UNIVERZITET U NISU

ELEKTRONSKI FAKULTET

**REPLIKACIJA KOD POSTGRESQL BAZE PODATAKA**

Seminarski rad

Studijski program: Racunarstvo i informatika

Modul: Softversko inzenjerstvo

Mentor: Student:

Prof. dr Aleksandar Stanimirovic Zeljko Vasic, br.ind. 1808

Sadrzaj

[1. Uvod u replikaciju 3](#_Toc5467)

[1.1. Faze replikacije 3](#_Toc12607)

[1.2. WAL(Write Ahead Log) 4](#_Toc13739)

[2. Tipovi replikacije 4](#_Toc20752)

[2.1. Asinhrona replikacija 4](#_Toc12820)

[2.2. Sinhrona replikacija 5](#_Toc11742)

[2.3. Hibridna replikacija 5](#_Toc29871)

[3. Vrste replikacije u PostgreSQL-u po broju master cvorova 6](#_Toc28066)

[3.1. Single-Master Replikacija 6](#_Toc26312)

[3.2. Multi-Master Replikacija 10](#_Toc10950)

[3.2.1. Strategije resavanja konflikata u multi-master sistemima 11](#_Toc32301)

[3.3. Masterless Replikacija 12](#_Toc23749)

[4. Vrste replikacije u PostgreSQL-u po nacinu prenosa podataka 12](#_Toc20933)

[4.1. Fizicka replikacija 12](#_Toc11996)

[4.2. Logicka replikacija 13](#_Toc22533)

[4.3. Snapshot Replikacija 14](#_Toc32042)

[5. HA setup 14](#_Toc17929)

[5.1. Load Balancer 14](#_Toc13301)

[5.2. Automatski failover 15](#_Toc31818)

[5.3. Monitoring i pracenje replikacije 15](#_Toc27115)

[5.3.1. View pg\_stat\_replication 17](#_Toc5719)

[5.3.2. View pg\_stat\_wal\_receiver 17](#_Toc5529)

[6. Replikacija u oblaku 18](#_Toc7412)

[7. Zakljucak 21](#_Toc32623)

[8. Literatura 22](#_Toc5770)

# Uvod u replikaciju

Replikacija je proces kopiranja podataka i njihove sinhronizacija izmedju vise baza podataka. Kod replikacije se podaci kopiraju iz jedne baze podataka u drugu i time se minimizira kolicina izgubljenih podataka prilikom pada neke baze, a ostale baze podataka su spremne da obrade zahteve koji bi inace bili izgubljeni jer su upuceni bazi koja je nedostupna. Baze podataka mogu biti smestene na istom serveru ali i da budu na potpuno razlicitim geografskim lokacijama. Replikacija moze postojati i u centralizovanom sistemu i u distirbuiranom sistemu. S obzirom da je za sisteme koji skladiste ogromne kolicine podataka jedna centralizovana baza podataka nedovoljna sa neometano funkcionisanje sistema, sve vise se koriste distribuirane baze podataka. Replikacija je osnovni nacin odrzavanja konzistentnosti izmedju tih distribuiranih baza odnosno cvorova. Sustina replikacije kod distribuiranih baza podataka je da obezbedi da svaka kopija poseduje identican skup podataka kao i master baza, uz sto manju vremensku razliku. Na taj nacin ako se desi kvar na glavnoj bazi, druga baza moze preuzeti njenu ulogu i korisnici nece imati problema sa funkcionisanjem sistema.

Cilj replikacije kod centralizovanih baza podataka je High Availability i backup podataka, a cilj replikacije kod distribuiranih baza podataka je skalabilnost i distirbucija.

PostgreSQL je dobar primer za replikaciju zato sto omogucava siguran failover i jednostavno skaliranje replika za citanje. Koristi se kao osnova mnogih High Availability sistema i mnoge cloud baze (AWS, Google Cloud SQL, Azure Database) su bazirane upravo na PostgreSQL replikaciji.

## Faze replikacije

Proces replikacije se moze razloziti na 5 osnovnih faza. Neke tehnike replikacije mogu preskociti odredjene faze, ili spojiti vise faza u jednu.

Faze koje cine replikaciju su:

1. Request (RE)
2. Server coordination (SC)
3. Execution (EX)
4. Agreement coordination (AC)
5. Response (END)

U toku **request faze** klijent salje zahtev sistemu. Klijent moze poslati direktno poslati zahtev svim replikama (samo u distribuiranim sistemima, ne obradjuje se u ovom radu), ili samo jednoj replici a ona ce proslediti taj zahtev ostalim replikama u toku server coordination faze. U bazama podataka klijenti nikada ne komuniciraju direktno sa svim repikama. Klijent treba da vidi jedinstvenu bazu podataka cak iako u pozadini postoji vise sinhronizovanih kopija koje sistem odrzava. Klijent komunicira sa jednim cvorom baze a taj cvor sa ostalima.

U toku **faze koordinacije servera** replike odredjuju redosled konkurentnih operacija nad bazom. Potrebna je saglasnost izmedju replika kojim redosledom se izvrsavaju operacije da ne bi doslo do nekonzistentnosti izmedju njih. Cilj je da sve replike vide sve relevantne operacije u istom redosledu. U koliko ne postoji zavisnost izmedju dve operacije jedne transakcije njihov redosled je nebitan s obzirom da to ne utice na konzistentnost replika i ne moze doci do hazarda.

**Execution phase** je faza gde se zapravo izvrsavaju operacije nad bazom podataka. Operacije se izvrsavaju redosledom koji je dogovoren u prethodnoj fazi. Replike primenjuju promene na svoje lokalne kopije.

**Agreement coordination** faza je faza gde replike potvrduju da su izvrsile operacije u redosledu koji je dogovoren u toku faze koordinacije servera. Ova faza je vazna za postizanje konsenzusa medju replikama. Sistem proverava da li su sve replike poslale potvrde pre nego sto se rezultati operacija mogu smatrati konacnim.

U **fazi odgovora** replika koja je inicijalno primila zahtev od klijenta salje nazad rezultat zahteva klijentu.

## **WAL(Write Ahead Log)**

**WAL(Write Ahead Log)** je mehanizam u PostgreSQL-u koji omogucava sigurnost i konzistentnost podataka u bazi podataka cak i u slucaju pada servera. WAL zapisuje svaku promenu podataka pre nego sto se ona stvarno primeni na glavne podatke baze. Promena se prvo zapisuje u WAL log a tek nakon toga se izvrsava SQL upit koji zapravo menja podatke. Nakon sto je WAL zapis snimljen transakcija se smatra za COMMITED, i tek onda se primenjuje na glavne podatke baze. WAL osigurava da se nijedna commitovana transakcija ne izgubi cak i ako baza padne pre nego sto se podaci fizicki izmene.

U zapisu loga se pamti i **LSN(Log Sequence Number)**, koji predstavlja adresu u WAL fajlu**.** Svaka operacija nad bazom podataka ima svoj LSN kada se zapise u WAL fajl**.** LSN je 64-bitna vrednost na osnovu koje je moguce precizno pratiti redosled svake promene u logu. Moguce je uporediti da li je operacija zapisana u logu stvarno izvrsena. U slucaju oporavka replike nakon pada, replika koristi LSN da utvrdi do kog trenutka je uspesno primenila promene nad svojim podacima. Replika uporedjuje svoj poslednji primenjeni LSN(replay\_lsn) sa LSN koji dolazi sa primarnog cvora (flush\_lsn). Ukoliko se ti zapisi ne poklapaju, replika pocinje sa primenom operacija pocevsi od poslednjeg izvrsenog zapisa, sve dok ne sustigne primarnu repliku. LSN se koristi iskljucivo u kontekstu logicke replikacije, gde se promene prenose na nivou SQL instrukcija.

Kako bi replika mogla brze da se oporavi nakon pada uvodi se kontrolna tacka. Kontrolna tacka predstavlja trenutak u kome se svi prethodno izvrseni zapisi iz WAL-a trajno upisuju na disk. Na ovaj nacin replika prilikom oporavka ne mora da proverava sve zapise od samog nastanka baze, vec moze da pocne od nekog stanja u skorijoj proslosti, poslednje poznate kontrolne tacke, cime se znacajno ubrzava proces sinhronizacije sistema.

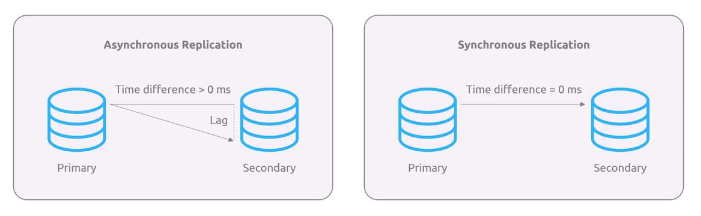
# Tipovi replikacije

## Asinhrona replikacija

U asinhronom rezimu replikacije primarni server ne ceka potvrdu od replike da zavrsi transkaciju. Podaci se salju replikama cim su zabelezeni u WAL-u i primarna baza nastavlja sa obradom transakcija bez cekanja. Primarni server se ne blokira cekajuci odgovor replika, sto povecava brzinu obradu transakcija. Konfiguracija je jednostavnija u odnosu na sinhronu replikaciju i moguce je imati vise read-only replika radi skaliranja citanja. Mana je to sto je moguc gubitak podataka. Ako primarna baza padne pre nego sto replike dobiju promene te transkacije ce biti izgubljene. Replike mogu kasniti u odnosu na primarnu bazu, odnosno imati starije podatke. Nije pogodna za sisteme gde je akcenat na konzistentnosti podataka, vec za one sisteme koji imaju veliki broj citanja i gde je prioritet performanse primarne baze, i gde je eventualno kasnjenje replika prihvatljivo.

## Sinhrona replikacija

U sinhronoj replikaciji primarni server ceka dok replike ne potvde prijem podataka pre nego sto se transakcija zavrsi. Time se osigurava da podaci nikada ne budu izgubljeni, cak i u slucaju pada primarnog servera. Uspostavlja se potpuna konzistentnost podataka jer su transkacije zasigurno zapisane i na primarnoj bazi i na replikama. Mana je vece vreme obrade zahteva s obzirom da primarni server mora da ceka na potvrdu replike, i moze doci do zastoja. Replika odmah moze da preuzme ulogu primarne baze, cime je obezbedjen siguran failover. Postavlja se pitanje kako ce se znati koja od replika ce uzeti ulogu primarne baze u slucaju njenog pada. PostgreSQL ne odlucuje samostalno o tome koja replika postaje primarna baza vec koristi alate za High Availability. Najpopularniji alat su Patroni. Ovaj alat automatski detektuje pad primarne baze i odlucuje koja replika preuzima njenu ulogu tako sto bira repliku koja ima najnovije podatke i koje je zdrava. Zdrava replika je ona replika koja je u stanju da preuzme primarnu ulogu bez rizika od gubitka podataka ili gresaka. To je replika koja je sinhronizovana sa primernim serverom, odmah dostupna i bez gresaka. Konfigurise je kao novi primary a ostale replike se rekonfigurisu tako da repliciraju podatke sa nove glavne baze, umesto sa one koja je pala.



## Hibridna replikacija

Predstavlja kombinaciju sinhrone i asinhrone replikacije. Balansira izmedju visoke dostupnosti, sto je karakteristicno za asinrhonu replikaciju, i performansi primarne baze, sto je karakterisitcno za sinhronu replikaciju.

Primarna baza ceka potvrdu samo od odredjenih replika (najcesce 1-2 replike), dok ostale replike primaju promene asinhrono. Replike koje primaju promene sinhrone moraju potvrditi svaku promenu. Ovim nacinom se obezbedjuje da kriticne transakcije budu sigurno zapisane u sinhronim replikama. Manje kriticne transakcije se salju asinhronim replikama. Moguce je skaliranje citanja povecavanjem broja asinhronih replika.

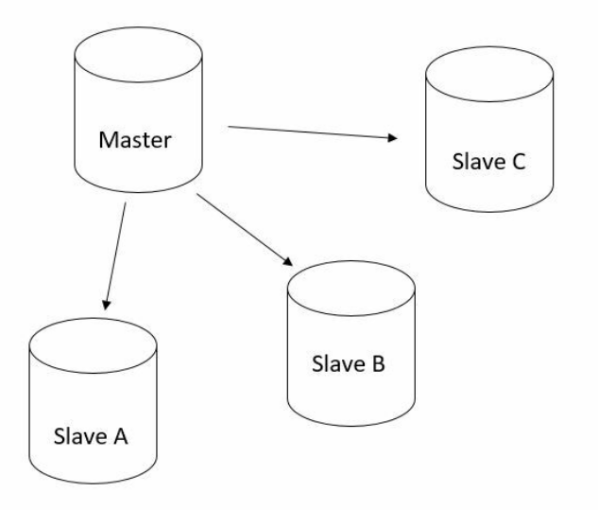
# Vrste replikacije u PostgreSQL-u po broju master cvorova

## Single-Master Replikacija

Najcesce koriscen model replikacije u PostgreSQL-u. Postoji samo jedan master cvor koji prima sve transakcije za upis, a sve ostale replike sluze za citanje podataka.

Master cvor je jedini cvor koji moze da izvsi transakcije za menjanje podataka (INSERT, UPDATE, DELETE). Ostale replike su iskljucivo read-only. Promene se prenose koristeci Write Ahead Log, sto omogucava visoku pouzdanost podataka i njihov integritet. Replike primaju promene sa master cvora i tako baza ostaje sinhronizovana. Nema konflikata prilikom pisanja s obzirom da samo jedan cvor moze da upisuje podatke. Replike se mogu skalirati i izvrsavati ogroman broj read upita. Ipak, ukoliko je veliki broj upisa to moze znacajno smanjiti performanse s obzirom da samo jedan cvor vrsi upis. Mana je takodje i to sto je master cvor Single Point of Failure, sto znaci da ako master cvor padne, pada ceo sistem. Dok failover ne bude izvrsen ceo sistem zavisi od primarnog servera.

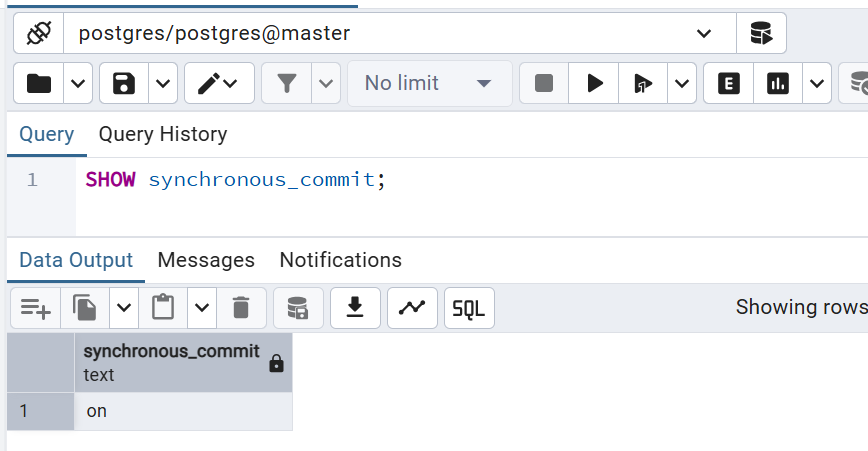
Single-Master replikaciju koriste sistemi i aplikacije sa velikim brojem citanja i ne tako velikim brojem upisa podataka.



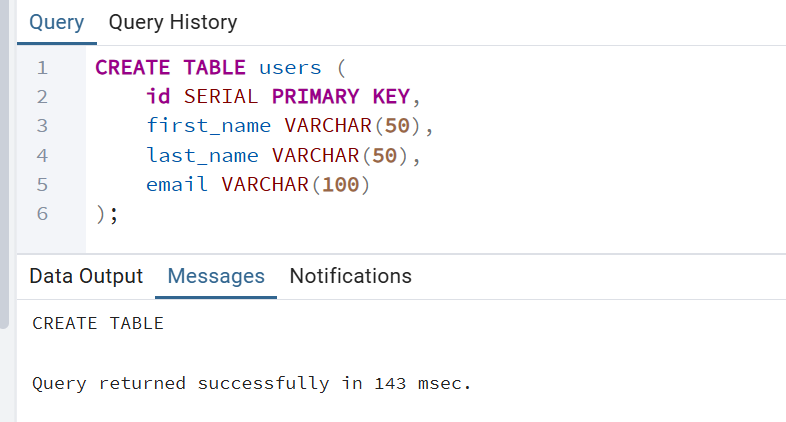
* Funkcija pg\_last\_wal\_receive\_lsn() vraca log sequence number poslednjeg WAL zapisa koji je slave primio od primarnog cvora.
* Funkcija pg\_last\_wal\_replay\_lsn() vraca loq sequence number poslednjeg WAL zapisa koji je slave primenio na svoju bazu.
* Funkcija pg\_last\_wal\_receive\_timestamp() vraca vreme prijema poslednjeg WAL zapisa
* Funkcija pg\_last\_wal\_replay\_timestamp() vraca vreme primene poslednjeg WAL zapisa

Primer komunikacije izmedju master cvora i replike kada su oba cvora dostupna:

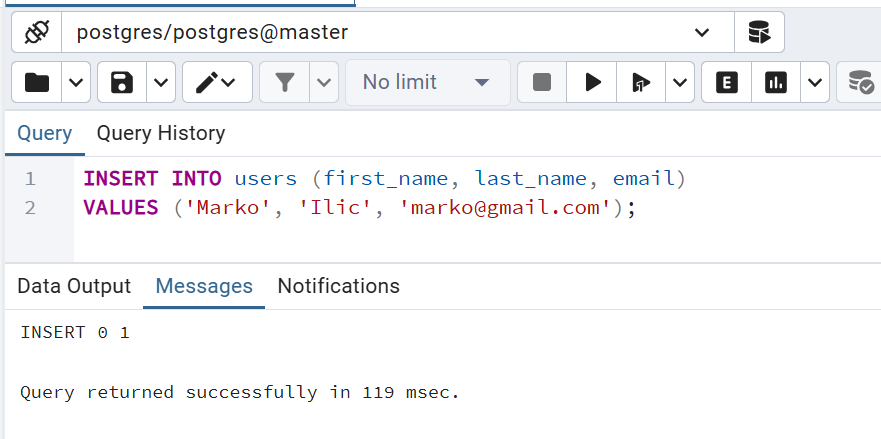
S obzirom da je ukljucena sinhrona replikacija, slave mora da potvrdi svaku promenu izvrsenu na primarnom cvoru.



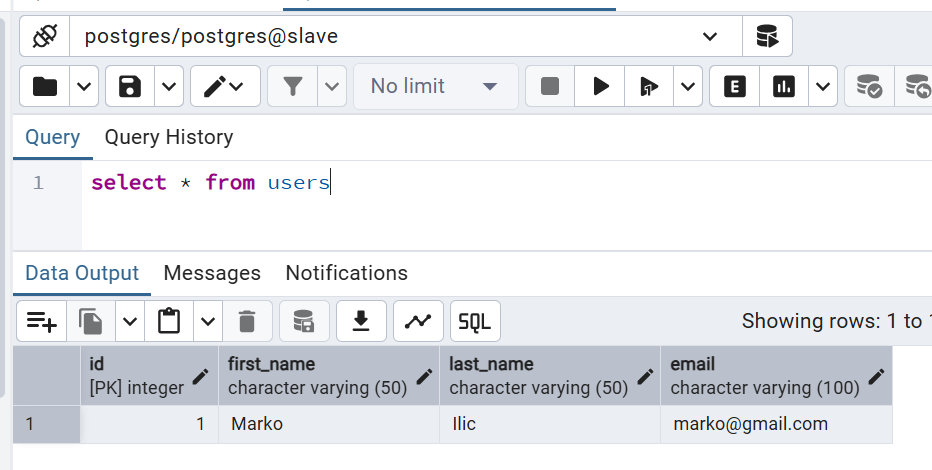
Kreiranje tabele u master tabeli

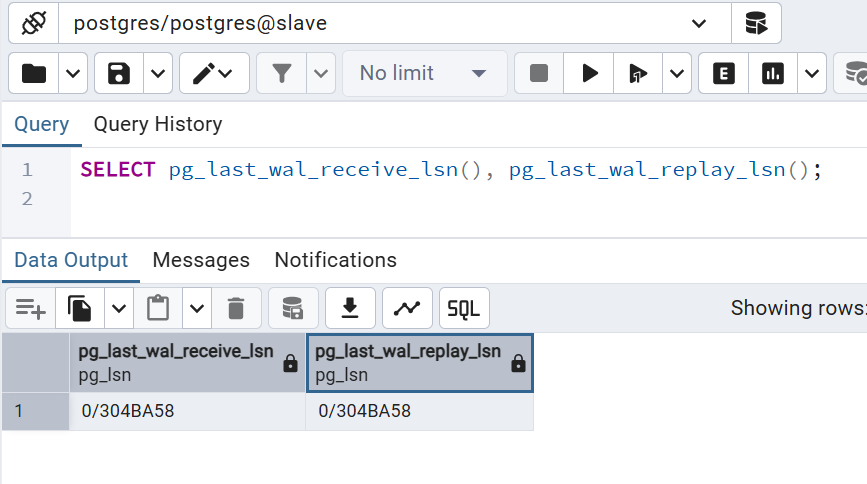


Dodavanje reda u kreiranoj tabeli:



Kada se izvrsi komanda INSERT u master tabeli, PostgreSQL salje WAL zapis ka replici, replika potvrdjuje da je primila promene, promene su vidljive na replici i master cvor moze da nastavi sa izvrsavanjem dalje.

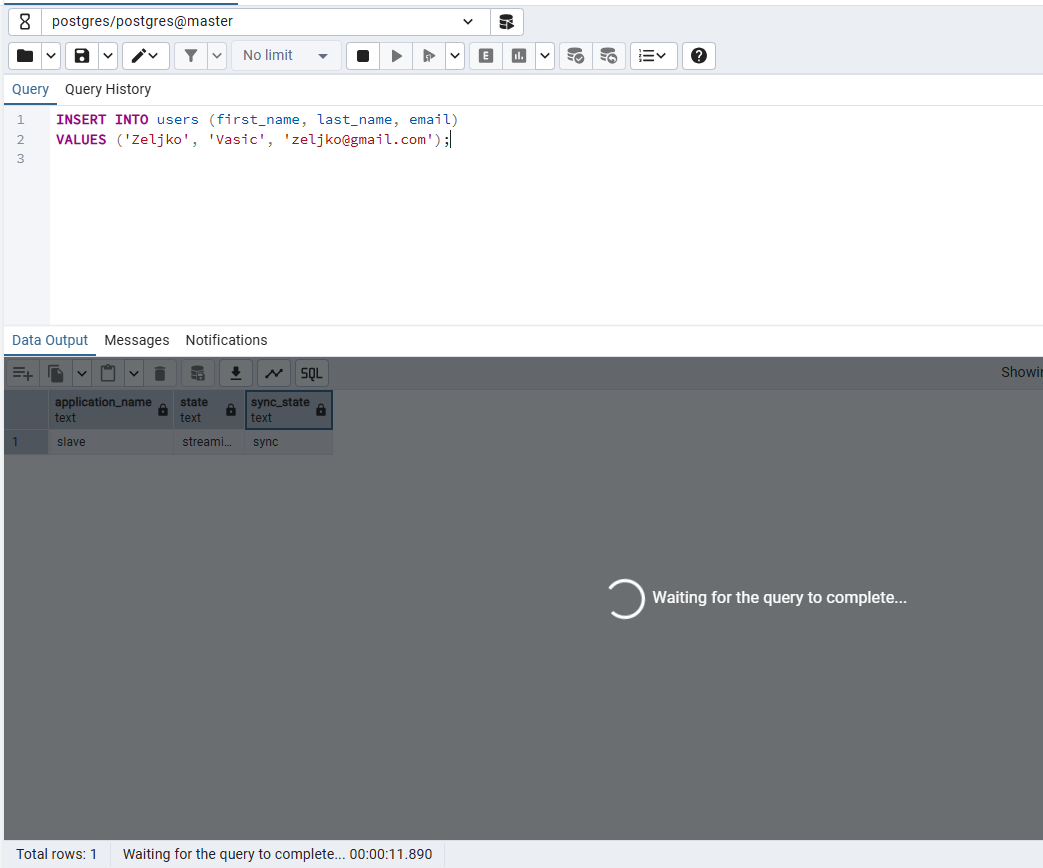




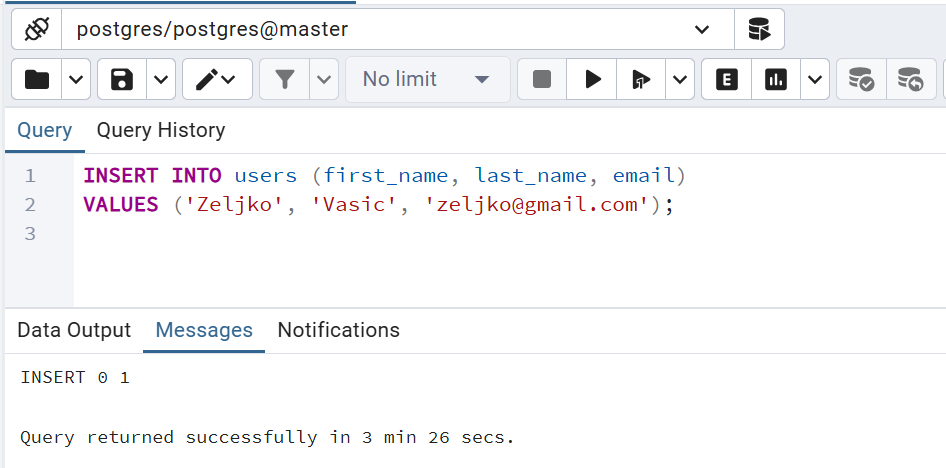
S obzirom da su obe vrednosti iste, moze se zakljuciti da je svaki wal koji je replika primila, primenjen na repliku.

Primer **sinhrone komunikacije** izmedju master cvora i replike u single-master sistemu dok je **slave nedostupan**:

Dodavanje reda u tabeli users na master cvoru:



S obzirom da je kod sinhrone komunikacije neophodno da slave odgovori masteru, ovaj upit ce cekati sve dok se ne podigne slave cvor. Master cvor ce se odblokirati tek kada sve replike potvrde da su primile WAL zapis, ili dok se replikacija ne prekonfikurise. S obzirom da PostgreSQL nema ugradjen timeout za sinhronu potvrdu, master sve to vreme ostaje blokiran.



Master cvor se odblokirao tek kada se slave podigao i poslao potvrdu da je primio podatke.

Ovaj problem se cesto resava konfiguracijom postgresq.config fajla na neki od nacina:

1. synchronous\_standby\_names = 'FIRST 1 (node1, node2)'

To znaci da samo jedna replika mora da odgovori masteru, i nakon njenog odgovara ce se master odblokirati.

1. Druga opcija je da nakon se primeti da je master usao u beskonacno cekanje izvrsiti

SET synchronous\_standby\_names = '';

Time master cvor prestaje da ceka replike i nastavlja da radi asinhrono sve dok se opet rucno ne vrati na sinhroni rad.

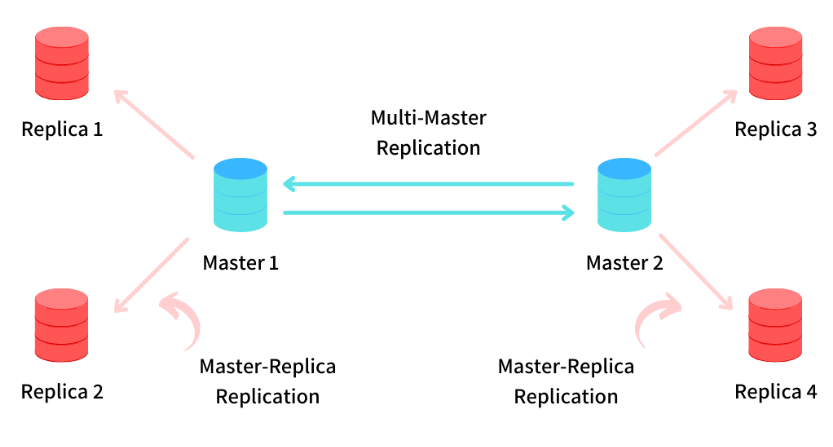
1. Bolja opcija je konfigurisati fajl da to radi automatski, komandom:

synchronous\_standby\_names = 'ANY 1 (node1, node2)'

To znaci da ako nijedna replika nije dostupna, master cvor ce raditi kao asinhroni cvor dok se barem jedna replika ne podigne. Ovo omogucava kontinuiran rad i sprecava ulazak u beskonacno cekanje master cvora ukoliko se nijedna replika ne podigne.

## Multi-Master Replikacija

Vise cvorova istovremeno moze da prima upite za pisanje i citanje. Povecana je tolerancija na kvarove i skalabilnost za upis, zato sto u ovom slucaju sistem ne zavisi samo od jednog cvora. Javlja se potreba za resavanjem konflikta s obzirom da vise cvorova sada moze istovremeno menjati isti podatak.



PostgreSQL po defaultu ne podrzava multi-master replikaciju vec je potrebno koristiti dodatne alate kao sto je bi-directional replication (BDR). Ovakav sistem je pogodan za koriscenje u sistemima koji su geografski distribuirani, gde je vazno da lokalni cvorovi obradjuju lokalne upite.

### Strategije resavanja konflikata u multi-master sistemima

**Last write wins** - pamti se poslednja zabelezena promena. Timestamp nije vezan za transakciju vec se dodeljuje po vremenu promene podatka, na primer azuriranje reda. Svaki podatak koji se menja ima svoj timestamp, vreme kada je poslednji put izmenjen. Kada se replike porede, zapravo se poredi vreme poslednje izmene i prihvataju se promene sa one replike koja ima skorije vreme. Jednostavno je za implementaciju i pogodno je za sisteme gde je najnoviji podatak uvek najvazniji ( na primer status korisnika, da li je online ili offline). U ostalim slucajevima, kada su obe promene vazne, moze doci do gubitka podataka, odnosno da prethodna promena bude pregazena novijom.

PostgreSQL ne podrzava last write wins strategiju, jer transakcija zakljucava red dok se ne zavrsi.

**Timestamp based -** ova strategija takodje koristi timestamps kao i last write wins ali na napredniji nacin. Svaka transakcija ima svoj timestamp. Ako dve ili vise transakcija azuriraju isti red, sistem moze da zadrzi obe verzije za analizu i razresavanje konflikata. Ukoliko vise transakcija menja isti red ali razlicitu kolonu u redu, sistem moze spojiti (mergovati) obe promene u jednu koju ce zadrzati. Ukoliko vise transakcija menja istu kolonu u istom redu, postoje dve opcije u zaviosnosti od implementacije. Sistem moze ponistiti stariju transakciju, i prihvatiti samo onu koja ima skoriji timestamp, ili moze zadrzati obe verzije dok se konflikti ne rese rucno. Ovo obezbedjuje preciznije resavanje konflikata i pogodno je za sisteme sa slozenijim transakcijama i visokom konzistentnoscu.

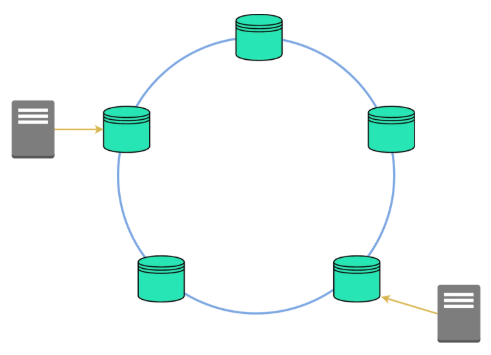
**Priority based** - Ako vise replika pokusava da izmeni isti podatak, pobedjuje ona replika koja ima veci prioritet. Svaka replika ima odredjen prioritet. Ukoliko replike imaju isti prioritet onda pobedjuje ona replika ciji je zahtev prvi stigao.

**Application-level merge -** najcesce koriscenastrategija za razresavanje konflikata u PostgreSQL distribuiranim sistemima. Logika spajanja podataka se ne odvija na nivou baze podataka vec u samoj aplikaciji, odnosno kodu koji kontrolise poslovnu logiku. Aplikacija dobija vise verzija istog podatka, analizira razlike izmedju tih verzija i donosi odluku o izgledu krajnje verzije. Logika koja se koristi za resavanje konflikata je pisana od strane programera, to povecava fleksibilnost ali i rizik od gresaka.

## Masterless Replikacija

Kao sto samo ime kaze ne postoji master cvor, vec su svi cvorovi jednaki. Svaki cvor u sistemu moze i da cita i da pise podatke. Kada se podaci promene u nekom od cvorova, taj cvor salje promenu i ostalim cvorovima. Ukolik je doslo do promene istih podataka u vise cvorova koriste se algoritmi za postizanje konsenzusa ili resavanje konflikata.

PostgreSQL po defaultu ne podrzava masterless replikaciju vec je potrebno koristiti dodatne alate kao sto je bi-directional replication (BDR). Prednosti su sto ne postoji single point of failure i lako se sistem se lako skalira dodajuci nove cvorove u sistem. Mane su sto je teze obezbediti konzistentnost, znaci sistem se fokusira na dostupnost i particionu toleranciju.

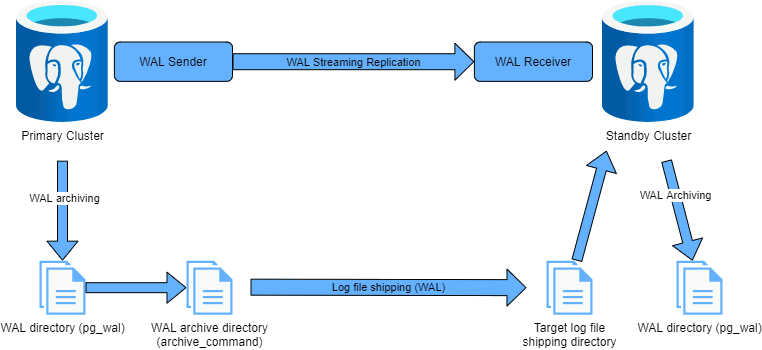


# Vrste replikacije u PostgreSQL-u po nacinu prenosa podataka

## Fizicka replikacija

Fizicka replikacija je tip replikacije gde se podaci prenose na nivou fizickih blokova i datoteka baze podataka. Podaci se prenose u binarnom obliku. Fizicka replikacija je uglavnom single-master tip gde postoji samo jedan master server i vise replika. Fizicka replikacija kopira celo stanje baze, ukljucujuci tabele, indekse i sistemske datoteke. Podaci se salju sa primarnog servera na replike kroz WAL.

Fizicka replikacija je vrlo efikasna s obzirom da se prenosi binarni zapis i nije potrebno parsiranje SQL komandi. Replike su potpuno identicne primarnom serveru, i replika moze postati primarni server u slucaju kvara, Mana je sto su replike samo read-only i sto se ne moze replicirati samo deo baze. Svi primeri obradjeni u prethodnim primerima se odnose na fizicku replikaciju.



Kako funkcionise fizicka replikacija u PostgreSQL-u:

1. Primarni server zapisuje svaku transakciju u WAL
2. Replika uspostavlja konekciju sa primarnim serverom i konstantno prima nove WAL zapise
3. Replika primenjuje primljene promene u istom redosledu u kom ih prihvata, cime ostaje konzistentna sa primarnim serverom.

## Logicka replikacija

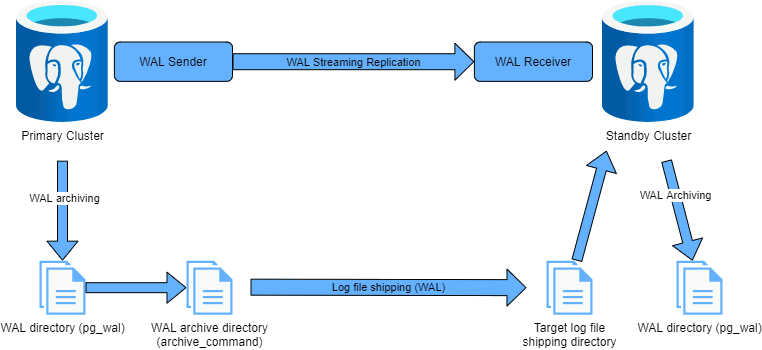
Replicira podatke na nivou tabela ili kolona koristeci publikaciju i subskripciju.

Postavlja se kroz CREATE PUBLICATION ili CREATE SUBSCRIPTION.

Omogucava selektivno repliciranje podataka.

Logicka replikacija je tip replikacije gde se podaci prenose na nivou SQL komandi ili promena po tabelama. To omogucava selektivno repliciranje i veci nivo fleksibilnosti nego kod fizicke replikacije.

Primarni server salje promene a replika ih prima i primenjuje. Jedan primarni server moze imati vise logickih replika i svaka od tih replika moze imati razlicite podatke, zato sto se ne salju kompletne baze replikama. U kombinaciji sa sinhronom replikom failover garantuje da se replika koja preuzima primarni server da bude potpuno konzistentna sa prethodnim primarnim serverom.

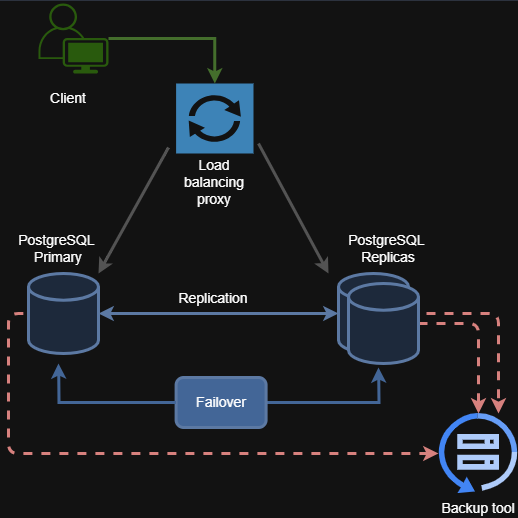


## Snapshot Replikacija

Snapshot replikacija podrazumeva periodicno kopiranje kompletne baze ili odabranih tabela baze sa jednog cvora na druge cvorove. Sve promene koje su se desile izmedju dva snapshota nisu vidljive ostalim cvorovima sve dok im cvor na kome su se desile promene ne posalje novi snapshot baze. Ne zahteva stalnu konekciju izmedju cvorova. Pogodno je za offline sisteme, gde replike nisu stalno povezane sa primarnim cvorom. Moze se koristiti za backup ili arhiviranje podataka. Nije pogodna za aplikacije koje zahtevaju stalno azuziranje podataka. Veca kolicina podataka u jednom trenutku moze izazvati opterecenje mreze i servera.

# HA setup

Replikacija je osnova svakog High Availability sistema. High Availability znaci da baza podataka ostaje dostupna cak i kada primarni server otkaze. Cilj je minimizirati downtime i gubitak podataka. U PostgreSQL-u se to postize kombinovanjem streaming(WAL) replikacije, automatskog failovera i load balancinga. Postize se visoka otpornost na greske, bez gubitka podataka i minimiziranje downtimea sistema. Zbog ovih osobina se PostgreSQL cesto koristi u sistemima gde su dostupnost, pouzdanost i performanse od vaznosti.



Osnovne komponente HA setup-a u PostgreSQL-u su:

1. Primarni server
2. Kopije primarnog servera
3. Load Balancer
4. Failover i promocija novog primarnog servera

## Load Balancer

Load balancing znaci da se upiti ravnomerno rasporedjuju na vise servera da bi se izbeglo preopterecenje jednog servera. U PostgreSQL-u se load balancing obicno koristi u single-master arhitekturi to znaci da ce master prihvatati sve write operacije, a standby replike ce prihvatati read upite. Load balancer rasporedjuje ravnomerno upite izmedju replika da ne bi dolazilo do zastoja. Ako neka replika padne, automatski se iskljucuje iz rotacije i njeni upiti se ravnomerno rasporedjuju na preostale replike. Raspodela upita po replikama se moze vrstiti po razlicitim kriterijumima:

* Round-robin - upiti se rasporedjuju redom izmedju svih dostupnih replika
* Least connections - upiti se salju onoj replici koja ima najmanje aktivnih veza
* Weighted balancing - neke replike imaju veci kapacitet od ostalih, pa one dobijaju vise upita u skladu sa tezinom (weight)

## Automatski failover

Automatski failover je mehanizam koji omogucava zamenu primarnog cvora servera sa stanby serverom (replikom) bez ljudske intervencije, cime se minimizira downtime i povecava HA. Failover oznacava proces prebacivanja odgornosti za upise i citanja sa neispravnog primarnog servera na repliku koja ce postati primarni server. Ta replika je sada zaduzena za izvrsavanje operacija citanja.

Kljucne komponente failover-a su:

1. Monitoring
2. Election
3. Promotion
4. Reconfiguration

Primarni server se stalno proverava da li je dostupan i da li je pao(**monitoring**). Kada primarni server padne vrsi izbor replike koja ce postati novi primarni server (**election**).Ako postoji vise replika sistem bira zdravu repliku koja je potpuno sinhronizovana sa WAL logovima. Takodje je bitno i da je prisutna minimalna latencija sa primarnim servom i dostupnost replike na mrezi. Izabrani standby server postaje primarni server i preuzima sve upite za pisanje(**promotion**). Izabrana replika sada prima INSERT, UPDATE i DELETE operacije. Ostale replike se rekonfigurisu tako da sada dobijaju podatke od novog primarnog servera (**reconfiguration**).

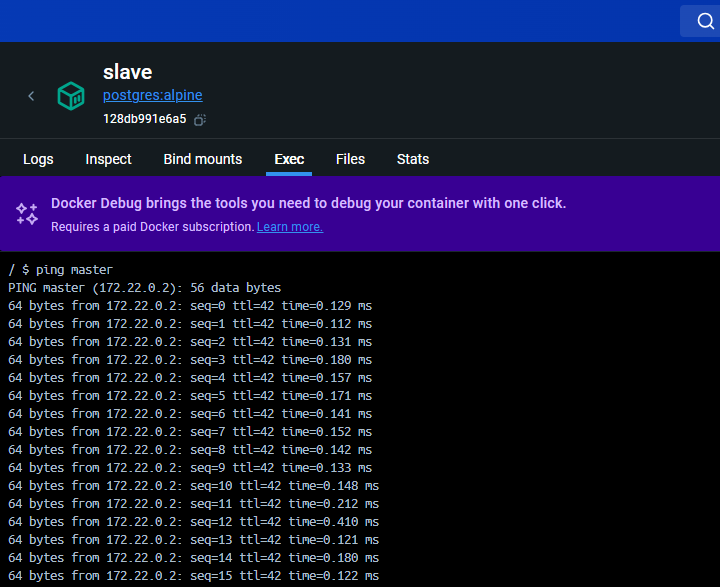
Automatski failover je vazan zato sto minimizira downtime, sto je od kljucnog znacaja u sistemima u kojima je bitno odrzati high availability.

## Monitoring i pracenje replikacije

Monitoring replikacije omogucava administratorima da prate perfomanse sistema i da blagovremeno reaguju na kvarove ukoliko je potrebno. Proverava se koji procesi su aktivni, da li je baza zdrava i da li moze da prihvati sve zahteve koji su joj upuceni.

Pracenje replikacije u PostgreSQL-u obuhvata vise parametara koji se mogu pratiti pomocu ugradjenih SQL komandi. Postoji aktivni i pasivni monitoring.

**Aktivni monitoring** znaci da sistem za nadgledanje aktivno testira performanse i dostupnost baze tako sto salje mock zahteve i meri vreme odgovora. Salje se ping da proveri da li je server dostupan, server ako je dostupan vraca poruku. Ukoliko nema odgovora za neko odredjeno vreme smatra se je ta baza down, i zapocinje se proces oporavka sistema.

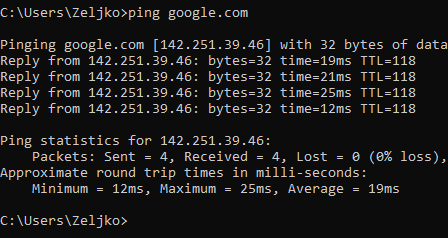


64 bytes from 172.22.0.2. znaci da je master cvor dostupan.

Time= 0.129 ms predstavlja vreme odgovora master cvora.

Seq predstavlja broj zahteva koji je poslat. Ova komanda se salje svake sekunde iz slave cvora, i svake sekunde se proverava da li je master cvor dostupan.

Ukoliko se ista komanda posalje ka nekoj udaljenoj bazi, na primer Google, moze se primetiti da je vreme odgovora servera 100 puta vece nego kod baza koje rade lokalno.



Reply from 142.251.39.46 znaci da je server dostupan.

Average = 19ms predstavlja prosecno vreme koliko je bilo potrebno serveru da odgovori(Windows po defaultu salje 4 paketa prema serveru koji se pinguje, racuna se prosecno vreme odgovora)

Lost = 0 znaci da nijedan od tih paketa nije izgubljen.

Velika razlika u vremenu odogovora nastaje zbog toga sto ping prema Google serveru putuje kroz vise mreznih cvorova, Google server je u oblaku, geografska udaljenost je mnogo veca i samim tim je vreme odgovora vece. Zbog toga su ovo kljucni faktori koji se trebaju uzeti u obzir prilikom dizajniranja sistema koji zahtevaju visoku dostupnost.

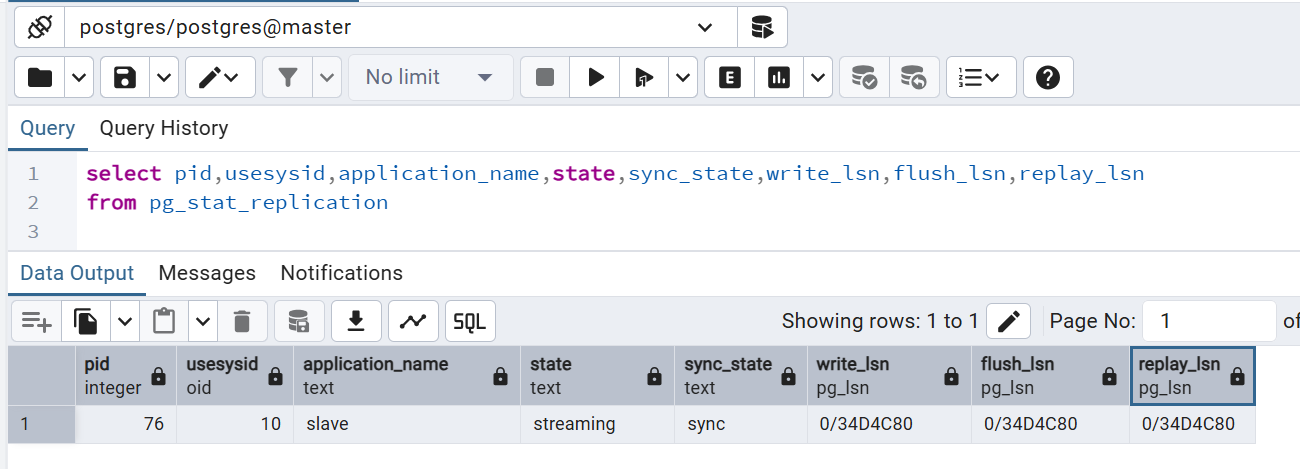
**Pasivni monitoring** je metoda nadgledanja sistema u kojoj se podaci prikupljaju i analiziraju bez aktivnog slanja test poruka. Pasivni monitoring samo nadgleda trenutni saobracaj i ono sto se vec desava na mrezi, i ne salje dodatni saobracaj. Proverava koliko resursa je trenutno iskorisceno u sistemu.

### View pg\_stat\_replication

Ovaj view se nalazi na master serveru i prikazuje informacije o svim povezanim replikama.

Kljucna polja u ovom viewu su:

* Pid - id procesa replikacije
* Usesyid - id korisnika koji koristi replikaciju
* Application\_name - ime aplikacije ili cvora koji je povezan kao sekundarni cvor
* State - trenutni status replikacije
* Sync\_state - status sinhronizacije (sync, async)
* Write\_lsn - Najveci WAL zapis koji je zapisan u baferu replike
* Flush\_lsn - Najveci WAL zapis koji je zapisan na disku replike
* Replay\_lsn - poslednji replayovani WAL na slave-u, podaci iz WAL zapisa su u tabelama



Ovaj rezultat znaci da je slave sa nazivom ‘slave’, koji ima proces id 76 potpuno sve zapise koje je primio od mastera vec primenio na svoje podatke, zato sto su mu write\_lsn, flush\_lsn i replay\_lsn jednaki.

Ipak to ne znaci da su potpuno sinhronizovani master i slave, ovo samo govori da je slave primenio sve zapise koje je primio.

### View pg\_stat\_wal\_receiver

Kljucna polja u ovom viewu su:

* Pid - proces ID procesa koji prima WAL zapisa
* Status - trenutni status WAL receiver procesa

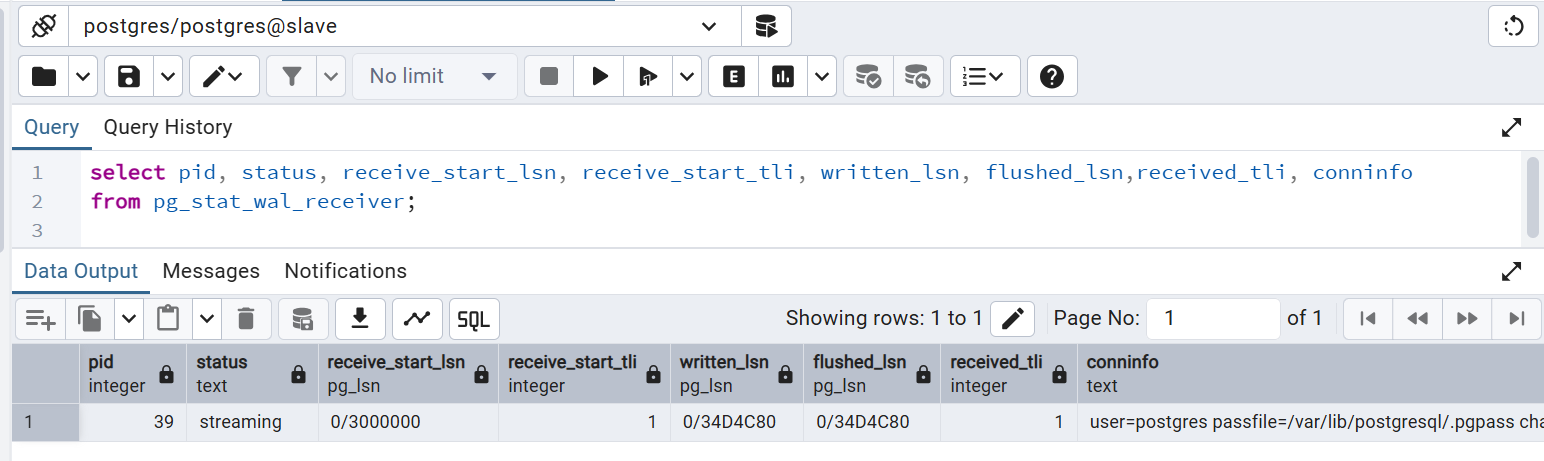
Moguce vrednosti su:

Startup- proces se pokrece i sprema za primanje zapisa

Catchup - pokusava da dostigne master jer je zaostao

Streaming - aktivno prima WAL zapise od mastera

* Receive\_start\_lsn - log sequence number od koga je replika pocela da prima WAL zapise
* Receive\_start\_tli - ID timeline-a od koje pocinje replikacija
* Written\_lsn - Najveci WAL zapis koji je zapisan u baferu replike
* Flushed\_lsn - Najveci WAL zapis koji je zapisan na disku replike
* Received\_tli - ID timelinea poslednje WAL zapisa koji je primljen
* Conninfo - informacije o konekciji ka master serveru, kao sto su user, port, host...



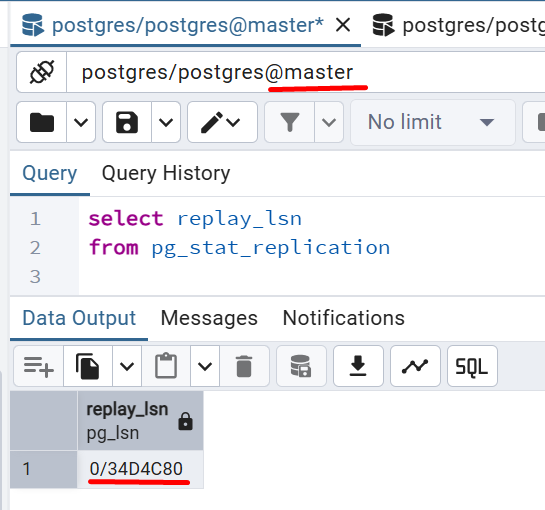
Da bi se sa sigurnoscu reklo da je replika u potpunosti sinhronizovana sa master cvorom potrebno je da vazi:

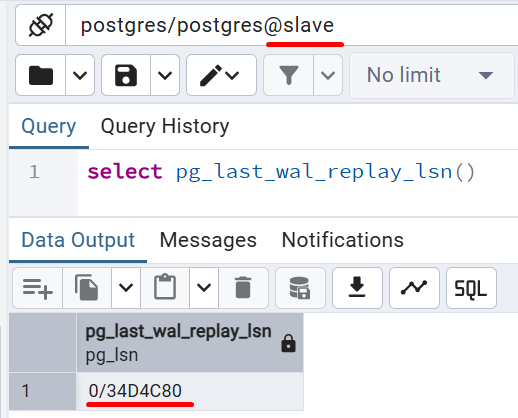
master.write\_lsn = slave.written\_lsn

master.flush\_lsn = slave.flushed\_lsn

**master.replay\_lsn = pg\_last\_wal\_replay\_lsn() na slave-u**

Posto je replay poslednji korak i zahteva da su write\_lsn i flush\_lsn jednaki, dovoljno je proveriti da li su im jednaki replay\_lsn, a ukoliko jesu to automatski implicira da su im jednaki i write\_lsn i flush\_lsn.





# **Replikacija u oblaku**

Replikacija u oblaku podrazumeva kopiranje i sinhronizaciju podataka izmedju vise baza podataka koje se nalaze na cloudu. Prednost je ta sto baze mogu da budu geografski distribuirane, i podaci dostupni globalno.

Glavna razlika u odnosu na klasicnu lokalnu replikaciju je ta da replikacija u oblaku ne zavisi od fizicke razdaljine servera.

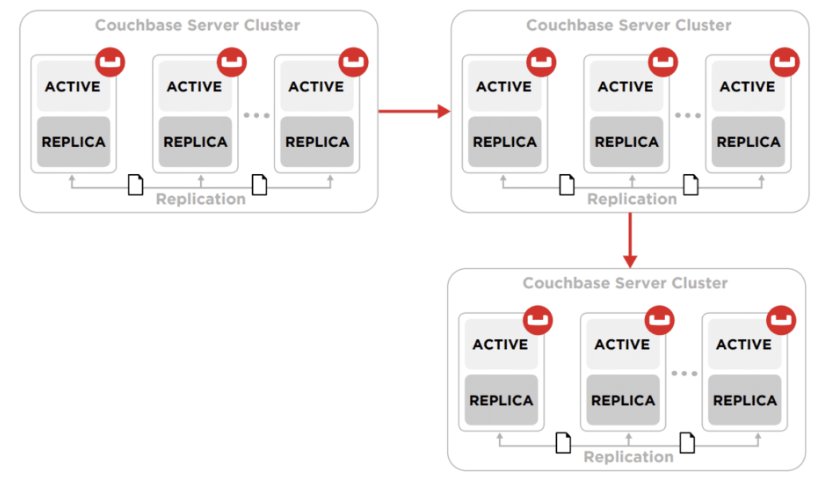
Klasicna replikacija se uglavnom koristi za ocuvanje podataka u slucaju pada master baze. U slucaju pada bi se zahtevala rucna konfiguracija da se serveri ponovo dignu.

Replike obicno nisu direktno dostupne aplikaciji, osim kada padne master cvor.

Kod klasicne replikacije se serveri obicno nalaze blizu jedni drugih, u istom data centru ili klasteru da bi mrezna latencija i kasnjenje podataka bili minimalni. Podaci se prenose preko lokalne mreze. Korak dalje od toga bi bilo **cross-data center replication**.

Replikacija u oblaku je dizajnirana za funkcionisanje nezavisno od rastojanja izmedju servera. Podaci se prenose preko interneta ili cloud mreza.

**Cloud-to-cloud replikacija podataka** koristi lokalnu mrezu kao master, i vise cloud servisa iz razlicitih regiona, ali unutar jednog clouda, kao replike ali mogu da sluze i pisanje i za citanje.



Organizacije cesto vec imaju sopstvenu mrezu koja je redudantna (ima backup u slucaju kvara). Ako organizijama fali geografska distribuiranost, ali uz odrzavanje reduntnosti, tada je idealno preci na cloud. Replikacija u oblaku omogucava organizacijama da sadrze svoju redudantnu privatnu mrezu i da je geografski prosire. Takodje se sprecava vendor lock-in, odnosno preveliku zavisnost od jednog provajdera. Proces prelaska sa jednog provajdera na drugog (Azzure, Google Cloud…) je jednostavan, bez potrebe za glomaznim migracijama podataka.

Cloud provajderi nude opcije koje umnogome olaksavaju proces replikacije:

* Automatsko kreiranje replika - Kada korisnik pokrene novu bazu u oblaku i oznaci opciju *enable replication* provajder automatski pravi jednu ili vise replika te baze, bez potrebe za rucnim podesavanjem. Korisnik ne mora da brine o konfiguraciji replika. Provajder takodje vodi racuna i o optimalnoj udaljenosti i mreznoj povezanosti izmedju replika i master cvora, da bi sinhronizacija izmedju njih bila sto optimalnija.
* Automatski failover - U slucaju kvara master cvora, sistem automatski unapredjuje jednu od replika u master cvor, minimizirajuci downtime.
* Automatsko skaliranje - Ukoliko sistem proceni da su replike preopterecene read zahtevima, moze automatski da pravi nove replike sto smanjuje opterecenje na postojecim replikama
* Kreiranje snapshot-a baze direktno sa replike. To znaci da master server ne mora da se dodatno opterecuje pravljenjem backup-a vec se moze fokusirati na izvrsavanje write zahteva.
* Replike se mogu koristiti za testiranje novih funkcionalnosti. Posto su podaci vec sinhronizovani sa serverom, replike se mogu koristiti, sto eliminise rizik da ce se ostetiti primarni cvor.
* Monitoring opterecenja servera i latencije i alarmi koji obavestavaju administratore u slucaju da postoji neki problem sa mrezom

Ako se replike nalaze u razlicitim regionima, latencija mreze moze uzrokovati kasnjenje u repliciranju WAL zapisa. To dalje znaci da podaci na replikama nisu trenutno konzistentni sa primarnom bazom. Zbog toga aplikacije sa replikacijom u cloudu trebaju biti dizajnirane da mogu raditi sa eventualnom konzistentnoscu ili da koriste sinhrone replike ukoliko je potrebna potpuna konzistentnost. Na primer kod gejminga gde je brzina odgovora od kljucnog znacaja, je cesta praksa da jedna ista igrica ima odvojene servere za svaki region. Server sa istocnu Evropu,Severnu Ameriku, Juznu Ameriku… Igraci u zavisnosti od regiona u kom se nalaze komuniciraju sa najblizim serverom kako bi dobili odgovor sto je brze moguce. Ukoliko igrac iz evrope pokusa da igra na Americkom serveru, imao bi kasnjenja, lagove i ne bi imao prijatno korisnicko iskustvo.

Podaci koji se prenose kroz mrezu moraju da budu enkriptovani zbog ocuvanja sigurnosti podataka. Cloud baze su zasticene i autentifikacijom i autorizacijom. Za pristup cloud bazi je potreban odgovarajuci korisnicki nalog. Autorizacija je role-based, odnosno korisnici imaju razlicite dozvole za upravljanje bazom i mogu izvrsavati razlicite SQL opetacije nad bazom u zavinosti od svoje role(read-only, read-write, admin…).

# Zakljucak

Replikacija predstavlja kljucni mehanizam za povecanje dostupnosti, skalabilnosti i pouzdanosti baza podataka. Svaki moderni sistem koristi replikaciju kako bi obezbedio kontinuirani rada i integritet podataka. Replikacija omogucava skalabilnost citanja i pisanja, povecanje otpornosti sistema na greske. U slucaju pada primarnog cvora zahvaljujuci failoveru sistem moze znacajno smanjiti downtime i izbeci gubitak informacija.

Replikacija omogucava horizontalnu skalabilnost, posebno za citanje, cime se rasterecuje primarni server. Razliciti tipovi replikacije omogucavaju prilagodjavanje performansi i sigurnosti zahtevima konkretne aplikacije. Fizicka replikacija se koristi kada se potrebno tacno kopiranje baze, dok se logicka replikacija koristi kada je potrebno selektivno repliciranje podataka. Visoka dostupnost se postize kombinacijom primarnog servera, replika, load balancera i automatskog failovera.

Uspesna implementacija replikacije ne zavisi samo od tehnicke implementacije baze podata, vec i od razumevanja poslovnih zahteva aplikacije koja je koristi i njenih prioriteta. Replikacija je alat koji mora biti pazljivo prilagodjen kontektstu same aplikacije, arhitekturi sistema i ocekivanjima korisnika.

# Literatura

1. ,,Understanding Replication in Databases and Distributed Systems”, M. Wiesmann, F. Pedone, A. Schiper
2. ,,Database Replication: A Survey of Open Source and Commercial Tools”, Salman Abdul Moiz
3. ,,Unraveling Multi-Master Replication in PostgreSQL: Architectures and Solutions”, Ibrar Ahmed
4. Mastering PostgreSQL 13, Fourth Edition, Hans-Jürgen Schönig
5. <https://www.postgresql.org/docs/current/protocol-replication.html>
6. <https://www.postgresql.org/docs/current/logical-replication.html>
7. <https://www.postgresql.org/docs/current/runtime-config-replication.html>
8. <https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-ver17>
9. <https://www.cybertec-postgresql.com/wp-content/uploads/2024/02/PostgreSQL_understanding_replication.pdf>
10. <https://www.tigerdata.com/learn/best-practices-for-postgres-database-replication>
11. <https://www.scylladb.com/glossary/high-availability-database/>