Baze de Date Distribuite în PostgreSQL

Cătălina-Gabriela Sfetcu

ETTI - Ingineria Informației și a Sistemelor de Calcul

Universitatea Politehnica din București

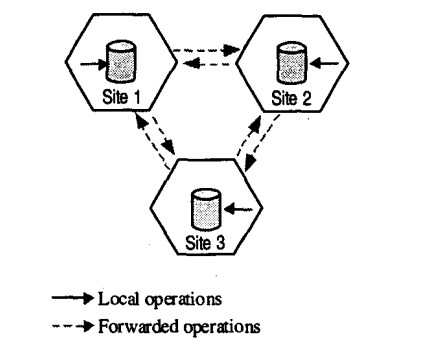
București, Romania

**Abstract — În ultimii ani evoluția comunicării datelor a modificat rapid și substanțial modul în care informația este gestionată. Creșterea conectivității la prețuri scăzute și îmbunătățirea tehnologiei de stocare a datelor a condus oamenii să necesite accesarea și mai multor informații. Au fost dezvoltate tehnologii noi pentru a partaja date pe internet (de exemplu MIDAS, CORBA), dar uneori este necesară și agregarea datelor. Spre exemplu, dacă vrem să obținem un set unic de date de la mai multe baze de date din site-uri diferite, aceste noi tehnologii necesită un mecanism de distribuire a datelor. Replicarea datelor este un proces care permite construirea unei baze de date distribuită prin intermediul managementului mai multor copii ale datelor, adăugând o copie pe fiecare site. Scopul prezentei lucrări a fost de a implementa o procedură specifică pentru ca un sistem PostgreSQL ORDBMS să gestioneze baze de date distribuite prin intermediul replicării asincrone[1].**

# **Introducere**

PostgreSQL, sau mai simplu - Postgres, este un sistem open-source de gestiune a bazelor de date relaționale cu accent pe extensibilitate, fiabilitate, robustețe, performanță și respectarea standardelor. Are capacitatea de a gestiona sarcini începând de la aplicații mici ce rulează pe o singură mașină până la aplicații mari de pe Internet (sau pentru depozitarea datelor) folosite de mulți utilizatori simultani. Este baza de date implicită de pe serverul macOS și este disponibilă pentru Windows și Linux (furnizată în majoritatea distribuțiilor). PostgreSQL își bazează dezvoltarea pe o comunitate răspândită la nivel global, precum și pe câteva companii dezvoltatoare. PostgreSQL este tranzacțională și compatibilă cu ACID. Este dezvoltată de PostgreSQL Global Development Group, un grup format din mai multe companii și contribuitori individuali. De asemenea, soluția este gratis și open-source, oferită sub termenii licenței software PostgreSQL[2].

Replicarea datelor este un proces care permite construirea unei baze de date distribuită prin intermediul managementului mai multor copii ale datelor, adăugând o copie pe fiecare site (figura 1). În special, replicarea sincronă (de asemenea numită si replicare de date în timp real) transmite informații în timp real către toate site-urile implicate. Pe partea opusă, replicarea asincronă (stocare și transmiterea replicării) stochează operațiile efectuate pe o bază de date într-o coadă locală pentru distribuirea ulterioară de către un proces de sincronizare dintr-o bază. Tehnologia de replicare sincronă asigură cea mai mare performanță la nivel de integritate a datelor, dar necesită o disponibilitate permanentă a serverelor și a benzii de transmisie. Pe de altă parte, replicarea asincronă oferă mai multa flexibilitate decât replicarea sincronă întrucât intervalul de timp de sincronizare poate varia de la minute la luni și, în plus, un singur site ar putea funcționa chiar dacă un server de la distanță este inaccesibil sau nefuncțional. În plus, sunt efectuate operațiuni de date mai repede și traficul în rețea este mai compact. Cu toate acestea, este necesară o planificare complexă de replicare în cazul replicării asincrone pentru a fi detectate și corectate conflictele provenite din modificările concurente apărute la diferite site-uri între două evenimente de sincronizare de date.



*Fig. 1. Replicarea datelor în 3 site-uri[1]*

# Concepte POSTGRESQL

PostgreSQL este cea mai avansată bază de date "open source" ce oferă multe caracteristici moderne. PostgreSQL are o arhitectura a sistemului realizată pe modelul client/server.

Există două tipuri de procese (programe) care stau la baza oricărei sesiuni PostgreSQL. Acestea sunt:

Procesul server numit postmaster ce se ocupă cu administrarea fișierelor bazei de date, acceptă conexiuni la baza de date de la aplicațiile client și realizează în numele clientului acțiuni asupra bazei de date.

Aplicația clientului care dorește să realizeze acțiuni asupra bazei de date. Tipurile de aplicații client sunt foarte diverse: de la un client care poate fi orice unealtă, aplicație grafică sau server web care dorește sa acceseze baza pentru a afișa informații, până la tool-uri care sunt specializate în întreținea unei baze de date[3].

Ca orice aplicație client/server, si in acest caz serverul si clientul pot sa fie pe diferite host-uri. Serverul PostgreSQL poate sa accepte mai multe conexiuni multiple de la clienți. Pentru fiecare conexiune se creează un nou proces ( \"fork\"), si de aici fiecare client si proces server vor comunica fără intervenția procesului postmaster. Tool-urile client oferite de PostgreSQL sunt doua, unul este un tool grafic - pgAdmin, iar al doilea este un tool de tip "command-line tool" și se numeste "psql".

După cum am amintit PostgreSQL oferă un set important de caracteristici moderne care îl fac un server complex si actual. Una dintre aceste caracteristici este numărul mare de tipuri de date disponibile pentru utilizator, punându-le la dispoziție si posibilitatea de a adăuga alte tipuri noi folosind comanda CREATE TYPE. Unele dintre aceste tipuri de date sunt unice in PostgreSQL, cum ar fi tipurile geometrice sau tipuri cu mai multe formate decât cele standard (date and time). O alta particularitate o regăsim în modul de manipulare a tranzacțiilor. În cazul PostgreSQL este posibil un control mai granulat al tranzacțiilor folosindu-ne de savepoints (puncte de salvare).

Acest concept permite definirea unui punct de salvare la care ne putem întoarce cu un ROLLBACK TO. Toate schimbările făcute asupra bazei de date de la savepoint la rollback to sunt înlăturate, dar modificările făcute până la savepoint sunt păstrate. Putem să ne întoarcem de mai multe ori la același savepoint, sau în caz contrar să îl folosim o singura daca, caz în care resursele vor fi eliberate. Ca și trăsătură specială putem aminti și noțiunea de moștenire care este preluată din alte limbaje și care se aplică pe componentele fundamentale a unei baze de date și anume a tabelelor[3].

# Replicarea

Odată cu dezvoltarea tehnologică, tot mai multe date se păstrează în format electronic.

Aplicațiile de calculator existente produc o cantitate din ce în ce mai mare de informație care trebuie păstrată și organizată. Bazele de date au rolul de a păstra informația într-o formă mai mult sau mai puțin organizată. Dacă la început bazele de date trebuiau să stocheze o cantitate relativ mică de informație, astăzi trebuie să stocheze zeci de GB. Există mai multe familii de baze de date, fiecare având anumite avantaje specifice, dar în general putem vorbi de baze de date relaționale SQL și baze de date NoSQL. Datorită cantității mari de informație care se dorește a fi stocată, în general, o bază de date poate fi distribuită pe mai multe noduri. O bază de date distribuită este caracterizată și trebuie să asigure: consistența datelor, disponibilitate datelor, și să fie toleranta la partiționare. Conform teoremei CAP aceste 3 caracteristici nu pot fi asigurate concomitent. Prin consistență se înțelege faptul că toate nodurile văd aceleași date. Prin valabilitate se înțelege faptul că datele sunt disponibile, iar baza de date este tolerantă la defectarea unor noduri. Prin toleranță la partiționare se înțelege faptul că baza de date poate fi distribuită mai multe noduri, ideal pe oricâte[5]. Aplicațiile de bussines au cerințe destul de exigente asupra bazelor de date. Datele trebuie să poată fi restabilite când sistemul este indisponibil. Sistemul trebuie să fie tolerant la defectarea unor noduri singulare. Sistemul trebuie să asigure timpi de răspuns cât mai mici posibili. Pentru a asigura cerințele necesare există mai multe tehnici partajare, copiere și sincronizare a datelor între nodurile pe care este distribuită baza de date. Replicarea este un proces care constă în realizarea și distribuirea de copii ale datelor și, în plus, permite ca modificările efectuate să fie propagate în mod consistent la copiile corespunzătoare. Distribuirea acestor replici are ca scop procesarea datelor la nivel local. Procesul de replicare sporeşte securitatea sistemului și îmbunătăţeşte viteza operaţiunilor de procesare de date. În afară de aceasta, aplicaţia poate funcţiona chiar şi dacă un server local ar eşua, dar alte servere cu baze de date replicate rămân accesibile. În cazul replicării datelor, este mai dificil să se asigure menţinerea consistenţei acestora pentru că actualizarea unui fragment dintr-o bază de date trebuie să fie propagată la toate copiile fragmentului din bazele de date replicate. Decizia replicării unui fragment este influenţată de raportul dintre numărul de procese de scriere şi citire ale fragmentului. Copierea datelor pe mai multe servere concomitent oferă un paralelism de procesare. Marea problema pe care apare la replicare este timpul de propagare a noilor date către celelalte noduri, timp care trebuie să fie cât mai mic, aceasta impune cerințe mari asupra sistemului hardware și software. Pentru a asigura replicarea, în sistemele în timp real, fiecare nod rulează o buclă de sincronizare care poate comunica cu nodurile vecine dar și cu alte servere. Teoretic, în urma replicării de k ori, sistemul poate fi văzut din exterior ca un sistem care are o singură copie a datelor cu un hard care este mai puternic de k ori. În acest context putem spune că am obținut o scalabilitate verticală. Prin scalabilitate verticală se înțelege creșterea performanței sistemului prin creșterea performanței sistemului hard. Viceversa, este scalabilitatea orizontală, în care creșterea performanței sistemului este făcută pe baza creșterii numărului de mașini pe care rulează sistemul. În ambele cazuri, scalabilitate orizontală/scalabilitate verticală, performanța nu crește liniar cu creșterea performanței hardului/numărului de mașini[5]. Replicare poate să difere în funcție de gradul de replicare a datelor. O bază de date poate fi:

-Nereplicată – sistemul are o bază de date care se află într-un singur nod

-Parțial replicată – este atunci când există date partajate care sunt replicate, și date specifice fiecărui nod

-Total replicată – toată datele sunt replicate pe mai multe noduri.

În sistemele ce se doresc a fi performante se optează pentru replicarea totală a datelor, dar a în funcție de distribuția geografică a nodurilor și specificul aplicație, la fel de bine se poate opta pentru o replicare parțială. În ceea ce privește replicarea în sistemele real-time, trebuie ținut cont de câteva aspecte importante:

*Consistența* – Fiecare server își poate sincroniza replicile sale într-un anumit mod, ceea ce determină un anumit nivel de consistență. Trebuie de făcut un compromis între nivelul de consistență dorit și disponibilitatea datelor. Un nivel de consistență mai mare înseamnă un overhead de sincronizare mai mare, ceea ce implică o performanță mai mică.

*Responsivitatea* – prin responsivitate înțelegem timpul de reacție la o cerea a utilizatorului. În general timpul de răspuns a aplicațiilor ce au baze de date trebuie să fie între 50-100 ms pentru a avea o performanță optimă. Prin replicare se pot aduce datele mai aproape de utilizator, micșorând timpul de răspuns al aplicației.

*Scalabilitatea* – Prin scalabilitate se înțelege posibilitatea de a extinde aplicația pe mai multe mașini cu scopul de a acomoda mai mulți utilizatori. Cum s-a explicat anterior replicarea oferă o scalabilitate pseudo verticală.

# reguli pentru replicare

Pentru ca un sistem de gestiune al bazei de date să fie cu adevărat distribuit, el ar trebui să respecte în totalitate cele 10 reguli introduse de C.J. Date in 1987. Putem afirma că aceste reguli sintetizează motivele principale pentru care este necesară distribuirea datelor pe mai multe calculatoare într-o rețea și constituie în același timp și obiectivele principale ale unui SGBDD. Cele 10 reguli de date formulate se constituie ca o metrică de evaluarea a bazelor de date distribuite, cu toate ca nu se cunoaște încă nici un produs care să satisfacă toate aceste cerințe. Sintetizate cele 10 reguli sunt[4]:

1. Autonomia locala. Un sistem de baze de date distribuit trebuie sa ofere pentru fiecare locație implicata un grad înalt al autonomiei locale. Ca urmare, fiecare locație trebuie să-și gestioneze propriile date și să ofere funcționalitate independent de celelalte locații.

2. Absenta unei dependente de o locație centrală. Este de dorit ca toate locațiile să aibă aceleași capabilități. Chiar dacă între locații exista o anumită ierarhie, nu trebuie exagerat. Într-un sistem distribuit nu trebuie sa existe o locație răspunzătoare în totalitate de realizarea unei anumite funcții sau activități.

3. Operare continua sau independenta la defectare. Obiectivul operării continue se refera la doua aspecte majore ce privesc îndeaproape performantele sistemelor distribuite, și anume: fiabilitatea și disponibilitatea. Fiabilitatea reprezintă capacitatea unui sistem de a funcționa fără întrerupere cu asigurarea unor parametri de calitate. Sistemele distribuite, spre deosebire de sistemele tranzacționale, nu urmăresc principiul atomicității. Acestea operează neîntrerupt, chiar și în cazul apariției unei defecțiuni la componente sau la mediul de comunicație. Disponibilitatea se refera la aspectul funcționării sistemului pe o perioadă prestabilită, fără posibilitatea ca, din anumite motive, o parte din date sa fie inaccesibile. În principal cerința presupune capabilitatea de ajustare a bazei de date fără a necesita trecerea acesteia offline.

4. Independenta de localizare. Locul in care au fost stocate sau de unde sunt accesate datele trebuie sa fie transparent, atât pentru utilizator, cat si pentru aplicațiile care rulează. Utilizatorul trebuie sa se comporte în aceeași maniera si sa fie servit de către sistem in consecință, chiar daca el lansează cereri de pe nodul A, iar datele solicitate se afla la nodul A, X sau Y. Acest principiu poarta uneori numele de “transparenta la localizare”.

5. Independenta de fragmentare. Spre deosebire de bazele de date centralizate, destinate unui nod central, baza de date distribuita este împarțită in fragmente ce sunt distribuite pe toate locațiile sistemului. Aceste porțiuni ale bazei globale se numesc fragmente si datorita dimensiunii lor in raport cu sursa din care provin vor duce la creșterea performantelor sistemului. Fragmentarea bazei de date are impact direct asupra cresterii concurentei bazei de date, întrucât in momentul in care un utilizator dorește să facă o actualizare, va bloca pentru o scurta perioada de timp doar o mica parte a unui fragment, deci o parte nesemnificativa a bazei de date globale[4].

6. Independenta la replicare. Atât din motive de securitate cat si de disponibilitate a datelor, anumite fragmente trebuie sa fie copiate la mai multe locații. Aceste copii se numesc replici sau reproduceri. Utilizarea replicilor si alocarea lor pe diferite locații trebuie sa fie făcută transparent față de utilizator. Acesta trebuie sa fie în continuare convins ca lucrează de fapt pe întreaga baza care se afla stocata local. Acest lucru este posibil daca replicile sunt plasate acolo unde sunt direcționate frecvent cererile specifice. Independenta de replicare se mai numește si transparenta la replicare. Conform acesteia, utilizatorul va trebui sa transmită cereri către fragmente bine Sisteme de baze de date distribuite – Dorin Cârstoiu 5 delimitate, fără a fi obligat totuși sa cunoască localizarea si nici faptul ca fragmentele consultate ar mai avea si alte amplasări posibile în rețea.

7. Gestionarea distribuită a tranzacțiilor. Gestiunea tranzacțiilor are ca obiect de studiu controlul accesului concurent si problematica tolerantei la defecte. Fata de abordarea cu un singur nod central, in cazul sistemelor distribuite lucrurile devin mult mai complicate. O tranzacție lansata in sistemul distribuit are nevoie de agenți, care sa verifice rezultatul finalizării tranzacției în fiecare din locațiile implicate. Evident ca și în mediul distribuit proprietățile tranzacțiilor (ACID - Atomicitate, Coerenta, Izolare, Durabilitate) trebuie sa fie respectate. Gestionarea distribuita a tranzacțiilor este strâns legata de problematica transparentei tranzacțiilor. Fără a lua in considerare aspectele privitoare la controlul concurentei si a rezistentei la defecte, într-un mediu distribuit, descompunerea unei tranzacții in subtranzacții duce la creșterea performantelor atât la nivelul vitezei de execuție, cat si din punctul de vedere al concurentei.

8. Independenta de sistemul de operare. Un sistem distribuit trebuie sa poate funcționa atât pe mașini diferite din punct de vedere al tehnologiei hardware, cat si pe calculatoare pe care sunt instalate sisteme de operare diferite.

9.Independenta de infrastructura de comunicație. Nodurile sistemului distribuit sunt conectate logic la aceeași rețea de comunicație. Chiar daca topologiile, vitezele de transfer, dimensiunile pachetelor, metodele de acces la mediu sau tehnologia subrețelelor, peste care acesta se întinde, sunt diferite, acestea nu trebuie sa devina un impediment in buna funcționare a sistemului.

10. Independenta de sistemul de gestiune al bazei de date. Acest deziderat este unul dintre cel mai greu de atins. Ideal ar fi ca pe toate mașinile sistemului distribuit sa ruleze același SGBD. Nu întotdeauna acest lucru poate fi pus in practica datorita eterogenității echipamentelor si a sistemelor de operare, fără a ne putea detașa si de latura financiara. Chiar daca SGBD-urile sunt diferite, ar fi cel puțin recomandat ca sa nu existe problema incompatibilității canalelor de comunicare intre locațiile având SGBD-uri diferite. Pentru aceasta, fiecare locație ar trebui sa recunoască același standard SQL.

Similar cu sistemele de baze de date cetralizate operațiile asupra bazei de date sunt grupate in tranzacții. Într-o baza de date distribuita tranzacțiile se împart în:

-tranzacții locale– acele tranzacții pentru execuția cărora nu sunt necesare resurse stocate pe un alt nod și tranzacția este ceruta de un utilizator conectat la baza de date locala;

-tranzacții distante (remote) – sunt tranzacții cerute de un utilizator conectat la baza de date de la alt nod, dar pentru execuția sa sunt necesare numai resurse aferente unui singur nod distant;

- tranzacții distribuite – tranzacții lansate de un utilizator conectat la baza de date aferenta unui nod pentru a căror execuție sunt necesare resurse găzduite de cel puțin două noduri, indiferent daca ambele sunt distante sau unul este local[4].

# avantaje și dezavantaje

În primul rând prin distribuirea datelor unei baze de date pe mai multe noduri obținem o extensie a spațiului de stocare și în același timp oferim resurse de procesare multiple. Chiar daca puterea de calcul a crescut foarte mult în ultimii ani, la prelucrarea volumelor mari de date, performanțele devin inacceptabile. Prin distribuția datelor pe mai multe centre de procesare se obțin avantaje considerabile privind performanțele în principal datorate prelucrării paralele pe mai multe noduri, dar garantarea îndeplinirii proprietăților ACID este mult mai greu de realizat. Pe lângă acestea, sistemele de baze de date distribuite oferă o serie de avantaje adiționale[4]:

-Se identifică cu structura organizațională a multor organizații, pe care o modelează mult mai bine, având în vedere ca multe companii sunt localizate "distribuit" din punct de vedere geografic;

-Cu toate ca datele sunt partajabile, administrarea lor se bucura de un grad înalt de autonomie locala. Datele păstrate pe fiecare site pot fi administrate independent de către administrator diferiți;

-Oferă o disponibilitate ridicata pentru accesul la baza de date fata de sistemele centralizate. Chiar si in cazul in care apar erori de comunicație sau defectări de noduri sistemul poate continua sa funcționeze în condiții satisfăcătoare;

-Se îmbunătățește fiabilitatea sistemului întrucât pot fi refăcute rapid fișierele distruse utilizând replici găzduite de alte noduri;

-Diminuarea costurilor de implementare si întreținere prin utilizarea de echipamente uzuale in nodurile de prelucrarea fata de un sistem centralizat care sa ofere aceeași putere de prelucrare. Prin realizarea a cat mai multe prelucrări locale, arunci când aplicațiile permit acest lucru, scade costul de exploatare prin diminuarea costului comunicațiilor.

-Creșterea scalabilității sistemului prin posibilitatea de adăugare sau extragere a noi noduri, funcție de necesitățile de exploatare. Ajustarea puterii de calcul la sistemele centralizate este mult mai costisitoare si in plus necesita timpi de oprire a funcționalităților; nu trebuie însă sa minimalizam efectele negative ale distribuției, dintre care principalele sunt:

-Creșterea complexității cu efecte in ceea ce privește complicațiile de implementare și gestionare, necesitatea existentei unui personal specializat, necesitatea unei infrastructuri de comunicație;

-Sensibilitatea la erori datorata algoritmilor de procesare distribuita pentru care consistenta datelor este greu de menținut;

-Necesitatea unor prelucrări suplimentare pentru asigurarea schimburilor de mesaje intre noduri si pentru coordonarea activității acestora;

-Deschiderea de noi breșe de securitate in comparație cu un sistem centralizat. Sistemul, prin distribuția sa geografica nu poate fi ascuns in spatele unui firewall, o serie de date sensibile sunt schimbate intre noduri pe infrastructura de comunicație;

-Existenta mai multor entități de administrare cu politici de administrare diferite;

-Asigurarea integrității datelor este greu de realizat fata de sistemul centralizat. Din cauza costurilor mari de comunicație, dar și a timpilor de răspuns, de cele mai multe ori se renunță la o parte din regulile ce trebuie verificate pentru datele din baza.

-Probleme legate de standardizare. Chiar daca nu se poate vorbi de o lipsa totala de standardizare putem afirma ca nu sunt încă standarde internaționale unanim acceptate care sa garanteze comunicarea eficienta, proiectarea, gestionarea si exploatarea datelor in sisteme distribuite.

-Este posibilă apariția unui flux mare de informații intre noduri care sa necesite rezolvarea problemelor legate de sincronizarea mesajelor, detectarea erorilor, inconsistenta datelor redundante[4].

# mecanisme de replicare și sincronizare

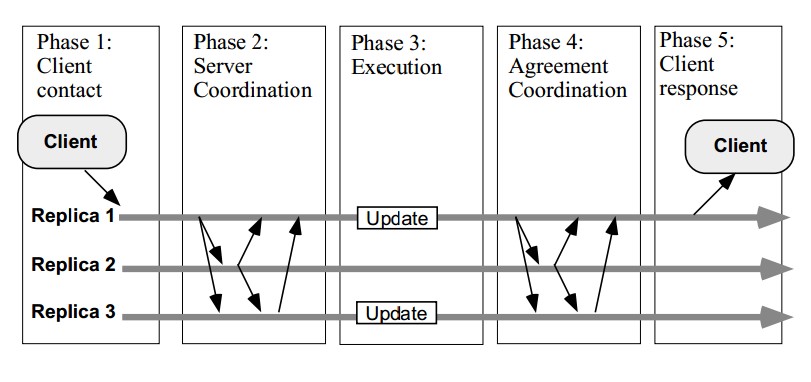
Replicarea datelor este un concept fundamental în sistemele de baze de date distribuite care a fost studiat o perioadă lungă de timp. Au fost dezvoltate mai multe tehnici de replicare pentru a crește scalabilitatea, fiabilitatea și performanța sistemelor distribuite. Aceste mecanisme trebuie să țină cont de dimensiunea datelor cu care lucrează sistemul, de numărul de noduri și performanța fiecărui nod, de lățimea de bandă pe care o au nodurile pentru a comunica între ele. Au fost dezvoltate tehnici de replicare activă și pasivă cu modalitate de sincronizare imediată și respectiv „leneșă”. Fiecare tehnică de repicare are caracteristicile sale specifice. Conform teoremei CAP, nu putem avea consistență, disponibilitate și toleranță la partiționare concomitent. Trebuie să facem un compromis între aceste trei caracteristici în funcție de aplicație. Într-un sistem bancar am dori să avem consistență maxim și să facem optimizarea între celelalte 2 caracteristici. Într-un sistem de distribuție de fișiere mai degrabă am vrea să avem toleranță la partiționare, ceea ce ar permite distribuirea datelor pe mai multe noduri. Există 2 criterii de clasificare a mecanismelor de replicare[5]:

1.Modul în care se sincronizează datele și selectarea datelor care trebuie sincronizate

2.Modul de selecție a nodurilor care trebuie să primească datele ce trebuie sincronizate

 Replicarea imediată (Eager replication)

Prin replicare rapidă se înțelege faptul că odată ce un nod master a suferit niște schimbări de date, nodul master are grijă să inițializeze imediat procesul de sincronizare a datelor. Tote nodurile replici primesc date pentru a împrospăta baza de date pe care o dețin. Acest tip de sincronizare folosește tranzacții imbricate bazate pe protocoale pentru tranzacții distribuite care asigură atomicitatea și consistența datelor. Overhead-ul de sincronizare este mult mai mare ca în cazul replicării leneșe.



*Fig.2. Schema de sincronizare rapidă[5]*

Din schema de mai sus observăm că clientul nu primește confirmare până când datele nu sunt copiate pe toate replicile necesare. Clientul fie execută tranzacția pe toate replicile, fie pe niciuna. Aceasta introduce un timp de răspuns foarte mare.

 Replicarea leneșă

Nu se face copierea imediată a datelor pe toate replicile. Clientul primește un răspuns de confirmare de la nodul master, iar nodul master își ia responsabilitatea să propage schimbările și pe replicile necesare dar peste un anumit timp. Acest mecanism de replicare scade timpul de răspuns al aplicației, dar necesită algoritmi mai complecși de sincronizare care să țină cont de eventuale conflicte ce pot apărea între date care au fost modificate pe noduri diferite. În acest sens se poate folosi algoritmul Lamport.

Avantajul replicării leneșe este creșterea responsivității sistemului cu prețul reducerii consistenței[5].

# tipuri de replicare

Pot fi următoarele tipuri de replicare:

-Replicarea snapshot – Se copie datele integral de pe un server pe altul, sau dintr-o baza de date în alta.

-Replicarea prin îmbinarea datelor – datele din mai multe baze de date sunt îmbinate și copiate într-o bază de date.

-Replicarea tranzacțională – este creată o replică inițială de tip snapshot după care se primesc update-uri periodice în funcție de schimbările survenite.

 Replicarea snapshot

Este una dintre cele mai simple metode de replicare. Replicarea periodică funcționează prin unor blocuri mari de date. Poate fi folosită atunci când replica funcționează într-un mod read only sau când se poate funcționa fără a avea cele mai recente date. Timpul dintre doua update-uri consecutive se numește latență, și în această perioadă se lucrează cu date care pot fi false.

Acest tip de replicare se poate folosi pentru date care se schimbă rar și pe lângă asta ne permitem să existe o probabilitate ca aceste date să fie eronate. De exemplu poate exista o bază centrală a unei companii cu inventarul companiei, și săptămânal să fie făcută replicarea pe bazele de date din filialele companiei. Aceste date sunt relativ constante și în cazul unor erori nu provoacă mari pierderi. Acest tip de replicare poate fi folosit în soluțiile în care conexiunea dintre noduri este instabila. Replicarea snapshot, este singurul tip de replicare care poate fi folosit pentru date care sunt organizate în tabele fără cheie primară. La replicarea snapshot nodul master creează niște fișiere numite snapshot-uri, care le distribuie către celelalte noduri. Din fișierele snapshot sunt extrase datele și introduse în bazele de date respective.

Cazuri în care poate fi folosită replicarea snapshot:

-Datele sunt relativ statice

-Este acceptabil să avem date nesincronizate

-Când volumul de date care se modifică e mare dar se întâmplă rar.

 Replicarea prin îmbinarea datelor (merge replication)

Replicarea prin îmbinarea datelor permite ca nodul care publică datele să salveze pentru fiecare schimbare intervenite un snapshot în cazul în care nu există legătura între cu nodurile replici, iar apoi când legătura este restabilită se extrag datele din snapshot-uri care se combină și se transmit către nodurile replici. Acest tip de replicare permite nodurilor să lucreze off-line ca mai apoi să se poată face sincronizările necesare.

Având în vedere că se pot acumula un număr mare de date nesincronizate, pot apărea conflicte între versiuni diferite de date de pe noduri diferite. În acest sens este necesară o modalitate de arbitrare a conflictelor. Modalitățile de rezolvare a conflictelor sunt configurabile. Acest tip de replicare este cel mai complex tip de replicare deoarece permite atât nodului sursă de date cât și nodului replică destinație să facă schimbări pe aceleași date. Nodurile pot lucra independent o anumită perioadă de timp.

Replicarea prin îmbinarea datelor este utilă când:

-Aceleși date se schimbă pe noduri diferite și necesită să fie aleasă o singură versiune.

-Există perioade în care nodurile sunt off-line

-Apar un număr acceptabil de conflicte și în cazul în care apar ne permitem să fie violate conceptele ACID.

###  Replicarea tranzacțională

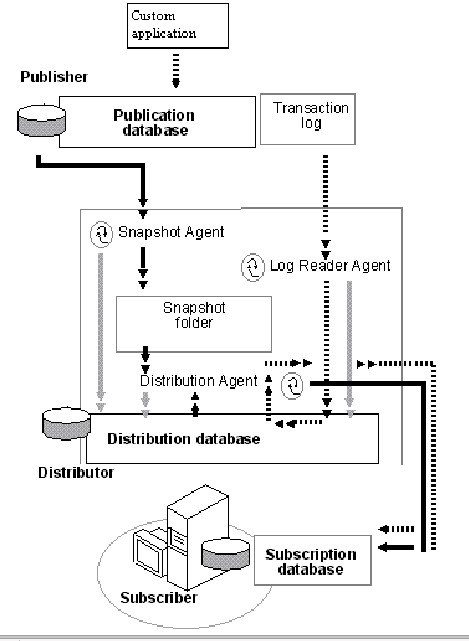
Poate fi considerată opusul replicării de tip snapshot. Sincronizarea este facută imdiat ce apar schimbări de date. Fiecare tranzacție care apare este imediat replicată și pe nodurile necesare. Poate fi cofigurată și opțiunea de a acumula datele de la mai multe tranzacții și să fie transmise toate odată. Acest tip de replicare necesită o conexiune bună între noduri, bandă destul de largă și principalul latență mică. Tranzacțiile se stochează într-un fișier de Log-uri. Dacă conexiunea pică acest fișier poate crește rapid. Replicarea tranzacțională începe cu un snapsot al datelor inițiale. Datele din snapshot sunt alterate apoi prin aplicarea tranzacțiilor. Se poate seta perioadă de sincronizare a snapshot-ului. Odată ce a fost creat snapshot-ul datelor inițiale, replicarea tranzacțională se face cu ajutorul unui agent care citește fișierele Log, extrage tranzacțiile și le stochează într-o bază de date distribuită. Un alt agent de distribuire extrage tranzacțiile din baza de date distribuită și le transmite către nodurile destinație unde fiecare nod își updatează snapshot-ul inițial.

Replicarea tranzacțională este utilă când[6]:

-Infrastructura ne permite, adică avem o conectare stabilă a nodurilor

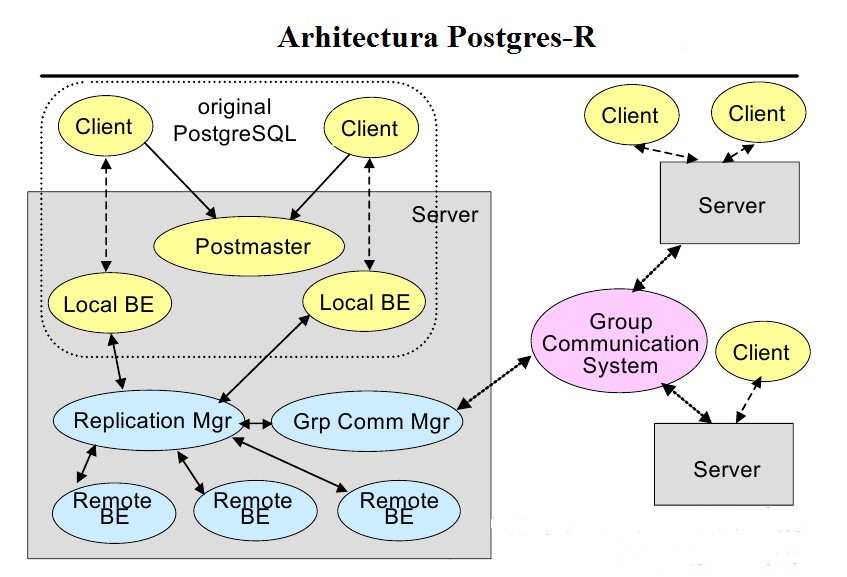
-Dorim ca schimbările să se propage către nodurile destinație imediat ce apar, adică să asigurăm o consistență sporită

-Toate copierile de date au caracteristici ACID.

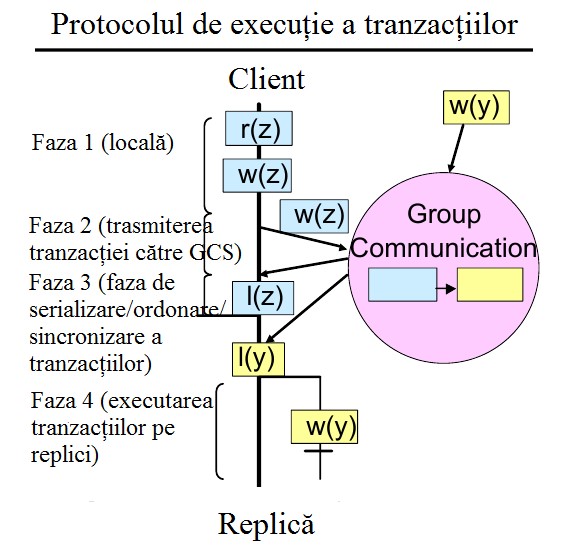
 *Fig.3. Schema replicării tranzacționale[6]*

# replicarea în postgresql

Există mai multe tool-uri pentru replicarea bazelor de date PostgreSQL printre care: Usogres, eRServer, PostgreSQL Replicator, Postgres-R. Postgres-R este o extensie PostgreSQL care oferă metode eficiente de replicare a bazelor de date cluster, totodată asigurând consistența. Este folosită în primul rînd pentru echilibrarea/distribuirea încărcării sistemului și disponibilitate sporită a sistemului. Postgres-R asigură o scalabilitate a procesului de replicare, este mai fiabil și flexibil în comparație cu sistemele centralizate de replicare. Postgres-R poate replica obiecte mari și folosește replicare asincronă de tipul store and forward. Se pot adăuga sau scote noduri în timpul rulării sistemului fără a fi nevoie de a fi oprit. Nodurile picate sunt detectate și scoase din sistem fără ca sistemul să fie afectat. Postgres-R folosește algoritmul 2PL pentru controlul tranzacțiilor concurente. Tranzacțiile sunt de tipul read-one-write-all și sunt execută în 4 faze. În prima fază tranzacția se execută local, pe nodul pe care a venit cererea. În faza a doua se propagă schimbările de date. Faza a treia este faza de sincronizare, în care nodurile replici primesc tranzacția. În faza a patra tranzacția este executată pe nodurile replici[7].

*Fig.4. Arhitectura Postgres-R[7]*

Pe fiecare server rulează câte o instanță de Postgres-R. Când un client vrea să acceseze baza de date, trimite cererea către Postmaster. Postmasterul creează câte un proces pentru fiecare bază de date locală pe care vrea să o acceseze clientul, după care se stabilește conexiunea dintre baza locală și client. Clientul se poate conecta la orice server disponibil. Tranzacțiile se execută în patru pași. Managerul de replicare „Replication Mgr” este responsabil de procesarea mesajelor externe.

*Fig.5. Replicarea tranzacțională în PostgresR[7]*

În primă fază tranzacțiile sunt executate local. În faza a doua sunt transmise operațiile care necesită scrieri de date către Group comunication system. GCS serializează mesajele primite stabilind între ele o ordine totală rezolvând conflictele, după care le transmite către nodurile replici. În faza a 4 mesajele sunt execute pe fiecare replică în parte[8].

Avantaje Postgres-R:

-Performanță în sensul de timpi de răspuns și rata datelor.

-Asigură consistență sporite și fiabilitatea sistemului (disponibilitate)

-Este construit peste SGBD PostgreSQL, deci este un modul în locul căruia poate fi folosit oricare altul, și nu depinde de versiunea PostgreSQL.

# failover

Failover-ul este capacitatea unui sistem de a continua să funcționeze chiar dacă una dintre componentele acestuia este defectă. Aceasta sugerează că funcțiile principale ale sistemului sunt realizate de componentele secundare în cazul în care cele primare sunt defecte. În cazul PostgreSQL, există diferite instrumente care permit să implementarea unui cluster de baze de date care sa fie rezistent la eșecuri. Un mecanism de redundanță disponibil nativ în PostgreSQL este replicarea datelor.

Așa cum deja am expus, replicarea reprezintă procesul de copiere și menținere a datelor actualizate într-unul sau mai multe noduri de bază de date. Este utilizat conceptul unui nod master care primește modificările asupra datelor și al unor noduri slave unde sunt replicate datele.

Există mai multe moduri de clasificare a replicării:

-Replicare sincronă: Nu există pierderi de date chiar dacă nodul nostru principal este defect total, dar noile componente secundare trebuie să aștepte o confirmare de la slave, ceea ce poate afecta performanța.

-Replicarea asincronă: Există o posibilitate de pierdere a datelor în cazul în care se pierde nodul principal. Dacă replica din anumite motive nu este actualizată în momentul incidentului, informațiile care nu au fost copiate pot fi pierdute.

-Replicarea fizică: se copiază blocurile de discuri.

-Replicarea logică: transmiterea fluxurilor de date.

-Warm Standby Slaves: Nu suportă conexiuni.

-Hot Standby Slaves: Suportă conexiuni numai pentru citire, utile pentru rapoarte sau interogări[9].

Dacă serverul primar eșuează, atunci server-ul de așteptare ar trebui să înceapă procedurile de reajustare. Dacă serverul în așteptare poate fi repornit, chiar și după un timp, procesul de recuperare poate fi, de asemenea, repornit imediat, profitând de recuperarea ulterioară. Dacă serverul în așteptare nu poate fi repornit, atunci ar trebui să fie creată o nouă instanță de rezervă nouă. Dacă serverul primar eșuează și serverul de așteptare devine noul primar și apoi vechiul server primar își revine, trebuie să existe un mecanism pentru a informa vechiul primar că acesta nu mai este principalul. Aceasta este uneori cunoscută sub numele de STONITH, care este necesară pentru a evita situațiile în care ambele sisteme consideră că acestea sunt primare, ceea ce va duce la confuzie și, în cele din urmă, la pierderea datelor. Multe sisteme failover utilizează doar două sisteme, primar și standby, conectate printr-un mecanism asemănător cu cel al bătăilor de inimă pentru a verifica în permanență conectivitatea dintre cele două și viabilitatea celui primar. Este, de asemenea, posibil să se utilizați un al treilea sistem (numit server de mărturii) pentru a preveni unele cazuri de eroare inadecvată, dar complexitatea suplimentară ar putea să nu fie utilă decât dacă este prevăzută cu o grijă și o testare riguroasă.

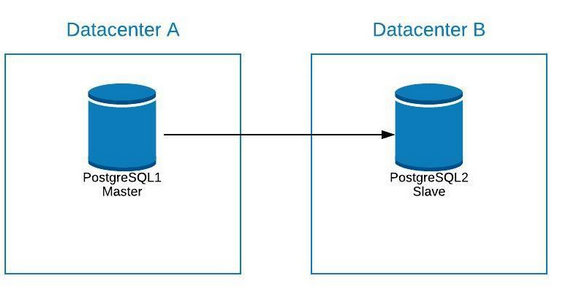
PostgreSQL nu furnizează un software de sistem necesar pentru a identifica erorile pe server-ul primar și pentru a notifica server-ul în așteptare. Există multe astfel de instrumente și sunt bine integrate cu facilitățile sistemului de operare necesare pentru o reușită, cum ar fi migrarea adreselor IP. După apariția failover-ului pe server-ul principal, mai rămâne un singur server în funcțiune. Aceasta este cunoscută ca o stare degenerată. Fosta stare de așteptare este acum principala, dar server-ul de bază este defect și s-ar putea să rămână în această stare. Pentru a reveni la funcționarea normală, un server în așteptare trebuie să fie recreat, fie pe fostul sistem primar, fie pe un al treilea sistem, eventual nou deși acest lucru complică în mod clar configurația sistemului și procesele operaționale. Odată finalizat, se poate considera că server-ul primar și server-ul de așteptare au schimbat rolurile. Astfel, trecerea de la serverul primar la cel de așteptare poate fi rapidă, dar necesită timp pentru a pregăti cluster-ul de failover. Comutarea regulată de la server-ul primar la server-ul de așteptare este utilă, deoarece permite întreruperi regulate pe fiecare sistem pentru întreținere. Acest lucru serveste, de asemenea, ca un test al mecanismului failover pentru a se asigura ca va funcționa atunci când va fi nevoie de el[10].

Există mai multe posibilități de utilizare a failover-ului[9]:

**Migrare**

Dacă vrem să migrăm de la un centru de date la altul prin reducerea la minimum a timpului de nefuncționare, putem utiliza failover-ul.

Să presupunem că master-ul se află în datacenter-ul A și se dorește migrarea sistemele la datacenter-ul B.



*Fig.6.* *Diagrama migrării master-ului[9]*

Astfel, se poate crea o replică în datacenter-ul B. Odată ce este sincronizat, trebuie să se oprească sistemul, și să se declare slave-ul ca noul master înainte de a se indica sistemului noul master din datacenter-ul B. Failover-ul nu se referă numai la baza de date, ci și la aplicațiile de pe acestea care nu vor ști la care dintre bazele de date să se conecteze. Cu siguranță nu este o soluție sa se modifice aplicația deoarece aceasta va extinde doar timpul de nefuncționare. Deci, se poate configura un loadbalancer astfel încât atunci când se defectează master-ul sistemului, acesta va direcționa automat către următorul server care este promovat. O altă opțiune este utilizarea DNS. Prin migrarea nodului principal în noul datacenter, se modifică direct adresa IP a numelui host-ului ce reprezintă master-ul. În acest fel, se evită modificarea aplicației și, deși nu se poate face automat, este o alternativă dacă nu se dorește implementarea unui load balancer.

Având o singură instanță de loadbalancer nu este chiar optim deoarece poate deveni un singur punct de eșec. Prin urmare, se poate implementa și failover pentru load balancer, utilizând un serviciu precum keepalived. În acest fel, dacă există o problemă cu load balancer-ul primar, keepalived este responsabil pentru migrarea IP-ului la load balancer-ul secundar și astfel, totul continuă să funcționeze în mod transparent.

**Întreținere**

Dacă se efectuează orice întreținere pe serverul master postgreSQL, se poate promova slave-ul pentru a executa sarcina și apoi sa se reconstruiască un slave pe vechiul master. După aceasta se poate promova vechiul master urmat de repetarea ​​procesului de reconstrucție al slave-ului, revenind la starea inițială. În acest fel, am putea lucra pe serverul nostru, fără a risca să fim offline sau să pierdem informații în timpul efectuării întreținerii.

# realizare practică

Replicarea bazei de date master / slave este un proces de copiere (sincronizare) a datelor dintr-o bază de date de pe un server (master) către o bază de date de pe un alt server (slave). Principalul beneficiu al acestui proces este distribuirea bazelor de date către mai multe mașini, astfel că atunci când serverul principal are o problemă, există o mașină de rezervă cu aceleași date disponibile pentru a gestiona cererile fără întrerupere. PostgreSQL oferă mai multe modalități de replicare a unei baze de date. În acest proiect este folosită replicarea PostgreSQL utilizând modul hot standby. Modul hot standby necesită 2 servere de baze de date. În acest proiect am folosit Ubuntu ca sistem de operare pe ambele servere.

Master Server - acceptă conexiuni de la client cu permisiuni de citire și scriere. Slave Server - serverul de rezervă rulează copia datelor de pe serverul principal.

MASTER SERVER

Adăugarea apt repository-ului pentru a putea instala postgres:

sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ $(lsb\_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

sudo apt-get install wget ca-certificates wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

sudo apt-get update

Instalarea PostgreSQL 10:

sudo apt-get install postgresql-10

Crearea user-ului de replicare:

Postgres-ul necesită un user cu drepturi de replicare pentru a trage datele. Prin convenție, utilizatorii folosesc un user numit replication cu permisiunea de replicare.

sudo su - postgres

psql

CREATE USER replication REPLICATION LOGIN ENCRYPTED PASSWORD 'password1';

Setarea permisiunilor de conectare pe user-ul replication:

După ce s-a creat user-ul replication, trebuie setată și permisiunea de a se conecta efectiv pe serverul master. Acest lucru se face în fișierul pg\_hba.conf de la calea /etc/postgresql/10/main/pg\_hba.conf unde se declară user-ul și baza de unde se face conectarea. În acest caz, ip-ul server-ului de pe care se face conectarea (serverul slave) este 192.168.56.101.

host replication replication 192.168.56.101/32 md5

Configurările pentru crearea arhivei de replicare:

Una dintre părțile cele mai critice ale configurației replicării este salvarea datelor pentru arhivarea fișierelor, astfel încât acestea să poată fi trase la replici.

Configurările se fac în fișierul de configurare de la calea /etc/postgresql/10/main/postgresql.conf astfel:

Listen\_addresses = `\*` wal\_level = replica max\_wal\_senders = 2 wal\_keep\_segments = 32 archive\_mode =on archive\_command = 'cp -i %p /var/lib/postgresql/10/main/archive/%f'

Crearea folder-ului de arhivare:

sudo su - postgres

mkdir /var/lib/postgresql/10/main/archive

În acest moment, server-ul master este configurat în totalitate și mai trebuie doar să dăm un restart pe clientul de postgres:

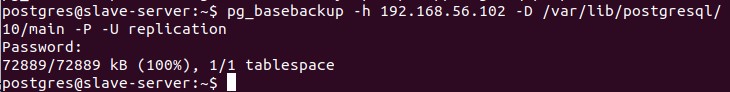
sudo service postgresql restart

SLAVE SERVER

Instalarea postgreSQL 10 așa cum am făcut la serverul master.

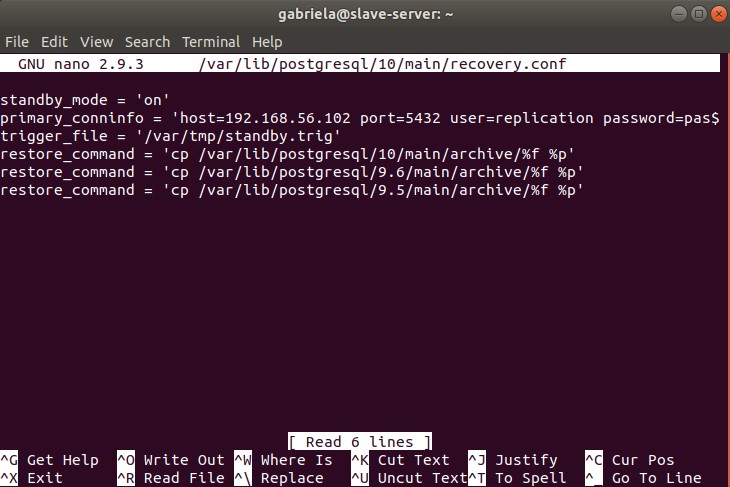
Configurarea fișierului postgresql.conf de la calea /etc/postgresql/10/main/postgresql.conf astfel:

Wal\_keep\_segments = 32 wal\_level = replica hot\_standby = on wal\_log\_hints = on

Sincronizarea cu serverul master pentru a putea extrage date și fișiere (192.168.56.102 este ip-ul serverului master) 

Crearea fișierului recovery.conf pentru continuarea replicării:

Până în acest moment am instalat baza postgres, am configurat-o și am tras date. Acum trebuie să adăugăm fișierul care se va conecta și va trage constant datele. De asemenea, va face comanda de restaurare a datelor. Fișierul trebuie să fie numit recovery.conf.

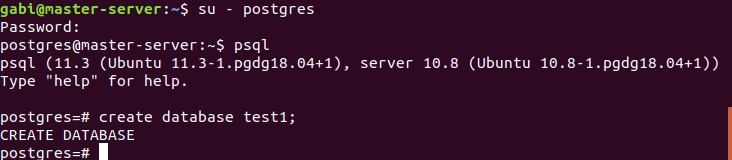


Restartarea serviciului postgres de pe serverul slave:

sudo systemctl restart postgresql

Testarea replicării:

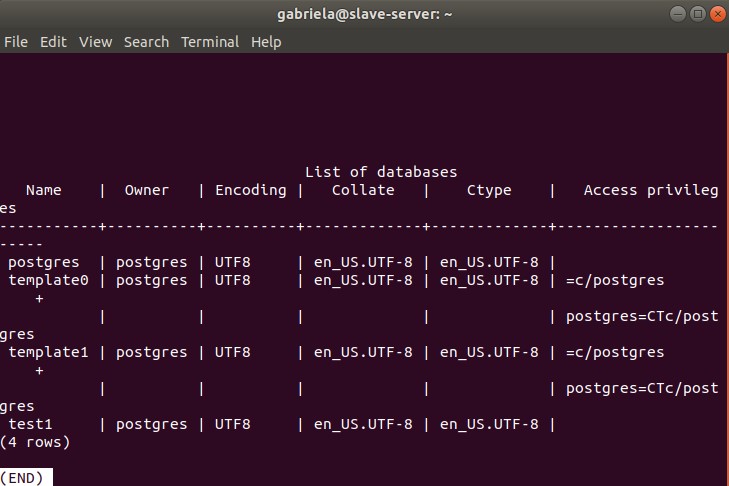
Crearea unei noi baze de date pe serverul primar, conectarea la replică și validarea acesteia:



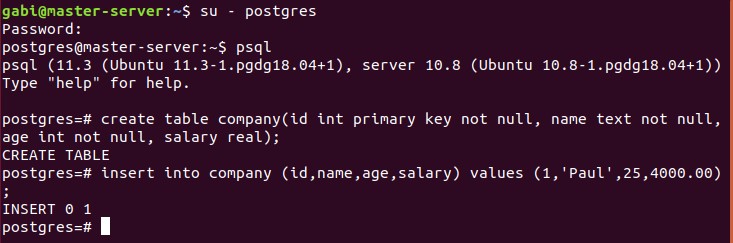
Verificarea existenței bazei “test1” pe serverul slave:

sudo su – postgres

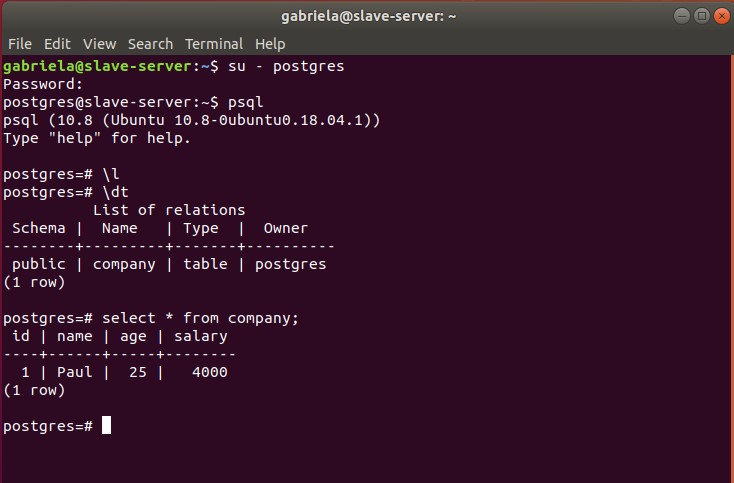
psql \l

Rezultatul:

Crearea unei tabele de serverul master și inserarea unei linii în aceasta:



Verificarea existenței datelor pe serverul slave:



# CONCLUZII

În această lucrare a fost prezentată și implementată o planificare complexă de replicare în cazul replicării asincrone pentru un sistem PostgreSQL de baze de date distribuite pentru a fi utilizat în scopul colectării datelor pe mai multe site-uri ale sistemului în vederea obținerii unei extensii a spațiului de stocare și în același timp oferirea unor resurse de procesare multiple. De asemenea, a fost prezentată și detectarea și corectarea conflictelor provenite din modificările concurente apărute la diferite site-uri între două evenimente de sincronizare de date.

Așa cum am expus în această lucrare, replicarea este un proces care constă în realizarea și distribuirea de copii ale datelor și, în plus, permite ca modificările efectuate să fie propagate în mod consistent către copiile corespunzătoare. Distribuirea acestor replici are ca scop procesarea datelor la nivel local.

Procesul de replicare sporește securitatea sistemului și îmbunătăţeşte viteza operaţiunilor de procesare de date. În afară de aceasta, sistemul poate funcţiona chiar și dacă un server local ar eşua, dar alte servere cu baze de date replicate rămân accesibile. În cazul replicării datelor, este mai dificil să se asigure menţinerea consistenţei acestora pentru că actualizarea unui fragment dintr-o bază de date trebuie să fie propagată la toate copiile fragmentului din bazele de date replicate.

# BIBLIOGRAFIE

**[1]**A set of tools for building PostgreSQL distributed databases in biomedical environment. M. Cavalleri, R. Prudentino, U. Pozzoli, G. Reni

**[2]**https://en.wikipedia.org/wiki/PostgreSQL

**[3]**<http://www.referatele.com/referate/noi/informatica/postgresql-225514248.php>

**[4]** Sisteme de baze de date distribuite, Dorin Cârstoiu

**[5]** Understanding Replication in Databases and Distributed Systems. M. Wiesmann, F. Pedone, A. Schipe, B. Kemme, G. Alonso.

**[6]** TRANSACTION MANAGEMENT IN A DISTRIBUTED DATABASE SYSTEM FOR LOCAL AREA NETWORKS, Marek Rusinkiewicz and Dimitrios Georgakopoulos

**[7]** International Journal of Recent Trends in Electrical & Electronics Sept 2011. ISSN: 22316612. Replication: Analysis & Tackle in Distributed Real Time Database System SANJAY KUMAR TIWARI, A.K.SHARMA, V.SWAROOP Department of Computer Sc. & Engg. M.M.M. Engineering College, Gorakhpur, U.P., India.

**[8]** PostgreSQL Database Replication Options. Darren Johnson

**[9]** Failover for PostgreSQL Replication 101, Sebastian Insausti

**[10]** https://www.postgresql.org/docs/9.5/warm-standby-failover.html