UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

OBRADA UPITA KOD POSTGRESQL BAZE PODATAKA
-Seminarski rad-

Student:
Sara Savić, br.ind. 1758

Mentor:
Prof. dr Aleksandar Stanimirović

Niš,april 2024.

SAŽETAK

U seminarskom radu na temu “Obrada upita kod PostgreSQL baze podataka”, obrađene su celine koje se odnose na upoznavanje sa terminima baze podataka i obrade upita kod ove baze podataka.

Centralni deo rada odnosi se na razmatranje obrade složenijih upita koji se koriste prilikom spajanja tabela kod PostgreSQL baze podataka i bavi se izučavanjem i opisom različitih načina za spajanje većeg broja tabela, metoda koje se koristi od strane baze podataka kako bi se izvršilo spajanje tabela, i naposletku procesa obrade upita koji se koriste prilikom spajanja tabela,sa posebnim fokusom na proces generisanja optimalnog plana izvršenja ovakve vrste upita. Rad se takođe fokusira i na prikaz praktičnih primera koji ilustruju opisane teorijske koncepte i koji na najbolji način prikazuju kako je moguće izvršiti spajanje tabela kod PostgreSQL baze podataka i koji način odabrati u zavisnosti od potreba, kao i u čemu se ogleda značaj spajanja tabela.

Sadržaj

1.UVOD

Koncept tehnologija baza podataka predstavlja suštinske elemente softverskih sistema, omogućavajući modelovanje, konstrukciju i efikasno upravljanje podacima. Baze podataka su neophodan deo mnogih profesionalnih, naučnih i privatnih delatnosti, pružajući osnovu za skladištenje i organizaciju podataka različitih formata. Same baze podataka predstavljaju kolekcije međusobno povezanih podataka, organizovanih u strukturu koja podržava jednu ili više aplikacija. Sistem baze podataka obuhvata podatke, sistem za upravljanje bazama podataka(DBMS) i aplikacije koje koriste te podatke.

DBMS predstavlja softverski sistem koji korisnicima omogućava kreiranje, upravljanje i korišćenje baza podataka i pruža različite funkcionalnosti kao što su manipulacija podacima, upravljanje pristupom i sigurnošću, itd. Postoji mnoštvo DBMS-ova a jedan od njih jeste i PostgreSQL.

PostgreSQL je objektno-relacioni sistem za upravljanje bazama podataka, koji se ističe svojim proširenim SQL upitnim jezikom i podrškom za ACID transkacije. Kao open-source platforma, PostgreSQL pruža bogat set funkcionalnosti kao što su kompleksni upiti, mehanizmi stranih ključeva i transkcioni integriteti. Pored svega ovoga, PostgreSQL može biti korišćen i modifikovan besplatno od strane korisnika u privatne, komercijalne ili akademske svrhe.

Jedan od najbitnijih procesa u okviru baza podataka, pa tako i okviru PostgreSQL-a jeste proces obrade upita,a njegovo razumevanje je neophodno za efikasno korišćenje PostgreSQL baze podataka.Međutim, pored obrade klasičnih upita, neophodno je razumevanje i obrade upita koji se koriste prilikom spajanja tabela, jer upravo takav jedan upit omogućava pribavljanje podataka iz većeg broja tabela kroz jedan upit.

Cilj seminarskog rada “Obrada upita kod PostgreSQL baze podataka” jeste detaljno istraživanje procesa obrade upita za spajanje tabela kod PostgreSQL sistema, sa posebnim osvrtom na različite tehnike spajanja tabela i njihovu primenu u praksi. Pored toga, u radu će biti reči o procesu obrade upita u relacionim bazama podataka, kao i samoj PostgreSQL bazi podataka, o njenim svojstvima i karakteristikama, i uopšteni proces obrade upita kod PostgreSQL-a.

2.OBRADA UPITA KOD RELACIONIH BAZA PODATAKA

Obrada upita odnosi se na kompilaciju i izvršavanje specifikacije upita napisanih na deklarativnom jeziku baze podataka, kao što je SQL koji koriste relacione baze podataka. U kontekstu relacionih baza podataka, obrada upita predstavlja proces prevođenja upita višeg nivoa na upit nižeg nivoa koji je baza podataka u stanju da izvrši. Proces obrade upita uključuje sledeće korake[1]:

1. Parsiranje i prevod upita-Prvi korak u obradi upita koji je prosleđen bazi podataka jeste pretvaranje upita u oblik koji može koristiti mašina za obradu upita. Upitni jezici visokog nivoa kao što je SQL, upite predstavljaju kao niz karaktera ili sekvencu karaktera. Određene sekvence karaktera predstavljaju različite vrste tokena poput ključnih reči, operatora, operanada, literalnih nizova itd. Kao i kod svih jezika, tako i kod upitnog jezika, postoje određena pravila-sintaksa i gramatika jezika, koja regulišu kako se tokeni mogu kombinovati u validne izjave. Samim tim, osnovni zadatak parsera jeste izvlačenje tokena iz niza karaktera i njihovo prevođenje u odgovarajuće interne podatke(operacije i operande relacione algebra) i strukture(stablo upita). Poslednji zadatak parsera jeste provera validnosti i sintakse originalnog niza karaktera upita.
2. Optimizacija upita-Optimizacija upita se odnosi na primenu određenih pravila na interne strukture podataka upita kako bi transformisala ove strukture u ekvivalentne, ali efikasnije reprezentacije. Pravila mogu biti zasnovana na matematičkim modelima relacione algebre izraza i stabla(heuristika), na procenama troškova različitih algoritama primenjenih na operacije ili na semantici unutar upita i relacija koje uključuje.
3. Evaluacija upita- Poslednji korak obrade upita jeste evaluacija. Evaluacija upita uključuje izbor najboljeg plana evaluacije koji je generisao optimizator i njegovo izvršenje.

Prilikom obrade upita kod relacionih baza podataka , pravi izazov može se javiti prilikom obrade složenijeg upita, tačnije upita koji vrši spajanje većeg broja tabela. Kako bi bilo moguće izvršiti ovakav upit neophodno je najpre izvršiti spajanje različitih tabela. Proces spajanja, koji se često naziva i operacija spajanja(***eng.join operation***) predstavlja složen proces koji zahteva pažljivo planiranje ali i optimizaciju, zarad dobijanja efikasnih performansi. Kod relacionih baza podataka , za spajanje tabela postoji nekoliko načina spajanja:operatorima spajanja,korišćenjem podupita, kreiranjem unije ili posebnih konstrukcija u okviru upita koje će vratiti podatke iz većeg broja tabela. Međutim prilikom izvršenja takvog upita, odgovornost je na bazi podataka da optimalno izvrši dobijeni upit. To se postiže korišćenjem fizičkih operatora spajanja. Oni određuju kako će baza podataka fizički spojiti podatke iz različitih tabela.Kako postoji veći broj fizičkih operatora, zadatak optimizatora jeste da odabere koji od ovih operatora će najoptimalnije izvršiti ovakav složen upit.Metode spajanja upita jesu sledeće[2]:

- Nested loop join-Nested Loop Join je osnovna metoda spajanja, koja podrazumeva iteriranje kroz svaki red u spoljnoj tabeli i poređenje sa svakim redom u unutrašnjoj tabeli. U slučaju podudaranja na osnovu određenog uslova spajanja, redovi koji se poklapaju se spajaju i uključuju u rezulutujući skup. Ova metoda je efikasna za male

tabele ili kada su odgovarajući indeksi na mestu, međutim ukoliko se veličine tabela povećaju, performanse se pogoršavaju, što može dovesti do dugih vremena izvršenja.

- Sort-Merge join- Ulazne tabele moraju biti sortirane pre operacije spajanja kako bi se izvršio sort-merge join. Nakon sortiranja, sistem baze podataka upoređuje i spaja odgovarajuće redove iterirajući kroz obe tabele. Ova metoda je efikasna za velike tabele, posebno ako su ulazni podaci već sortirani ili skoro sortirani, međutim sortiranje ulaznih tabela može zahtevati mnogo resursa, posebno za velike skupove podataka.
- Hash join-Hash join predstavlja naprednu tehniku koja koristi heš tabelu za uparivanje redova prema njihovim ključevima spajanja. Algoritam uključuje fazu izgradnje i fazu pretrage. U fazi izgradnje, heš tabela se generiše koristeći ključeve spajanja manje ulazne tabele. U fazi pretrage, sistem baze podataka skenira kroz veću ulaznu tabelu, koristeći istu heš funkciju na ključevima spajanja i traži podudaranja u heš tabeli.
- Adaptive join-Adaptive join je savremeni pristup metodi spajanja koji prilagođava svoju strategiju prema karakteristikama ulaznih podataka u toku izvršavanja. Obično počinje sa nested loop-om i nadgleda napredak spajanja.Ako performanse sistema nisu zadovoljavajuće može se preći na drugi metod spajanja, na primer sort-merge ili hash join kako bi se povećala efikasnost.

Ovim poglavljem predstavljene su osnove obrade upita kod relacionih baza podataka, sa posebnim osvrtom na obradu upita koji se dobijaju spajanjem više tabela, kao i metodama koje baze koristi za najefikasnije spajanje tabela. U nastavku seminarskog rada, biće prikazano kako se vrši obrada upita, kao i spajanje tabela i izvršenje upita iz više tabela, na primeru jedne od najpoznatijih relacionih baza podataka, PostgreSQL baze podataka.

3.POSTGRESQL BAZA PODATAKA

PostgreSQL predstavlja moćan sistem otvorenog koda za upravljanje objektno-relacionim bazama podataka koji koristi i proširuje SQL jezik u kombinaciji sa mnogim funckijama koje bezbedno skladište i skaliraju najkomplikovanija radna opterećenja podataka. Poreklo PostgreSQL-a datira iz 1986.godine kao deo projekta POSTGRES na Univerzitetu Kalifornije u Berkliju i ima više od 35 godina aktivnog razvoja na osnovnoj platformi [3].

PostgreSQL stekao je snažnu reputaciju zahvaljujući svojoj arhitekturi, pouzdanosti, integritetu podataka i robusnom skupu funkcija. Ova baza podataka podržana je na svim operativnim sistemima, poput Linux-a, macOS-a, Windows-a, a od 2001.godine je ACID kompatabilna. Pored toga,PostgreSQL ima moćne dodatke, poput popularnog proširenja geoprostorne baze podataka PostGIS. Dodatno, još jedna od značajnih karakteristika ove baze podataka, jeste konstantan rad na unapređenju postojećih verzija, tako da se korisnicima uvek nude najnovije verzije PostgreSQL baze podataka. Trenutno aktuelna verzija jeste PostgreSQL 16, koja je izašla septembra 2023.godine. Za prikaz praktične relazivacije teorijskih elemenata, obrađenih u seminarskom radu, korišćena je najnovija verzija, verzija PostgreSQL 16.2.

3.1 Arhitektura PostgreSQL baze podataka

Arhitektura PostgreSQL baze podataka zasniva se na klijent-server modelu [4]. Njegov glavni program funkcioniše kao servis odgovoran za definisanje struktura podataka, skladištenje podataka i obradu upita. Ova arhitektura omogućava PostgreSQL-u da opslužuje više klijenata, bez obzira da li se oni povezuju lokalno ili preko mreže. Kada glavni proces primi zahtev od klijenta stvara se novi proces koji je posvećen tom konkretnom zahtevu. Ukoliko se više klijenata povezuje istovremeno, svaki od njih dobija svoj sopstveni proces. Kako svaki proces zahteva CPU jezgra i RAM memoriju, broj klijenata koji se mogu povezati istovremeno ograničen je raspoloživim resursima CPU jezgara i RAM-a. Nakon što server iscrpi svoje resurse svaki novi zahtev za povezivanjem će biti odbijen. U takvim situacijama, klijenti moraju ponovo da pokušaju da se povežu. Ovaj problem rešava mehanizam koji se naziva “Pooling” konekcija. Pooler-i za konekciju iniciraju više konekcija ka serveru tokom pokretanja i pružaju ih klijentu kako zahtevi stižu. Ukoliko su sve dostupne konekcije zauzete, novi zahtevi će biti stavljeni u red i biće opsluženi čim jedna konekcija postane dostupna. Pooler-i za konekciju, pored toga što rešavaju problem sa konekcijama, takođe poboljšavaju performanse baze podataka izbegavajući vreme potrebno za kreiranje konekcije za svaki novi zahtev klijenta.

PostgreSQL se sastoji od različitih procesa od kojih svaki ima svoju ulogu u osiguravanju nesmetanog funkcionisanja baze podataka. Ovi procesi, se međutim, šire gledano, mogu klasifikovati u tri kategorije:

1. Procesi servera PostgreSQL-a-Ovi procesi predstavljaju glavne nadzorne entitete koji su odgovorni za upravljanje klijentskim konekcijama i pokretanje novih backend procesa. Oni osluškuju zahteve za konekcijom klijenata i omogućavaju besprekornu komunikaciju između klijenata i baze podataka.
2. Backend procesi-Backend procesi su zaduženi za svaku klijentsku konekciju. Oni se staraju o izvršavanju upita i upravljanju transakcijama baze podataka u ime klijenata,

direktno komunicirajući sa istim i osiguravajući efikasnost operacija nad bazom podataka.

3. Procesi pozadine(***eng.Backgorund Worker Process***)-Za razliku od procesa servera i backend procesa, ovi procesi imaju nekoliko podtipova, pri čemu svaki od tih podtipova obavlja specifične zadatke. Ovi procesi obavljaju ključne zadatke pozadine, kao što su održavanje baze podataka i administracija na nivou sistema.

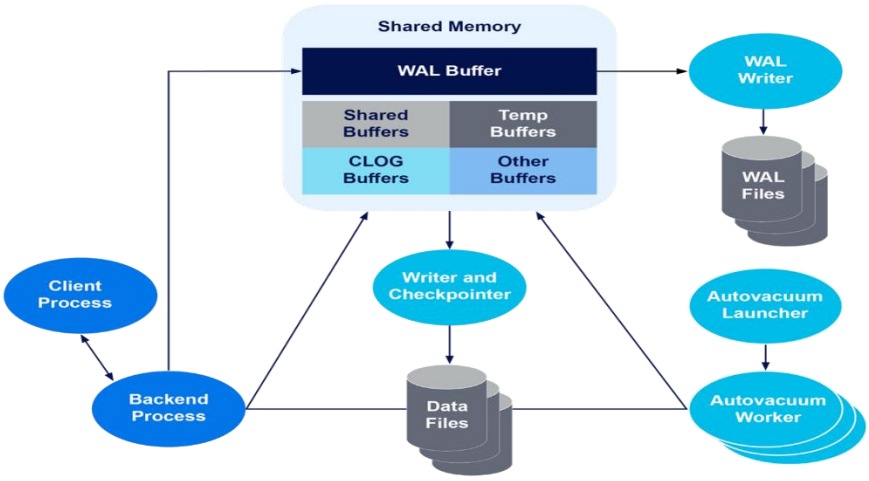
Još jedna bitna komponenta u samoj arhitekturi PostgreSQL baze podataka jeste memorija, koja igra ključnu ulogu u međuprocesnoj komunikaciji i performansama PostgreSQL baze podataka. Memorija u PostgreSQL-u može se klasifikovati u 2 kategorije na osnovu načina njenog korišćenja [4]:

- Lokalna memorija koju koriste pojedinačni backend procesi za izvršavanje upita.
- Deljena memorija koju koristi PostgreSQL server proces. Najvažnije komponente deljene memorije jesu deljeni baferi i WAL(***eng.Write-Ahead Logging***) baferi. Deljeni baferi služe kao keš za sve IO operacije, poboljšavajući performanse baze tako što često korišćene podatke čuvaju u memoriji radi bržeg pristupa. S druge strane, WAL baferi omogućavaju zapisivanje transkacija na disk i osiguravaju otpornost baze podatka tako što omogućavaju oporavak neizvršenih transkacija u slučaju pada sistema.

Naposletku priče o arhitekturi PostgreSQL baze podataka, biće reči i o samoj strukturi baze podataka [4]. To se odnosi na logičku i fizičku strukturu baze podataka.

Logička struktura: Klaster u PostgreSQL-u je skup baza podataka kojima upravlja jedan server. Svaka baza podataka se sastoji od šema, koje sadrže objekte poput tabela, pogleda(***eng.views***) i indeksa. PostgreSQL koristi tabele koje se nazivaju “Catalog” tabele za čuvanje informacija o objektima baze podataka. Tabele čine osnovu baze podataka i sadrže redove i kolone za čuvanje podataka. Indeksi omogućavaju brži pristup podacima tako što referišu specifične redove u tabeli, dok pogledi(***eng.views***) predstavljaju virtuelne tabele koje predstavljaju podskup podataka iz izvornih tabela.

Fizička struktura:PostgreSQL obezbeđuje arhitekturu koja omogućava efikasno čuvanje logičkih objekata u fizičkim datotekama. Svaka baza podataka ima svoj sopstveni direktorijum, a svaka tabela u bazi podataka biće datoteka unutar tog direktorijuma. PostgreSQL upisuje podatke u heap datoteke, vrstu organizacije datoteka koja se koristi za čuvanje i upravljanje podacima u bazi podataka. U heap datoteci, podaci su organizovani kao kolekcija zapisa od kojih svaki ima fiksnu dužinu i zauzima fiksnu količinu prostora na disku. Prilikom dodavanja novog zapisa, taj zapis se dodaje na kraj datoteke. Heap datoteka veličine 1GB organizovana je kao kolekcija stranica veličine 8KB(ove veličine se mogu menjati), od kojih svaka može da sadrži više redova podataka. PostgreSQL upisuje u novu datoteku kada se postojeća popuni, tako da tabela može obuhvatiti više datoteka.



Slika 1-Primer osnovne arhitekture PostgreSQL sistema[1]

3.2 Upravljanje PostgreSQL bazom podataka

Upravljanje PostgreSQL bazom podataka[5] obuhvata ključne akcije administracije serverske baze podataka PostgreSQL. Postoje različiti alati, kako open-source alati, tako i plaćeni, koji olakšavaju ovaj proces, međutim u okviru seminarskog rada korišćeni su:

- psql:psql je front-end terminalni alat koji omogućava izvršavanje SQL upita direktno ili iz datoteka, kao i pregled rezultata. Pored toga, psql pruža i različite meta komande.
- pgAdmin:pgAdmin predstavlja jednu od najpopularnijih open-source platformi za upravljanje i razvoj PostgreSQL baza podataka. pgADmin je napredan alat koji se može koristiti na različitim operativnim sistemima.

U okviru ovog poglavlja, dat je pregled osnovnih koncepata same PostgreSQL baze podataka, kao i opis same arhitekture ove baze. Ovaj uvid pružio je jasniju sliku o samoj bazi, pre nego bude opisano i prikazano kako se vrši obrada upita u PostgreSQL bazi podataka.

4.OBRADA UPITA KOD POSTGRESQL BAZE PODATAKA

Ovo poglavlje pružiće uvid u samu obradu upita kod PostgreSQL baze podataka. Upit predstavlja proces preuzimanja ili komandu za preuzimanje podataka iz baze podataka. Kako je već ranije rečeno(videti poglavlje 3), za pisanje upita PostgreSQL baza podataka koristi upitni jezik koji predstavlja varijantu SQL upitnog jezika.Kod PostgreSQL baze podataka, upiti započinju odgovarajućom komandom(SELECT,INSERT,UPDATE,DELETE), nakon čega se formuliše odgovarajuća sintaksa(recimo, FROM klauzula iza SELECT naredbe ili VALUES u okviru INSERT komande). Naposletku, moguće je dodati odgovarajuće filtere i uslove kako bi se ograničili rezultati koje upiti vraćaju ili izvršile složene operacije.

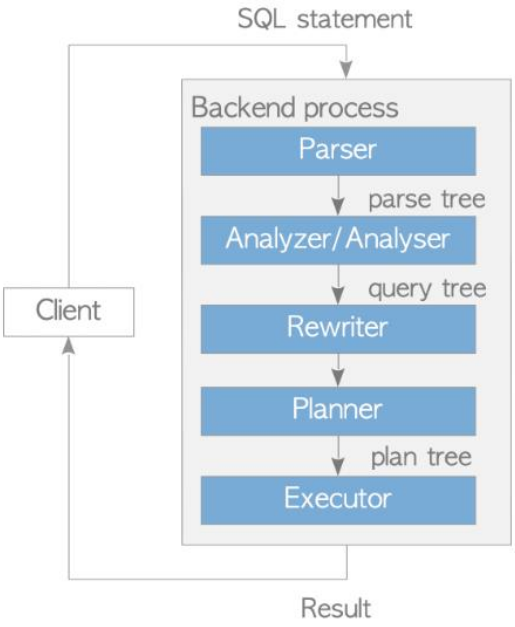
4.2 Proces obrade upita

Kako je već ranije pomenuto, (videti poglavlje 3.1),PostgreSQL baza podataka implementira klijent-server arhitekturu, gde klijentska strana šalje zahteve serverskoj strani

[1] Izvor slike-https://www.instaclustr.com/blog/postgresql-architecture

kako bi dobila željene podatke. Zahtevi se šalju kucanjem odgovarajućih SQL upita, nakon čega serverska strana vraća rezultat, međutim koraci koji dovode do rezultata enkapsulirani su kao crne kutije. Ti koraci zapravo predstavljaju proces obrade upita[6], koji, kako je već objašnjeno u poglavlju 2, podrazumeva prevođenje upita visokog nivoa, kao što je SQL upit, na upit nižeg nivoa koji je baza podataka u stanju da izvrši. Obrada upita jeste jedan od najsloženijih podsistema PostgreSQL-a. Obrada upita predstavlja backend proces(videti poglavlje 3.1) i sastoji od 5 delova:

1. Parser:Parser proverava sintaksu SQL upita i parsira SQL upit iz običnog teksta u stablo parsiranja.
2. Analizator: Analizator vrši semantičku analizu stabla parsiranja, nakon čega generiše stablo upita.
3. Rewriter:Rewriter transformiše stablo upita prosleđeno od analizatora na osnovu pravila koja postoje u sistemu pravila.
4. Planer:Planer generiše stablo plana koje može najefikasnije da se izvrši na osnovu stabla upita.
5. Izvršilac(Executor): Izvršilac(Executor) izvršava upit pristupajući tabelama i indeksima u redosledu koji je kreiran od strane stabla upita.



Slika 2-Proces obrade upita[2]

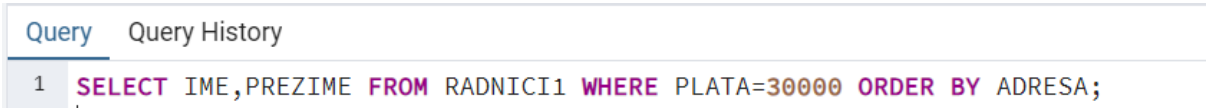
U nastavku potpoglavlja biće dati detaljni opisi svakog od ovih koraka.

4.2.1 Parser

Prvi podsistem ovog backend procesa jeste parser koji konvertuje plain tekst prosleđenog upita u stablo parsiranja. Zadatak parsera jeste da proveri sintaksnu strukturu upita, pritom osiguravajući da budu ispoštovana sva gramatička pravila PostgreSQL jezika.

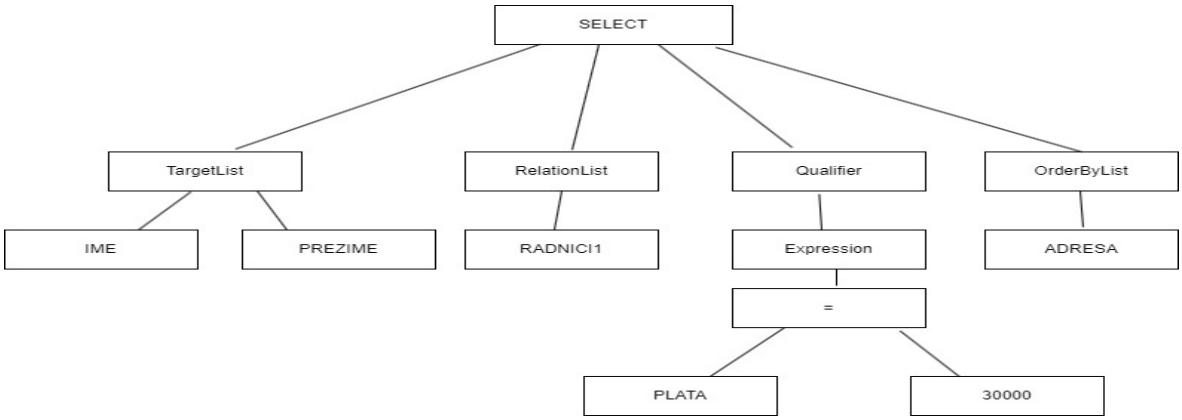
[2] Izvor slike-https://www.interdb.jp/pg/pgsql03/01.html

Parser provera gramatičke greške, validira samu strukturu upita i naposletku kreira stablo parsiranja . Kreirano stablo parsiranja koristiće drugi podsistemi obrade upita.



Slika 3-Primer upita koji služi za kreiranje stable parsiranja

Na primer, za upit na slici 3 koji za kreiranu tabelu RADNICI1, koja sadrži podatke o imenu, prezimenu, godinama, plati radnika i adresi radnika stablo parsiranja bilo bi sledeće:



Slika 4-Primer jednostavnog stable parsiranja za upit sa slike 3

Važno je napomenuti, da parser provera samo sintakse greške prilikom generisanja stabla, ali ne i semantičke greške, pa tako ukoliko se u upitu navede ime tabele koja ne postoji parser neće vratiti grešku. Proveru semantičkih grešaka vrši analizator.[6]

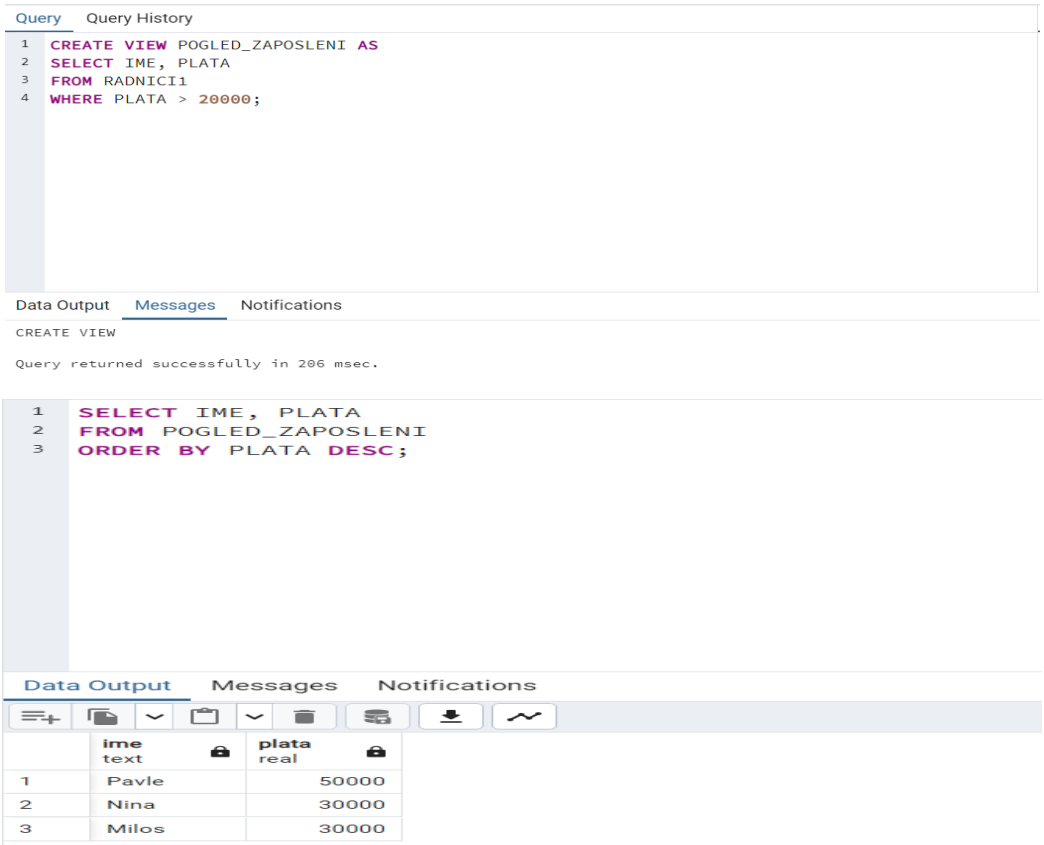
4.2.2 Analizator

Analizator vrši semantčku analizu nad stablom parsiranja koje je kreirao parser. Analizira sintaksu upita, proverava postojanje i ispravnost imena tabela i kolona, te rešava eventualne nejasnoće. Ova analiza rezultira generisanjem stabla upita. Stablo upita predstavlja internu reprezantciju SQL naredbe i sastoji se od sledećih bitnih delova[7]:

- Tip komande(***eng.command type***), koji predstavlja jednostvanu vrednost i govori koja komanda(SELECT,INSERT,UPDATE,DELETE) je proizvela stablo upita.
- Tabela opsega(***eng. range table***) je lista relacija koje se koriste u upitu. U SELECT naredbi, to su relacije koje su navedene nakon ključne reči FROM.
- Relacija rezultata(***eng. result relation***) predstavlja indeks u tabeli opsega koji identifikuje relaciju u koju idu rezultati upita. SELECT naredbe nemaju relaciju rezultata, dok je kod naredbi kao što su INSERT, UPDATE i DELETE relacija rezultata tabela(ili pogled(***eng.view***)) gde će promene biti primenjene.
- Lista ciljeva(***eng.target list***) je lista izraza koji definišu rezultat upita.Kod SELECT naredbe ovi izrazi čine konačan izlaz upita.DELETE naredbe ne zahtevaju listu ciljeva jer ne proizvode nikakav rezultat.Za INSERT naredbe lista ciljeva opisuje nove redove koji treba da idu u relaciju rezultata, dok za UPDATE naredbe lista ciljeva opisuje nove redove koji treba da zamene stare.
- Stablo spajanja upita(***eng.join tree***) prikazuje strukturu FROM klauzule.

4.2.3 Rewriter

Rewriter upita transformiše stablo upita na osnovu unapred definisanih pravila. On pojednostavljuje i proširuje upit, eliminiše redundatne operacije i rešava reference ka objektima poput tabela i kolona. Ta pravila se koriste, recimo, za kreiranje pogleda(***eng.view***).Pogledi predstavljaju virtuelnu tabelu koja se koristi kako bi se pojednostavili složeniji upiti, jer se ti upiti definišu samo jednom u pogledu nakon čega im se može direktno pristupati. Pogled može predstavljati podskup stvarne tabele, birajući određene kolone ili određene redove iz obične tabele. Kada se definiše pogled, odgovarajuće pravilo se automatski generiše i čuva u "catalog" tabeli[3]. Kada se izda upit koji sadrži kreirani pogled, parser kreira stablo parsiranja. U ovoj fazi, rewriter prevodi čvor tabele opsega u stablo parsiranja podupita, što predstavlja odgovarajući pogled.[6]



Slika 4-Primer kreiranja i izvršenja pogleda za prethodno pomenutu tabelu RADNICI1

4.2.4 Planer

Planer upita generiše optimalni plan izvršenja za kreirano stablo upita.Plan može uključivati sekvencijalno pretraživanje cele tabele, ili ukoliko su kreirani, pretraživanje indeksa. Ukoliko upit uključuje dve ili više tabela, planer može predložiti nekoliko različitih metoda za spajanje tabela(pogledati potpoglavlje 5.2).Izvršni planovi su razvijeni u smislu operatora upita, kao što su operatori SeqScan, Sort ili Index-Scan koji se koriste za pribavljanje traženih podataka. Svaki operator upita transformiše jedan ili više ulaznih skupova u srednji skup rezultata. Na primer, operator SeqScan transformiše ulazni skup(fizičku tabelu), u skup rezultata, pritom filtrirajući sve redove koji ne ispunjavaju ograničenja upita. Operator Sort proizvodi skup rezultata tako što menja redosled ulaznog

[3] "Catalog" tabela- tabele koje se koriste za čuvanje informacija o objektima baze podataka

skupa prema jednom ili više ključeva za sortiranje. Nakon što su generisani svi mogući planovi, optimizator traži najefikasniji plan izvršenja. Izbor najefikasnijeg plana zasniva se na proceni troškova, pri čemu svaki od operator upita ima drugačiju procenu troškova.[8]

4.2.5 Executor

Nakon izbora najpovoljnijeg plana izvršenja, izvršilac upita kreće od početka plana i traži od najvišeg operatora da proizvede skup rezultata. Svaki operator transformiše svoj ulazni skup u skup rezultata- taj ulazni skup može dolaziti od drugog operatora niže u stablu. Kada najviši operator završi svoju transformaciju, rezultati se vraćaju klijentu.[9]

5.UPITI KOJI SE KORISTE PRILIKOM SPAJANJA TABELA

U prethodnom poglavlju, detaljno je opisan proces obrade upita.Međutim, sama obrada upita može postati komplikovanija kod PostgreSQL baze podataka,ukoliko upit uključuje više tabela, kako je i objašnjeno u uvodnom poglavlju obrade upita kod samih relacionih baza podataka. SQL pruža nekoliko različitih načina za kreiranje upita koji se koriste za spajanje tabela,a poznavanje koji bi trebalo primeniti donosi efikasne rezultate.

Ovo poglavlje predstavlja ključni deo seminarskog rada i posvećeno je načinima spajanja tabela i njhovom značaju, prikazu metoda koje objašnjavaju kako su tabele fizički spojene u bazi podataka i naposletku primeru kreiranja i obrade upita koji se koriste za spajanje tabela u PostgreSQL bazi podataka.

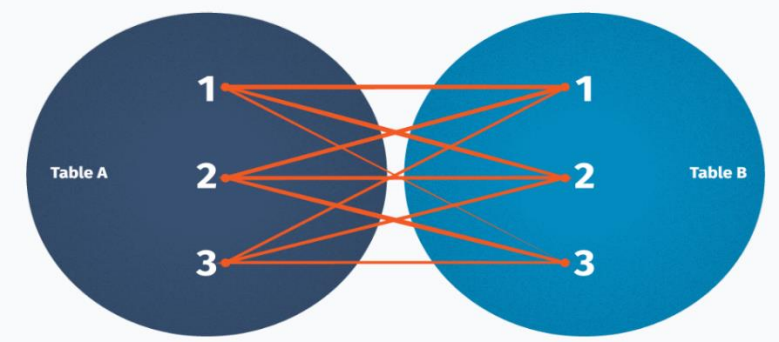
5.1 Spajanje tabela i značaj spajanja tabela u PostgreSQL bazi podataka

Prilikom spajanja zapisa između dve ili više tabele, u PostgreSQL bazi podataka, postoji nekoliko načina:osnovno spajanje tabela korišćenjem operatora spajanja, spajanje kreiranjem unije i spajanje korišćenjem podupita. U nastavku potpoglavlja biće opisani svaki od ovih načina.

1. Osnovno spajanje tabela korišćenjem operatora spajanja :Najosnovniji tip spajanja jeste kombinovanje polja iz dve ili više tabele korišćenjem vrednosti koje su zajedničke za obe tabele.Postoji više tipova osnovnog spajanja tabela (***eng.Join Types***)[10]
   - Unakrsno spajanje(***eng.CROSS JOIN***)
   - Unutrašnje spajanje(***eng.INNER JOIN***)
   - Levo spoljašnje spajanje(***eng.LEFT OUTER JOIN***)
   - Desno spoljašnje spajanje(***eng.RIGHT OUTER JOIN***)
   - Puno spoljašnje spajanje(***eng.FULL OUTER JOIN***)

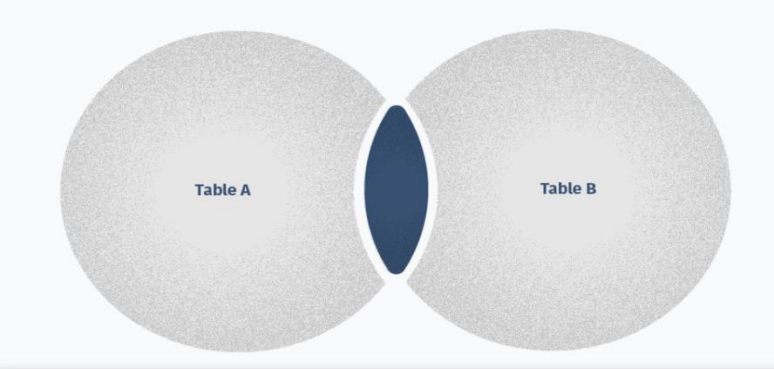
U nastavku će biti opisani i prikazani svaki od ovih tipova spajanja.

Unakrsno spajanje(eng.CROSS JOIN)- Ova tehnika spajanja uparuje svaku vrstu prve tabele sa svakom vrstom druge tabele. Ukoliko ulazne tabele imaju x i y kolona, rezultujuća tabela će u ovom slučaju imati x+y kolona,dok će broj vrsta biti proizvod broja vrsti u prvoj tabeli i broja vrsti u drugoj tabeli. Unakrsno spajanje(***eng.CROSS JOIN***) je poznato još pod nazivom Kartezijanov spoj(***eng.cartesian join)*** , jer se ovakvim spajanjem kreira karetzijanov proizvod.



Slika 5- Prikaz načina spajanja tabela korišćenjem CROSS JOIN-a[4]

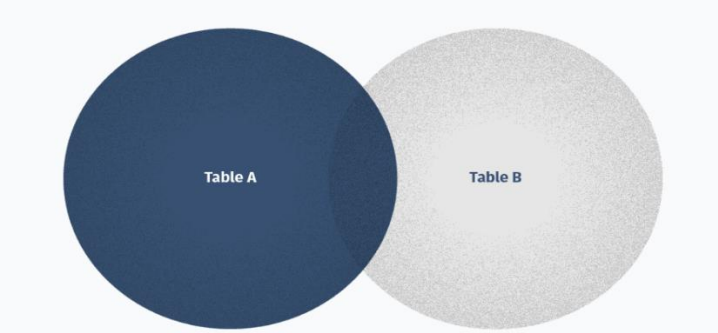
Unutrašnje spajanje(eng.INNER JOIN)- Unutrašnje spajanje(***eng.INNER JOIN***) kreira novu rezultujuću tabelu kombinovanjem vrednosti kolona dve tabele na osnovu predikata spajanja(***eng.join-predicate***). U okviru ove tehnike, vrši se poređenje svake vrste prve tabele sa svakom vrstom druge tabele, kako bi se pronašli svi parovi vrsti, koji zadovoljavaju predikat spajanja. Kada je predikat spajanja zadovoljen, vrednosti kolona za svaki par vrste obe tabele se kombinuju u rezultujuću vrstu.Dakle,unutrašnje spajanje vraća samo vrednosti koje se pojavljuju u obe tabele.Unutrašnje spajanje je najčešći tip spajanja, i kao takav on predstavlja podrazumevani tip spajanja.



Slika 6– Prikaz načina spajanja tabela korišćenjem INNER JOIN-a[4]

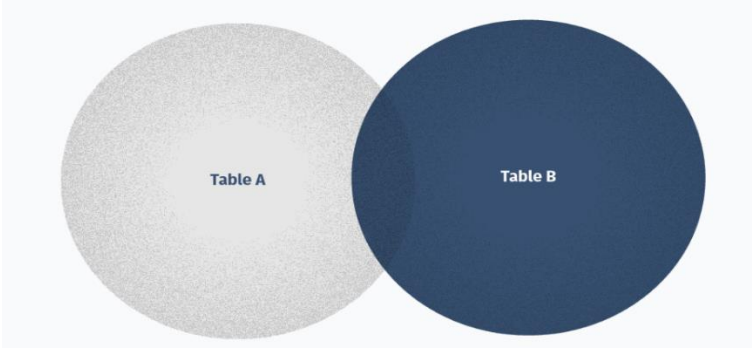
Levo spoljašnje spajanje(eng.LEFT OUTER JOIN)-Spoljašnje spajanje(eng.***OUTER JOIN***) je produžetak unutrašnjeg spajanja.SQL standard definiše tri tipa spoljašnjeg spajanja :levo, desno i potpuno, a PostgreSQL podržava sva tri tipa. U slučaju levog spoljašnjeg spajanja(eng.***LEFT OUTER JOIN***), prvo se vrši unutrašnje spajanje. Zatim se za svaku vrstu u prvoj tabeli koja ne zadovoljava uslov spajanja ni sa jednom drugom vrstom u drugoj tabeli, dodaje spojena vrsta sa null vrednostima u kolonama druge tabele. Dakle, spojena tabela uvek ima bar jednu vrstu za svaku vrstu u prvoj tabeli.

[4] Izvor slike- https://www.devart.com/dbforge/sql/sqlcomplete/sql-join-statements.html



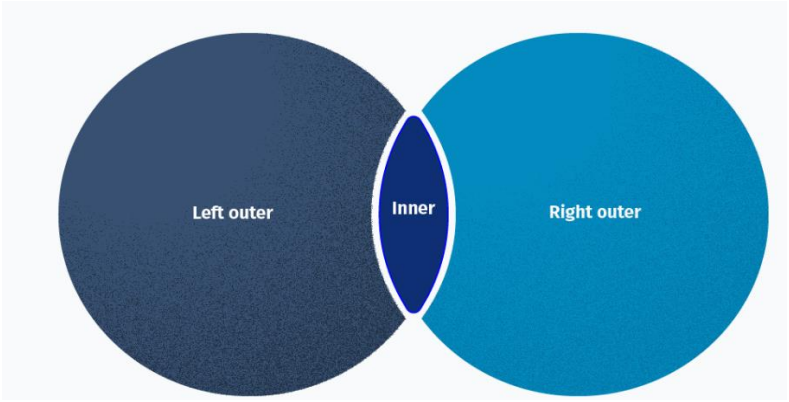
Slika 7 – Prikaz načina spajanja tabela korišćenjem LEFT OUTER JOIN-a[5]

Desno spoljašnje spajanje(eng.RIGHT OUTER JOIN)-U ovom slučaju, prvo se vrši unutrašnje spajanje. Zatim se za svaku vrstu u drugoj tabeli koja ne zadovoljava uslov spajanja ni sa jednom vrstom u prvoj tabeli, dodaje spojena vrsta sa null vrednostima u kolonama prve tabele. Ovaj način spajanja je suprotan od levog spoljašnjeg spajanja; rezultujuća tabela će uvek imati vrstu za svaku vrstu u drugoj tabeli.



Slika 8 – Prikaz načina spajanja tabela korišćenjem RIGHT OUTER JOIN-a[6]

Puno spoljašnje spajanje(eng.FULL OUTER JOIN)- Prvo se izvršava unutrašnje spajanje.Zatim se za svaku vrstu u prvoj tabeli koja ne zadovoljava uslov spajanja ni sa jednom vrstom u drugoj tabeli, dodaje spojena vrsta sa null vrednostima u kolonama druge tabele. Dodatno, za svaku vrstu u drugoj tabeli koja ne ispunjava uslov spajanja ni sa jednom vrstom u prvoj tabeli, dodaje se spojena vrsta sa null vrednostima u kolonama prve tabele.

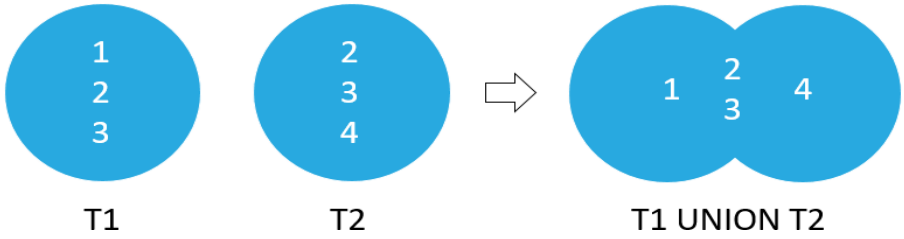


[5] Izvor slike- https://www.devart.com/dbforge/sql/sqlcomplete/sql-join-statements.html

Slika 9 – Prikaz načina spajanja tabela korišćenjem FULL[6] OUTER JOIN-a[6]

2. Spajanje kreiranjem unije-Prilikom spajanja tabela kreiranjem unije, postupak je sličan, kao i uopšteni postupak kreiranja unije u relacionoj algebra[11].Dakle, i ovde, postupak kreiranja unije ne predstavlja ništa više nego kombinovanje informacija iz više tabela koje su unija kompatabilne. Tabele se nazivaju unija kompatabilnim ako ispunjavaju sledeće uslove:
    a. Tabele koje se kombinuju moraju imati isti broj kolona istog tipa podataka.
    b. Broj vrsta ne mora biti isti.

Za kreiranje unije koristi se operator UNION, u SQL-u, koji, kada su gore navedeni kriterijumi ispunjeni, vraća sve vrste iz više tabela, nakon eliminisanja duplikata vrsti, kao rezultućuju tabele.Dodatno, imena kolona prve tabele postaće imena kolona rezultujuće tabele, a sadržaj druge tabele biće spojen u kolone rezultujuće tabele koje su istog tipa.



Slika 10-Prikaz postupka spajanja tabela kreiranjem unije[7]

3. Spajanje korišćenjem podupita-Podupit je upit koji je ugnježden unutar nekog drugog upita. Podupit se takođe naziva i unutrašnji upit ili ugnježdeni upit. Sam podupit se izvršava jednom pre glavnog upita,a zatim glavni upit koristi rezultate tog podupita. Spoljašnji upit i unutrašnji upit mogu biti povezani po vrednostima većeg broja atributa. Ukoliko se upoređuju argumenti koji se sastoje od više atributa, oba argumenta moraju da imaju jednak broj atributa, a upoređuje se prvi atribut sa prvim ,drugi sa drugim,itd. Atributi koji se upoređuju moraju biti istog ili kompatabilnog tipa podataka. Podupiti se koriste kada sve informacije koje je potrebno prikazati u upitu se nalaze u jednoj tabeli ili kada se kolone preko kojih se postavljaju uslovi nalaze u drugim tabelama,što označava proces spajanja tabele. U tom slučaju bi jedna tabela sadržala podatke koji se vraćaju, dok bi se druga tabela koristila za postavljanje uslova da se odrede koji podaci se zapravo vraćaju iz prve tabele[12].

Prilikom spajanja tabela korišćenjem nekog od prethodno opisanih načina neophodno je znati kada se koj način primenjuje.Recimo, podupiti bi se koristili za spajanje tabela koje imaju kompleksne ili ugnježdene odnose ili kad je potrebno filtrirati ili grupisati podatke na osnovu vrednosti u drugoj tabeli. Kreiranje unije bi bilo pogodno ukoliko su tabele unijski kompatabilne i ukoliko imaju isti broj kolona sa podacima istog tipa. Takođe, i kod spajanja tabela korišćenjem JOIN operatora, neophodno je odlučiti u kom trenutku je koj tip operacije spajanja primenljiv.Ukoliko je potrebno prikazati sve podatke iz obe tabele, koristiće se unutrašnje spajanje(***eng.INNER JOIN***).Levo spoljašnje spajanje ***(eng.LEFT OUTER JOIN)*** bi bilo iskorišćeno ukoliko je potrebno prikazati sve podatke iz prve tabele, ali nije potrebno prikazati podatke iz druge tabele, dok ukoliko je potrebno suprotno(prikazati sve

[6] Izvor slike- https://www.devart.com/dbforge/sql/sqlcomplete/sql-join-statements.html
[7] Izvor slike- https://www.sqlitetutorial.net/sqlite-union/

podatke iz druge tabele ali nije potrebno prikazati podatke iz prve tabele) bilo bi korisšćeno desno spoljašnje spajanje ***(eng.RIGHT OUTER JOIN)***. Ako je potrebno prikazati sve podatke iz obe tabele, ali je potrebno i spojiti tabele na osnovu jedne ili više kolona, koristilo bi se poptuno spoljašnje spajanje ***(eng.FULL OUTER JOIN),*** dok ukoliko nije potrebno spajati tabele na osnovu kolona, koristilo bi se unakrsno spajanje(***eng.CROSS JOIN***).

5.1.2 Značaj spajanja tabela prilikom obrade upita

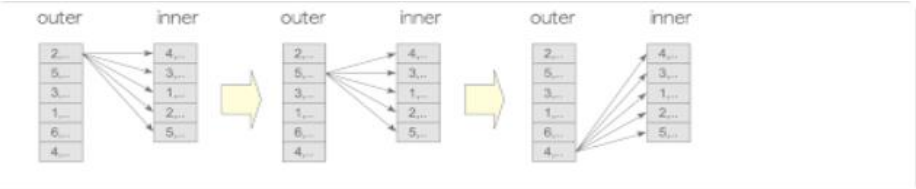
Svrha operacije spajanja jeste povezivanja podataka iz više tabela kako bi se dobio jedinstven rezultat koji sadrži informacije iz obe tabele. To je posebno važno prilikom obrade upita, jer omogućava rad sa podacima iz većeg broja tabela u okviru jednog upita, dok bi u suprotnom bilo potrebno izvršavati odvojene upite za svaku tabelu, a zatim kombinovati ručno rezultate iz različitih tabela, što bi otežalo analizu podataka.

5.2 Metode za spajanje tabela

Metode spajanja su algoritmi koji su zapravo korišćeni od samih baza podataka da obavljaju operacije spajanja tabela. Prilikom spajanja tabela, postoji više različitih verzija, odnosno prilagođenih oblika ovih algoritama koji se koriste u različitim situacijama.Na primer, i unutrašnje spajanje(eng.inner join) i levo spoljašnje spajanje(eng.left outer join), mogu koristiti istu metodu spajanja, ali će je koristiti na različit način.Postoji nekoliko metoda spajanja koje koriste relacione baze podataka koje su opisane u poglavlju 2, međutim PostgreSQL podržava Nested Loop Join, Merge Join i Hash Join. Ove različite metode se koriste u različitim uslovima, a zadatak planera jeste, da prilikom izvršavanja upita odabere najoptimalniji.

5.2.1 Nested Loop Join

U kontekstu programiranja, može se reći da je nested loop join “logička struktura u kojoj se jedna petlja(iteracija) nalazi unutar druge, odnosno za svaku iteraciju spoljne petlje izvršavaju se/procesiraju sve iteracije unutrašnje petlje”[13]. Na identičan način funkcioniše i nested loop join kod samih baza podataka; jedna od tabela koje se spajaju označava se kao spoljašnja tabela, a druga kao unutrašnja tabela. Za svaku vrstu spoljašnje tabele, sve vrste iz unutrašnje tabele se upoređuju jedna po jedna. Ukoliko se vrsta iz druge tabele poklapa, uključuje se u rezultujući skup, inače se ignoriše. Zatim se uzima sledeća vrsta iz spoljnašnje tabele i proces se ponavlja, sve dok se ne dođe do kraja spoljašnje tabele.



Slika 11-Prikaz načina rada Nested Loop Join-a[8]

```
for all the rows in outer table
  for all the rows in the inner table
    if outer_row and inner row satisfy the join condition
      emit the rows
  next inner
next outer
```

[8] Izvor slike- https://www.interdb.jp/pg/pgsql03/05/01.html

Slika 12-Prikaz pseudokoda Nested Loop Join-a[9]

Pseudokod koji je prikazan na slici 12 upravo to ilustruje; za svaku vrstu u spoljašnjoj tabeli(***eng.outer table***), prolazi se kroz sve vrste unutrašnje tabele(***eng.inner table***) i dodaju se one vrste koji zadovoljavaju uslov spajanja, nakon čega prelazi na sledeću vrstu unutrašnje tabele(***eng.inner table***) i tako sve dok se ne obrade najpre sve vrste iz unutrašnje tabele, (***eng.inner table***) i zatim i iz spoljašnje tabele(***eng.outer table***).

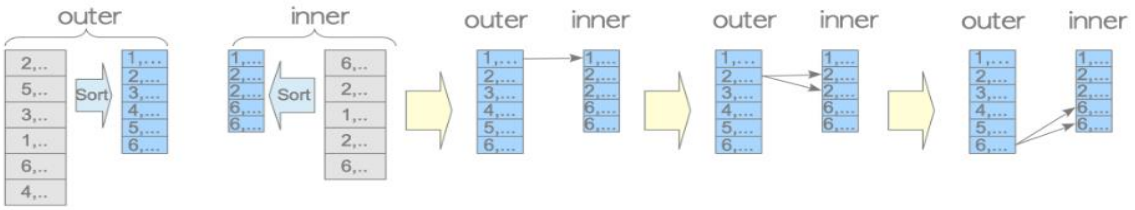
Ova metoda jeste najjednostavnija metoda spajanja, ali ujedno i najsporija.

5.2.2 Merge Join

Ova metoda spajanja zahteva da su obe ulazne tabele sortirane po kolonama koje se spajaju. Planer upita prilikom izvršavanja upita, skenira indeks ukoliko indeks postoji u kolonama koje se spajaju, ili ukoliko ne postoji, izvršiće sortiranje pre operacije Merge Join.Kada se obe ulazne tabele sortirane, Merge Join operator uzima po jednu vrstu iz svake tabele i upoređuje ih. Recimo, kod unutrašnjeg spajanja(***eng.inner join***), vrste se vraćaju ukoliko su jednake, a ukoliko nisu vrsta sa manjom vrednošću se odbacuje i uzima se nova vrsta iz iste te tabele. Ovaj proces se ponavlja dok sve vrste ne budu obrađene. Merge Join koristi privremenu tabelu za čuvanje vrsta. Ukoliko postoji neki dodatni uslov, pored uslova spajanja,sve vrste koje zadovoljavaju uslov spajanja, se vrednuju i sa tim dodatnim uslovom, a vraćaju se samo vrste koje ga zadovoljavaju.

Ova metoda spajanja je sama po sebi jako brza, pa se u većini slučajeva ona smatra najbržom meodom spajanja tabela.

Na slici 12 prikazan je postupak spajanja tabela metodom Merge Join.



Slika 13-Prikaz načina rada Merge Join-a[10]

```
Table1_sorted = table1.sort()
Table2_sorted = table2.sort()
Row1 = table1_sorted.first()
Row2 = table2_sorted.first()

while row1 is not Null and row2 is not Null:
        while row1 >= row2:
                if row1 == row2:
                        Add_to_result(row1, row2)
                Row2++
        Row1++
```

[9] Izvor slike-https://www.navicat.com/en/company/aboutus/blog/1948-nested-joins-explained

[10] Izvor slike-https://www.interdb.jp/pg/pgsql03/05/02.html

Slika 14-Primer jednostavnog pseudo-koda za Merge Join[11]

Upravo pseudo kod prikazan na slici 14 opisuje taj postupak. Pre svega se tabele sortiraju, a zatim se iz svake uzima po jedna vrsta i proverava uslov spajanja. Ukoliko zadovoljavaju taj uslov, vrste se uzimaju u rezultujuću tabele, ukoliko ne , prelazi se na sledeću vrstu. Ovaj postupak se ponavlja dok se ne obiću sve vrste, obe tabele.[14]

5.2.3 Hash Join

Hash join je vrsta metode spajanja tabela koja se sastoji od dve faze. U prvoj fazi se kreira hash tabela iz jedne od tabela koje se spajaju. U drugoj fazi prolazi se kroz vrste druge tabela, kako bi se našla poklapanja sa hash tabelom. [6]

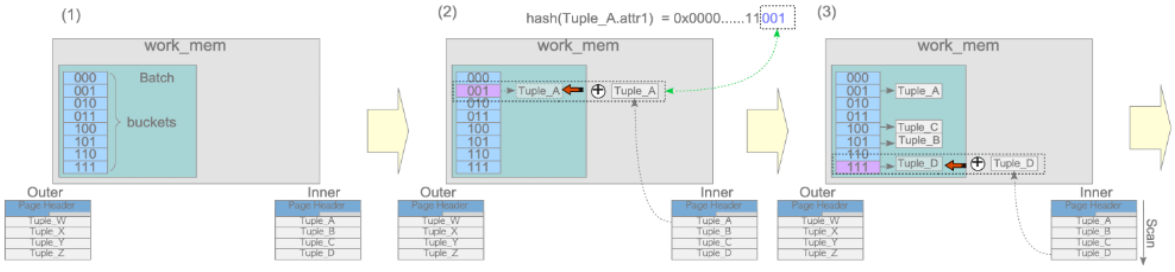
```
For row1 in table1:
        hashtable.add(row1.id, row1)

For row2 in table2:
        Row1 = hashtable.get(row2.id)
        If (row1 == row2):
                Add_to_result(row1, row2)
```

Slika 15-Primer pseudo koda Hash Join-a[12]

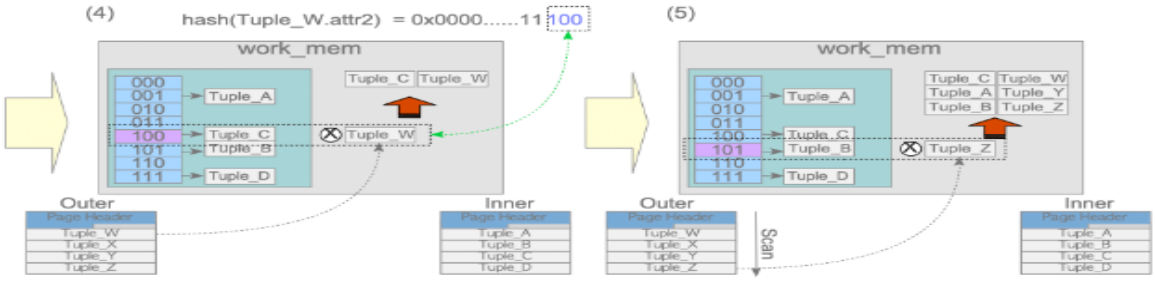
Postoje dve vrste Hash join-a:"In-memory" Hash Join i hibridni Hash Join(***eng.Hybrid Hash Join***).

1. "In-memory" Hash Join:Ova vrsta hash join-a se sastoji od dve faze:"build"faze i "probe" faze.U okviru "build" faze svaki red unutrašnje tabele se ubacuje u batch,oblast hash tabele.Batch se sastoji od hash slotova, koje nazivaju jednim imenom "buckets". Prilikom "probe" faze svaki red spoljašnje tabele se upoređuje sa unutrašnjim redovima u okviru batch-a i ukoliko je uslov spajanja zadovoljen, spajaju se.Postupak "build" faze prikazan je na slici 16, a "probe faze" na slici 17.



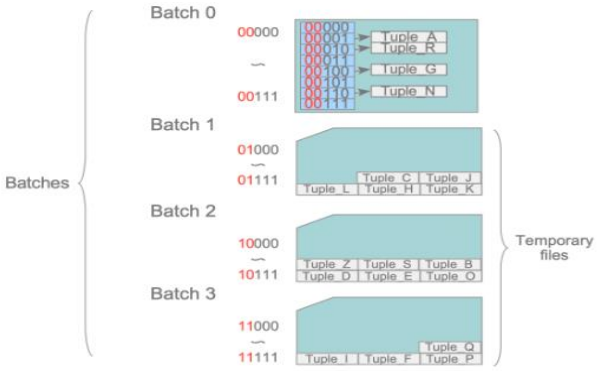
Slika 16-Prikaz "build" faze.[13]

[11] Izvor slike- https://www.metisdata.io/blog/understanding-join-strategies-in-postgresql
[12] Izvor slike-https://www.metisdata.io/blog/understanding-join-strategies-in-postgresql#hash-join
[13] Izvor slike- https://www.interdb.jp/pg/pgsql03/05/03.html



Slika 17-Prikaz "probe" faze[14]

2. Hibridni Hash Join(***eng.Hybrid Hash Join***)-Kod ove vrste hash join-a, u prvim "build" i "probe" fazama priprema se više "batch"-eva,pri čemu se samo jedan "batch" alocira u radnoj memoriji dok, dok se svi ostali čuvaju kao privremene datoteke. U hibridnom hash join-u, "build" i "probe" faze se izvršavaju onoliko puta koliko ima batch-eva., zato što su unutrašnja i spoljašnja tabela smeštene u istom broju batch-eva.Prvi batch-evi obe tabele se obrađuju u istoj etapi u kojoj se kreiraju.Međutim, kasnije etape zahtevaju ponovno učitavanje podataka, što je jako skup proces.Iz tih razloga, PostgreSQL kreira poseban batch koji se naziv "skew", kako bi efikasnije obradio veliki broj redova u prvoj etapi. "Skew" batch čuva redove unutrašnje tabele koji će biti spojeni sa redovima iz spoljašnje tabele, čije su najčešće vrednosti atributa, uključenih u uslov spajanja, relativno velike. Zato se ova modifikacija hash join-a naziva hibridni hash join sa "skew" batch-em(eng.***Hybrid Hash Join With Skew***).



Slika 18-Prikaz izgleda većeg broja batch-eva u hibrdnom hash join-u.[15]

5.3 Kreiranje i obrada upita koji se koriste prilikom spajanja tabela

U okviru ovog potpoglavlja biće objašnjeno i prikazano na praktičnim primerima kako izgleda kreiranje upita koji se koriste za spajanje tabela korišćenjem prethodno opisanih načina(opisano u potpoglavlju 5.1), a zatim i sam postupak njihove obrade, sa posebnim akcentom na generisanje optimalnog plana od strane planera.Proces obrade upita koji spajaju tabele se ne razlikuje značajno od obrade klasičnih upita koji vrše pribavljanje informacija iz jedne tabele u prva dva koraka.Naime, na samom početku same obrade, bi najpre parser

[14] Izvor slike- https://www.interdb.jp/pg/pgsql03/05/03.html
[15] Izvor slike- https://www.interdb.jp/pg/pgsql03/05/03.html

proverio sintaksu ovog upita i kreirao stablo parsiranja, koje bi zatim prosledio analizatoru da izbrši semantičku analizu i generiše stablo upita.Generisano stablo upita se dalje prosleđuje rewriter-u a potom i u koji generiše optimalni plan izvršenja.I prilikom obrade ovakve vrste upita, sam upit prolazi kroz faze preprocesiranja, kreiranja najjeftinijeg puta i naposletku generisanja stabla planiranja.Takođe, se i u ovom slučaju generisanje puta zasniva na proceni troškova, ali se sada, pored načina pretraživanja podataka razmatra i algoritam spajanja tabele. U PostgreSQL postoje tri vrste troškova:"start-up" trošak, koji predstavlja vreme potrebno da se pribavi prva torka(***eng.tuple***), "run" je trošak koji se koristi za pribavljanje svih torki(***eng.tuple***) i poslednja vrednost je ukupni trošak (***eng.total cost***), koji predstavlja zbir prethodna dva troška[6]. Kako je već pomenuto u potpoglavlju 4.3.3, za pretraživanje podatka postoje različite metode poput:Sekvancijalnog skeniranja(***eng.SeqScan***), Indeksnog skeniranja(***eng.Index Scan***) i Bitmap Indeks Skeniranja(***eng.Bitmap Index Scan***). Sekvencijalno skeniranje(***eng.SeqScan***)[15] je postupak koji temeljno prolazi kroz sve delove glavne tabele, odnosno datoteke, bez preskakanja. Na svakoj stranici, sistem pažljivo proverava svaku verziju vrste i odbacuje one koje ne odgovaraju upitu. Ovo skeniranje se izvršava pomoću keš memorije, gde se malo-prstenasti bafer koristi kako bi se sprečilo da veće tabele izbace korisne podatke iz memorije. Kada neki drugi proces treba da pristupi istoj tabeli, pridružuje se tom baferu, čime se smanjuje vreme čekanja na čitanje sa diska. Zbog toga, skeniranje ne mora uvek da počne od početka datoteke. Sekvencijalno skeniranje je najefikasniji način za prolazak kroz celu tabelu ili veći deo nje. Drugim rečima, sekvencijalno skeniranje je efikasno kad je selektivnost niska. U situacijama sa većom selektivnošću, kada samo mali broj vrsti u tabeli ispunjava uslove filtera, bolje je koristiti indeksno skeniranje.Indeksno skeniranje(***eng.Index Scan***)[16] je tehnika pretraživanja podataka u tabeli na osnovu indeksa.Indeksno skeniranje(***eng.Index Scan***) se sastoji iz dva koraka, prvo se dobija lokacija vrste iz indeksa, a zatim se prikupljaju stvarni podaci iz heap-a ili stranica tabele. Dakle, svaki pristup indeksnom skeniranju zahteva dva čitanja. Ipak, ovo je jedan od najefikasnijih načina za dobijanje podataka iz tabele. Planer bira ovu metodu skeniranja kada je broj vrsti koje je potrebno izvući mali, tako da je izvođenje dvostepenih operacija indeksnog skeniranja "jefitnije" i brže od prikupljanja podataka putem pojedinačne obrade stranica tabele.Bitmap indeks skeniranje(***eng.Bitmap Index Scan***)[16] je metoda koju planer bira kada upit zahteva dovoljno veliku količinu podataka koja može iskoristiti prednosti masovnog čitanja, kao što je slučaj kod sekvencijalnog skeniranja, ali nije toliko velika da zahteva obradu cele tabele. Dakle, Bitmap indeks skeniranja(***eng.Bitmap Index Scan***) bi bilo nešto između sekvencijalnog i indeksnog skeniranja. Ova vrsta skeniranja uvek radi u paru sa Bitmap Heap skeniranjem(***eng.Bitmap Heap Scan***);prvo skeniranje pretražuje indeks kako bi pronašao sve odgovarajuće lokacije vrsti i formirao bit-mapu, a zatim drugo skeniranje koristi tu bit-mapu da pregleda stranice heap-a jednu po jednu i sakuplja vrste. Nakon generisanja troškova i procene koju metodu pretraživanja bi trebao da koristi za svaku tabelu, planer računa i troškove svake od metoda spajanja(opisane u poglavlju 5.2) i određuje koju je najbolje primeniti za spajanje kako bi kreirao optimalan plan izvršenja.To bi uključivalo kombinovanje svake metode spajanja sa svakom metodom pretrage podataka, što je jako skup proces. Taj proces se radi iz nekoliko nivoa, pri čemu se u prvom određuje najoptimalniji put u odnosu na procenu troškova za svaku tabelu ponaosob i bira neka od metoda. Zatim se u razmatranje uključuje procena troškova spajanja tabela i određuje koju metodu spajanja treba primeniti; ukoliko postoje dve tabele generisanje plana se tu završava, a ukoliko ne postupak

se dalje nastavlja dok se ne dostigne nivo jednak broju tabela. Jednom kada je optimalan plan izgenerisan, kreira se i stablo planiranja i prosleđuje izvršiocu(executor-u)[6].

Za praktičan prikaz funkcionisanja ovih načina spajanja tabela, kreirane su dve tabele:ZAPOSLENI i DEPARTMANI.Tabela ZAPOSLENI prikazuje zaposlene radnike u jednoj firmi, dok departmani prikazuju odgovorajući departman kojima zaposleni pripadaju .

Query Query History

```
1 SELECT *
2 FROM ZAPOSLENI;
```

Data Output Messages Notifications

	zaposleni_id [PK] integer	ime character varying (100)	deaprtman_id integer
1	2	Nikola Brankovic	2
2	3	Ivan Misic	5
3	4	Lara Krstic	3
4	5	Ivona Misic	4
5	6	Petar Zivic	4
6	7	Jovan Jovanovic	5
7	8	Luna Jaksic	4
8	1	Jovan Milic	1
9	10	Jovan Ilic	6
10	11	Masa Masic	10

Query Query History

```
1 SELECT *
2 FROM DEPARTMANI;
```

Data Output Messages Notifications

	departman_id [PK] integer	naziv character varying (100)
1	1	HR
2	2	Marketing
3	3	Finansije
4	4	Informacione tehnologije
5	5	Pravna služba
6	6	Operacije
7	7	Tehnicka sluzba

Slika 19-Prikaz tabela ZAPOSLENI i DEPARTMANI

5.3.1 Kreiranja upita koji koristi JOIN operator za spajanje tabela(Osnovno spajanje)

Za kreiranje ovakve vrste upita, koristi se JOIN operator, ispred koga stoji ključna reč koja govori o kom tipu spajanja se radi (CROSS,INNER,LEFT OUTER,RIGHT OUTER, FULL OUTER).Sama sintaksa upita zavisi od konkretnog tipa spajanja, međutim upiti uglavnog započinju naredbom SELECT , iza kojih sledi ključna reč FROM koja definiše tabele iz kojih se izdvajaju podaci i naposletku se specificira konkretan tip spajanja.

1.Kreiranje i obrada upita koji prikazuje unakrsno spajanje(***eng.CROSS JOIN***) i primer

U PostgreSQL-u, za predstavljanje unakrsnog tipa spajanja tabela koristi se operator JOIN ispred koga stoji ključna reč CROSS, koja označava da je reč o unakrsnom spajanju. Sintaksa ovakog upita može biti prikazana na sledeći način: SELECT ... FROM table1 CROSS JOIN table2 ...,[10] pri čemu SELECT naredba označava početak upita, dok … označavaju mesto namenjeno nekim dodatnim uslovima(WHERE,GROUP BY,HAVING,ORDER BY).FROM označava prvu tabelu iz koje se izvlače podaci, zatim sledi sama vrsta spajanja u ovom slučaju CROSS JOIN, i iza nje druga tabela koja će se koristiti za spajanje.

Primer kreiranja upita:

Query | Query History

```
1  SELECT *
2  FROM ZAPOSLENI
3  CROSS JOIN DEPARTMANI;
4
```

Slika 20-Prikaz kreiranja upita koji vrši unakrsno spajanje tabela

	zaposleni_id integer	ime character varying (100)	deaprtman_id integer	departman_id integer	naziv character varying (100)
1	2	Nikola Brankovic	2	1	HR
2	3	Ivan Misic	5	1	HR
3	4	Lara Krstic	3	1	HR
4	5	Ivona Misic	4	1	HR
5	6	Petar Zivic	4	1	HR
6	7	Jovan Jovanovic	5	1	HR
7	8	Luna Jaksic	4	1	HR
8	1	Jovan Milic	1	1	HR
9	10	Jovan Ilic	6	1	HR
10	11	Masa Masic	10	1	HR
11	2	Nikola Brankovic	2	2	Marketing
12	3	Ivan Misic	5	2	Marketing
13	4	Lara Krstic	3	2	Marketing
14	5	Ivona Misic	4	2	Marketing
15	6	Petar Zivic	4	2	Marketing
16	7	Jovan Jovanovic	5	2	Marketing
17	8	Luna Jaksic	4	2	Marketing
18	1	Jovan Milic	1	2	Marketing
19	10	Jovan Ilic	6	2	Marketing
20	11	Masa Masic	10	2	Marketing
21	2	Nikola Brankovic	2	3	Finansije
22	3	Ivan Misic	5	3	Finansije

	zaposleni_id integer	ime character varying (100)	deaprtman_id integer	departman_id integer	naziv character varying (100)
25	6	Petar Zivic	4	3	Finansije
26	7	Jovan Jovanovic	5	3	Finansije
27	8	Luna Jaksic	4	3	Finansije
28	1	Jovan Milic	1	3	Finansije
29	10	Jovan Ilic	6	3	Finansije
30	11	Masa Masic	10	3	Finansije
31	2	Nikola Brankovic	2	4	Informacione tehnologije
32	3	Ivan Misic	5	4	Informacione tehnologije
33	4	Lara Krstic	3	4	Informacione tehnologije
34	5	Ivona Misic	4	4	Informacione tehnologije
35	6	Petar Zivic	4	4	Informacione tehnologije
36	7	Jovan Jovanovic	5	4	Informacione tehnologije
37	8	Luna Jaksic	4	4	Informacione tehnologije
38	1	Jovan Milic	1	4	Informacione tehnologije
39	10	Jovan Ilic	6	4	Informacione tehnologije
40	11	Masa Masic	10	4	Informacione tehnologije
41	2	Nikola Brankovic	2	5	Pravna služba
42	3	Ivan Misic	5	5	Pravna služba
43	4	Lara Krstic	3	5	Pravna služba
44	5	Ivona Misic	4	5	Pravna služba
45	6	Petar Zivic	4	5	Pravna služba
46	7	Jovan Jovanovic	5	5	Pravna služba

	zaposleni_id integer	ime character varying (100)	deaprtman_id integer	departman_id integer	naziv character varying (100)
49	10	Jovan Ilic	6	5	Pravna služba
50	11	Masa Masic	10	5	Pravna služba
51	2	Nikola Brankovic	2	6	Operacije
52	3	Ivan Misic	5	6	Operacije
53	4	Lara Krstic	3	6	Operacije
54	5	Ivona Misic	4	6	Operacije
55	6	Petar Zivic	4	6	Operacije
56	7	Jovan Jovanovic	5	6	Operacije
57	8	Luna Jaksic	4	6	Operacije
58	1	Jovan Milic	1	6	Operacije
59	10	Jovan Ilic	6	6	Operacije
60	11	Masa Masic	10	6	Operacije
61	2	Nikola Brankovic	2	7	Tehnicka sluzba
62	3	Ivan Misic	5	7	Tehnicka sluzba
63	4	Lara Krstic	3	7	Tehnicka sluzba
64	5	Ivona Misic	4	7	Tehnicka sluzba
65	6	Petar Zivic	4	7	Tehnicka sluzba
66	7	Jovan Jovanovic	5	7	Tehnicka sluzba
67	8	Luna Jaksic	4	7	Tehnicka sluzba
68	1	Jovan Milic	1	7	Tehnicka sluzba
69	10	Jovan Ilic	6	7	Tehnicka sluzba
70	11	Masa Masic	10	7	Tehnicka sluzba

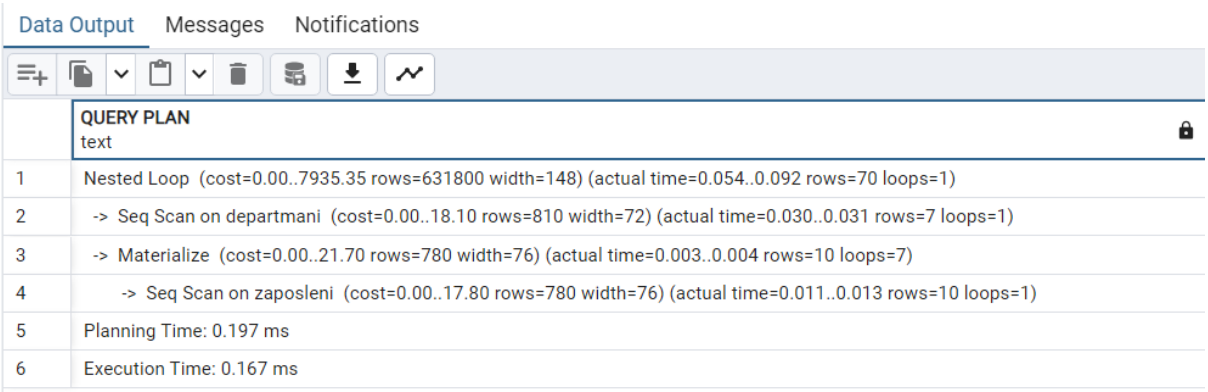
Slika 21-Prikaz rezulata izvršenja upita koji vrši unakrsno spajanje

U ovom konkretnom primeru, za tabele ZAPOLSENI i DEPARTMANI, unakrsno spajanje vratiće rezultujuću tabelu koja je uparila sve moguće vrste tabele “ZAPOLSENI” sa vrstama tabele DEPARTMANI, što dovodi do toga da rezultujuća tabela bude jako velika.Stoga, je ovaj tip spajanja pogodno koristiti samo ako je potrebno vratiti sve moguće kombinacije tabela ZAPOSLENI i DEPARTMANI.

- Obrada upita i generisanje optimalnog plana koji prikazuje unakrsno spajanje(***eng.CROSS JOIN***)

Prilikom obrade ovog upita, najpre se kreće kao i kod svakog drugog, od parser-a i analizatora-a, koji proveravaju sintaksne i semantičke greške, generišući stabla parsiranja i upita, redom, i ukoliko je sve u redu stablo upita prosleđuje se rewriter-u koji transforimše stablo upita prema unapred definisanim pravilima. Nakon rewriter-a, stablo se prosleđuje planer-u kako iz generisao optimalni plan izvršenja. PostgreSQL omogućuje detaljni pregled plana jednog upita korišćenjem konstrukcije EXPLAIN ANALYZE. Izvršavanjem ovih naredbi prikazuje se koje su metode iskorišćene za pretragu podataka,

i u ovom slučaju i za spajanje tabela.Optimalni plan ovog upita bi izgledao kako je prikazano na slici 22.



Slika 22-Prikaz plana izvršenja upita koji prikazuje unakrsno spajanje

Sa slike se može videti da je za spajanje tabela korišćen Nested Loop Join, koji i predstavlja najefikasniji način za spajanje kad je reč o korišćenju unakrsnog spajanja[17]. Čvor Nested Loop je mesto gde se izvršava algoritam(linija 1).Ovaj čvor uvek ima dva podčvora:gornji čvor je spoljašnja tabela(tabela DEPARTMANI), dok je donji čvor unutrašnja tabela(tabela ZAPOSLENI).Unutašnja tabela je u ovom slučaju predstavljena čvorom Materialize(linija 3), što znači , da kada se pozove, čvor čuva izlaz svog potčvora u RAM-u i zatim ga vraća.Prilikom daljih poziva, čvor vraća podatke iz memorije, izbgevajući time ponovna skeniranja tabele. Za skeniranje podataka je planer odabrao sekvencijalno skeniranje,kao najefikasniji način skeniranja, čiji je trošak u ovom slučaju, 17.80 jedinica i to predstavlja ukupni trošak, jer je početni trošak kod SeqScan-a jednak 0.Drugi parametar u liniji 2 u delu (cost=0.00..18.10 rows=810 width=72), označava broj vrsti koje SeqScan vraća, dok width označava veličinu vrsti u bajtovima.Nakon što su pribavljeni podaci iz unutrašnje tabele, Materialize omogućava štednju na daljim troškovima pribavljanja podataka. Zatim se vrši pribavljanje podataka skeniranjem spoljašnje tabele DEPARTMANI, pri čemu je trošak 18.10 jedinica. Kako se koristi konstrukcija EXPLAIN ANALYZE, moguće je videti i stvarno vreme izvršenja operacije. Za prvu operaciju skeniranja unutrašnje tabele (actual_time=0.011..0.013 rows=10 loops 1), prvi parametar označava da je potrebno vreme između 0.011 ms i 0.013 ms, da se skeniranjem vraćaju 10 vrsti i da je metoda izvršena 1 put.Kod drugog skeniranja, za drugu operaciju skeniranja(linija 2), vreme izvršenja je između 0.030 i 0.031, a vraćaju se 7 vrsti i metoda je izvršena jednom. Kada je reč, o trošku samog spajanja(linija 1,(cost=0.00..7935 rows=631800 width=148)), početni trošak je jednak zbiru početnih troškova njegovih podčvorova, a ukupni trošak je jednak zbiru:troška pribavljanja vrsti za spoljašnju tabelu(za svaku vrstu), trošku jednokratnog dohvatanja vrsti za unutrašnju tabelu(za svaku vrstu) i trošku obrade svake izlazne vrste, i u ovom slučaju iznosi 7935 jedinica.Procenjeno vreme izvršenja algoritma je između 0.054 i 0.092 milisekunde, a vreme planirano za izvršenje celog upita je 0.197ms, dok je vreme izvršenja je 0.167ms.

2.Kreiranje i obrada upita koji vrši unutrašnje spajanje(***eng.INNER JOIN***)

U slučaju kreiranja ovakvog upita, ispred JOIN operatora stoji ključna reč INNER, koja označava vrstu unutrašnjeg spajanja.Sintaksa ovakovog upita takođe započinje SELECT naredbom,sa razlikom što ovde SELECT naredba specifira koje kolone iz svih tabela koje se

spajaju se uključuju u rezultujući skup, i nakon nje ide FROM klauzula koja definiše iz koje tabele se izvlače podaci. Za njima sledi, INNER JOIN, koji spaja tabele.Ono što bitno razlikuje ovakav upit od prethodno kreiranog upita jeste definisanje uslova spajanja korišćenjem operatora ON.Sintaksa ovakvog upita bila bi sledeća[10]:

SELECT table1.column1, table2.column2...

FROM table1

INNER JOIN table2

ON table1.common_filed = table2.common_field;

Query Query History

```
1  SELECT ZAPOSLENI.ZAPOSLENI_ID, ZAPOSLENI.IME, DEPARTMANI.NAZIV
2  FROM ZAPOSLENI
3  INNER JOIN DEPARTMANI ON ZAPOSLENI.DEAPRTMAN_ID = DEPARTMANI.DEPARTMAN_ID;
4
5
```

Slika 23- Prikaz kreiranja upita koji vrši unutrašnje spajanje tabela

Data Output Messages Notifications

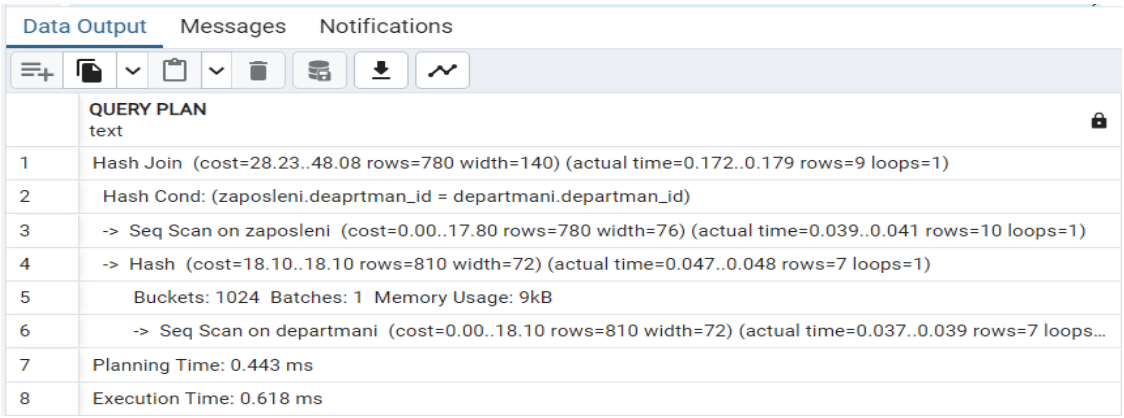
	zaposleni_id integer	ime character varying (100)	naziv character varying (100)
1	2	Nikola Brankovic	Marketing
2	3	Ivan Misic	Pravna služba
3	4	Lara Krstic	Finansije
4	5	Ivona Misic	Informacione tehnologije
5	6	Petar Zivic	Informacione tehnologije
6	7	Jovan Jovanovic	Pravna služba
7	8	Luna Jaksic	Informacione tehnologije
8	1	Jovan Milic	HR
9	10	Jovan Ilic	Operacije

Slika 24- Prikaz rezulata izvršenja upita koji vrši unutrašnje spajanje

U ovom konkretnom primeru, u okviru upita se nalaže, da se iz tabela izdvoje kolone koje se odnose na ID zaposlenog, ime zaposlenog i naziv departmana kome taj zaposleni pripada, a zatim se vrši spajanje dve tabele INNER JOIN-om, i na kraju se definiše uslov spajanja podataka koji kaže da bi trebalo spojiti one vrste iz tabele kojima je ID departmana u okviru tabele ZAPOSLENI identičan kao i id departmana u tabeli DEPRATMANI. Rezltujuća tabela samim tim ima samo tri kolone(ZAPOSLENI_ID,IME,NAZIV) i prikazuje spojene samo one vrste koje zadovoljavaju ovaj uslov spajanja.Dakle, INNER JOIN je pogodan za korišćenje ukoliko su potrebni samo rezultati gde se vrednosti podudaraju u obe tabele.

- Obrada upita i generisanje plana koji vrši unutrašnje spajanje(***eng.INNER JOIN***)

  Proces obrade upita je identičan kao i u prethodnom primeru, a generisani i kasnije izvršeni plan bi izgledao kao što je prikazano na slici 25.



Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=28.23..48.08 rows=780 width=140) (actual time=0.172..0.179 rows=9 loops=1)
2	Hash Cond: (zaposleni.deaprtman_id = departmani.departman_id)
3	-> Seq Scan on zaposleni (cost=0.00..17.80 rows=780 width=76) (actual time=0.039..0.041 rows=10 loops=1)
4	-> Hash (cost=18.10..18.10 rows=810 width=72) (actual time=0.047..0.048 rows=7 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on departmani (cost=0.00..18.10 rows=810 width=72) (actual time=0.037..0.039 rows=7 loops...
7	Planning Time: 0.443 ms
8	Execution Time: 0.618 ms

Slika 25-Prikaz plana izvršenja upita koji vrši unutrašnje spajanje

Za razliku od prethodnog upita, u okviru ovog upita tabele su spojene korišćenjem Hash Join-a[18], dok je za skeniranje odabrana od strane planera i iskorišćena, takođe metoda sekvencijalnog skeniranja, za obe tabele.Međutim, sada je izvršeno skeniranje najpre tabele DEPARTMANI koja predstavlja unutrašnju tabelu u ovom slučaju , dok tabela ZAPOSLENI predstavlja spoljašnju tabelu. Zatim , čvor Hash Join poziva čvor Hash(linija 4) koji pribavlja sve vrste tabele DEPARTMANI i organizuje ih u hash tabelu. Hash tabela čuva podatke kao parove ključ, vrednost, što činu pretragu ovih parova konstantnom i nezavisnom od veličine tabele. Hash funkcija raspoređuje ključeve nasumično i ravnomerno preko ograničenog broja “buckets”-a.Ukupan broja buckets-a jeste uvek 2 na stepen od N, a broj bucket-a za određeni hash ključ jeste poslednjih N bitova rezultata njegove hash funkcije.Tokom prvog koraka, hash join počinje skeniranjem svih vrsti unutrašnje tabele,što je u ovom slučaju tabela DEPARTMANI.Hash vrednost svake vrste se računa primenom hash funkcije na atribute spajanja(linija 2:Hash Cond),i sva polja iz jedne vrste koja su zahtevana se čuvaju u hash tabeli.Idealno, Hash join teži da pročita sve podatke u jednom prolazu, ali mu je potrebno dovoljno memorije da cela hash tabela stane odjednom. U ovom slučaju, alocirana memorija je bila dovoljna:hash tabela je zauzela 9KB, sa 1024 buckets-a.Ovo dozvoljava hash join-u da se izvrši u jednom prolazu.Nakon što je izgrađena hash tabela, Hash Join zove svoj drugi podređeni čvor kako bi pribavio spoljašni set(tabelu ZAPOSLENI, linija 3).Za svaku pribavljenu vrstu, hash tabela se pretražuje za odgovarajuću vrstu, pri čemu se izračunava hash vrednost za attribute spajanja(koristeći istu hash funkciju kao i ranije).Start-up trošak izvršenja Hash Join-a koštao je 28.43 dok je ukupan trošak koštao 48.08 jedinica, a vreme izvršenja je između 0.172 i 0.179.Izvršenje sekevencijalnog skeniranja koštalo je 18.10 jedinica, a trajalo je između 0.037 i 0.039 miliseknd za skeniranje tabele DEPARTMANI, dok je za skeniranje tabele ZAPOSLENI izvršenje trajalo između 0.039 i 0.041 miliseknd, a koštalo je 17.80 jedinica.Planirano vreme izvršenja upita bilo je 0.433ms, a vreme za koje je izvršeno je 0.618ms.

3.Kreiranje i obrada upita koji vrši levo spoljašnje spajanje(***eng.LEFT OUTER JOIN***)

Sintaksa upita koji za spajanja tabela koristi levo spoljašnje spajanje se ne razlikuje od sintakse upita koji koristi unutrašnje spajanje, osim što ispred JOIN operatora stoje ključne reči LEFT OUTER: SELECT ... FROM table1 LEFT OUTER JOIN table2 ON conditional_expression ...[10].U okviru ove sintakse na mestu … može stajati * koja označava da se biraju sve kolone ili se može odrediti koje tačno kolone će biti izbarane, dok

conditional_expression nakon operatora spajanja predslavlja uslov pod kojim će tabele biti spojene.

Query | Query History

```
1  SELECT ZAPOSLENI.ZAPOSLENI_ID, ZAPOSLENI.IME, DEPARTMANI.NAZIV
2  FROM ZAPOSLENI
3  LEFT OUTER JOIN DEPARTMANI ON ZAPOSLENI.DEAPRTMAN_ID = DEPARTMANI.DEPARTMAN_ID;
```

Slika 26- Prikaz kreiranja upita koji vrši levo spoljašnje spajanje tabela

Data Output | Messages | Notifications

	zaposleni_id integer	ime character varying (100)	naziv character varying (100)
1	2	Nikola Brankovic	Marketing
2	3	Ivan Misic	Pravna služba
3	4	Lara Krstic	Finansije
4	5	Ivona Misic	Informacione tehnologije
5	6	Petar Zivic	Informacione tehnologije
6	7	Jovan Jovanovic	Pravna služba
7	8	Luna Jaksic	Informacione tehnologije
8	1	Jovan Milic	HR
9	10	Jovan Ilic	Operacije
10	11	Masa Masic	[null]

Slika 27- Prikaz rezulata izvršenja upita koji vrši levo spoljašnje spajanje

LEFT OUTER JOIN vraća tabelu koja sadrži izabrane kolone iz obe tabele(i u ovom slučaju ZAPOSLENI_ID,IME,NAZIV), ali za razliku od INNER JOIN-a vraća i vrstu iz prve tabele(u ovom slučaju tabela ZAPOSLENI) koja nije zadovoljila uslov spajanja sa nekom vrstom iz druge tabele(u ovom slučaju tabela DEPARTMANI), sa null vrednostima u kolonama druge tabele(uokvireno crnim pravougaonikom),što i jeste suština korišćenja ovog tipa spajanja:da vrati sve podatke iz prve tabele, ne obazirući se na podatke iz druge tabele.

- Obrada upita i generisanje plana koji vrši levo spoljašnje spajanje spajanje(***eng.LEFT OUTER JOIN***)
  Proces obrade upita koji vrši levo spajanje se ne razlikuje značajmo od obrade upita koji vrši unutrašnje spajanje.Ono što je bitno naglasiti prilikom generisanja plana ovakvog upita,jeste da planer ukoliko postoji mogućnost transformiše spoljašnje spajanje u unutrašnje spajanje.Plan izvršenja korišćenjem EXPLAIN ANALYZE komande prikazan je na slici 28.

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Left Join (cost=28.23..48.08 rows=780 width=140) (actual time=0.057..0.061 rows=10 loops=1)
2	Hash Cond: (zaposleni.deaprtman_id = departmani.departman_id)
3	-> Seq Scan on zaposleni (cost=0.00..17.80 rows=780 width=76) (actual time=0.031..0.032 rows=10 loops=1)
4	-> Hash (cost=18.10..18.10 rows=810 width=72) (actual time=0.015..0.015 rows=7 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on departmani (cost=0.00..18.10 rows=810 width=72) (actual time=0.009..0.010 rows=7 loops...
7	Planning Time: 0.199 ms
8	Execution Time: 0.104 ms

Slika 28- Prikaz plana izvršenja upita koji vrši levo spoljašnje spajanje

I u ovom slučaju iskorišćena je Hash Join metoda, kao i prethodnom slučaju, tačnije iskorišćena je Hash Left Join metoda koja koristi hash tabelu za spajanje takođe, ali se leva tabela koristi kao osnovna, odnosno rezultujući skup sadrži sve vrste iz leve tabele(u ovom slučaju ZAPOSLENI) uz odgovarajuće parove iz desne tabele (tabela DEPARTMANI), ali vrstama iz desne kolone biće dodata null vrednost. Procenjeno vreme izvršenja ove metode jeste između 0.0.57 i 0.0.61 milisekundi, start-up trošak iznosi 28.23 jedinica, a totalo kost iznosi 48.08 jedinica. Za skeniranje tabela se takođe koristi sekvancijalno skeniranje. Planirano vreme izvršenja upita bilo 0.199 ms, a vreme za koje je upit zapravo izvršen je 0.104ms.

4.Kreiranje i obrada upita koji koristi vrši desno spoljašnje spajanje(***eng.RIGHT OUTER JOIN***)

Prilikom kreiranja ovakvog upita, sintaksa ostaje ista kao i kod LEFT OUTER JOIN-a, samo što se koriste ključne reči RIGHT OUTER ispred samog operatora: SELECT ... FROM table1 RIGHT OUTER JOIN table2 ON conditional_expression ...[10]

Query | Query History

```
1  SELECT ZAPOSLENI.ZAPOSLENI_ID, ZAPOSLENI.IME, DEPARTMANI.NAZIV
2  FROM ZAPOSLENI
3  RIGHT OUTER JOIN DEPARTMANI ON ZAPOSLENI.DEAPRTMAN_ID = DEPARTMANI.DEPARTMAN_ID;
```

Slika 29- Prikaz kreiranja upita koji vrši desno spoljašnje spajanje tabela

Data Output | Messages | Notifications

	zaposleni_id integer	ime character varying (100)	naziv character varying (100)
1	2	Nikola Brankovic	Marketing
2	3	Ivan Misic	Pravna služba
3	4	Lara Krstic	Finansije
4	5	Ivona Misic	Informacione tehnologije
5	6	Petar Zivic	Informacione tehnologije
6	7	Jovan Jovanovic	Pravna služba
7	8	Luna Jaksic	Informacione tehnologije
8	1	Jovan Milic	HR
9	10	Jovan Ilic	Operacije
10	[null]	[null]	Tehnicka sluzba

Slika 30- Prikaz rezulata izvršenja upita koji vrši desno spoljašnje spajanje

Nasuprot LEFT OUTER JOIN-u, RIGHT OUTER JOIN vraća sve vrste druge tabele,tj.tabele DEPARTMANI, čak i one koje ne zadovoljavaju uslove spajanja sa nekom od vrsti iz prve tabele(u ovom slučaju ZAPOSLENI), što je i cilj korišćenja ovog tipa spajanja:da prikaže sve podatke iz druge tabele.

- Obrada upita i generisanje plana koji vrši desno spoljašnje spajanje(***eng.RIGHT OUTER JOIN***)

  Obrada ovakvog upira razlikuje se od obrade prethodna dva upita, samo što je prilikom generisanja optimalnog plana iskorišćena metoda Hash Right Join,koja nalaže da je sada desna tabela(tabela DEPARTMANI) glavna, i da će se prikazivati sve vrste ove tabele zajedno sa odgovarajućim parovima iz leve tabele, dok će

vrstama iz desne tabele biti dodelje null vrednosti ukoliko nemaju para.Vreme izvršenja ovakvog upita je duže od upita koji vrši levo spajanje i iznosi 0.246 ms, iako je bilo planirano 0.257. Za skeniranje podataka obe tabele i u ovom slučaju korišćeno je sekvencijalno skeniranje.Plan obrade prikazan je na slici 31.

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Right Join (cost=28.23..48.08 rows=810 width=140) (actual time=0.179..0.192 rows=10 loops=1)
2	Hash Cond: (zaposleni.deaprtman_id = departmani.departman_id)
3	-> Seq Scan on zaposleni (cost=0.00..17.80 rows=780 width=76) (actual time=0.032..0.033 rows=10 loops=1)
4	-> Hash (cost=18.10..18.10 rows=810 width=72) (actual time=0.045..0.045 rows=7 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on departmani (cost=0.00..18.10 rows=810 width=72) (actual time=0.035..0.037 rows=7 loops...
7	Planning Time: 0.257 ms
8	Execution Time: 0.246 ms

Slika 31- Prikaz plana izvršenja upita koji vrši desno spoljašnje spajanje

5.Kreiranje i obrada upita koji vrši puno spoljašnje spajanje(***eng.FULL OUTER JOIN***)

Sintaksa upita koji za spajanje tabela koristi FULL OUTER JOIN: SELECT ... FROM table1 FULL OUTER JOIN table2 ON conditional_expression ...[10]

Query | Query History

```
1  SELECT ZAPOSLENI.ZAPOSLENI_ID, ZAPOSLENI.IME, DEPARTMANI.NAZIV
2  FROM ZAPOSLENI
3  FULL OUTER JOIN DEPARTMANI ON ZAPOSLENI.DEAPRTMAN_ID = DEPARTMANI.DEPARTMAN_ID;
```

Slika 32- Prikaz kreiranja upita koji vrši puno spoljašnje spajanje tabela

Data Output | Messages | Notifications

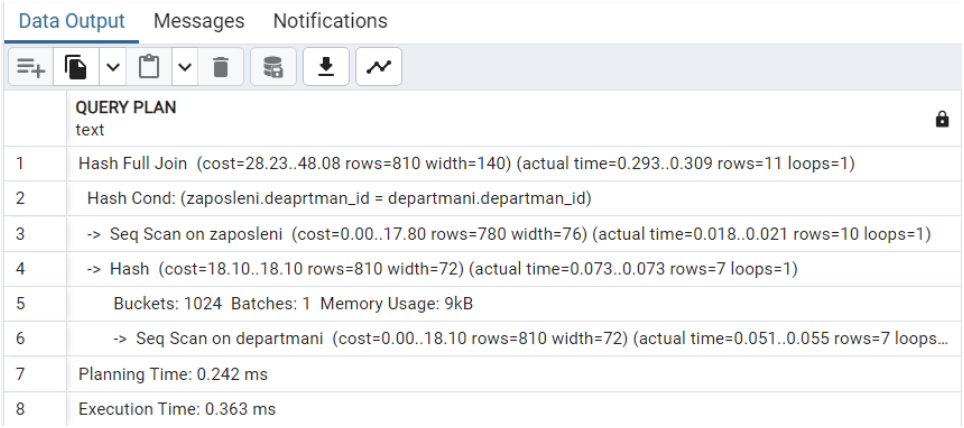
	zaposleni_id integer	ime character varying (100)	naziv character varying (100)
1	2	Nikola Brankovic	Marketing
2	3	Ivan Misic	Pravna služba
3	4	Lara Krstic	Finansije
4	5	Ivona Misic	Informacione tehnologije
5	6	Petar Zivic	Informacione tehnologije
6	7	Jovan Jovanovic	Pravna služba
7	8	Luna Jaksic	Informacione tehnologije
8	1	Jovan Milic	HR
9	10	Jovan Ilic	Operacije
10	11	Masa Masic	[null]
11	[null]	[null]	Tehnicka sluzba

Slika 33- Prikaz rezulata izvršenja upita koji vrši puno spoljašnje spajanje

Ovakav upit vratiće rezultujuću tabelu koja će sadržati i vrste iz prve tabele(u ovom slučaju ZAPOSLENI) koje nisu spojene ni sa jednom vrstom iz druge tabele(u ovom slučaju DEPARTMANI), i vrste iz druge tabele koje nisu spojene ni sa jednom vrstom iz druge tabele(na slici 26 uokvireno crnim pravougaonikom).

- Obrada upita koji vrši puno spoljašnje spajanje(***eng.FULL OUTER JOIN***)
  Za generisanje plana prilikom obrade ovakvog upita korišćena je Hash Full Join metoda.Ova metoda će prilikom kreiranja hash tabele, zadržati sve vrste iz obe tabele,

i ako se ne pronađu parovi za neku vrstu iz leve ili desne vrste, biće dodate null vrednosti u odgovarajućim kolonama te tabele; baš kao što funkcioniše i sama operacija punog spajanja.Dakle, FULL JOIN vraća vrste koje se ne podudaraju iz obe tabele, što znači da se on koristi ukoliko je potrebno da rezultujući skup uključuje nesparene vrste iz obe tabele.


Data Output | Messages | Notifications

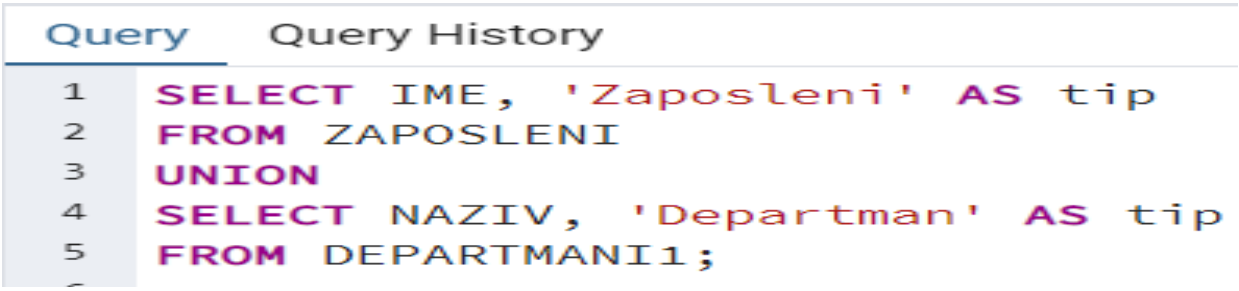
	QUERY PLAN text
1	Hash Full Join (cost=28.23..48.08 rows=810 width=140) (actual time=0.293..0.309 rows=11 loops=1)
2	Hash Cond: (zaposleni.deaprtman_id = departmani.departman_id)
3	-> Seq Scan on zaposleni (cost=0.00..17.80 rows=780 width=76) (actual time=0.018..0.021 rows=10 loops=1)
4	-> Hash (cost=18.10..18.10 rows=810 width=72) (actual time=0.073..0.073 rows=7 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on departmani (cost=0.00..18.10 rows=810 width=72) (actual time=0.051..0.055 rows=7 loops...
7	Planning Time: 0.242 ms
8	Execution Time: 0.363 ms

Slika 34- Prikaz plana izvršenja upita koji vrši puno spoljašnje spajanje
Na slici 34 prikazan je plan obrade upita. I ovde je kao i u prethodnom primeru(desno spoljašnje spajanje) vreme izvršenja duže (0.363ms) nego planirano vreme izvršenja(0.242ms).Za skeniranje podataka obe tabele i u ovom slučaju korišćeno je sekvencijalno skeniranje.

5.3.2 Kreiranje upita koji koristi UNION operator za spajanje tabela(Kreiranje unije)

Za spajanje dve ili više tabela, kako je već opisano u prethodnom potpoglavlju, moguće je kreirati i uniju ovih tabela, a za kreiranje unije u PostgreSQL se koristi operator UNION.Sintaksa ovakvog upita ima sledeći oblik[19]:

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

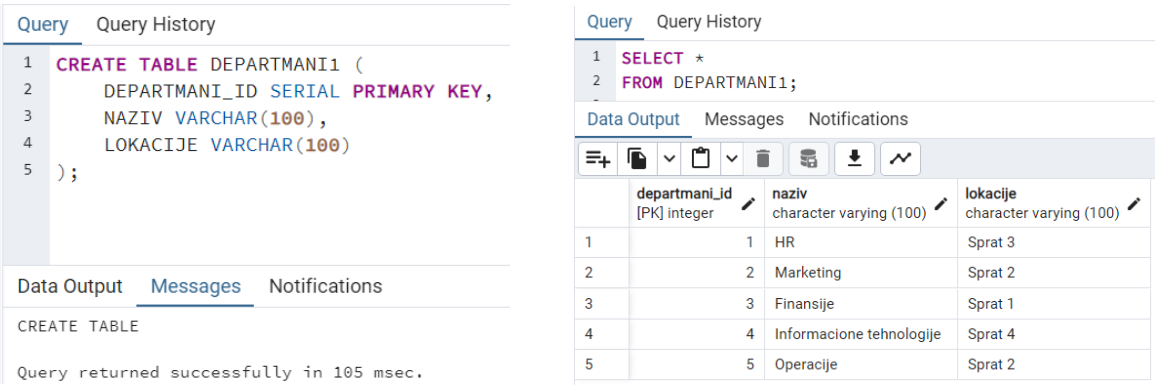

Query | Query History

```
1  SELECT IME, 'Zaposleni' AS tip
2  FROM ZAPOSLENI
3  UNION
4  SELECT NAZIV, 'Departman' AS tip
5  FROM DEPARTMANI1;
```

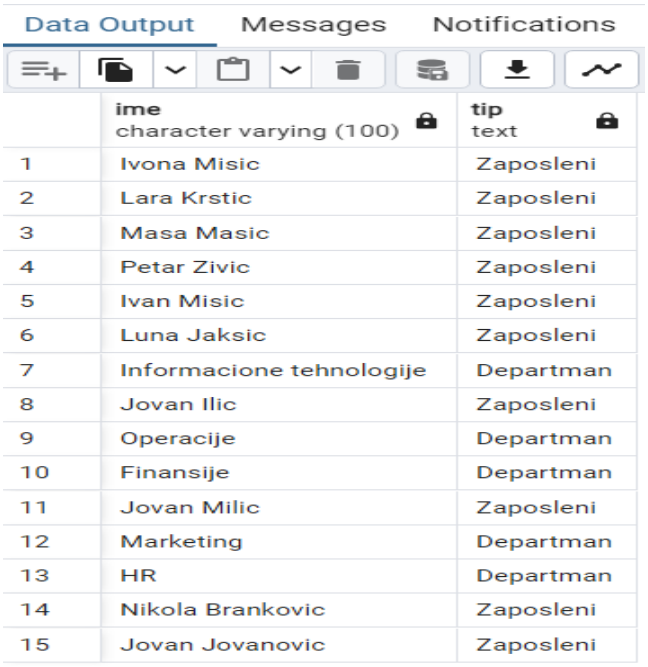
Slika 35-Prikaz kreiranja upita koji koristi operator UNION za spajanje tabela

Ovaj upit zadovoljava sintaksu koja je gore navedena i kombinuje rezultate iz dve različite tabele:tabele ZAPOSLENI i tabele DEPARTMANI1. Prvom SELECT naredbom izdvaja kolone , kao što je IME iz tabele ZAPOSLENI, ali dodaje i novu kolonu “tip” koja će sadržati vrednost Zaposleni za svaku vrstu iz tabele ZAPOSLENI. Zatim se koristi ključna reć UNION koja će kombinovati rezultate iz prve SELECT naredbe(koji predstavljaju zaposlene), sa rezultatima druge SELECT naredbe (koji predstavljaju departmane). Druga

SELECT naredba bira takođe kolone ali iz tabele DEPARTMANI1,i takođe dodaje dodatnu kolonu “tip” koja će sadržati vrednost Departman za svaku vrstu iz tabele DEPARTMANI1. Ono što je važno napomenuti jeste da unija radi sa tabela koje imaju isti broj kolona i sadrže podatke koje su istog tipa, te je zato kreirana tabela DEPARTMANI1, koja pored kolona ID-ja i naziva departmana, sadrži i kolonu za lokaciju.



Slika 36-Prikaz kreiranja tabela DEPARTMANI1

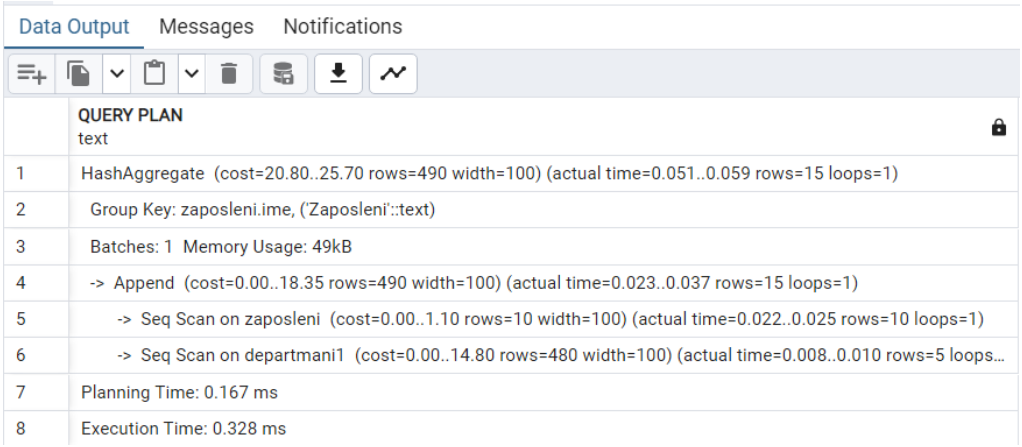


Slika 37-Prikaz rezultata izvršenja unije nad tabelama ZAPOSLENI i DEPARTMAN1

Unije će vratiti rezultujuću tabelu koja će imati dve kolone, kolonu ime koja će sadržati ime zaposlenog ili departmana i tip, i koja će sadržati sve vrednosti iz tabele ZAPOSLENI i sve vrednosti iz tabele DEPARTMANI1.

- Obrada upita i generisanje plana koji koristi UNION operator
  U ovom slučaju obrada upita je ista u prve dve faze, ali se prilikom generisanja plana ne koristi nijedna metoda spajanja već posebna tehnika koja omogućava agregaciju, Hash aggregate[20]. Ovaj proces, kao i Hash Join uključuje kreiranje hash tabele za upit.Hash aggregate zahteva operator agregacije ili grupni ključ kolone. Dok

PostgreSQL prolazi kroz svaku vrstu, on hešira ključeve po grupnom ključu(linija 2), a odgovarajuća vrsta se smešta prema izračunatoj vrednosti ključa. Vrednosti koje pripadaju istom hash-u smeštaju u isti “bucket”(koristi se na identičan način kao i kod Hash Join-a).Kada su sve neophodne vrste obrađene, PostgreSQL računa konačni prosek za svaku grupu koristeći podatke iz ove hash tabele, a zatim prikazuje te rezultate.Početni trošak ove tehnike je 20.80 jedinica, dok je totalni 25.70 jedinica, a vreme za koje se izvrši jeste između 0.051 i 0.059 milisekundi.



	QUERY PLAN text
1	HashAggregate (cost=20.80..25.70 rows=490 width=100) (actual time=0.051..0.059 rows=15 loops=1)
2	Group Key: zaposleni.ime, ('Zaposleni'::text)
3	Batches: 1 Memory Usage: 49kB
4	-> Append (cost=0.00..18.35 rows=490 width=100) (actual time=0.023..0.037 rows=15 loops=1)
5	-> Seq Scan on zaposleni (cost=0.00..1.10 rows=10 width=100) (actual time=0.022..0.025 rows=10 loops=1)
6	-> Seq Scan on departmani1 (cost=0.00..14.80 rows=480 width=100) (actual time=0.008..0.010 rows=5 loops...
7	Planning Time: 0.167 ms
8	Execution Time: 0.328 ms

Slika 38-Prikaz plana izvršenja upita koji koristi operator UNION

I ovde su je za skeniranje podataka obe tabele iskorišćeno sekvencijalno skeniranje.Planirano vreme izvršenja je 0.167ms, a vreme izvršenja samo upita je 0.328ms.

5.3.3 Kreiranje podupita koji se koristi za spajanje tabela

Podupiti, kako je već rečeno, predstavljaju upite u okviru jednog većeg upita, a opšta sintaksa koja se koristi za kreiranje podupita ima sledeći oblik:

```
SELECT
  select_list
FROM
  table1
WHERE
  columnA operator (
    SELECT
      columnB
    from
      table2
    WHERE
      condition
  );
```

I ovde se kao i kod većine upita, prvo u okviru SELECT naredbe koriste kolone koje bi se trebale vratiti ,nakon čega se FROM klauzulom specificiraju tabele iz kojih se vraćaju rezultati. WHERE klauzulom se definiše uslov koji bi trebalo da bude ispunjen, međutim umesto da se "columnA" direktno uporedi sa nekom vrednošću kreira se podupit, kako bi vratio vrednost iz neke druge tabele sa kojom će se "columnA" porediti.

Podupit koji bi spajao dve tabele izgledao bi ovako:

Query Query History

```
1 SELECT * FROM ZAPOSLENI AS Z, (SELECT * FROM DEPARTMANI) as D WHERE Z.DEAPRTMAN_ID = D.DEPARTMAN_ID;
```

Slika 39-Prikaz kreiranja podupita koji vrši spajanje tabela

U ovom primeru, se u okviru podupita nalaže, korišćenjem komande SELECT vraćanje svih kolona tabele DEPARTMANI a rezultati se dodeljuju kao priveremena tabela definisana sa D. Glavni upit vrši selektovanje podataka iz tabele ZAPOSLENI, a zatim se uslovom spajanja definiše način na koji se ove tabele spajaju(WHERE klauzulom).Rezultujuća tabela će imati kolone iz obe tabele, spojene vrstama koje se poklapaju u tabelama.

Podupiti mogu biti korisni,kako je ranije pomenuto, prilikom spajanja tabela koje imaju komplikovane ili ugnjždene odnose, ili kada je potrebno izvršiti filtriranje ili grupisanje na osnovu vrednosti u drugoj tabeli.Oni takođe mogu olakšati upit i učiniti ga lakšim za čitanje i razumevanje.

Data Output Messages Notifications

	zaposleni_id integer	ime character varying (100)	deaprtman_id integer	departman_id integer	naziv character varying (100)
1	2	Nikola Brankovic	2	2	Marketing
2	3	Ivan Misic	5	5	Pravna služba
3	4	Lara Krstic	3	3	Finansije
4	5	Ivona Misic	4	4	Informacione tehnologije
5	6	Petar Zivic	4	4	Informacione tehnologije
6	7	Jovan Jovanovic	5	5	Pravna služba
7	8	Luna Jaksic	4	4	Informacione tehnologije
8	1	Jovan Milic	1	1	HR
9	10	Jovan Ilic	6	6	Operacije

Slika 40-Prikaz rezultata podupita koji vrši spajanje tabela

- Obrada podupita i generisanje plana izvršenja

Prilikom obrade podupita, u fazi planiranja, ukoliko podupit ne sadrži klauzule poput, GROUP BY, HAVING, ORDERD BY, LIMIT ili DISTINCT(kao što je slučaj sa upitom prikazanim ovde), planer će tranformisati podupit na sledeći način[6]:

Query Query History

```
1 SELECT * FROM ZAPOSLENI AS Z, (SELECT * FROM DEPARTMANI) as D WHERE Z.DEAPRTMAN_ID = D.DEPARTMAN_ID;
```

Query Query History

```
1 EXPLAIN ANALYZE SELECT * FROM ZAPOSLENI AS Z, DEPARTMANI AS D WHERE Z.DEAPRTMAN_ID=D.DEPARTMAN_ID;
2
```

Slika 41-Prikaz transformacije podupita tokom faze planiranja

Nakon, toga i za ovakav upit se generiše plan, na sličan način kao i kod prethodnih primera.Najpre se vrši procena troškova i izbor metoda skeniranja, i na kraju se vrši procena troškova metoda spajanja i bira najjeftinija.Plan ovog podupita prikazan je na slici 38.

Data Output | Messages | Notifications

	QUERY PLAN text
1	Nested Loop (cost=0.15..196.37 rows=780 width=148) (actual time=0.043..0.062 rows=9 loops=1)
2	-> Seq Scan on zaposleni z (cost=0.00..17.80 rows=780 width=76) (actual time=0.021..0.023 rows=10 loops=1)
3	-> Index Scan using departmani_pkey on departmani d (cost=0.15..0.23 rows=1 width=72) (actual time=0.003..0.003 rows=1 loops=10)
4	Index Cond: (departman_id = z.deaprtman_id)
5	Planning Time: 0.203 ms
6	Execution Time: 0.092 ms

Slika 42-Prikaz plana izvršenja podupita

I ovde je za spajanje tabela, kao najoptimalnija metoda, od strane planera odabrana metoda Nested Loop Join, koji takođe ima dva podčvora koji se koriste za skeniranje i pribavljanje podataka, međutim u okviru ovog podupita iskorišćen je za skeniranje unutrašnje tabele Index Scan.Unutrašnja tabela je sada DEPARTMANI. Indeksno skeniranje je izvršeno na primarnom ključu tabele DEPARTMANI. Početni trošak je u ovom slučaju 0.15 dok je krajnji 0.23, a vreme izvršenja je 0.003 ms.Očekuje se da će ovo skeniranje vratiti 1 vrstu širine width, što i jeste slučaj, jer je je operacija skeniranja izvršena 10 puta pri čemu je svaki put vraćena samo jedna vrsta.U liniji 4, naveden je Index Cond koji predstavlja uslov koji se koristi za pronalaženja lokacije vrsti iz indeksa.Konkretno, korišćenjem ovog uslova se navodi da se izflitriraju vrste iz tabele DEPARTMANI koji se podudaraju sa vrednošću “DEPARTMAN_ID” iz tabele ZAPOSLENI. Skeniranje spoljašnje tabele izvršeno je sekvencijalnim skeniranjem, kao i u prethodnim primerima. Takođe, početni i ukupni trošak metode spajanja(Nested Loop) se računaju kao i u prvom primeru.Planirano vreme izvršenja jeste 0.203ms, a vreme izvršenja je 0.092ms.

Ovim praktičmim primerima, prikazano je kako izgleda kreiranje različitih upita koji se koriste za spajanje tabela i kako izgleda postupak obrade ovakvog upita, pogotovu generisanje plana.EXPLAIN ANALYZE-om je prikazano na koj način je za svaki od ovih načina izvršeno spajanje tabela ali i skeniranje podataka. Upoređivanjem, vremena izvršenja, ustanovljeno je da se spajanje tabela ZAPOSLENI i DEPARTMANI, najbrže izvršilo korišćenjem CROSS JOIN-a, a najsporije INNER JOIN-a.Međutim, kako je već rečeno , najefikasniji način zavisi od toga šta se želi postići spajanjem tabela. Recimo, i pored jednostvanosti i citljivosti podupita, ponekad će biti potrebno transformisati podupit, u upit koji spajanje vrši nekim operatorom spajanja, najčešće unutrašnje spajanje. S druge strane, nekada će biti potrebno i samo spoljašnje spajanje tranformisati u unutrašnje spajanje.

Kreiranjem posebne tabele, kako bi bilo moguće, izvršiti uniju pokazano je da je uniju moguće primeniti za spajanje tabela samo ukoliko tabele imaju isti broj kolona i sadrže podatke istog tipa(tabele DEPARTMANI1 i ZAPOSLENI).

5.3.4 Kreiranje upita koji spaja tri tabele kombinacijom INNER JOIN-A i podupita

Svi dosada opisani i prikazani primeri, predstavljaju korišćenje nekog od načina spajanja na primeru dve tabele.Međutim, nekada je moguće i kombinovati načine spajanja.U nastavku će biti prikazan primer upita koji spaja tri tabele.Za potrebe kreiranja ovakvog upita, kreirana je tabele PROJEKTI3 koja sadrži informacije o projektima koji se izvode u određenom DEPARTMANU i ima kolone NAZIV I STATUS koje predstavljaju osnovne informacije o projektu, dok polje DEPARTMAN_ID označava departman koji je zadužen za taj projekat.

Query | Query History

```
CREATE TABLE PROJEKTI3 (
    PROJEKAT_ID SERIAL PRIMARY KEY,
    NAZIV VARCHAR(100),
    DEPARTMAN_ID INT,
    STATUS VARCHAR(50)
);
INSERT INTO PROJEKTI3 (NAZIV, DEPARTMAN_ID, STATUS) VALUES
    ('Izgradnja web aplikacije', 4, 'U toku'),
    ('Redizajn korisnickog interfejsa', 4, 'Zavrsen'),
    ('Raspisvanje konkursa za zaposljavanje', 1, 'U pripremi'),
    ('Istrazivanje trzista', 3, 'U toku'),
    ('Unapredjenje sistema za upravljanje lagerom', 2, 'Zavrsen');
```

Slika 43-Prikaz kreiranja tabele PROJEKTI3

Upit koji bi spojio ove tri tabele zasniva se na INNER JOIN-U i podupitu:

Query | Query History | Scratch Pad

```
SELECT * FROM ZAPOSLENI AS Z
INNER JOIN (
    SELECT *
    FROM DEPARTMANI
    ORDER BY DEPARTMAN_ID
) AS D ON Z.DEAPRTMAN_ID = D.DEPARTMAN_ID
INNER JOIN (
    SELECT *
    FROM PROJEKTI3
    ORDER BY DEPARTMAN_ID
) AS P ON D.DEPARTMAN_ID = P.DEPARTMAN_ID
ORDER BY P.DEPARTMAN_ID;
```

Data Output | Messages | Notifications

	zaposleni_id integer	ime character varying (100)	deaprtman_id integer	departman_id integer	naziv character varying (100)	projekat_id integer	naziv character varying (100)	departman_id integer	status character varying (50)
1	12	Jovan Milic	1	1	HR	3	Raspisvanje konkursa za zaposljavanje	1	U pripremi
2	2	Nikola Brankovic	2	2	Marketing	5	Unapredjenje sistema za upravljanje lagerom	2	Zavrsen
3	4	Lara Krstic	3	3	Finansije	4	Istrazivanje trzista	3	U toku
4	5	Ivona Misic	4	4	Informacione tehnologije	1	Izgradnja web aplikacije	4	U toku
5	6	Petar Zivic	4	4	Informacione tehnologije	2	Redizajn korisnickog interfejsa	4	Zavrsen
6	5	Ivona Misic	4	4	Informacione tehnologije	2	Redizajn korisnickog interfejsa	4	Zavrsen
7	8	Luna Jaksic	4	4	Informacione tehnologije	2	Redizajn korisnickog interfejsa	4	Zavrsen
8	8	Luna Jaksic	4	4	Informacione tehnologije	1	Izgradnja web aplikacije	4	U toku
9	6	Petar Zivic	4	4	Informacione tehnologije	1	Izgradnja web aplikacije	4	U toku

Slika 44-Prikaz kreiranja upita koji spaja tri tabele

U okviru ovog upita, spajanje tabele ZAPOSLENI,DEPARTMANI I PROJEKTI3 je izvršeno korišćenjem INNER JOIN-a,dok se podupit koristi kako bi se tabele “DEPARTMANI” i “PROJEKTI3” sortirale pre spajanja sa tabelom ZAPOSLENI. Prvi podupit bira sve kolone

iz tabele DEPARTMANI i zatim ih sortira po primarnom ključu, a zatim se korišćenjem INNER JOIN-a vrši spajanje rezultata podupita sa tabelom zaposleni.Drugi upit takođe vraća sve kolone tabele PROJEKTI3 i sortira ih, da bi se na kraju drugim JOIN-om, spojili rezultati podupita i rezultati prethodnog JOIN-a.Na kraju, rezultat JOIN-a se sortira jedinstvenom ključu tabele DEPRTMANI.

Generisanje plana ovakvog upita, je slično kao i kada se radi sa dve tabele, osim što se sada najpre vrši izbor najjeftinijeg puta za sve tri tabele ponaosob, pa se zatim vrši izbor najjeftinijeg puta spajanja za svaku kombinaciju ove tri tabele, da bi se na kraju odbarao najjeftiniji put[6].

Data Output | Messages | Notifications

	QUERY PLAN text
1	Merge Join (cost=3.60..3.70 rows=4 width=267) (actual time=0.189..0.197 rows=9 loops=1)
2	Merge Cond: (departmani.departman_id = z.deaprtman_id)
3	-> Sort (cost=1.17..1.19 rows=7 width=72) (actual time=0.043..0.044 rows=5 loops=1)
4	Sort Key: departmani.departman_id
5	Sort Method: quicksort Memory: 25kB
6	-> Seq Scan on departmani (cost=0.00..1.07 rows=7 width=72) (actual time=0.030..0.031 rows=7 loops=1)
7	-> Sort (cost=2.43..2.44 rows=5 width=195) (actual time=0.143..0.144 rows=9 loops=1)
8	Sort Key: z.deaprtman_id
9	Sort Method: quicksort Memory: 25kB
10	-> Hash Join (cost=1.18..2.37 rows=5 width=195) (actual time=0.090..0.097 rows=9 loops=1)
11	Hash Cond: (z.deaprtman_id = projekti3.departman_id)
12	-> Seq Scan on zaposleni z (cost=0.00..1.10 rows=10 width=76) (actual time=0.014..0.015 rows=10 loops=1)
13	-> Hash (cost=1.12..1.12 rows=5 width=119) (actual time=0.034..0.034 rows=5 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB
15	-> Sort (cost=1.11..1.12 rows=5 width=119) (actual time=0.027..0.027 rows=5 loops=1)
16	Sort Key: projekti3.departman_id
17	Sort Method: quicksort Memory: 25kB
18	-> Seq Scan on projekti3 (cost=0.00..1.05 rows=5 width=119) (actual time=0.014..0.015 rows=5 loop...
19	Planning Time: 0.532 ms
20	Execution Time: 0.248 ms

Slika 45-Prikaz generisanog plana izvršenja za upit koji spaja tri tabele

Prilikom generisanja plana za spajanje tabela ZAPOSLENI i PROJEKTI3 planer je odabro Hash Join metodu kao najoptimalniju.Spajanje je izvršeno na sličan način kao i kod primera upita koji vrši unutrašnje spajanje,pri čemu je tabela PROJEKTI3 unutrašnja,a tabela ZAPOSLENI spoljašnja tabela. I u ovom slučaju, alocirana memorija je bila dovoljna:hash tabela je zauzela 9KB, sa 1024 buckets-a, što dozvoljava hash join-u da se izvrši u jednom prolazu.Za skeniranje podataka obe tabele izabrano je sekvencijalno skeniranje.Međutim, tabele DEPARTMANI i PROJEKTI3 spojene su Merge Join-om, pri čemu je uslov po kome se tabele spajaju takođe da primarni ključ deprtmana u tabeli DEPARTMAN i tabeli PROJEKTI3 , budu isti.I u ovom slučaju je za skeniranje podataka izabrano sekvencijalno skeniranje. Ono što je bitno istaći, da zbog ORDER BY klauzule, sve tri tabele su sortirane prema DEPARTMAN_ID-u korišćenjem operatora Sort, a metoda koja je iskorišćena jeste quicksort jer se podaci koji se sortiraju uklapaju u alociranu memoriju koja u sva tri slučaja iznosi 25KB. Ukupno vreme izvršenja je 0.248ms, dok je planirano vreme bilo 0.532ms.

Prethodni primer je samo jedan od načina na koji je moguće izvšiti spajanje tri ili više tabela.Moguće je koristiti samo jednu join operaciju za spajanje sve tri tabele ili kombinovati operacije, moguće je kreirati ih samo podupitima, ili kombinacijom operacija osnovnog spajanja i podupita kako je i ovde prikazano.

6.ZAKLJUČAK

Proces obrade upita kod svih relacionih baza podataka, pa tako i kod PostgreSQL-a , predstavlja vrlo važan element, jer upravo razumevanje koncepta obrade upita, omogućava efikasno korišćenje same baze podataka.Posebno je značajan proces obrade složenijih upita koji vrše spajanje tabela, jer oni omogućavaju da se korišćenjem i kreiranjem samo jednog upita pribave i koriste podaci iz većeg broja tabela.

U seminarskom radu, opisan je najpre postupak obrade upita kod relacionih baza podataka, dajući osvrt i na process obrade upita koji se koriste prilikom spajanja tabela. Zatim je opisana sama PostgreSQL baza podataka i njena svojstva i najznačajnije karakteristike. Takođe je opisan postupak obrade upita kod PostgreSQL sistema, detaljno objašnjavajući svaki od koraka koji se preduzimaju u toku ovog procesa. Posebna pažnja je posvećena obradi složenih upita koji se koriste za samo spajanje tabela u PostgreSQL-u. Opisani su načini na koji je moguće spojiti tabele, kao i koje se metode koriste od strane POstgreSQL-a kako bi se tabele fizički spojile.

Naposletku, pregledom seminarskog rada i poređenjem rezultata izvršenja praktične realizacije opisanih načina spajanja, može se ustanoviti važnost pravilnog izbora određenog načina za spajanje tabela, kao i značaj razumevanja koncepta spajanja tabela, a posebno obradi upita koji to omogućavaju.

7.KORIŠĆENA LITERATURA

[1] Michael L. Rupley, Jr., “Introduction to Query Processing and Optimization”,Indiana University at South Bend

Dostupno na: https://clas.iusb.edu/computer-science-informatics/research/reports/TR-20080105-1.pdf

[2]” Exploring the Different Join Methods in Relational Database Systems”

Dostupno na: https://www.dwhpro.com/exploring-join-methods-in-relational-database-systems/

[3]PostgreSQL About, The PostgreSQL Global Development Group 1996–2024.

Dostupno na: https://www.postgresql.org/about/

[4] Sharath Punreddy, “Understanding the Fundamentals of PostgreSQL Architecture”

Dostupno na: https://www.instaclustr.com/blog/postgresql-architecture/

[5] PostgreSQL Tutorial,Java T Point

Dostupno na : https://www.javatpoint.com/postgresql-tutorial

[6] H. Suzuki, "The Internals of PostgreSQL for database administrators and system developers",2019.

Dostupno na : https://www.interdb.jp/pg/index.html

[7] PostgreSQL 16 Documentation -The Rule System(The Query Tree), The PostgreSQL Global Development Group 1996–2024.

Dostupno na: https://www.postgresql.org/docs/current/querytree.html

[8] “Understanding How PostgreSQL Executes a Query”

Dostupno na: https://etutorials.org/SQL/Postgresql/Part+I+General+PostgreSQL+Use/Chapter+4.+Performance/Understanding+How+PostgreSQL+Executes+a+Query/

[9] “Understanding Query Processing in PostgreSQL: Part 1”

Dostupno na: https://dev.to/fatemasamir/understanding-query-processing-in-postgresql-part-1-45h7

[10] "PostgreSQL - JOINS"

Dostupno na: https://www.tutorialspoint.com/postgresql/postgresql_using_joins.html

[11]”UNION VS JOIN”

Dostupno na: https://www.tutorialspoint.com/sql/sql-union-vs-join.htm

[12]” PostgreSQL Subquery”

Dostupno na: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-subquery/

[13]Dostupno na: https://www.mssqltips.com/sqlservertip/2115/understanding-sql-server-physical-joins/

[14] "Understanding Merge Joins"

Dostupno na: https://learn.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms190967(v=sql.105)

[15] "Queries in PostgreSQL: 3. Sequential scan"

Dostupno na: https://postgrespro.com/blog/pgsql/5969403

[16] "One Index, Three Different PostgreSQL Scan Types"

Dostupno na: https://www.percona.com/blog/one-index-three-different-postgresql-scan-types-bitmap-index-and-index-only/

[17] "Queries in PostgreSQL: 5. Nested loop"

Dostupno na: https://postgrespro.com/blog/pgsql/5969618

[18]" Queries in PostgreSQL: 6. Hashing"

Dostupno na: https://postgrespro.com/blog/pgsql/5969673

[19]" SQL UNION Operator"

Dostupno na: https://www.w3schools.com/sql/sql_union.asp

[20]" Understanding Hash aggregates and Hash Joins in PostgreSQL"

Dostupno: https://stormatics.tech/blogs/understanding-hash-aggregates-and-hash-joins-in-postgresql