se preporučuje privremeno povećanje max\_wal\_size varijable kako bi se smanjio broj checkpoint operacija koje su neophodne tokom procesa učitavanja velikih količina podataka.

Privremeno povećanje max\_wal\_size varijable zahteva pažljivo planiranje, jer veće vrednosti ove varijable znače da će se koristiti više prostora na disku za čuvanje WAL fajlova. Međutim, pravilno podešavanje ove varijable može značajno ubrzati proces učitavanja velikih količina podataka u PostgreSQL bazu podataka.

## Isključenje WAL arhiviranja i strimovanja aplikacije

Kada se vrši upis velike količine podataka u PostgreSQL bazi podataka, korišćenje WAL arhiviranja ili strimovanje replikacije može značajno usporiti proces. WAL arhiviranje podrazumeva upisivanje svih promena koje se dešavaju u bazi podataka u posebnu datoteku, dok strimovana replikacija znači sinhronizovanje promena koje se dešavaju na primarnom serveru sa svim pridruženim replikama. Međutim, prilikom velikih opterećenja, ova funkcionalnost može usporiti proces, jer se veliki broj promena zapisuje u WAL datoteku, što dovodi do potrebe za čestim arhiviranjem ili slanjem ovih datoteka.

Da bi se smanjilo opterećenje, moguće je privremeno onemogućiti arhiviranje i strimovanu replikaciju. To se može postići podešavanjem parametara wal\_level na minimal, archive\_mode na off i max\_wal\_senders na nulu. Ovo će smanjiti broj upisivanja u WAL datoteku, ali treba imati u vidu da promena ovih podešavanja zahteva restartovanje servera i čini da su sve prethodno kreirane backup datoteke nedostupne za oporavak ili primarni server, što može dovesti do gubitka podataka. Pored ubrzanja procesa zbog manjeg broja upisivanja u WAL datoteku, ovo takođe može učiniti određene naredbe bržim, jer ne moraju pisati u WAL datoteke ukoliko je wal\_level minimalan.

## Pokretanje ANALYZE komande nakon upisa

Kada postoji znatna izmena distribucije podataka u tabeli, preporučljivo je pokreneti ANALYZE komandu nakon te izmene. Ovo uključuje učitavanje velikih količina podataka u tabelu. Pokretanje ANALYZE (ili VACUUM ANALYZE) osigurava da planer ima ažurirane statistike o tabeli. Ako ne postoje statistike ili su zastarele, planer može doneti loše odluke tokom planiranja upita, što dovodi do loših performansa na tabelama sa nepreciznim ili nepostojećim statistikama. Važno je napomenuti da ako je daemon za automatsko vakumiranje (autovacuum) omogućen, on može automatski pokrenuti ANALYZE komandu.

# Zaključak

U ovom radu je obrađena tema optimizacije upita u PostgreSQL bazi podataka. U prvom delu su objašnjeni termini: plan upita, struktura plana upita, kao i komande EXPLAIN, ANALYZE i VACUUM koje služe za analizu i optimizaciju upita.

U drugom delu je fokusi bio na genetskom optimizatoru upita, opisan je genetski algoritam, način rada GEQO u PostgreSQL bazi podataka, kao i način podešavanja parametara za genetski optimizator upita.

U trećem delu su predstavljene neke od tehnika za optimizaciju upisa u bazu podataka, kao što su isključivanje AUTOCOMMIT opcije, korišćenje COPY i PREPARE opcije, uklanjanje indeksa, brisanje stranih ključeva i povećanje promenljivih maintenence\_wal\_size i max\_wal\_size. Takođe je razmotrana i mogućnost isključenja WAL arhiviranja i strimovanja aplikacije, kao i pokretanje ANALYZE komande nakon upisa.

Kroz ovaj rad je prikazano da postoje brojne tehnike za optimizaciju upita i upisa u PostgreSQL bazu podataka kroz praktične primere[4]. Pored toga, istaknuto je da je jako važno pravilno podešavanje parametara i indeksa u bazi kako bi se postigle optimalne performanse baze. U skladu sa tim, potrebno je pažljivo analizirati upite i izvršiti adekvatne optimizacije, što će pomoći u postizanju boljih performansi baze podataka.

# Literatura

1. <https://www.postgresql.org/files/documentation/pdf/15/postgresql-15-A4.pdf> [Accessed: 25-Maj-2023].
2. https://sematext.com/blog/postgresql-performance-tuning/ [Accessed: 2-April-2023].
3. https://www.cybertec-postgresql.com/en/how-the-postgresql-query-optimizer-works/ [Accessed: 5-April-2023].
4. https://github.com/catherinedevlin/opensourceshakespeare [Accessed: 20-Maj-2023].