**Univerzitet u Nišu, Elektronski fakultet**

**Katedra za računarstvo**

**SEMINARSKI RAD**

**Replikacija kod MySQL-a**

**Doc. dr Aleksandar Stanimirović Student: Vesna Stojanović 1339**

**Niš, 2022.**

**Sadržaj**

# 

# 

[Uvod](#_q28ngwr867rq) 2

[Problemi koje rešava replikacija](#_1fob9te) 3

Prednosti replikacije u MySql-u 4

Šema replikacije 4

Rad [replikacij](#_3znysh7)e 6

[Replikacione topologije](#_2jxsxqh) 9

[Master i više replika](#_z337ya) 9

[Master-master u aktivan-aktivan modu](#_3j2qqm3) 10

[Master-master u aktivan-pasivan modu](#_1y810tw) 11

[Master-master sa replikama](#_4i7ojhp) 12

[Prsten replikacija](#_2xcytpi) 12

[Master, distribucioni master i replikacije](#_1ci93xb) 13

[Stablo ili piramida](#_3whwml4) 15

Primer replikacije 16

[Administracija i održavanje replikacije](#_2bn6wsx) 22

[Merenje zaostajanja replikacije](#_qppsxbp5e8z) 23

[Ponovna sinhronizacija replike sa masterom](#_1pxezwc) 23

[Promena mastera](#_49x2ik5) 24

[Planirane promocije](#_2p2csry) 24

[Neplanirane promocije](#_147n2zr) 25

[Pronalaženje željenih pozicija loga](#_3o7alnk) 26

Primer failover master-slave replikacije 28

Zaključak 32

Literatura 33

# **Uvod**

Replikacija omogućava da se podaci sa jednog MySql servera baze podataka (izvora) kopiraju na jedan ili više MySql servera baze podataka (replike). Replikacija je podrazumevano asinhrona; replike ne moraju biti stalno povezane da bi primale ažuriranja iz source-a. U zavisnosti od konfiguracije, možete replicirati sve baze podataka, izabrane baze podataka ili čak izabrane tabele unutar baze podataka.

Osnovni problem koji replikacija rešava je čuvanje podataka jednog servera sinhronizovanih sa podacima drugog. Mnoge replike se mogu povezati sa jednim masterom i ostati sinhronizovane sa njim, a replika, za uzvrat, može da se ponaša kao master.

MySQL podržava dve vrste replikacija:

* logičke replikacije (*eng. statement-based replication*)
* sirove replikacije (*eng. row-based replication*).

Replikacije zasnovane na naredbi ili logičke replikacije su dostupne od MySQL 3.23. Sirove replikacije su dodate u MySQL 5.1. Obe vrste rade tako što snimaju promene u binarnom logu mastera i reprodukuju log na repliku, i obe su asinhrone – to jest, nije zagarantovano da kopija podataka na replici bude ažurna u datom trenutku. Ne postoji garancija koliko velike latencije mogu biti na replici. Veliki upiti mogu učiniti da replika zaostaje sekundama, minutima ili čak satima iza mastera.

MySQL-ova replikacija je uglavnom kompatibilna unazad. Odnosno, noviji server obično može da bude replika starijeg servera bez problema. Međutim, starije verzije servera često ne mogu da služe kao replike novije verzije: one možda ne razumeju nove funkcije ili SQL sintaksu novijeg servera, a mogu postajati i razlike u formatima fajla koje replikacija koristi.

Replikacija zahteva da binarno logovanje bude omogućeno na masteru, što može imati značajne troškove, ali to nam je svakako potrebno za ispravno pravljenje backup-ova i oporavak u određenom trenutku. Pored binarnog logovanja, svaka dodata replika dodaje i dodatna opterećenja na masteru tokom normalnog rada. Ako replike čitaju stare binarne logove sa mastera, a ne samo prate najnovije događaje, troškovi mogu biti mnogo veći zbog I/O zahteva za čitanjem starih logova.

Replikacija je relativno dobra za skaliranje čitanja, koja možemo da usmerimo ka replici, ali nije dobar način za skaliranje upisa ako je ne dizajniramo ispravno. Pridruživanje više replika masteru prouzrokuje da se više puta obavlja upis, po jednom na svaku repliku. Ceo sistem je ograničen brojem upisa koji najslabiji deo može da izvede.

## **Problemi koje rešava replikacija**

Ovo su neke uobičajene upotrebe replikacije:

* **Distribucija podataka** – MySQL replikacija obično ne zahteva veliku propustnost, iako ćemo videti kasnije da sirova replikacija može da koristi mnogo veću propustnost od tradicionalnih logičkih replikacija. Takođe možemo da je zaustavimo i pokrenemo kada želimo. Stoga, replikacija je korisna za održavanje kopije podataka na geografski udaljenoj lokaciji, kao što je drugi data centar. Udaljena replika može čak da radi i kada dođe do prekida veze. Međutim, ako želimo da naše replike imaju veoma mali zastoj replikacije, biće nam potrebna stabilna veza sa malim kašnjenjem.
* **Balansiranje opterećenja** – MySQL replikacija može da nam pomogne da distribuiramo upite za čitanje na više servera. Možemo da uradimo osnovno balansiranje opterećenja sa nekoliko jednostavnih promena u kodu. Možemo da koristimo pojednostavljene, ali i sofisticiranije pristupe. Standarna rešenja za balansiranje opterećenja, kao što su mrežni proizvodi za balansiranje opterećenja, mogu dobro da funkcionišu za distribuciju opterećenja među MySQL serverima.
* **Pravljenje backup-ova** – Replikacija je dragocena tehnika za pomoć pri pravljenju backup-ova. Međutim, replika nije backup niti je zamena za backup-ove.
* **Visoka dostupnost i prevazilaženje grešaka** - Replikacija nam može pomoći da izbegnemo da MySQL postane jedinstvena tačka neuspeha u našoj aplikaciji. Dobar sistem za prevazilaženje grešaka koji uključuje replikaciju može značajno da smanjenji vreme zastoja.
* **Testiranje MySQL upgrade-ova** – Uobičajena je praksa da postavimo repliku sa upgrade-ovanom MySQL verzijom i da je koristimo kako bismo osigurali da naši upiti rade onako kako se očekuje, pre upgrade-a svake instance.

**Prednosti replikacije u MySql-u**

* **Rešenja za skaliranje** – raspoređivanje opterećenja između više replika radi poboljšanja performansi. U ovom okruženju, sva upisivanja i ažuriranja se moraju odvijati na izvornom serveru. Čitanja se, međutim, mogu odvijati na jednoj ili više replika. Ovaj model može poboljšati performanse upisivanja (pošto je izvor posvećen ažuriranjima), dok dramatično povećava brzinu čitanja u sve većem broju replika.
* **Sigurnost podataka** – pošto replika može pauzirati proces replikacije, moguće je pokrenuti usluge rezervnog kopiranja na replici bez oštećenja odgovarajućih izvornih podataka.
* **Analitika** – podaci uživo se mogu kreirati na izvoru, dok se analiza informacija može odvijati na replici bez uticaja na performanse izvora.
* **Distribucija podataka na velike udaljenosti** – možete koristiti replikaciju da biste kreirali lokalnu kopiju podataka za udaljenu lokaciju za korišćenje, bez trajnog pristupa izvoru.

**Šema replikacije**

Trenutno postoje dve šeme replikacije koje podržava MySql Replikaciju:

* Asinhrona replikacija
* Polusinhrona replikacija

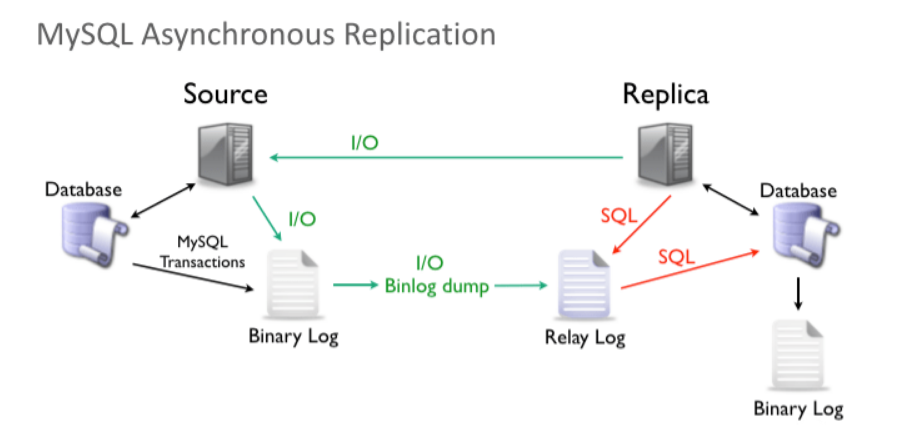
**Asinhrona replikacija**

MySql replikacija je podrazumevano asinhrona. Ovo je najstarija, najpopularnija i široko rasprostranjena šema replikacije. Sa asinhronom replikacijom, master upisuje događaje u svoj binarny log i podređeni ih zahtevaju kada su spremni. Ne postoji garancija da će bilo koji događaj ikada stići do nekog slave-a. To je slabo povezan odnos master-slave, gde:

* Master ne čeka Slave-a.
* Slave određuje koliko treba čitati i sa koje tačke u binarnom dnevniku.
* Slave može proizvoljno da zaostaje za masterom u čitanju ili primeni promena.

Ako master pada, transakcije koje je izvršio možda neće biti prenete nijednom slave-u. Shodno tome, prelazak sa glavnog na podređeni u ovom slučaju može dovesti do prebacivanja greške na server kome nedostaju transakcije u odnosu na master.

Asinhrona replikacija obezbeđuje niže kašnjenje u pisanju, pošto upis lokalno priznaje master pre nego što se upiše na slave. Odličan je za skaliranje čitanja jer dodavanje više replika ne utiče na kašnjenje replikacije. Dobri slučajevi upotrebe asinhrone replikacije uključuju primenu replika čitanja za skaliranje čitanja, živu rezervnu kopiju za oporavak od katastrofe i analitiku/izveštavanje.



Slika 1:Asinhrona replikacija

**Polusinhrona replikacija**

MySql takođe podržava polusinhronu replikaciju, gde master ne potvrđuje transakcije klijentu sve dok bar jedan slave ne kopira promenu u svoj relay log i prenese je na disk. Da bi se omogućila polusinhrona replikacija, potrebni su dodatni koraci za instalaciju plugin-ova i moraju biti omogućeni na određenom MySql master-u i slave-u.

Čini se da je polusinhrona replikacija dobro i praktično rešenje za mnoge slučajeve gde je važna visoka dostupnost i nedostatak gubitka podataka. Ali treba uzeti u obzir da polusinhrona replikacija ima uticaj na performanse zbog dodatnog povratnog puta i ne pruža jake garancije protiv gubitka podataka. Kada se commit uspešno vrati, poznato je da podaci postoje na najmanje dva mesta (na glavnom i najmanje jednom slave-u). Ako master izvrši, ali dođe do pada dok master čeka potvrdu od slave-a, moguće je da transakcija nije stigla nijednom slave-u. Ovo nije tako veliki problem jer se commit neće vratiti u aplikaciju u ovom slučaju. Zadatak aplikacije je da ponovo pokuša da izvrši transakciju u budućnosti. Ono što je važno imati na umu je da, kada je master otkazao i slave je unapređen, stari master ne može da se pridruži lancu replikacije. Pod nekim okolnostima ovo može dovesti do sukoba sa podacima na slave-ovima (kada se master srušio nakon što je slave primio događaj binarnog dnevnika, ali pre nego što je master dobio potvrdu od slave-a). Stoga je jedini siguran način odbaciti podatke na starom masteru i obezbediti ih od nule koristeći podatke sa novo unapređenog mastera.

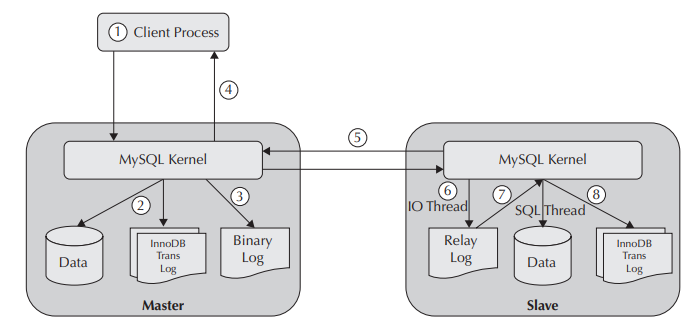
Dobar slučaj upotrebe za polusinhronu replikaciju je master rezervne kopije koji smanjuje uticaj kvara glavnog master-a minimiziranjem rizika od gubitka podataka.

**Globalni identifikator transakcije (GTID)**

Globalni identifikatori transakcija (GTID) predstavljeni su u MySql 5.6. GTID je jedinstveni identifikator kreiran i povezan sa svakom transakcijom izvršenom na serveru porekla (master). Ovaj identifikator je jedinstven ne samo za server na kome je nastao, već je jedinstven za sve servere u datom podešavanju replikacije. Postoji mapiranje jedan-na-jedan između svih transakcija i svih GTID-ova.

# **Rad replikacije**

Prvo ćemo pogledati ilustraciju na slici 2 i ukratko nabrojati korake a kasnije ćemo videti i detaljan opis rada replikacije.



Slika 2: Rad replikacije

1. MySql transakcija se pokreće na masteru (tačka 1).

2. Jedan ili više SQL izraza se primenjuju na master (tačka 2).

Prava primena fizičkog rezultata zavisi od storage engine-a za skladištenje koji se koristi. Generalno, bez obzira na mehanizam za skladištenje, operacija promene podataka se prvo beleži u okviru primenljivog memorijskog bafera. Za InnoDB, izraz je zabeležen u InnoDB evidenciji transakcija (InnoDB podatke na disk upisuje zasebna pozadinska nit). Za MyISAM, operacija je upisan direktno u primenljivu datoteku podataka tabele.

3. Po završetku transakcije, glavni binarni log beleži rezultat primenjenih DML ili DDL izraza (tačka 3). MySql podržava različite režime koji mogu da snime iskaz(e) ili se stvarne promene podataka.

4. Indikator uspeha se vraća klijentskom programu koji označava završetak transakcije (tačka 4).

5. Slave server otkriva da je došlo do promene u glavnom položaju binarnog loga (tačka 5).

6. Podređeni server prima promene (tj. proces povlačenja). i upisan u dnevnik slave relay od strane slave I/O niti (tačka 6)

7. Podređena SQL nit čita nove događaje iz relay log-a (tačka 7) i primenjuje sve izraze u transakciji (tačka 8). Ove promene mogu biti zabeležene kao izraz koji treba da se izvrši ili kao

fizička modifikacija reda.

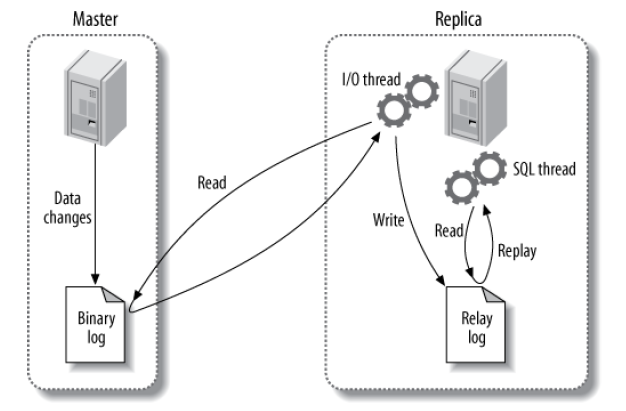
8. Indikator uspeha se vraća podređenoj replikaciji upravljanje kada se transakcija završi.

Ukratko, SQL transakcije se beleže u glavnom binarnom logu, a promena ovog log-a se koristi kao okidač za slave da povuče promena.

Na visokom nivou, replikacija je jednostavan proces koji se sastoji iz tri dela:

1. Master snima promene svojih podataka u svom binarnom logu. (Ovi zapisi se nazivaju događaji binarnog loga.)
2. Replika kopira događaje binarnog loga mastera u svoj binarni log (eng. relay log).
3. Replika reprodukuje događaje iz binarnog loga, primenjujući promene na svoje sopstvene podatke.

Svaki od ovih koraka je složen. Na slici 1 je detaljnije ilustrovan proces replikacije.



Slika 3: Proces replikacije

Prvi deo procesa je binarno logovanje na masteru. Neposredno pre nego što se završi svaka transakcija koja ažurira podatke na masteru, master snima promene u svom binarnom logu. MySQL serijski upisuje transakcije u binarnom logu, čak i ako su naredbe u transakcijama isprepletene tokom izvršenja. Nakon što upiše događaje u binarnom logu, master kaže mehanizmu za skladištenje da izvrši transakcije.

Sledeći korak je da replika kopira binarni log mastera na svoj sopstveni hard uređaj, u takozvani relay log (*eng. relay log*). Za početak, ona pokreće radnu nit, koja se naziva U/I radna nit *(eng. I/O slave thread).* U/I nit otvara uobičajenu klijentsku konekciju ka masteru, a zatim pokreće poseban *binlog dump* proces. *Binlog dump* proces čita događaje iz binarnog loga mastera. Ako sustigne mastera, on spava i čeka da master signalizira kada dođu novi događaji. U/I nit upisuje događaje u relay log replike.

SQL radna nit *(eng. SQL slave thread)* upravlja poslednjim delom procesa. Ova nit čita i reprodukuje događaje iz relay loga, ažurirajući tako podatke replike tako da se podudaraju sa podacima mastera. Sve dok ova nit drži korak sa U/I niti, relay log obično ostaje u kešu operativnog sistema. Događaji koje SQL nit izvršava mogu opciono da pređu u binarni log replike, što je korisno za scenarije koje ćemo spomenuti kasnije u radu.

Slika 1 prikazuje samo dve niti za replikaciju koje se izvršavaju na replici, ali takođe postoji i nit na masteru, kao bilo koja konekcija sa MySQL serverom, konekcija koju replika otvara ka masteru kako bi pokrenula nit na masteru.

Ova arhitektura replikacije razdvaja procese pribavljanja i reprodukovanja događaja na replici, što im omogućava da budu asinhroni. Odnosno, U/I nit može da radi nezavisno od SQL niti. Ona takođe postavlja ograničenja u procesu replikacije, od kojih je najvažnije da se replikacija izvršava serijski na replici. Ovo znači da ažuriranja koja se možda izvršavaju paralelno (u različitim nitima) na masteru ne mogu da budu paralelizovana na replici, jer se izvršavaju u jednoj niti. Postoje neka rešenja za ovo, ali većina korisnika i dalje podležu jednonitnom ograničenju.

# **Replikacione topologije**

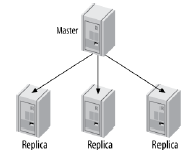
Možemo da postavimo MySQL replikaciju za skoro bilo koju konfiguraciju mastera i replika, uz ograničenje da data instanca MySQL replike može da ima samo jednog mastera. Mnoge složene topologije su moguće, ali čak i jednostavne mogu da budu veoma fleksibilne. Jedna topologija može da ima više različitih upotreba.

Već smo videli kako da podesimo master sa jednom replikom. U ovom odeljku ćemo videti neke druge uobičajene topologije i razmotriti njihove prednosti i mane. Osnovna pravila su:

* Instanca MySQL replike može da ima samo jednog mastera.
* Svaka replika mora da ima jedinstven ID servera.
* Master može da ima više replika.
* Replika može da propagira promene od mastera i da bude master drugim replikama, ako je omogućeno log\_slave\_updates.

## **Master i više replika**

Pored osnovne postavke dva servera master-replika koju smo već pokazali, master sa više replika je najjednostavnija replikaciona topologija. Zapravo, ona je jednostavna kao i osnovna postavka, jer replike ne interaguju međusobno, one su povezane samo sa masterom. Ovo je ilustrovano na slici 4.



Slika 4:Master i više replika

Ova konfiguracija je najkorisnija kada imamo malo upisa i mnogo čitanja. Možemo da propagiramo čitanja na bilo koji broj replika, sve dok replike ne opterećuju previše master ili mrežna propustnost od mastera do replika ne postane problem. Možemo da postavimo više replika odjednom ili da dodajemo replike po potrebi, korišćenjem istih koraka koje smo pokazali ranije.

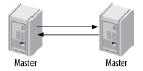
Iako je ovo veoma jednostavna topologija, ona zadovoljava mnoge potrebe, kao što su:

* Korišćenje različitih replika za različite uloge.
* Postavljanje jedne od replika kao master pripravnosti (čekanja), bez saobraćaja osim replikacije.
* Stavljanje jedne od replika u udaljeni data centar za oporavak od katastrofe.
* Vremensko odlaganje jedne ili više replika radi oporavka od katastrofe.
* Korišćenje jedne od replika za pravljenje backup-a ili za razvoj i postavljanje servera.

Jedan od razloga zašto je ova topologija popularna je taj što ona izbegava mnoge kompleksnosti koje dolaze sa drugim konfiguracijama. Na primer, lako je uporediti jednu repliku sa drugom u odnosu na poziciju binarnog loga na masteru, jer će sve biti iste. Drugim rečima, ako zaustavimo sve replike u istoj logičkoj tački u replikaciji, sve će one čitati sa iste fizičke pozicije u logovima mastera, Ovo je dobra osobina koja pojednostavljuje mnoge administrativne zadatke, kao što je promovisanje replike da bude master. Ovo svojstvo ima samo među braćama replika. Komplikovanije je poređenje pozicije logova između servera koji nisu u direktnoj master-replika relaciji ili nisu braća.

## **Master-master u aktivan-aktivan modu**

Replikacija master-master (poznata kao i dvosmerna replikacija) uključuje dva servera, oba konfigurisana i kao master i kao replika drugog – drugim rečina, par komastera. Postavka je prikazana na slici 5.



Slika 5:Master-master u aktivan-aktivan modu

Replikacija master-master u aktivan-aktivan mod se uglavnom koristi za specijalne namene. Jedna moguća upotreba je za geografski odvojene kancelarije, gde svaka kancelarija treba da ima sopstvenu kopiju podataku na kojoj može lokalno da upisuje.

Najveći problem sa takvom konfiguracijom je kako izaći na kraj sa konfliktnim promenama. Velika je lista mogućih problema prouzrokovanih posedovanjem dva mastera koji mogu da upisuju. Problemi obično nastaju kada upit menja isti red istovremeno na oba servera ili dodaje u tabelu sa AUTO\_INCREMENT kolonom istovremeno na oba servera.

MySQL 5.0 je dodao: podešavanja auto\_increment\_increment i auto\_increment\_offset. Ova podešavanja omogućavaju serverima da automatski generišu nekonfliktne vrednosti za INSERT upite. Međutim, dopuštanje upisa oba mastera je idalje veoma opasno. Ažuriranja koja se dešavaju u različitom redosledu na dve mašine mogu još izazvati da podaci postanu nesinhronizovani. Na primer, pretpostavimo da imamo jednu kolonu i jedan red u tabeli koji sarži vrednost 1. Sada pretpostavimo da se na prvom komasteru izvršava naredba kao na slici 6, a na drugom naredba kao na slici 7.



Slika 6

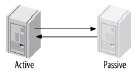


Slika 7

Jedan server će imati vrednost 4, a drugi 3, a ipak uopšte nema grešaka u replikaciji. To što podaci postaju nesinhronizovani je samo početak. Šta ako se zaustavi normalna replikacija sa greškom, ali aplikacije nastavljaju da upisuju na oba servera? Ne možemo samo da kloniramo jedan server od drugoga, jer svaki od njih će imati promene koje moramo da kopiramo drugom. Ovoj problem će se veoma teško rešiti. Ako pažljivo podesimo master-master aktivan-aktivan konfiguraciju, možda sa dobro particionisanim podacima i privilegijama, i ako stvarno znamo šta radimo, možemo da izbegnemo neke od ovih problema. Uopšteno, omogućavanje upisa na oba servera prouzrokuje više problema nego koristi.

## **Master-master u aktivan-pasivan modu**

Ovo je varijacija master-master replikacije koja izbegava zamke o kojima smo diskutovali i veoma je snažan način za dizajniranje sistema koji su tolerantni na otkaze i visoko dostupni. Glavna razlika je u tome što je jedan od servera read-only pasivan server, kao što je prikazano na slici 8.



Slika 8:Master-master u aktivan-pasivan modu

Ova konfiguracija nam omogućava da lako i jednostavno menjamo aktivne i pasivne uloge servera, jer su konfiguracije servera simetrične. Ovo nam omogućava lako prevazilaženje grešaka. Takođe nam omogućava održavanje, optimizaciju tabela, nadogradnju operativnog sistema (ili aplikacije ili hardvera) i obavljanje drugih zadataka bez ikakvih zastoja.

Na primer, pokretanje ALTER TABLE naredbe zaključava celu tabelu, blokirajući čitanja i upise u njoj. Ovo može da traje dugo i poremeti uslugu. Međutim, master-master konfiguracija nam omogućava da zaustavimo replikacione niti na aktivnom serveru (tako da ne obrađuje ažuriranja sa pasivnog servera), izmenimo tabelu na pasivnom serveru, zamenimo uloge i restartujemo replikaciju na prethodno aktivnom serveru. Taj server onda čita svoj relay log i izvršava istu ALTER TABLE naredbu.

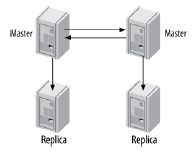
Master-master par ćemo konfigurisati izvršenjem sledećih koraka na oba servera, tako da oni imaju simetričnu konfiguraciju:

* Osigurati da serveri imaju potpuno iste podatke.
* Omogućiti binarno logovanje, izabrati jedinstvene ID-jeve servera i dodati replikacione naloge.
* Omogućiti logovanje ažuriranja replike. Ovo je važno za prevazilaženje i oporavak od greške.
* Opciono konfigurisati pasivan server samo za čitanje kako bi se sprečile promene koje mogu da budu u konfliktu sa promenama na aktivnom serveru.
* Startovati svaku instancu MySQL servera.
* Konfigurisati svaki server kao repliku drugog, počevši od novokreiranog binarnog loga.

Kada dođe do promene na aktivnom serveru, promena se upisuje u njegov binarni log i prenosi se preko replikacije u relay log pasivnog servera. Pasivni server izvršava upit i upisuje događaj u svom binarnom logu, jer je omogućeno log\_slave\_updates. Zatim aktivan server ignoriše događaj, jer se ID servera u događaju poklapa sa njegovim.

## **Master-master sa replikama**

U ovoj konfiguraciji se dodaje jedna ili više replika svakom ko-masteru, kao što je prikazano na slici 9.



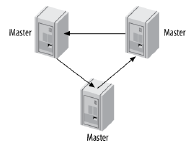
*Slika 9:*Master-master sa replikama

Prednost ove konfiguracije je dodatna redundantnost. U geografski distribuiranoj replikacionoj topologiji, ovo uklanja jedinstvenu tačku neuspeha na svakoj lokaciji. Takođe možemo da učitamo upite sa intenzivnim čitanjem u replike.

Ako lokalno koristimo master-master topologiju za brzo prevazilaženje grešaka, ova konfiguracija je još uvek korisna. Moguće je promovisanje jedne od replika za zamenu palog mastera, iako je malo složenije. Isto važi i za premeštanje jedne od replika da ukazuje na drugi master.

## **Prsten replikacija**

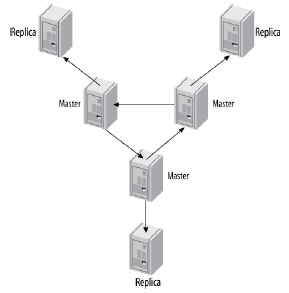
Master-master konfiguracija je samo specijalan slučaj prsten replikacione konfiguracije, koja je prikazana na slici 10. Prsten ima tri ili više mastera. Svaki server je replika servera pre njega u prstenu, a master servera nakon njega. Ova topologija se takođe naziva i kružna replikacija.



Slika 10:Prsten replikacija

Prsteni nemaju neke od ključnih prednosti master-master postavke, kao što su simetrična konfiguracija i lako prevazilaženje greške. Oni takođe upotpunosti zavise od svakog čvora u prstenu koji je dostupan, što mnogo povećava verovatnoću pada celog sistema. Ako uklonimo jedan od čvorova iz prstena, svi događaji replikacije koji su nastali na tom čvoru mogu da uđu u beskonačnu petlju: oni će zauvek kružiti kroz lanac servera, jer jedini server koji će filtrirati događaj na osnovu njegovog ID-a servera je server koji ga je kreirao. Generalno, prstene treba izbegavati.

Neke od rizika prsten replikacije možemo da ublažimo dodavanjem replika kako bismo obezbedili redundantnost na svakoj lokaciji, kao što je prikazano na slici 11. Ovo samo štiti od rizika od neuspeha servera. Gubitak struje ili bilo koji drugi problem koji utiče na bilo kakvu vezu između lokacija će idalje prekinuti ceo prsten.



Slika 11

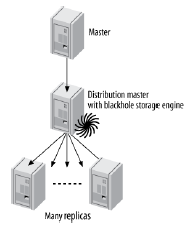
## 

## **Master, distribucioni master i replikacije**

Replike mogu dosta da opterete master ako ih ima mnogo. Svaka replika kreira novu nit na masteru, koja izvršava posebnu binlog dump komandu. Ova komanda čita podatke iz binarnog loga i šalje ih replici. Ovaj posao se ponavlja za svaku repliku.

Ako postoji mnogo replika i postoji posebno veliki događaj binarnog loga, opterećenje mastera može značajno da se poveća. Masteru čak može i da ponestane memorije i padne jer sve replike traže isti ogroman događaj u isto vreme. Sa druge strane, ako sve replike zahtevaju različite događaje binarnog loga koji više nisu u kešu fajl sistema, to može da izazove mnoštvo traženja diska, što može takođe da ometa performanse mastera.

Iz tog razloga, ako nam je potrebno mnogo replika, često je dobra ideja ukloniti učitavanje sa mastera i koristiti distribucioni master. Distribucioni master je replika čija je jedina namena da čita i servira binarne logove od mastera. Više replika može da se poveže sa distribucionim masterom, koji originalni master izoluje od opterećenja. Da bi uklonili posao izvršenja upita na distribucioni master, možemo da promenimo njegove tabele u Blackhole mehanizmu skladištenja, što je prikazano na slici 12.



Slika 12

Teško je reći tačno koliko replika master može da hendluje pre nego što mu je potreban distribucioni master. Grubo pravilo je da ako master radi skoro punim kapacitetom, ne treba staviti više od 10 replika na njemu. Ako postoji veoma malo aktivnosti upisa ili repliciramo samo deo tabela, master može da opslužuje mnogo više replika. Uz to, ne moramo da se ograničimo na samo jednog distribucionog mastera. Možemo da koristimo nekoliko ako treba da repliciramo ogroman broj replika ili čak možemo da koristimo piramidu distribucionih mastera.

Takođe možemo da koristimo distribucioni master za druge namene, kao što su primena filtera ili prepisivanje pravila u događaje binarnog loga. Ovo je mnogo efikasnije od ponavljanja logovanja, prepisivanja i filtriranja na svakoj replici.

Ako koristimo Blackhole tabele na distribucionom masteru, on će moći da uslužuje više replika nego što bi inače mogao. Distribucioni master će izvršavati upite, ali upiti će biti veoma jeftini, jer Blackhole tabele neće imati nikakve podatke. Mana Blackhole tabela je što one imaju greške, kao što je da u nekim okolnostima zaborave da stave autoinkrementalne ID-jeve u svoje binarne logove, tako da treba da budemo oprezni sa Blackhole tabelama ako ih koristimo.

Možemo da osiguramo da sve tabele na distribucionom masteru koriste Blackhole mehanizam skladištenja postavljanjem storage\_engine opcije servera, kao na slici 13. Ovo će uticati samo na CREATE TABLE naredbe koje eksplicitno ne specificiraju mehanizam skladištenja. Možemo da onemogućimo InnoDB pomoću opcije skip\_innodb, ali ne možemo da onemogućimo MyISAM.

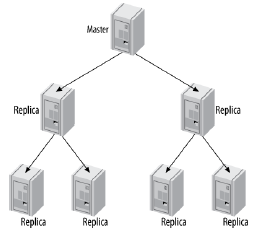


Slika 13

Drugi veliki nedostatak su potreškoće prilikom zamene mastera jednom od replika. Teško je promovisati jednu od replika na njegovo mesto, jer master osigurava da one skoro uvek imaju drugačije koordinate binarnog loga od originalnog mastera.

## **Stablo ili piramida**

Ako repliciramo master na velikom broju replika, upotreba dizajna piramide može da bude lakša za upravljanje, što je ilustrovano na slici 14.

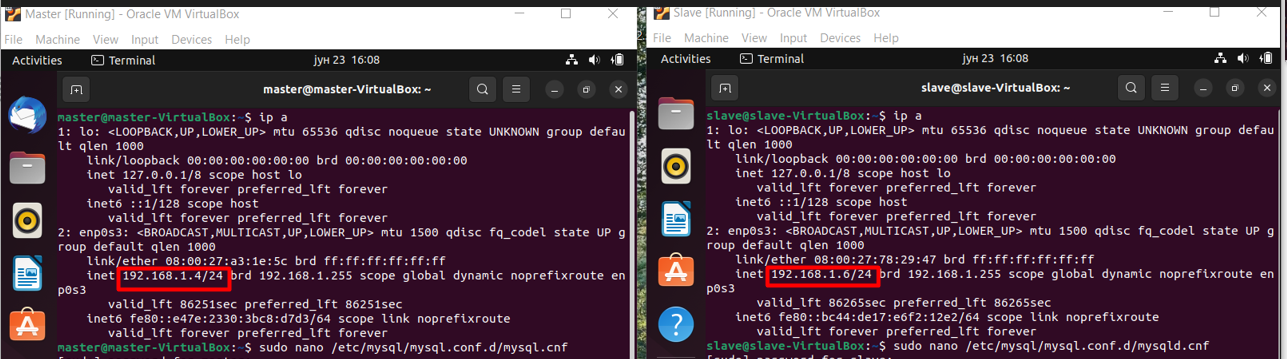


Slika 14:Stablo ili piramida

Prednost ovog dizajna je u tome što smanjuje opterećenje na masteru, kao što je to radio distribucioni master u prethodnom odeljku. Nedostatak je u tome što svaki neuspeh u srednjem nivou će uticati na više servera, što se ne bi desilo da su sve replike direktno povezane na master. Takođe, što više imamo srednjih nivoa, teže je i komplikovanije rešavanje neuspeha.

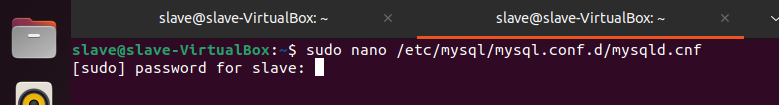
**Primer**

Sada ćemo videti kako da podesimo replikaciju zasnovanu na osnovu pozicije binary log-a. Prvo instaliramo Virtualbox na njoj Linux, tačnije u ovom primeru je korišćena verzija Ubuntu 20.4. Zarad ovog primera napravljene su virtuelne mašine koje su podešene i nakon toga je na svakoj od njih instaliran MySql. Nakon konfiguracije i instalacije na virtuelnim mašinama, prvo je pronađena adresa svake mašine pomoću naredbe **ip a,** a nakon togaurađeno je dodatna konfiguracija u okviru conf fajla.



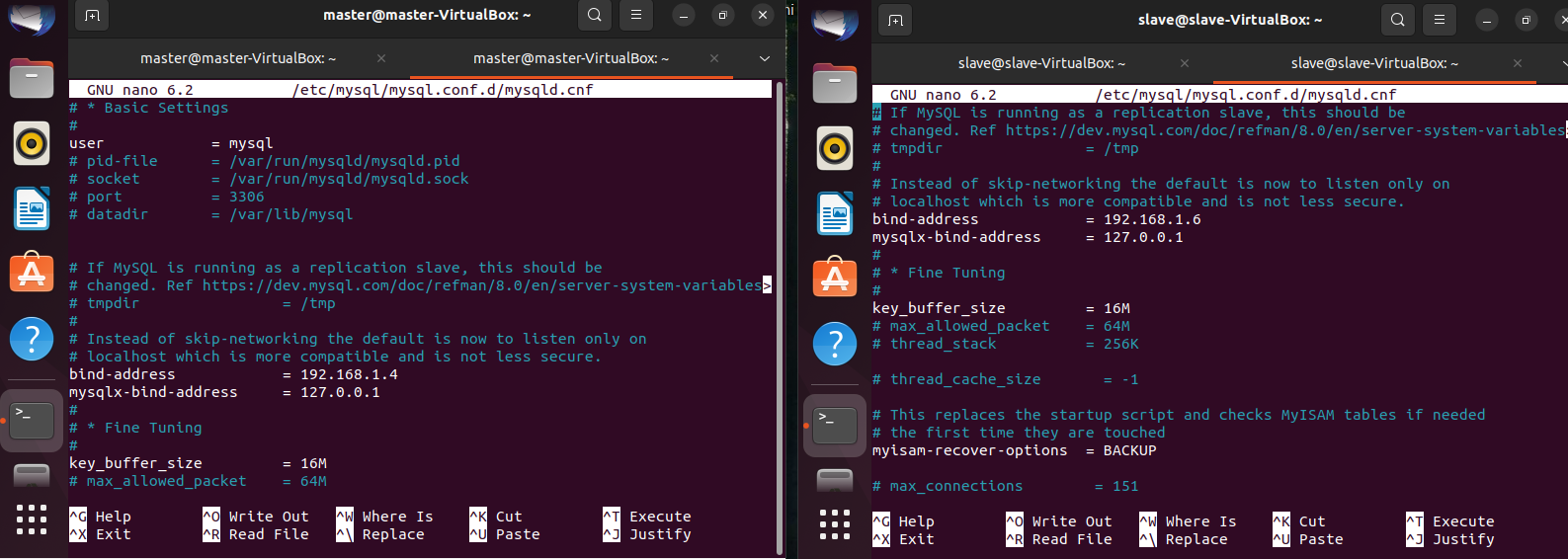
Slika 15: ip adrese mašina

Do conf fajla možemo doći pomoću naredne komande na slici.



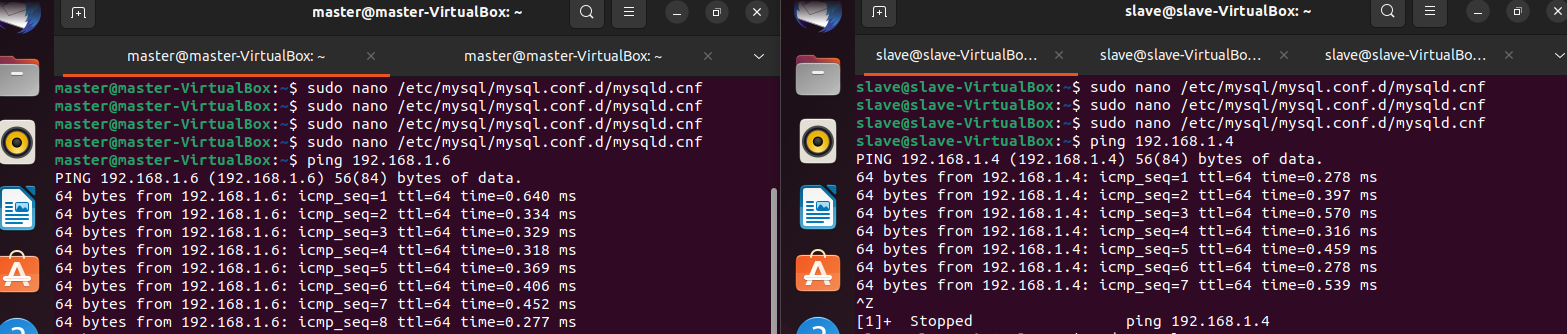
Slika 16: Naredba za prikaz conf fajla

Na slici možemo videti izgled config fajla. U njemu su dodatno promenjeni server-id, bind address i binlog\_do\_db (ovde navodimo ime baze koju ćemo kasnije replicirati).



Slika 17: conf fajl

Sada se proba ping-ovanje mašina da bismo videli da li komuniciraju jedna sa drugom.



Slika 18: ping-ovanje mašina

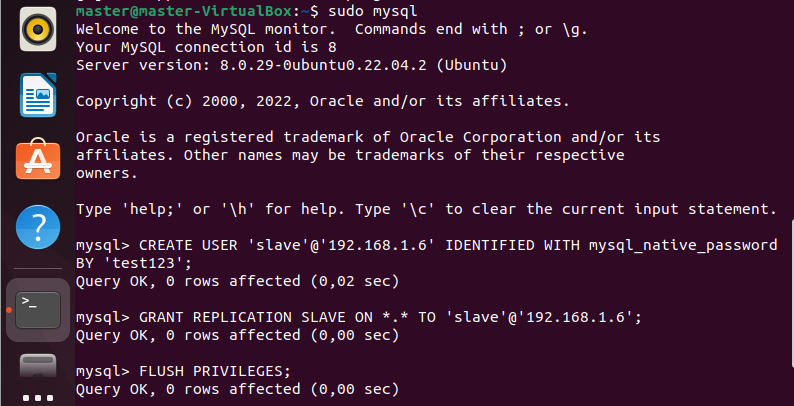
Nakon što podesimo master i slave i proverimo ping-ovanje istih, onda na master serveru kreiramo MySql korisnika, a to je slave server kao što je prikazano na Slici 19, komandom:

CREATE USER ‘<replicaUserName>@’<replicaIP>’ IDENTIFIED WITH my\_sql\_native\_password BY ‘<somePassword>’;

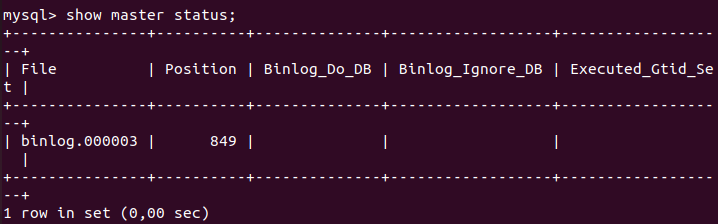
Ovom korisniku dozvoljavamo replikaciju narednom naredbom:

*GRANT REPLICATION SLAVE ON \*.\* TO ‘<imeKreiranogKorisnika>’@’<IPAdresa>’;*

Na kraju koristimo FLUSH PRIVILEGES da bismo očistili keš koji je prethodno sašuvan. Možemo proveriti da li sve radi kako treba naredbom show master status.

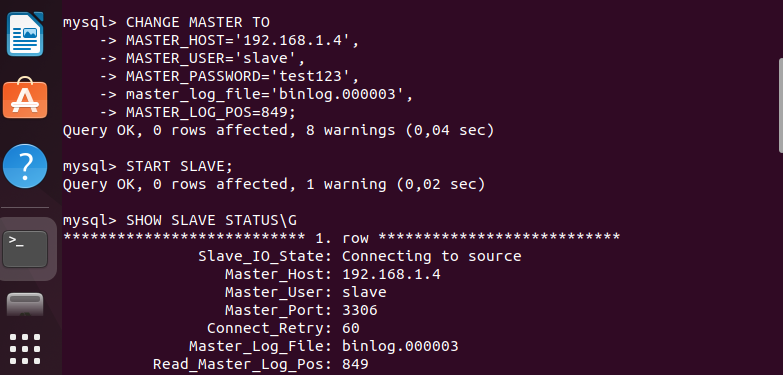


Slika 19: Kreiranje usera, grant replication, flush privileges

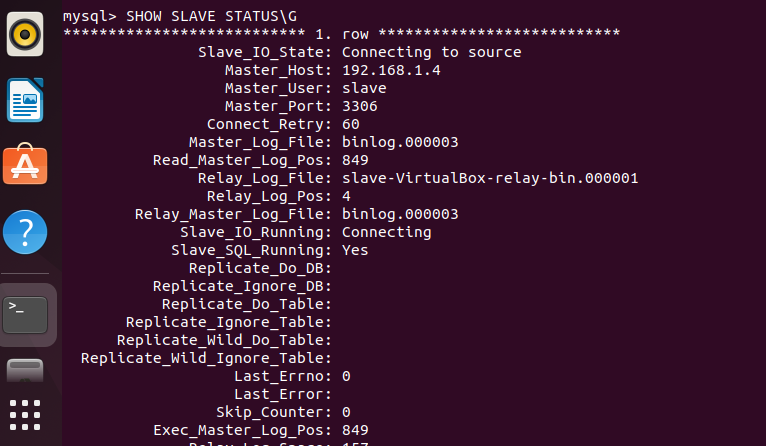


Slika 20: show master status

Nakon toga, podešavamo slave kao što možemo videti na narednoj slici i videti njegov status i da li je sve kako treba.



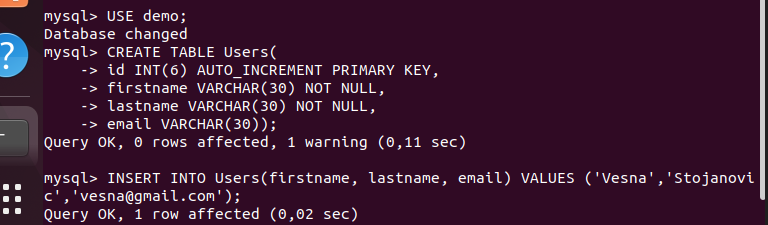
Slika 21: podešavanje jednog slave-a



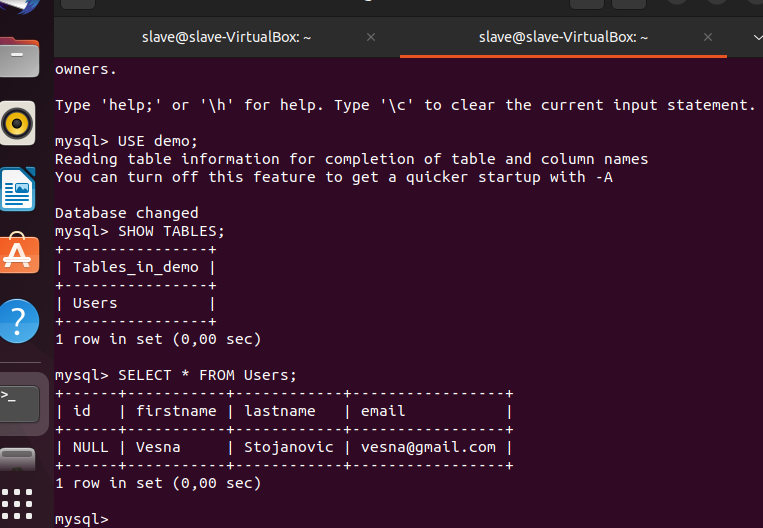
Slika 22:show slave status

Nakon toga možemo otključati tabele naredbom UNLOCK TABLES;

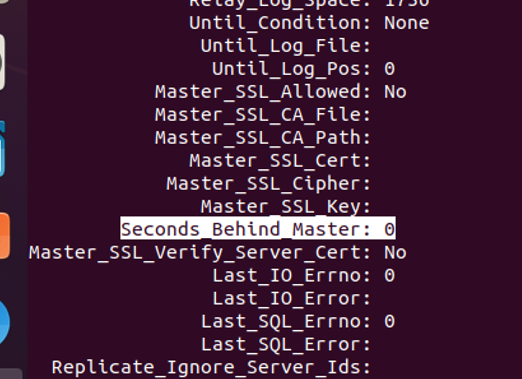
Potom kreiramo tabelu na masteru i popunimo podacima a kasnije vidimo da je replikacija već odrađena i da imamo tu istu tabelu i na slave-u i takođe vidimo kolko je kašnjenje za masterom.



Slika 23: Kreiranje tabele na masteru



Slika 24: replicirana tabela na slave-u



Slika 25: Kašnjenje za master-om

# **Administracija i održavanje replikacije**

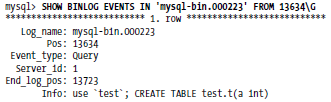
Kada jednom budemo postavili replikaciju, monitoring i administriranje naše replikacione topologije će biti redovan posao, bez obzira na to koliko imamo servera.

**Monitoring replikacije**

Replikacija povećava složenost monitoringa MySQL-a. Iako se replikacija zapravo odvija i na masteru i na replici, najveći deo posla se vrši na replici i upravo se tu javljaju najčešći problemi.

Na masteru možemo da koristimo naredbu SHOW MASTER STATUS kako bismo videli trenutnu poziciju binarnog loga mastera i konfiguraciju. Takođe možemo da pitamo mastera koji sve binarni logovi postoje na disku pomoću naredbe SHOW MASTER LOGS.

Takođe možemo da vidimo replikacione događaje u binarnom logu pomoću komande SHOW BINLOG EVENTS. Na primer, nakon pokretanja prethodne komande, kreirali smo tabelu na drugom neiskorišćenom serveru. Pošto smo znali da je ovo bila jedina naredba koja je promenila podatke, znali smo da je pomeraj naredbe u binarnom logu 13634, tako da smo mogli da vidimo to kao na slici .



## 

## **Merenje zaostajanja replikacije**

Jedna od najčešćih stvari koju treba da nadgledamo je koliko replika zaostaje iza mastera. Iako kolona Seconds\_behind\_master u SHOW SLAVE STATUS teorijski pokazuje zaostajanje replikacije, ovo nije uvek tačno, iz više razloga:

* Replika izračunava Seconds\_behind\_master tako što upoređuje trenutni timestamp servera sa snimljenim timstampom u binarnom logu događaja, tako da replika ne može čak ni da prijavi svoje zaostajanje ako se ne obrađuje upit.
* Replika će obično prijaviti NULL ako se procesi replikacije ne izvršavaju.
* Neke greške (na primer, neusklađeno podešavanje max\_allowed\_packet između mastera i replike ili nestabilne mreže) mogu da prekinu replikaciju ili zaustave replikacione niti, ali Seconds\_behind\_master će prijaviti 0 umesto da ukaže na grešku.
* Replika ponekad ne može da izračuna zaostajanje čak i ako se procesi replikacije izvšavaju. Ako se ovo dogodi, replika može da prijavi ili 0 ili NULL.
* Veoma duga transakcija može da izazove promenu prijavljenog kašnjenja. Na primer, ako imamo transakciju koja ažurira podatke, ostaje otvorena sat vremena, a zatim se izvrši, ažuriranje će ući u binarni log sat vremena nakon što se zapravo dogodilo. Kada replika obradi naredbu, ona će privremeno da prijavi da je sat vremena iza mastera, a zatim će skočiti nazad na nula sekundi.
* Ako zaostaje distribucioni master i ima svoje replike, replike će prijaviti da su zaostale nula sekundi, čak i ako postoji kašnjenje u odnosu na krajnjeg mastera.

Rešenje za ove probleme je zanemariti Seconds\_behind\_master i nadgledati zaostajanje replike nečim što možemo direktno posmatrati i meriti. Najbolje rešenje je heartbeat zapis, koji je timestamp koga ažuriramo jednom u sekundi na masteru. Kako bi izračunali kašnjenje, jednostavno možemo da oduzmemo heartbeat od trenutnog timestampa na replici. Ova metoda je imuna na sve probleme koje smo spomenuli i ima dodatnu prednost kreiranja timestampa koji pokazuje u kom trenutku su trenutno podaci replike.

## **Ponovna sinhronizacija replike sa masterom**

Tradicionalni savet za popravku nesinhronizovane replike je da se ona zaustavi i ponovo klonira iz mastera. Ako je nekonzistentna replika kritičan problem, potrebno je da je zaustavimo i uklonimo iz produkcije čim je pronađemo. Zatim možemo da je ponovo kloniramo ili povratimo iz mastera.

Mana ovog pristupa je kada imamo mnogo podataka. Ako možemo da saznamo koji se podaci razlikuju, verovatno možemo to da učinimo efikasnije nego ponovnim kloniranjem čitavog servera. I ako nekonzistentnost koju smo otkrili nije kritična, možda ćemo moći da ostavimo repliku online i ponovno sinhronizujemo samo pogođene podatke.

Najjednostavnija popravka je odložiti i ponovo učitati samo pogođene podatke pomoću mysqldump-a. Ovo može dobro da funkcioniše ako se podaci ne menjaju dok to radimo. Možemo jednostavno da zaključamo tabelu na masteru, odložimo tabelu, sačekamo da replika sustigne mastera i onda importujemo tabelu na replici. Moramo da sačekamo da replika sustigne mastera kako ne bismo uneli još nekonzistentnosti u druge tabele.

Iako ovo funkcioniše prihvatljivo za mnoge scenarije, često je nemoguće raditi ovo na zauzetom serveru. Takođe ima manu kada se manjaju podaci replike van replikacije. Promena podataka replike kroz replikaciju (pravljenjem promena na masteru) je obično najsigurnija tehnika, jer se izbegavaju uslovi trke i druga iznenađenja. Ako je tabela veoma velika ili je propustnost mreže ograničena, odlaganje i ponovno učitavanje je izuzetno skupo.

## **Promena mastera**

Često će biti potrebno da usmerimo repliku na novog mastera. Možda je došlo do kvara i mora da promovišemo repliku da bude master ili možda samo realociramo kapacitet. Bez obzira na razlog, moramo da obavestimo repliku o njenom novom masteru.

Kada je proces isplaniran, to je lako. Jednostavno treba da izvršimo komandu CHANGE MASTER TO na replici, korišćenjem odgovarajućih vrednosti, od kojih su većina opcione. Replika će odbaciti svoju trenutnu konfiguraciju i relay logove i počeće repliciranje sa novog mastera. Ona će takođe da ažurira master.info fajl novim parametrima, tako da će promena biti perzistentna tokom ponovnog pokretanja replike. Najteži deo ovog procesa je pronaći željenu poziciju na novom masteru, tako da replika počinje u istoj logičkoj poziciji u kojoj je stopirana na starom masteru.

Promovisanje replike da bude master je malo teže. Postoje dva osnovna scenarija za zamenu mastera jednom od njegovih replika. Prvi je kada je planirana promocija, a drugi kada je neplanirana.

### **Planirane promocije**

Promovisanje replike da bude master uključuje sledeće korake:

1. Stopirati upise na starom masteru.
2. Opciono dopustiti da njegove replike sustignu replikaciju.
3. Konfigurisati repliku da bude novi master.
4. Usmeriti replike i saobraćaj ka novom masteru, zatim omogućiti upise na njemu.

Međutim, problem su detalji. Moguća su nekoliko scenarija u zavisnosti od replikacione topologije. Na primer, koraci su malo drugačiji u master-master topologiji od koraka u master-replika postavci. Ovo su neki koraci koje ćemo verovatno morati da preduzmemo za većinu postavki:

1. Zaustaviti sve upise na trenutnom masteru. Možda ćemo morati da prisilimo sve klijentske programe da se zatvore.
2. Opciono zaustaviti sve aktivnosti upisa na masteru pomoću FLUSH TABLES WITH READ LOCK. Takođe možemo da postavimo da master bude samo za čitanje pomoću opcije read\_only. Od ovog trenutka, treba zabraniti sve upise na masteru koji će uskoro biti zamenjen, jer jednom kada više nije master, pisanje na njemu znači gubitak podataka. Međutim, postavka read\_only ne sprečava izvršenje postojećih transakcija. Treba ubiti sve otvorene transakcije.
3. Izabrati jednu od replika da bude novi master i osigurati da je ona upotpunosti sustigla replikaciju (odnosno dopustiti da završi izvršenje svih relay logova koje je pribavila od starog mastera).
4. Opciono proveriti da li novi master sadrži iste podatke kao i stari master.
5. Izvršiti STOP SLAVE na novom master.
6. Izvrštiti CHANGE MASTER TO MASTER\_HOST=’’, a zatim RESET SLAVE na novom masteru, kako bismo se diskonektovali od starog mastera i odbacili informacije o konekciji u njegovom master.info fajlu.
7. Treba saznati koordinate binarnog loga novog mastera pomoću SHOW MASTER STATUS.
8. Proveriti da li su sve ostale replike sustigle replikaciju.
9. Isključiti stari master.
10. U MySQL 5.1 i novijim verzijama, aktivirati događaje na novom masteru ako je potrebno.
11. Dopustiti klijente da se povežu sa novim masterom.
12. Izvršiti komandu CHANGE MASTER TO na svakoj replici, ukazujući na novog mastera. Korisititi koordinate binarnog loga koje smo pribavili iz SHOW MASTER STATUS.

### **Neplanirane promocije**

Ako master padne i moramo da promovišemo repliku da ga zameni, proces možda neće biti lak. Ako postoji samo jedna replika, samo ćemo iskoristiti repliku. Ali ako ih imamo više, moraćemo da uradimo nekoliko dodatnih koraka kako bismo promovisali repliku da bude novi master.

Takođe postoji i dodatni problem zbog potencijalno izbugljenih replikacionih događaja. Moguće je da neka ažuriranja koja su se dogodila na masteru još uvek nisu replicirana ni na jednoj od njegovih replika. Moguće je čak i da je naredba izvršena i onda poništena na masteru, ali nije poništena na replici – tako da bi replika zapravo bila ispred pozicije logičke replikacije mastera. Ako možemo da povratimo podatke mastera u nekom trenutku, možda ćemo moći da povratimo izgbljene naredbe i primenimo ih ručno.

Ovo je procedura za promociju replike u master-replike topologiji:

1. Odrediti koja replika ima najnovije podatke. Proveriti izlaz iz SHOW SLAVE STATUS na svakoj replici i izabrati onu čije su najnovije koordinate Master\_Log\_File/ Read\_Master\_Log\_Pos
2. Dopustiti da sve replike završe izvršenje relay logova koje su pribavile od starog mastera pre nego što je pao. Ako promenimo master replike pre nego što se izvrši relay log, odbaciće se preostali događaji loga i nećemo znati gde se zaustavio.
3. Izvrštiti korake 5-7 iz liste u predhodnom odeljku.
4. Uporediti Master\_Log\_File/ Read\_Master\_Log\_Pos koordinate svake replike sa koordinatama novog mastera.
5. Izvršiti korake 10-12 iz liste u predhodnom odeljku.

### **Pronalaženje željenih pozicija loga**

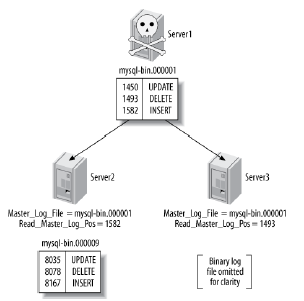
Ako neka replika nije na istoj poziciji kao novi master, moraćemo da pronađemo poziciju u binarnim logovima novog mastera koja odgovara poslednjem događaju koji je replika izvršila, i da je iskoristimo u CHANGE MASTER TO. Možemo koristiti mysqlbinlog alat za pregledavanje zadnjeg upita koji je replika izvršila i pronalaženje ovog istog upita u binarnom logu novog mastera.

Da bismo ovo ilustrovali, pretpostavićemo da događaji loga inkrementiraju ID i da je najnovija replika – novi master – upravo pribavio događaj 100 kada je stari master pao. Pretpostavimo da postoje još dve replike, replica2 i replica3; replica2 je pribavila događaj 99, a replica3 je pribavila događaj 98. Ako obe replike usmerimo na trenutnu poziciju binarnog loga novog mastera, one će početi da repliciraju događaj 101, tako da neće biti sinhronizovane. Međutim, sve dok je omogućen binarni log novog mastera pomoću log\_slave\_updates, možemo da pronađemo događaje 99 i 100 u binarnom logu novog mastera, tako da možemo da vratimo replike u konzistentno stanje.

Zbog restartovanja servera ili različitih konfiguracija isti događaji mogu da postoje u različitim bajt pomerajima na različitim serverima. Pronalaženje događaja može da bude sporo, ali obično nije teško. Samo treba da pregledamo poslednji događaj koji se izvršio na svakoj replici, tako što ćemo pokrenuti mysqlbinlog u binarnom logu ili relay logu replika. Zatim treba da pronađemo isti upit u binarnom logu novog mastera, takođe pomoću mysqlbinlog; ovo će odštampati pomeraj upita u bajtovima i možemo da koristimo taj pomeraj u CHANGE MASTER TO upitu.

Ovaj proces možemo da ubrzamo tako što ćemo oduzeti pomeraje u bajtovima na kojima su se zaustavili novi master i replika, što će nam dati razliku u njihovim pozicijama. Ako onda oduzmemo tu vrednost od trenutne pozicije binarnog loga novog mastera, velike su šanse da će željeni upit biti na toj poziciji. Na taj način smo pronašli poziciju na kojoj treba da startujemo repliku.

Prikazaćemo ovo na konkretnom primeru. Pretpostavimo da je server 1 master serverima 2 i 3, i on se srušio. Prema Master\_Log\_File/Read\_Master\_Log\_Pos u SHOW SLAVE, server 2 je uspeo da replicira sve događaje koji su bili u binarnom logu servera 1, ali server 3 nije ažuriran. Slika 26 ilustrira ovaj scenario.



Slika 26

Prema slici, možemo da budemo sigurni da je server 2 replicirao sve događaje u binarnom logu mastera jer se njegovi Master\_Log\_File i Read\_Master\_Log\_pos poklapaju sa poslednjim pozicijama na serveru 1. Stoga, možemo da promovišemo server 2 da bude novi master, a da server 3 bude njegova replika.

Server 3 se zaustavio na pomeraju 1493, što je 89 bajta iza pomeraja 1582, poslednje komande koju je server 2 izvršio. Server 2 trenutno upisuje na poziciju 8167 u svom binarnom logu. 8167 – 89 = 8078, tako da teoretski treba da ukažemo server 3 na taj pomeraj u logovima servera 2. Međutim, dobra je ideja da istražimo događaje loga oko ove pozicije i proverimo da server 2 stvarno ima ispravne događaje na tom pomeraju u svojim logovima. Možda će tamo biti nešto drugo, na primer zbog ažuriranja podataka koja su se desila samo na serveru 2. Pod pretpostavkom da su događaji isti nakon pregleda, komanda na slici 27 će server 3 prebaciti da bude replika servera 2.



Slika 27

Šta ako je server 1 završio izvršenje i logovanje još jednog događaja, izvan pomeraja 1582, kada se srušio? Pošto je server 2 pročitao i izvršio samo do pomeraja 1582, možda smo zauvek izgubili jedan događaj. Međutim, ako disk starog mastera nije oštećen, idalje možemo da povratimo događaj koji nedostaje iz njegovog binarnog loga pomoću mysqlbinlog.

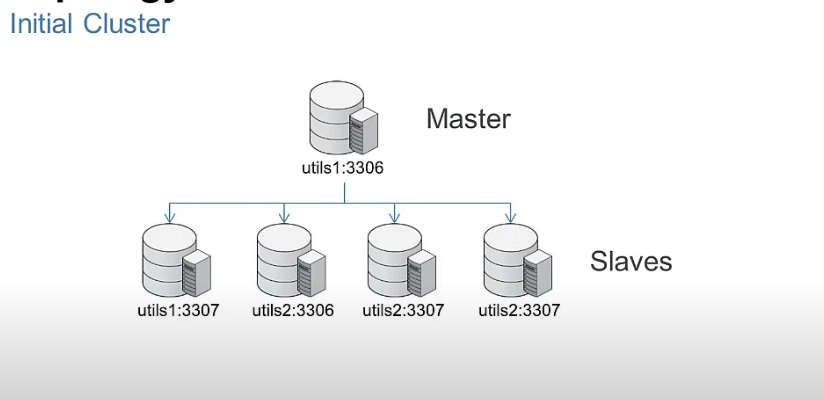
Ako je potrebno da povratimo događaje koji nedostaju iz starog mastera, to treba da uradimo nakon što postavimo novi master, ali pre nego što dopustimo klijentske konekcije. Na ovaj način nećemo morati da izvršavamo događaje koji nedostaju na svakoj replici; replikacija će se pobrinuti o tome za nas. Ako je master potpuno nedostupan, možda ćemo morati da sačekamo da to uradimo kasnije.

Varijacija ovog postupka je korišćenje pouzdanog načina za skladištenje fajlova binarnog loga mastera, kao što su SAN ili DRBD. Čak i ako se master upotpunosti srušio, i dalje ćemo imati njegove binarne logove. Možemo da postavimo server loga, da replike usmerimo ka njemu i zatim ih pustimo da dođu do tačke u kojoj se master srušio. Ovo olakšava promociju jedne od replika da postane novi master – ovo je suštinski isti postupak kao što smo pokazali za planiranu promociju.

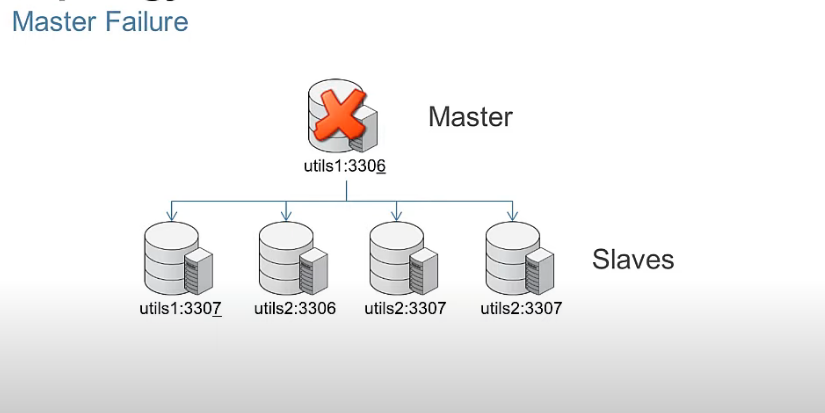
**Primer failover master-slave replikacije**

U potpuno automatizovanom režimu, uslužni programi će kontinuirano pratiti stanje glavnog i u slučaju njegovog kvara identifikovati najboljeg slave-a za unapređenje – podrazumevano će izabrati onaj koji je najažurniji, a zatim primeniti sve promene koji su dostupni na drugim slave-ovima, ali ne i na ovom pre nego što ga promovišu u novog master-a. Korisnik može poništiti ovo ponašanje (na primer, ograničavanjem koji od podređenih jedinica ispunjava uslove za unapređenje). Korisnik takođe može da se poveže sa sopstvenim programima koji će biti pokrenuti pre i posle prelaska na grešku (na primer, da obavesti aplikaciju).

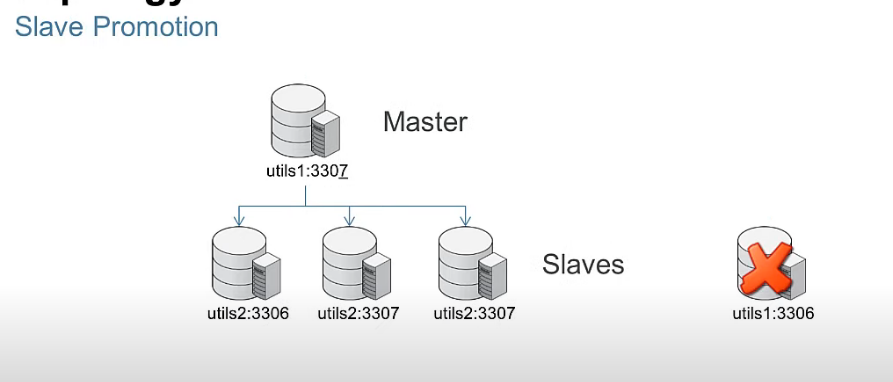
U režimu nadgledanja, uslužni program i dalje kontinuirano proverava dostupnost mastera i obaveštava korisnika ako ne uspe. Korisnik zatim izvršava jednu komandu da bi prešao na željeni slave.

****

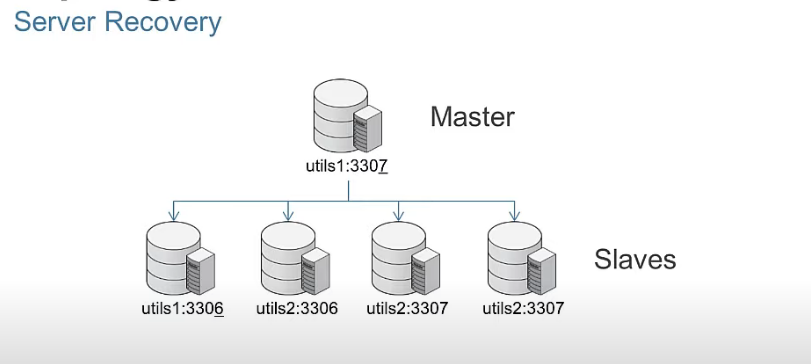
Slika 28: Inicijalni klaster

****

Slika 29: Master failure



Slika 30: Slave promotion

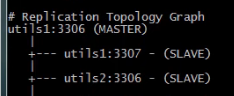


Slika 31: Server recovery

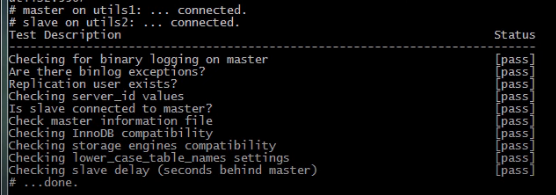
Prvo se proveri da li su svi slave-ovi i master konfigurisani pravilno.

Dodelimo im privilegije "grant all on \*.\* to root@'%' with grant option;".

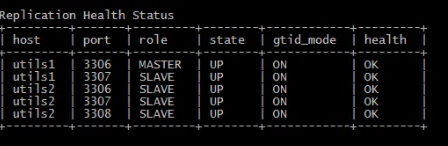
Možemo videti topologiju grafa na narednoj slici.



Slika 32: Toplogija replikacije



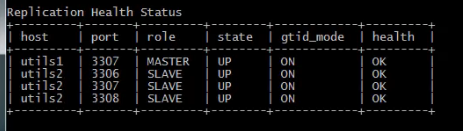
Slika 33: Nadgledanje replikacije



Slika 34: Automatsko nadgledanje

Nakon toga možemo izazvati pad master naredbom shutdown i nakon toga možemo videti kako slave preuzima ulogu mastera.

Ograničavanje koji slave-ovi ispunjavaju uslove za unapređenje u mastera podrazumevano je da se uzme najažurniji slave.



Slika 35: Slave postaje master

Ako želimo da dodamo oporavljeni MySQL server nazad u topologiju nakon ponovnog pokretanja neuspelog MySQL servera, on se može vratiti u topologiju kao podređeni novom masteru. Output iz mysqlfailover (još uvek radi) potvrđuje da je dodat.

Ako je bilo važno da se oporavljeni MySql server vrati kao master onda treba ručno pokrenuti promociju (nakon napuštanja mysqlfailover).

**Zaključak**

Cilj ovog rada je bio da predstavimo osnove replikacije kod MySQL-a, vidimo kako replikacija zapravo funkcioniše i objasnimo kako MySQL replicira podatke. U radu prošli smo kroz primere, kako podesiti master, slave, kako podesiti replikaciju korak po korak i videli smo kako izgleda sam proces replikacije jedne bae, odnosno tabele u okviru te baze. Osim toga videli smo kako slave preuzima ulogu master kada se desi pad masters i kako master nakon toga opet može biti dodat u toplogoju replikacije. U radu smo objasnili nekoliko replikacionih topologija koje mogu da imaju više različitih upotreba. Takođe objašnjen je monitoring i administriranje replikacione topologije, jer je to takođe veoma bitno u radu pogotovo ukoliko imamo više servera. Ovim radom je prikazan značaj replikacija i videli smo kako možemo sačuvati podatke čak i u slučaju pada servera što je veoma bitno.

**Literatura**

<https://hevodata.com/learn/mysql-master-slave-replication/>

High performance MySQL - Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko

<https://dev.mysql.com/doc/refman/8.0/en/replication.html>

<https://dev.mysql.com/doc/mysql-replication-excerpt/5.6/en/replication-howto.html>

<https://www.digitalocean.com/community/tutorials/how-to-set-up-replication-in-mysql>

<https://www.educative.io/answers/what-is-mysql-replication>

<https://www.toptal.com/mysql/mysql-master-slave-replication-tutorial>

<https://kovidrathee.medium.com/mysql-replication-primer-980782e9054c>

<https://severalnines.com/database-blog/introduction-failover-mysql-replication-101-blog>

<https://www.digitalocean.com/community/tutorials/how-to-set-up-replication-in-mysql#step-5-%E2%80%94-configuring-the-replica-database>